

# **webMethods Integration Server Built- In Services Reference**

Version 9.10

April 2016

This document applies to webMethods Integration Server Version 9.10 and to all subsequent releases.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Copyright © 2007-2017 Software AG, Darmstadt, Germany and/or Software AG USA Inc., Reston, VA, USA, and/or its subsidiaries and/or its affiliates and/or their licensors.

The name Software AG and all Software AG product names are either trademarks or registered trademarks of Software AG and/or Software AG USA Inc. and/or its subsidiaries and/or its affiliates and/or their licensors. Other company and product names mentioned herein may be trademarks of their respective owners.

Detailed information on trademarks and patents owned by Software AG and/or its subsidiaries is located at <http://softwareag.com/licenses>.

Use of this software is subject to adherence to Software AG's licensing conditions and terms. These terms are part of the product documentation, located at <http://softwareag.com/licenses> and/or in the root installation directory of the licensed product(s).

This software may include portions of third-party products. For third-party copyright notices, license terms, additional rights or restrictions, please refer to "License Texts, Copyright Notices and Disclaimers of Third Party Products". For certain specific third-party license restrictions, please refer to section E of the Legal Notices available under "License Terms and Conditions for Use of Software AG Products / Copyright and Trademark Notices of Software AG Products". These documents are part of the product documentation, located at <http://softwareag.com/licenses> and/or in the root installation directory of the licensed product(s).

## Table of Contents

<b>About this Guide</b> .....	<b>21</b>
Document Conventions.....	25
Online Information.....	26
<b>ART Folder</b> .....	<b>27</b>
Summary of Elements in this Folder.....	28
pub.art.listRegisteredAdapters.....	31
pub.art.connection:disableConnection.....	31
pub.art.connection:enableConnection.....	32
pub.art.connection:getConnectionStatistics.....	32
pub.art.connection:listAdapterConnections.....	33
pub.art.connection:queryConnectionState.....	34
pub.art.listener:disableListener.....	34
pub.art.listener:enableListener.....	35
pub.art.listener:listAdapterListeners.....	36
pub.art.listener:queryListenerState.....	37
pub.art.listener:resumeListener.....	37
pub.art.listener:setListenerNodeConnection.....	38
pub.art.listener:suspendListener.....	38
pub.art.notification:disableListenerNotification.....	39
pub.art.notification:disablePollingNotification.....	39
pub.art.notification:disablePublishEvents.....	40
pub.art.notification:enableListenerNotification.....	40
pub.art.notification:enablePollingNotification.....	41
pub.art.notification:enablePublishEvents.....	41
pub.art.notification:listAdapterListenerNotifications.....	42
pub.art.notification:listAdapterPollingNotifications.....	42
pub.art.notification:queryListenerNotificationState.....	43
pub.art.notification:queryPollingNotificationState.....	44
pub.art.notification:resumePollingNotification.....	45
pub.art.notification:setListenerNotificationNodeListener.....	46
pub.art.notification:setPollingNotificationNodeConnection.....	46
pub.art.notification:suspendPollingNotification.....	47
pub.art.service:listAdapterServices.....	47
pub.art.service:setAdapterServiceNodeConnection.....	48
pub.art.transaction:commitTransaction.....	49
pub.art.transaction:rollbackTransaction.....	50
pub.art.transaction:setTransactionTimeout.....	50
pub.art.transaction:startTransaction.....	51
<b>Assets Folder</b> .....	<b>53</b>

Summary of Elements in this Folder.....	54
pub.assets:getChecksums.....	54
<b>Cache Folder.....</b>	<b>57</b>
About Checkpoint Restart.....	58
Summary of Elements in this Folder.....	59
pub.cache:containsKey.....	62
pub.cache:get.....	63
pub.cache:getAll.....	64
pub.cache:getKeys.....	65
pub.cache:put.....	66
pub.cache:putAll.....	67
pub.cache:remove.....	69
pub.cache:removeAll.....	70
pub.cache:search.....	70
pub.cache.admin:clearAllCaches.....	75
pub.cache.admin:clearCache.....	76
pub.cache.admin:disableCache.....	77
pub.cache.admin:enableCache.....	78
pub.cache.admin:evictExpiredElements.....	79
pub.cache.admin:isCacheDisabled.....	80
pub.cache.atomic:putIfAbsent.....	81
pub.cache.atomic:remove.....	82
pub.cache.atomic:replace.....	83
pub.cache.atomic:replaceIfKeyExists.....	84
pub.cache.bulk:isClusterBulkLoadEnabled.....	85
pub.cache.bulk:isNodeBulkLoadEnabled.....	86
pub.cache.bulk:setNodeBulkLoadEnabled.....	87
pub.cache.bulk:waitUntilClusterBulkLoadComplete.....	88
pub.cache.lock:acquireLock.....	88
pub.cache.lock:isLockedByCurrentThread.....	90
pub.cache.lock:releaseLock.....	91
<b>Client Folder.....</b>	<b>93</b>
Summary of Elements in this Folder.....	94
pub.client:ftp.....	98
pub.client.ftp:append.....	102
pub.client.ftp:cd.....	103
pub.client.ftp:cdls.....	103
pub.client.ftp:delete.....	104
pub.client.ftp:dir.....	105
pub.client.ftp:get.....	105
pub.client.ftp:getCompletedNotification.....	108
pub.client.ftp:login.....	109
pub.client.ftp:logout.....	112
pub.client.ftp:ls.....	112

---

pub.client.ftp:mdelete.....	113
pub.client.ftp:mget.....	114
pub.client.ftp:mput.....	115
pub.client.ftp:put.....	116
pub.client.ftp:putCompletedNotification.....	117
pub.client.ftp:quote.....	118
pub.client.ftp:rename.....	119
pub.client.ftp:sessioninfo.....	119
pub.client.http.....	121
pub.client.ldap:add.....	132
pub.client.ldap:bind.....	134
pub.client.ldap:cancelNotification.....	135
pub.client.ldap:compare.....	136
pub.client.ldap:delete.....	137
pub.client.ldap:modify.....	138
pub.client.ldap:registerNotification.....	140
pub.client.ldap:rename.....	142
pub.client.ldap:search.....	143
pub.client.oauth:executeRequest.....	145
pub.client.sftp:cd.....	149
pub.client.sftp:chgrp.....	149
pub.client.sftp:chmod.....	150
pub.client.sftp:chown.....	151
pub.client.sftp:get.....	151
pub.client.sftp:login.....	152
pub.client.sftp:logout.....	153
pub.client.sftp:ls.....	154
pub.client.sftp:mkdir.....	155
pub.client.sftp:put.....	155
pub.client.sftp:pwd.....	156
pub.client.sftp:rename.....	157
pub.client.sftp:rm.....	157
pub.client.sftp:rmdir.....	158
pub.client.sftp:symlink.....	158
pub.client.smtp.....	159
pub.client.soapClient.....	163
pub.client.soapHTTP.....	180
pub.client.soapRPC.....	183
<b>Date Folder.....</b>	<b>189</b>
Pattern String Symbols.....	190
Time Zones.....	191
Examples.....	193
Notes on Invalid Dates.....	194
Summary of Elements in this Folder.....	194

pub.date:calculateDateDifference.....	195
pub.date:compareDates.....	196
pub.date:currentNanoTime.....	197
pub.date:dateBuild.....	198
pub.date:dateTimeBuild.....	199
pub.date:dateTimeFormat.....	201
pub.date:elapsedNanoTime.....	202
pub.date:formatDate.....	203
pub.date:getCurrentDate.....	204
pub.date:getCurrentDateString.....	204
pub.date:getWorkingDays.....	205
pub.date:incrementDate.....	206
<b>Db Folder.....</b>	<b>209</b>
Summary of Elements in this Folder.....	210
pub.db:call.....	212
pub.db:clearTransaction.....	214
pub.db:close.....	215
pub.db:closeAll.....	216
pub.db:commit.....	216
pub.db:connect.....	217
pub.db:delete.....	219
pub.db:execSQL.....	221
pub.db:getProclInfo.....	225
pub.db:getProcs.....	227
pub.db:getTableInfo.....	228
pub.db:getTables.....	230
pub.db:insert.....	231
pub.db:query.....	233
pub.db:rollback.....	235
pub.db:startTransaction.....	236
pub.db:update.....	237
<b>Document Folder.....</b>	<b>239</b>
Summary of Elements in this Folder.....	240
pub.document:bytesToDocument.....	241
pub.document:deleteDocuments.....	242
pub.document:documentListToDocument.....	242
pub.document:documentToBytes.....	244
pub.document:documentToDocumentList.....	246
pub.document:documentToXMLValues.....	248
pub.document:groupDocuments.....	248
pub.document:insertDocument.....	250
pub.document:searchDocuments.....	250
pub.document:sortDocuments.....	251
pub.document:XMLValuesToDocument.....	253

<b>Event Folder.....</b>	<b>255</b>
Summary of Elements in this Folder.....	256
pub.event:addSubscriber.....	260
pub.event:alarm.....	264
pub.event:alarmInfo.....	265
pub.event:audit.....	265
pub.event:auditError.....	266
pub.event:auditErrorInfo.....	268
pub.event:auditInfo.....	269
pub.event:callstackItem.....	270
pub.event:deleteSubscriber.....	271
pub.event.eda:event.....	272
pub.event.eda:eventToDocument.....	274
pub.event.eda:schema_event.....	276
pub.event:error.....	276
pub.event:errorInfo.....	277
pub.event:exception.....	277
pub.event:exceptionInfo.....	279
pub.event:gdEnd.....	280
pub.event:gdEndInfo.....	281
pub.event:gdStart.....	281
pub.event:gdStartInfo.....	282
pub.event:getEventTypes.....	283
pub.event:getSubscribers.....	284
pub.event:jmsReceiveErrorEvent.....	285
pub.event:jmsSendErrorEvent.....	287
pub.event:journal.....	288
pub.event:journalInfo.....	290
pub.event:modifySubscriber.....	291
pub.event.nerv:eventToDocument.....	295
pub.event.nerv:send.....	296
pub.event.nerv:subscribe.....	299
pub.event.nerv:unsubscribe.....	302
pub.event:portStatus.....	303
pub.event:portStatusInfo.....	303
pub.event:reloadEventManagerSettings.....	304
pub.event:replication.....	304
pub.event:replicationInfo.....	305
pub.event.routing:send.....	306
pub.event.routing:eventAcknowledgement.....	308
pub.event.routing:subscribe.....	309
pub.event.routing:unsubscribe.....	311
pub.event:saveEventManagerSettings.....	313
pub.event:security.....	313

pub.event:securityInfo.....	315
pub.event:sessionEnd.....	316
pub.event:sessionEndInfo.....	317
pub.event:sessionExpire.....	318
pub.event:sessionExpireInfo.....	318
pub.event:sessionStart.....	319
pub.event:sessionStartInfo.....	320
pub.event:stat.....	320
pub.event:statInfo.....	322
pub.event:txEnd.....	324
pub.event:txEndInfo.....	324
pub.event:txStart.....	325
pub.event:txStartInfo.....	326
<b>File Folder.....</b>	<b>327</b>
File Access Control Configuration for the pub.file Services.....	328
Parameter Settings.....	328
Summary of Elements in this Folder.....	330
pub.file:bytesToFile.....	330
pub.file:checkFileExistence.....	331
pub.file:copyFile.....	332
pub.file:deleteFile.....	333
pub.file:getFile.....	333
pub.file:listFiles.....	335
pub.file:moveFile.....	336
pub.file:readerToFile.....	337
pub.file:streamToFile.....	338
pub.file:stringToFile.....	339
<b>Flat File Folder.....</b>	<b>341</b>
Summary of Elements in the Flat File Folder.....	342
pub.flatFile:convertToString.....	344
pub.flatFile:convertToValues.....	349
pub.flatFile:FormatService.....	356
pub.flatFile:getSupportedEncodings.....	359
pub.flatFile.generate:createDocumentType.....	359
pub.flatFile.generate:createFFDictionary.....	360
pub.flatFile.generate:deleteFFDictionary.....	360
pub.flatFile.generate:deleteFFDictionaryEntry.....	361
pub.flatFile.generate:deleteFFSchema.....	362
pub.flatFile.generate:FFDictionary.....	363
pub.flatFile.generate:FFSchema.....	365
pub.flatFile.generate:findDependants.....	368
pub.flatFile.generate:findReferences.....	369
pub.flatFile.generate:getFFDictionaryAsXML.....	369
pub.flatFile.generate:getFFDictionaryEntryAsXML.....	370

pub.flatFile.generate:getFFSchemaAsXML.....	371
pub.flatFile.generate:listFFDictionaryEntries.....	371
pub.flatFile.generate:saveXMLAsFFDictionary.....	372
pub.flatFile.generate:saveXMLAsFFSchema.....	374
pub.flatFile.generate:updateFFDictionaryEntryFromXML.....	375
<b>Flow Folder.....</b>	<b>379</b>
Summary of Elements in this Folder.....	380
pub.flow.clearPipeline.....	382
pub.flow.debugLog.....	382
pub.flow.getCallingService.....	384
pub.flow.getLastError.....	384
pub.flow.getRetryCount.....	385
pub.flow.getSession.....	386
pub.flow.getTransportInfo.....	386
pub.flow.invokeService.....	387
pub.flow.iterator.....	388
pub.flow.restorePipeline.....	390
pub.flow.restorePipelineFromFile.....	391
pub.flow.savePipeline.....	392
pub.flow.savePipelineToFile.....	392
pub.flow.setCustomContextID.....	394
pub.flow.setResponse.....	395
pub.flow.setResponse2.....	397
pub.flow.setResponseCode.....	399
pub.flow.setResponseHeader.....	399
pub.flow.setResponseHeaders.....	402
pub.flow.throwExceptionForRetry.....	403
pub.flow.tracePipeline.....	404
pub.flow.transportInfo.....	404
<b>Hashtable Folder.....</b>	<b>415</b>
Summary of Elements in this Folder.....	416
pub.hashtable.containsKey.....	416
pub.hashtable.createHashtable.....	417
pub.hashtable.get.....	417
pub.hashtable.listKeys.....	417
pub.hashtable.put.....	418
pub.hashtable.remove.....	418
pub.hashtable.size.....	419
<b>IO Folder.....</b>	<b>421</b>
Summary of Elements in this Folder.....	422
pub.io.bytesToStream.....	423
pub.io.close.....	424
pub.io.createByteArray.....	424

pub.io:mark.....	425
pub.io:markSupported.....	426
pub.io:read.....	427
pub.io:readAsString.....	427
pub.io:readerToString.....	428
pub.io:reset.....	429
pub.io:skip.....	429
pub.io:streamToBytes.....	430
pub.io:streamToReader.....	431
pub.io:streamToString.....	431
pub.io:stringToReader.....	432
pub.io:stringToStream.....	432
<b>JDBC Folder.....</b>	<b>433</b>
Summary of Elements in this Folder.....	434
pub.jdbc:getPoolInfo.....	434
<b>JMS Folder.....</b>	<b>437</b>
Summary of Elements in This Folder.....	438
pub.jms:acknowledge.....	439
pub.jms:batchTriggerSpec.....	440
pub.jms:createConsumer.....	440
pub.jms:documentResolverSpec.....	444
pub.jms:JMSMessage.....	445
pub.jms:receive.....	449
pub.jms:reply.....	454
pub.jms:send.....	461
pub.jms:sendAndWait.....	473
pub.jms:sendBatch.....	492
pub.jms:triggerSpec.....	502
pub.jms:waitForReply.....	502
pub.jms.wmjms:receiveStream.....	508
pub.jms.wmjms:sendStream.....	510
<b>JSON Folder.....</b>	<b>513</b>
Data Type Mapping.....	514
Summary of Elements in This Folder.....	515
pub.json:documentToJSONString.....	515
pub.json:jsonStreamToDocument.....	517
pub.json:jsonStringToDocument.....	518
<b>List Folder.....</b>	<b>521</b>
Summary of Elements in this Folder.....	522
pub.list:addItemToVector.....	522
pub.list:appendToDocumentList.....	523
pub.list:appendToStringList.....	524

---

pub.list:sizeofList.....	525
pub.list:stringListToDocumentList.....	525
pub.list:vectorToArray.....	526
<b>Math Folder.....</b>	<b>529</b>
Summary of Elements in this Folder.....	530
pub.math:absoluteValue.....	532
pub.math:addFloatList.....	532
pub.math:addFloats.....	533
pub.math:addIntList.....	534
pub.math:addInts.....	535
pub.math:addObjects.....	535
pub.math:divideFloats.....	536
pub.math:divideInts.....	537
pub.math:divideObjects.....	538
pub.math:max.....	538
pub.math:min.....	539
pub.math:multiplyFloatList.....	539
pub.math:multiplyFloats.....	540
pub.math:multiplyIntList.....	541
pub.math:multiplyInts.....	542
pub.math:multiplyObjects.....	543
pub.math:randomDouble.....	543
pub.math:roundNumber.....	544
pub.math:subtractFloats.....	544
pub.math:subtractInts.....	546
pub.math:subtractObjects.....	546
pub.math:toNumber.....	547
<b>Metadata Folder.....</b>	<b>549</b>
Summary of Elements in this Folder.....	550
pub.metadata.assets:publishPackages.....	550
<b>MIME Folder.....</b>	<b>553</b>
Summary of Elements in this Folder.....	554
pub.mime:addBodyPart.....	555
pub.mime:addMimeHeader.....	559
pub.mime:createMimeData.....	561
pub.mime:getBodyPartContent.....	564
pub.mime:getBodyPartHeader.....	565
pub.mime:getContentType.....	567
pub.mime:getEnvelopeStream.....	568
pub.mime:getMimeHeader.....	569
pub.mime:getNumParts.....	571
pub.mime:getPrimaryContentType.....	571
pub.mime:getSubContentType.....	572

pub.mime:mergeHeaderAndBody.....	573
<b>OAuth Folder.....</b>	<b>575</b>
.....	
Summary of Elements in this Folder.....	576
pub.oauth:authorize.....	576
pub.oauth:getAccessToken.....	578
pub.oauth:refreshAccessToken.....	580
<b>Packages Folder.....</b>	<b>583</b>
Summary of Elements in this Folder.....	584
pub.packages:activatePackage.....	584
pub.packages:backupPackage.....	585
pub.packages:disablePackage.....	586
pub.packages:enablePackage.....	587
pub.packages:installPackage.....	587
pub.packages:recoverPackage.....	589
pub.packages:reloadPackage.....	590
pub.packages.hotdeployment:cancel.....	590
<b>Publish Folder.....</b>	<b>593</b>
Summary of Elements in this Folder.....	594
pub.publish:deliver.....	595
pub.publish:deliverAndWait.....	598
pub.publish:documentResolverSpec.....	602
pub.publish:envelope.....	604
pub.publish:getRedeliveryCount.....	609
pub.publish:publish.....	610
pub.publish:publishAndWait.....	613
pub.publish:reply.....	618
pub.publish:syncToBroker.....	620
pub.publish:syncToProvider.....	621
pub.publish:waitForReply.....	622
pub.publish.notification:error.....	624
<b>Remote Folder.....</b>	<b>627</b>
Summary of Elements in this Folder.....	628
pub.remote:invoke.....	629
pub.remote.gd:end.....	631
pub.remote.gd:getStatus.....	631
pub.remote.gd:invoke.....	632
pub.remote.gd:restart.....	633
pub.remote.gd:retrieve.....	633
pub.remote.gd:send.....	634
pub.remote.gd:start.....	635
pub.remote.gd:submit.....	635

<b>Replicator Folder.....</b>	<b>637</b>
Summary of Elements in this Folder.....	638
pub.replicator:addReleaseRegistryEntry.....	639
pub.replicator:deleteReleaseRegistryEntry.....	640
pub.replicator:distributeViaFTP.....	641
pub.replicator:distributeViaSvcPull.....	642
pub.replicator:distributeViaSvcPush.....	643
pub.replicator:generateReplicationEvent.....	643
pub.replicator:getLocalReleasedList.....	643
pub.replicator:getRemoteReleasedList.....	644
pub.replicator:notifyPackageRelease.....	645
pub.replicator:packageCreation.....	646
<b>Report Folder.....</b>	<b>649</b>
Summary of Elements in this Folder.....	650
pub.report:runFileTemplate.....	650
pub.report:runFileTemplateOnPipe.....	651
pub.report:runStringTemplate.....	651
pub.report:runStringTemplateOnPipe.....	652
pub.report:runTemplate.....	652
pub.report:runTemplateOnPipe.....	653
<b>Scheduler Folder.....</b>	<b>655</b>
Summary of Elements in this Folder.....	656
pub.scheduler:addComplexTask.....	657
pub.scheduler:addOneTimeTask.....	660
pub.scheduler:addRepeatingTask.....	662
pub.scheduler:cancelTask.....	665
pub.scheduler:getTaskIDs.....	665
pub.scheduler:getTaskInfo.....	666
pub.scheduler:getUserTaskList.....	670
pub.scheduler:migrateTasksToJDBC.....	670
pub.scheduler:resumeTask.....	671
pub.scheduler:suspendTask.....	672
pub.scheduler:updateComplexTask.....	673
pub.scheduler:updateOneTimeTask.....	676
pub.scheduler:updateRepeatingTask.....	678
<b>Schema Folder.....</b>	<b>683</b>
Summary of Elements in this Folder.....	684
pub.schema:createXSD.....	685
pub.schema:validate.....	687
pub.schema:validatePipeline.....	690
pub.schema.w3c.....	691
pub.schema.w3c:datatypes.....	692

pub.schema.w3c:structures.....	692
pub.schema.w3c:xml.....	692
pub.schema.w3c:xsi.....	692
<b>Security Folder.....</b>	<b>693</b>
About the Security Elements.....	694
Summary of Elements in this Folder.....	695
pub.security:clearKeyAndChain.....	698
pub.security:setKeyAndChain.....	699
pub.security:setKeyAndChainFromBytes.....	699
pub.security.enterpriseGateway:alertSpec.....	700
pub.security.enterpriseGateway:customFilterSpec.....	702
pub.security.keystore:getCertificate.....	703
pub.security.keystore:getKeyAndChain.....	704
pub.security.keystore:getTrustedCertificates.....	704
pub.security.keystore:setKeyAndChain.....	705
pub.security.keystore.pkcs7:sign.....	705
pub.security.outboundPasswords:setPassword.....	707
pub.security.outboundPasswords:getPassword.....	708
pub.security.outboundPasswords:listKeys.....	709
pub.security.outboundPasswords:removePassword.....	710
pub.security.outboundPasswords:updatePassword.....	710
pub.security.pkcs7:sign.....	711
pub.security.pkcs7:verify.....	712
pub.security.userInfoSpec.....	715
pub.security.util:createMessageDigest.....	718
pub.security.util:getCertificateInfo.....	718
pub.security.util:loadPKCS7CertChain.....	720
pub.security.util:createSecureString.....	720
pub.security.util:convertSecureString.....	721
pub.security.util:destroySecureString.....	721
pub.security.xml:decryptXML.....	722
pub.security.xml:encryptXML.....	724
pub.security.xml:signXML.....	726
pub.security.xml:verifyXML.....	732
<b>SMIME Folder.....</b>	<b>735</b>
Summary of Elements in this Folder.....	736
pub.smime:createCertsOnlyData.....	737
pub.smime:createEncryptedData.....	737
pub.smime:createSignedAndEncryptedData.....	738
pub.smime:createSignedData.....	740
pub.smime:processCertsOnlyData.....	742
pub.smime:processEncryptedData.....	742
pub.smime:processSignedData.....	744
pub.smime.keystore:createSignedAndEncryptedData.....	747

pub.smime.keystore:createSignedData.....	748
pub.smime.keystore:processEncryptedData.....	748
<b>SOAP Folder.....</b>	<b>751</b>
Summary of Elements in this Folder.....	752
pub.soap.handler:addBodyBlock.....	762
pub.soap.handler:addFaultBlock.....	764
pub.soap.handler:addHeaderBlock.....	766
pub.soap.handler:addHeaderElement.....	770
pub.soap.handler:generateDocumentTypesFromWSDL.....	773
pub.soap.handler:getBodyBlock.....	776
pub.soap.handler:getBodyBlockQNames.....	778
pub.soap.handler:getFaultBlock.....	778
pub.soap.handler:getHeaderBlock.....	780
pub.soap.handler:getHeaderBlockQNames.....	783
pub.soap.handler:getHeaderElement.....	783
pub.soap.handler:getInitialSOAPRequest.....	787
pub.soap.handler:getMessageAddressingProperties.....	787
pub.soap.handler:getProperty.....	790
pub.soap.handler:getServicePipeline.....	791
pub.soap.handler:getSOAPMessage.....	793
pub.soap.handler:getWebServiceInvocationProperties.....	794
pub.soap.handler:handlerSpec.....	795
pub.soap.handler:hasFaultMessage.....	796
pub.soap.handler:listConsumer.....	797
pub.soap.handler:listProvider.....	798
pub.soap.handler:registerConsumer.....	800
pub.soap.handler:registerProvider.....	800
pub.soap.handler:registerWmConsumer.....	800
pub.soap.handler:registerWmProvider.....	802
pub.soap.handler:removeBodyBlock.....	804
pub.soap.handler:removeHeaderBlock.....	804
pub.soap.handler:removeHeaderElement.....	806
pub.soap.handler:removeProperty.....	807
pub.soap.handler:setProperty.....	808
pub.soap.handler:setSOAPMessage.....	809
pub.soap.handler:unregisterConsumer.....	810
pub.soap.handler:unregisterProvider.....	810
pub.soap.handler:updateFaultBlock.....	810
pub.soap.processor:list.....	813
pub.soap.processor:processMessage.....	814
pub.soap.processor:processRPCMessage.....	815
pub.soap.processor:registerProcessor.....	815
pub.soap.processor:unregisterProcessor.....	817
pub.soap.schema:encoding.....	817

---

pub.soap.schema:encoding_1_2.....	817
pub.soap.schema:envelope.....	818
pub.soap.schema:envelope_1_2.....	818
pub.soap.utils:addBodyEntry.....	818
pub.soap.utils:addHeaderEntry.....	819
pub.soap.utils:addTrailer.....	821
pub.soap.utils:callbackServiceSpec.....	822
pub.soap.utils:convertToVersionSpecificSOAPFault.....	823
pub.soap.utils:createSoapData.....	827
pub.soap.utils:createXOObject.....	829
pub.soap.utils:exitUnableToUnderstand.....	830
pub.soap.utils:getActor.....	830
pub.soap.utils:getBody.....	831
pub.soap.utils:getBodyEntries.....	832
pub.soap.utils:getDocument.....	832
pub.soap.utils:getEncoding.....	833
pub.soap.utils:getHeader.....	834
pub.soap.utils:getHeaderEntries.....	834
pub.soap.utils:getMustUnderstand.....	835
pub.soap.utils:getQName.....	836
pub.soap.utils:getTrailers.....	837
pub.soap.utils:getXOObjectContent.....	838
pub.soap.utils:QName.....	840
pub.soap.utils:removeBodyEntry.....	840
pub.soap.utils:removeHeaderEntry.....	841
pub.soap.utils:removeTrailer.....	842
pub.soap.utils:requestResponseSpec.....	843
pub.soap.utils:resetWSDEffectivePolicy.....	843
pub.soap.utils.setWSDEffectivePolicy.....	844
pub.soap.utils:soapDataToBytes.....	845
pub.soap.utils:soapDataToString.....	845
pub.soap.utils:soapFault.....	846
pub.soap.utils:streamToSoapData.....	847
pub.soap.utils:stringToSoapData.....	848
pub.soap.utils:validateSoapData.....	849
pub.soap.wsa:action.....	850
pub.soap.wsa:faultTo.....	851
pub.soap.wsa:from.....	852
pub.soap.wsa:messageID.....	853
pub.soap.wsa:problemAction.....	853
pub.soap.wsa:problemHeaderQName.....	854
pub.soap.wsa:problemIRI.....	855
pub.soap.wsa:relatesTo.....	855
pub.soap.wsa:replyTo.....	856
pub.soap.wsa:retryAfter.....	857

pub.soap.wsa:to.....	858
pub.soap.wsa:schema_wsa.....	858
pub.soap.wsa.submission:action.....	858
pub.soap.wsa.submission:faultTo.....	859
pub.soap.wsa.submission:from.....	861
pub.soap.wsa.submission:messageID.....	863
pub.soap.wsa.submission:relatesTo.....	863
pub.soap.wsa.submission:replyTo.....	864
pub.soap.wsa.submission:retryAfter.....	866
pub.soap.wsa.submission:to.....	867
pub.soap.wsa.submission:schema_wsa_submission.....	867
pub.soap.wsrn:closeSequence.....	867
pub.soap.wsrn:createSequence.....	869
pub.soap.wsrn:sendAcknowledgementRequest.....	873
pub.soap.wsrn:terminateSequence.....	875
pub.soap.wsrn:waitUntilSequenceCompleted.....	876
<b>Storage Folder.....</b>	<b>879</b>
About the Storage Elements.....	880
Locking Considerations.....	881
Entry Locking.....	881
Data Store Locking.....	881
Automatic Promotion to Exclusive Lock.....	882
Wait Time and Duration.....	882
Sample Flow Service for Checkpoint Restart.....	883
Summary of Elements in this Folder.....	884
pub.storage:add.....	886
pub.storage:closeStore.....	886
pub.storage:deleteStore.....	887
pub.storage:get.....	888
pub.storage:keys.....	889
pub.storage:listLocks.....	889
pub.storage:lock.....	890
pub.storage:put.....	893
pub.storage:registerStore.....	894
pub.storage:releaseLocks.....	894
pub.storage:remove.....	895
pub.storage:shutdown.....	895
pub.storage:startup.....	895
pub.storage:unlock.....	896
<b>String Folder.....</b>	<b>897</b>
Summary of Elements in this Folder.....	898
pub.string:base64Decode.....	900
pub.string:base64Encode.....	900
pub.string:bytesToString.....	901

pub.string:compareStrings.....	902
pub.string:concat.....	902
pub.string:HTMLDecode.....	903
pub.string:HTMLEncode.....	904
pub.string:indexOf.....	904
pub.string:isAlphanumeric.....	905
pub.string:isDate.....	906
pub.string:isNullOrBlank.....	906
pub.string:isNumber.....	907
pub.string:length.....	907
pub.string:lookupDictionary.....	908
pub.string:lookupTable.....	908
pub.string:makeString.....	909
pub.string:messageFormat.....	910
pub.string:numericFormat.....	910
pub.string:objectToString.....	912
pub.string:padLeft.....	913
pub.string:padRight.....	914
pub.string:replace.....	914
pub.string:stringToBytes.....	915
pub.string:substitutePipelineVariables.....	916
pub.string:substring.....	916
pub.string:tokenize.....	917
pub.string:toLowerCase.....	917
pub.string:toUpperCase.....	918
pub.string:trim.....	919
pub.string:URLDecode.....	919
pub.string:URLEncode.....	919
<b>Sync Folder.....</b>	<b>921</b>
Summary of Elements in this Folder.....	922
pub.sync:notify.....	922
pub.sync:shutdown.....	923
pub.sync:wait.....	923
<b>Synchronization Folder.....</b>	<b>925</b>
Summary of Elements in this Folder.....	926
pub.synchronization.latch:closeLatch.....	926
pub.synchronization.latch:isLatchClosed.....	927
pub.synchronization.latch:openLatch.....	928
pub.synchronization.xref:createXReference.....	929
pub.synchronization.xref:deleteByObjectId.....	930
pub.synchronization.xref:deleteXReference.....	930
pub.synchronization.xref:getCanonicalKey.....	931
pub.synchronization.xref:getNativeId.....	932
pub.synchronization.xref:insertXReference.....	933

<b>Trigger Folder</b> .....	<b>935</b>
Summary of Elements in this Folder.....	936
pub.trigger:createJMSTrigger.....	937
pub.trigger:createTrigger.....	951
pub.trigger:deleteJMSTrigger.....	964
pub.trigger:deleteTrigger.....	964
pub.trigger:disableJMSTriggers.....	965
pub.trigger:enableJMSTriggers.....	967
pub.trigger:resourceMonitoringSpec.....	968
pub.trigger:resumeProcessing.....	969
pub.trigger:resumeRetrieval.....	972
pub.trigger:suspendJMSTriggers.....	974
pub.trigger:suspendProcessing.....	976
pub.trigger:suspendRetrieval.....	978
<b>TX Folder</b> .....	<b>981</b>
Summary of Elements in this Folder.....	982
pub.tx:init.....	982
pub.tx:shutdown.....	983
pub.tx:resetOutbound.....	983
<b>UniversalName Folder</b> .....	<b>985</b>
Summary of Elements in this Folder.....	986
pub.universalName:find.....	986
pub.universalName:findDocumentType.....	987
pub.universalName:list.....	987
pub.universalName:listAll.....	988
<b>Utils Folder</b> .....	<b>991</b>
Summary of Elements in this Folder.....	992
pub.utils:deepClone.....	992
pub.utils:executeOSCommand.....	993
pub.utils:generateUUID.....	995
pub.utils:getServerProperty.....	995
pub.utils.messaging:migrateDocTypesTriggersToUM.....	996
pub.utils:transcode.....	1006
pub.utils.ws:setCompatibilityModeFalse.....	1009
<b>VCS Folder</b> .....	<b>1015</b>
Summary of Elements in this Folder.....	1016
pub.vcs.admin:getUsers.....	1017
pub.vcs.admin:removeCurrentUser.....	1018
pub.vcs.admin:removeMultipleUsers.....	1019
pub.vcs.admin:setCurrentUser.....	1019
pub.vcs.admin:setMultipleUsers.....	1020

---

<b>XML Folder</b> .....	<b>1023</b>
Summary of Elements in this Folder.....	1024
pub.xml:documentToXMLString.....	1025
pub.xml:freeXMLNode.....	1032
pub.xml:getNextXMLNode.....	1033
pub.xml:getXMLNodeIterator.....	1034
pub.xml:getXMLNodeType.....	1037
pub.xml:loadEnhancedXMLNode.....	1037
pub.xml:loadXMLNode.....	1047
pub.xml:queryXMLNode.....	1055
pub.xml:xmlNodeToDocument.....	1058
pub.xml:xmlStringToEnhancedXMLNode.....	1069
pub.xml:xmlStringToXMLNode.....	1072
<b>XMLData Folder</b> .....	<b>1075</b>
Summary of Elements in this Folder.....	1076
pub.xmldata.domNodeToXMLData.....	1076
pub.xmldata:getAttributes.....	1080
pub.xmldata:getInstanceTag.....	1081
pub.xmldata:getNamespaceTags.....	1082
pub.xmldata:setAttribute.....	1084
pub.xmldata:setInstanceTag.....	1084
pub.xmldata:setNamespaceTag.....	1085
pub.xmldata:xmlDataToXMLString.....	1086
<b>XSLT Folder</b> .....	<b>1089</b>
Summary of Elements in this Folder.....	1090
pub.xslt.Transformations:transformSerialXML.....	1090
pub.xslt.Cache:removeAllTemplates.....	1093
pub.xslt.Cache:removeTemplate.....	1094
<b>Index</b> .....	<b>1095</b>

---

## About this Guide

---

The *webMethods Integration Server Built-In Services Reference* describes the built-in services provided with a standard installation of the webMethods Integration Server. Services are also installed with webMethods add-on packages, such as adapters and monitoring tools. You will find documentation for those services in the user guide provided with the add-on product.

**Note:** This guide describes features and functionality that may or may not be available with your licensed version of webMethods Integration Server. For information about the licensed components for your installation, see the **Settings > License** page in the webMethods Integration Server Administrator.

The descriptions in this book are divided into the following folders. These folders reside in the WmPublic package, unless otherwise specified.

Folder	Contains services you use to...
<a href="#">ART Folder</a>	(WmART package) Manage adapter components, including connections, adapter services, listeners, and notifications.
<a href="#">Assets Folder</a>	Retrieve information about assets on Integration Server.
<a href="#">Cache Folder</a>	Perform tasks on caches.
<a href="#">Client Folder</a>	Formulate and submit requests to HTTP, FTP, e-mail, and LDAP servers.
<a href="#">Date Folder</a>	Generate and format date values.
<a href="#">Db Folder</a>	(WmDB package) Access JDBC-enabled databases. The webMethods Adapter for JDBC also provides services that perform operations against JDBC-enabled databases. See the <i>webMethods Adapter for JDBC Installation and User's Guide</i> for information.
<a href="#">Document Folder</a>	Perform operations on documents in the pipeline.
<a href="#">Event Folder</a>	Build audit and event handler services.

Folder	Contains services you use to...
File Folder	Perform operations on the local file system.
Flat File Folder	(WmFlatFile package) Convert between Flat File documents and IS documents.
Flow Folder	Perform debugging and utility-type tasks in a flow service.
Hashtable Folder	Create, update, and get information about the hashtable.
IO Folder	Convert data between byte[ ] and InputStream representations.
JDBC Folder	Obtain information about Integration Server JDBC pools.
JMS Folder	Send and receive JMS messages.
JSON Folder	Parse JSON content in the pipeline.
List Folder	Retrieve, replace, or add elements in an Object List, Document List, or String List; convert String Lists to Document Lists.
Math Folder	Add, subtract, multiply, or divide string-based numeric values.
Metadata Folder	Publish metadata about Integration Server packages and administrative assets to the CentraSite shared registry.
MIME Folder	Create MIME messages and extract information from MIME messages.
OAuth Folder	Authorize a client application to access data on Integration Server using the OAuth 2.0 Authorization Framework.
Packages Folder	Install, load, and/or alter the status of a package on the Integration Server.

Folder	Contains services you use to...
Publish Folder	Publish and deliver documents to other Integration Servers via webMethods Universal Messaging or webMethods Broker.
Remote Folder	Invoke services on remote webMethods Integration Servers.
Replicator Folder	Replicate packages across webMethods Integration Servers.
Report Folder	Apply an output template to the values in the pipeline.
Scheduler Folder	Schedule services to execute at the times you specify.
Schema Folder	Validate objects or values in the pipeline.
Security Folder	Control which client certificates are sent to other services and digitally sign data and process digital signatures. Store and retrieve outbound passwords to access secure resources.
SMIME Folder	Create digitally signed and/or encrypted MIME messages. Process signed and encrypted MIME messages.
SOAP Folder	Send, receive, and retrieve data from SOAP messages. Register custom SOAP processors.
Storage Folder	Create, close, delete, and register repository data stores. Insert and retrieve information from data stores.
String Folder	Perform string manipulation and substitution operations.
Sync Folder	Coordinate the execution of services.
Synchronization Folder	Perform latching and cross-referencing operations in a publish-and-subscribe integration.
Trigger Folder	Create and delete triggers and manage document retrieval and document processing for individual

Folder	Contains services you use to...
	webMethods messaging triggers. Create, delete, enable, disable, or suspend one or more JMS triggers.
TX Folder	Perform administrative tasks for guaranteed delivery transactions.
UniversalName Folder	List the contents of the Universal Registry and look up services by their universal names.
Utils Folder	Retrieve the values of server properties
VCS Folder	<i>Deprecated</i> - Use the local service development feature (Local Version Control Integration) to check package elements and their supporting files into and out of a version control system (VCS) directly from Designer. Manage user associations for the VCS Integration feature.
XML Folder	Perform operations on XML documents.
XMLData Folder	Work with XML documents in the form of XMLData documents.
XSLT Folder	(WmXSLT package) Transform an XML stream into a byte array, file, or XML node, and to maintain the XSLT stylesheet cache.

Built-in services generally have the following default access permissions:

For this type of permission...	<u>Built-in services are assigned to this ACL...</u>	
List	Developers	Members of the Developers ACL can see, in webMethods Integration Server or Software AG Designer, that a service exists.
Read	WmPrivate	The WmPrivate ACL is a virtual ACL designed to protect the proprietary code in the built-in services. As this ACL has no members, no user can edit a service or view its source.
Write	WmPrivate	

---

## For this type of permission...

## Built-in services are assigned to this ACL...

Execute	Internal	Members of the Internal ACL can execute a service.
---------	----------	--

These default access permissions cannot be changed (that is, another ACL cannot be selected).

---

## Document Conventions

Convention	Description
<b>Bold</b>	Identifies elements on a screen.
Narrowfont	Identifies storage locations for services on webMethods Integration Server, using the convention <i>folder.subfolder:service</i> .
UPPERCASE	Identifies keyboard keys. Keys you must press simultaneously are joined with a plus sign (+).
<i>Italic</i>	Identifies variables for which you must supply values specific to your own situation or environment. Identifies new terms the first time they occur in the text.
Monospace font	Identifies text you must type or messages displayed by the system.
{ }	Indicates a set of choices from which you must choose one. Type only the information inside the curly braces. Do not type the { } symbols.
	Separates two mutually exclusive choices in a syntax line. Type one of these choices. Do not type the   symbol.
[ ]	Indicates one or more options. Type only the information inside the square brackets. Do not type the [ ] symbols.
...	Indicates that you can type multiple options of the same type. Type only the information. Do not type the ellipsis (...).

---

## Online Information

---

### Software AG Documentation Website

You can find documentation on the Software AG Documentation website at <http://documentation.softwareag.com>. The site requires Empower credentials. If you do not have Empower credentials, you must use the TECHcommunity website.

### Software AG Empower Product Support Website

You can find product information on the Software AG Empower Product Support website at <https://empower.softwareag.com>.

To submit feature/enhancement requests, get information about product availability, and download products, go to [Products](#).

To get information about fixes and to read early warnings, technical papers, and knowledge base articles, go to the [Knowledge Center](#).

### Software AG TECHcommunity

You can find documentation and other technical information on the Software AG TECHcommunity website at <http://techcommunity.softwareag.com>. You can:

- Access product documentation, if you have TECHcommunity credentials. If you do not, you will need to register and specify "Documentation" as an area of interest.
- Access articles, code samples, demos, and tutorials.
- Use the online discussion forums, moderated by Software AG professionals, to ask questions, discuss best practices, and learn how other customers are using Software AG technology.
- Link to external websites that discuss open standards and web technology.

# 1 ART Folder

---

■ Summary of Elements in this Folder .....	28
--	----

You use the elements in the art folder to manage adapter components, including connections, adapter services, listeners, and notifications.

## Summary of Elements in this Folder

The following elements are available in this folder:

Element	Package and Description
<a href="#">pub.art:listRegisteredAdapters</a>	WmART. Returns the display name and adapter type name of all registered adapters.
<a href="#">pub.art.connection:disableConnection</a>	WmART. Disables a connection node.
<a href="#">pub.art.connection:enableConnection</a>	WmART. Enables an existing connection node.
<a href="#">pub.art.connection:getConnectionStatistics</a>	WmART. Returns current usage statistics for a connection node.
<a href="#">pub.art.connection:listAdapterConnections</a>	WmART. Lists connection nodes associated with a specified adapter.
<a href="#">pub.art.connection:queryConnectionState</a>	WmART. Returns the current connection state (enabled/disabled) and error status for a connection node.
<a href="#">pub.art.listener:disableListener</a>	WmART. Disables a listener.
<a href="#">pub.art.listener:enableListener</a>	WmART. Enables an existing listener.
<a href="#">pub.art.listener:listAdapterListeners</a>	WmART. Lists listeners associated with a specified adapter.
<a href="#">pub.art.listener:queryListenerState</a>	WmART. Returns the current state for a listener.

Element	Package and Description
<a href="#">pub.art.listener:resumeListener</a>	WmART. Resumes a specified listener.
<a href="#">pub.art.listener:setListenerNodeConnection</a>	WmART. Changes the connection node used by a specified listener.
<a href="#">pub.art.listener:suspendListener</a>	WmART. Suspends a specified listener.
<a href="#">pub.art.notification:disableListenerNotification</a>	WmART. Disables a listener notification.
<a href="#">pub.art.notification:disablePollingNotification</a>	WmART. Disables a polling notification.
<a href="#">pub.art.notification:disablePublishEvents</a>	WmART. Disables the publishing of events for an adapter notification.
<a href="#">pub.art.notification:enableListenerNotification</a>	WmART. Enables an existing listener notification.
<a href="#">pub.art.notification:enablePollingNotification</a>	WmART. Enables an existing polling notification.
<a href="#">pub.art.notification:enablePublishEvents</a>	WmART. Enables the publishing of events for an adapter notification.
<a href="#">pub.art.notification:listAdapterListenerNotifications</a>	WmART. Lists the listener notifications associated with a specified adapter.
<a href="#">pub.art.notification:listAdapterPollingNotifications</a>	WmART. Lists the polling notifications associated with a specified adapter.
<a href="#">pub.art.notification:queryListenerNotificationState</a>	WmART. Returns the current state (enabled/disabled) for a listener notification.

Element	Package and Description
<a href="#">pub.art.notification:queryPollingNotificationState</a>	WmART. Returns the current state for a polling notification.
<a href="#">pub.art.notification:resumePollingNotification</a>	WmART. Resumes a specified polling notification node.
<a href="#">pub.art.notification:setListenerNotificationNodeListener</a>	WmART. Changes the listener used by a specified listener notification.
<a href="#">pub.art.notification:setPollingNotificationNodeConnection</a>	WmART. Changes the connection node used by a specified polling notification.
<a href="#">pub.art.notification:suspendPollingNotification</a>	WmART. Suspends a specified polling notification.
<a href="#">pub.art.service:listAdapterServices</a>	WmART. Lists adapter services associated with a specified adapter.
<a href="#">pub.art.service:setAdapterServiceNodeConnection</a>	WmART. Changes the connection node used by a specified adapter service.
<a href="#">pub.art.transaction:commitTransaction</a>	WmART. Commits an explicit transaction.
<a href="#">pub.art.transaction:rollbackTransaction</a>	WmART. Rolls back an explicit transaction.
<a href="#">pub.art.transaction:setTransactionTimeout</a>	WmART. Manually sets a transaction timeout interval for implicit and explicit transactions.
<a href="#">pub.art.transaction:startTransaction</a>	WmART. Starts an explicit transaction.

---

## pub.art:listRegisteredAdapters

WmART. Returns the display name and adapter type name of all registered adapters.

### Input Parameters

---

None.

### Output Parameters

---

*registeredAdapterList* **Document List** Information for each adapter registered with the WmART package.

<u>Key</u>	<u>Description</u>
<i>adapterDisplayName</i>	<b>String</b> The localized name that the Integration Server Administrator displays.
<i>adapterTypeName</i>	<b>String</b> The name of the adapter as registered with the WmART package. This value can be used as input for the inventory services that take <i>adapterTypeName</i> as input.

---

## pub.art.connection:disableConnection

WmART. Disables a connection node.

### Input Parameters

---

*connectionAlias* **String** Name of the connection node you want to disable.

### Output Parameters

---

None.

### See Also

[pub.art.connection:enableConnection](#)

---

## pub.art.connection:enableConnection

WmART. Enables an existing connection node.

### Input Parameters

---

*connectionAlias*      **String** Name of the connection node you want to enable.

### Output Parameters

---

None.

### See Also

[pub.art.connection:disableConnection](#)

---

## pub.art.connection:getConnectionStatistics

WmART. Returns current usage statistics for a connection node.

### Input Parameters

---

*aliasName*      **String** Name of the connection node for which you want usage statistics returned.

### Output Parameters

---

*connectionStatistics*      **Document List** Information for each connection node.

<b>Key</b>	<b>Description</b>
<i>TotalConnections</i>	<b>Integer</b> Current number of connection instances.
<i>BusyConnections</i>	<b>Integer</b> Number of connections currently in use by services, notifications, and listeners.
<i>FreeConnections</i>	<b>Integer</b> Total number of connections created and available for use.

<i>TotalHits</i>	<b>Integer</b> Number of times this connection node successfully provided connections since the last reset.
<i>TotalMisses</i>	<b>Integer</b> Number of times this connection node unsuccessfully provided connections since the last reset (when the request timed out).

**See Also**

[pub.art.connection:queryConnectionState](#)

---

## pub.art.connection:listAdapterConnections

WmART. Lists connection nodes associated with a specified adapter.

### Input Parameters

---

<i>adapterTypeName</i>	<b>String</b> The name of the adapter as registered with the WmART package.
------------------------	---

### Output Parameters

---

<i>connectionDataList</i>	<b>Document List</b> Information for each connection node registered with the specified adapter.
---------------------------	--

<u>Key</u>	<u>Description</u>
<i>connectionAlias</i>	<b>String</b> The name of the connection node.
<i>packageName</i>	<b>String</b> The name of the package in which the connection node resides.
<i>connectionState</i>	<p><b>String</b> Current state of the connection node. The state will have one of these values:</p> <ul style="list-style-type: none"> <li>■ disabled - Connection node is disabled</li> <li>■ enabled - Connection node is enabled.</li> <li>■ shuttingdown - Connection node is in the process of shutting down.</li> </ul>

- `unknown` - Connection node is registered but has not yet established its state.

### See Also

[pub.art:listRegisteredAdapters](#)

[pub.art.connection:queryConnectionState](#)

---

## pub.art.connection:queryConnectionState

WmART. Returns the current connection state (enabled/disabled) and error status for a connection node.

### Input Parameters

---

*connectionAlias*      **String** Name of the connection node for which you want the connection state and error status returned.

### Output Parameters

---

*connectionState*      **String** Current connection state (enabled/disabled).

*hasError*              **Boolean** Flag indicating if any error was detected on connection. The values are:

- `true` if an error was detected.
- `false` if no error was detected.

### See Also

[pub.art.connection:getConnectionStatistics](#)

[pub.art.connection:enableConnection](#)

[pub.art.connection:disableConnection](#)

---

## pub.art.listener:disableListener

WmART. Disables a listener.

---

## Input Parameters

---

*listenerName*      **String** Name of the listener you want to disable. The listener should have a state of `enabled` or `suspended`.

*forceDisable*      **String** Optional. Flag to disable the listener regardless of whether it is still waiting for data from a back-end resource. The string may have one of these values:

- `true` to disable the listener.
- `false` to keep the listener enabled.

## Output Parameters

---

None.

### See Also

[pub.art.listener:enableListener](#)

---

# pub.art.listener:enableListener

WmART. Enables an existing listener.

---

## Input Parameters

---

*listenerName*      **String** Name of the listener you want to enable.

## Output Parameters

---

None.

### Usage Notes

If you do not enable the connection resource associated with the listener, this service will return without performing any action, and the listener will remain disabled. Therefore, you should invoke [pub.art.listener:queryListenerState](#) before calling this service to confirm that the listener has been enabled.

### See Also

[pub.art.listener:queryListenerState](#)

[pub.art.listener:disableListener](#)

---

## pub.art.listener:listAdapterListeners

WmART. Lists listeners associated with a specified adapter.

### Input Parameters

---

*adapterTypeName*      **String** The name of the adapter as registered with the WmART package.

### Output Parameters

---

*listenerDataList*      **Document List** Information for each listener registered with the specified adapter.

<u>Key</u>	<u>Description</u>
<i>listenerNodeName</i>	<b>String</b> The name of the listener.
<i>packageName</i>	<b>String</b> The name of the package in which the listener resides.
<i>listenerEnabled</i>	<b>String</b> Current state of the listener. The state will have one of these values: <ul style="list-style-type: none"><li>■ <code>disabled</code> if the listener is disabled.</li><li>■ <code>enabled</code> if the listener is enabled.</li><li>■ <code>enablePending</code> if the listener is in the process of starting.</li><li>■ <code>disablePending</code> if the listener is in the process of disabling.</li><li>■ <code>suspended</code> if the listener is suspended.</li><li>■ <code>suspendPending</code> if the listener is in the process of suspending.</li></ul>

### See Also

[pub.art.listRegisteredAdapters](#)

[pub.art.listener:queryListenerState](#)

---

## pub.art.listener:queryListenerState

WmART. Returns the current state for a listener.

### Input Parameters

---

*listenerName*      **String** Name of the listener for which you want the current state returned.

### Output Parameters

---

*listenerState*      **String** Current state of the listener. The state will have one of these values:

- `disabled` if the listener is disabled.
- `enabled` if the listener is enabled.
- `enablePending` if the listener is in the process of starting.
- `disablePending` if the listener is in the process of disabling.
- `suspended` if the listener is suspended.
- `suspendPending` if the listener is in the process of suspending.

### See Also

[pub.art.listener:enableListener](#)

[pub.art.listener:disableListener](#)

---

## pub.art.listener:resumeListener

WmART. Resumes a specified listener.

### Input Parameters

---

*listenerName*      **String** The name of the suspended listener you want to resume. The service returns an error if you specify an invalid listener.

### Output Parameters

---

None.

---

### Usage Notes

If the requested transition is not valid (for example, trying to resume a disabled listener or a listener that is already resumed), the service ignores the request.

After you use this service, you can use [pub.art.listener:queryListenerState](#) to verify `pub.art.listener:resumeListener` correctly changed the state of the listener.

### See Also

[pub.art.listener:queryListenerState](#)

[pub.art.listener:suspendListener](#)

---

## pub.art.listener:setListenerNodeConnection

WmART. Changes the connection node used by a specified listener.

---

### Input Parameters

<i>listenerName</i>	<b>String</b> Name of the listener for which you want to change the connection node.
<i>connectionAlias</i>	<b>String</b> Name of the new connection node to use with the listener.

---

### Output Parameters

None.

### Usage Notes

Calling this service for a listener that is disabled is permitted.

Calling this service for a listener that is suspended changes the state of the listener to `disabled`. The user must enable the listener before using it.

### See Also

[pub.art.listener:disableListener](#)

---

## pub.art.listener:suspendListener

WmART. Suspends a specified listener.

---

### Input Parameters

---

*listenerName*      **String** The name of the listener you want to suspend. The service returns an error if you specify an invalid listener.

### Output Parameters

---

None.

### Usage Notes

If the requested transition is not valid (for example, trying to suspend a disabled listener or a listener that is already suspended), the service ignores the request.

After you use this service, you can use [pub.art.listener:queryListenerState](#) to verify `pub.art.listener:suspendListener` correctly changed the state of the listener.

### See Also

[pub.art.listener:queryListenerState](#)

[pub.art.listener:resumeListener](#)

---

## pub.art.notification:disableListenerNotification

WmART. Disables a listener notification.

### Input Parameters

---

*notificationName*      **String** The name of the listener notification you want to disable.

### Output Parameters

---

None.

### See Also

[pub.art.notification:enableListenerNotification](#)

---

## pub.art.notification:disablePollingNotification

WmART. Disables a polling notification.

### Input Parameters

---

*notificationName*      **String** The name of the polling notification you want to disable. The polling notification should have a state of `enabled` or `suspended`.

### Output Parameters

---

None.

### See Also

[pub.art.notification:enablePollingNotification](#)

---

## pub.art.notification:disablePublishEvents

WmART. Disables the publishing of events for an adapter notification.

### Input Parameters

---

*notificationName*      **String** The name of the notification for which you want to disable the publishing of events.

### Output Parameters

---

None.

### See Also

[pub.art.notification:enablePublishEvents](#)

---

## pub.art.notification:enableListenerNotification

WmART. Enables an existing listener notification.

### Input Parameters

---

*notificationName*      **String** The name of the listener notification you want to enable.

### Output Parameters

---

None.

**See Also**

[pub.art.notification:disableListenerNotification](#)

---

## pub.art.notification:enablePollingNotification

WmART. Enables an existing polling notification.

**Input Parameters**

---

*notificationName*      **String** Name of the polling notification you want to enable.

**Output Parameters**

---

None.

**Usage Notes**

You must schedule the polling notification before you can run this service. See your adapter user documentation for instructions to schedule the polling notification.

**See Also**

[pub.art.notification:disablePollingNotification](#)

---

## pub.art.notification:enablePublishEvents

WmART. Enables the publishing of events for an adapter notification.

**Input Parameters**

---

*notificationName*      **String** The name of the notification for which you want to enable the publishing of events.

**Output Parameters**

---

None.

**See Also**

[pub.art.notification:disablePublishEvents](#)

---

## pub.art.notification:listAdapterListenerNotifications

WmART. Lists the listener notifications associated with a specified adapter.

### Input Parameters

---

*adapterTypeName*      **String** The name of the adapter as registered with the WmART package.

### Output Parameters

---

*notificationDataList*      **Document List** Information for each listener notification registered with the specified adapter.

<u>Key</u>	<u>Description</u>
<i>notificationNodeName</i>	<b>String</b> The name of the listener notification.
<i>packageName</i>	<b>String</b> The name of the package in which the listener notification resides.
<i>notificationEnabled</i>	<p><b>String</b> The current state of the listener notification. The state will have one of these values:</p> <ul style="list-style-type: none"> <li>■ <code>no</code> if the listener notification is disabled.</li> <li>■ <code>yes</code> if the listener notification is enabled.</li> </ul>

### See Also

[pub.art:listRegisteredAdapters](#)

[pub.art.notification:queryListenerNotificationState](#)

---

## pub.art.notification:listAdapterPollingNotifications

WmART. Lists the polling notifications associated with a specified adapter.

---

## Input Parameters

*adapterTypeName*      **String** The name of the adapter as registered with the WmART package.

## Output Parameters

*notificationDataList*      **Document List** Information for each polling notification registered with the specified adapter.

<u>Key</u>	<u>Description</u>
<i>notificationNodeName</i>	<b>String</b> The name of the polling notification.
<i>packageName</i>	<b>String</b> The name of the package in which the polling notification resides.
<i>notificationEnabled</i>	<p><b>String</b> The current state of the polling notification. The state will have one of these values:</p> <ul style="list-style-type: none"> <li>■ <code>no</code> if the polling notification is disabled.</li> <li>■ <code>yes</code> if the polling notification is enabled.</li> <li>■ <code>pending</code> if the polling notification is in the process of shutting down.</li> <li>■ <code>suspended</code> if the polling notification is suspended.</li> </ul>

## See Also

[pub.art:listRegisteredAdapters](#)

[pub.art.notification:queryPollingNotificationState](#)

---

## pub.art.notification:queryListenerNotificationState

WmART. Returns the current state (enabled/disabled) for a listener notification.

---

### Input Parameters

---

*notificationName*      **String** The name of the listener notification for which you want the current state (enabled/disabled) returned.

### Output Parameters

---

*notificationState*      **String** The current state (enabled/disabled) for the listener notification.

### See Also

[pub.art.notification:enableListenerNotification](#)

[pub.art.notification:disableListenerNotification](#)

---

## pub.art.notification:queryPollingNotificationState

WmART. Returns the current state for a polling notification.

### Input Parameters

---

*notificationName*      **String** The name of the polling notification for which you want the current state and schedule settings returned.

### Output Parameters

---

*notificationState*      **String** The current state (enabled, disabled, pending disable, pending suspend, or suspended) for the polling notification.

*scheduleSettings*      **IData** Object that contains the notification's schedule settings as follows:

<b>Key</b>	<b>Description</b>
<i>notificationInterval</i>	<b>Integer</b> Polling frequency of the notification.
<i>notificationOverlap</i>	<b>Boolean</b> Flags whether the notification can overlap. The values are:

	<ul style="list-style-type: none"> <li>■ <code>true</code> if the notification can overlap.</li> <li>■ <code>false</code> if the notification cannot overlap.</li> </ul>
<i>notificationImmediate</i>	<p><b>Boolean</b> Flags whether the notification can fire immediately. The values are:</p> <ul style="list-style-type: none"> <li>■ <code>true</code> if the notification can fire immediately.</li> <li>■ <code>false</code> if the notification cannot fire immediately.</li> </ul>

**See Also**

[pub.art.notification:enablePollingNotification](#)

[pub.art.notification:disablePollingNotification](#)

---

## pub.art.notification:resumePollingNotification

WmART. Resumes a specified polling notification node.

**Input Parameters**

<i>notificationName</i>	<b>String</b> The name of the polling notification you want to resume. The service returns an error if you specify an invalid polling notification.
-------------------------	---

**Output Parameters**

None.

**Usage Notes**

If the requested transition is not valid (for example, trying to resume a disabled polling notification or a polling notification that is already resumed), the service ignores the request.

After you use this service, you can use [pub.art.notification:queryPollingNotificationState](#) to verify `pub.art.notification:resumePollingNotification` correctly changed the state of the polling notification to `enabled`.

**See Also**

[pub.art.notification:queryPollingNotificationState](#)

---

[pub.art.notification:suspendPollingNotification](#)

---

## pub.art.notification:setListenerNotificationNodeListener

WmART. Changes the listener used by a specified listener notification.

### Input Parameters

---

<i>notificationName</i>	<b>String</b> Name of the listener notification for which you want to change the listener.
<i>listenerNode</i>	<b>String</b> Name of the new listener to use with the listener notification.

### Output Parameters

---

None.

### Usage Notes

This service returns an error if the listener notification is enabled.

You can use this service for synchronous and asynchronous listener notifications.

### See Also

[pub.art.notification:disableListenerNotification](#)

---

## pub.art.notification:setPollingNotificationNodeConnection

WmART. Changes the connection node used by a specified polling notification.

### Input Parameters

---

<i>notificationName</i>	<b>String</b> Name of the polling notification for which you want to change the connection node.
<i>connectionAlias</i>	<b>String</b> Name of the new connection node to use with the polling notification.

### Output Parameters

---

None.

### Usage Notes

The polling notification must be in a `disabled` or `suspended` state before you call this service. This service returns an error if the polling notification is enabled.

If you use this service on a suspended polling notification, the service changes the state of the polling notification to `disabled`.

### See Also

[pub.art.notification:disablePollingNotification](#)

---

## pub.art.notification:suspendPollingNotification

WmART. Suspends a specified polling notification.

### Input Parameters

---

<i>notificationName</i>	<b>String</b> The name of the polling notification you want to suspend. The service returns an error if you specify an invalid polling notification.
-------------------------	--

### Output Parameters

---

None.

### Usage Notes

If the requested transition is not valid (for example, trying to suspend a disabled polling notification or a polling notification that is already suspended), the service ignores the request.

After you use this service, you can use [pub.art.notification:queryPollingNotificationState](#) to verify `pub.art.notification:suspendPollingNotification` correctly changed the state of the polling notification to `suspended`.

### See Also

[pub.art.notification:queryPollingNotificationState](#)

[pub.art.notification:resumePollingNotification](#)

---

## pub.art.service:listAdapterServices

WmART. Lists adapter services associated with a specified adapter.

---

### Input Parameters

*adapterTypeName*      **String** The name of the adapter as registered with the WmART package.

### Output Parameters

*serviceDataList*      **Document List** Information for each adapter service registered with the specified adapter.

<u>Key</u>	<u>Description</u>
<i>serviceName</i>	<b>String</b> The name of the adapter service.
<i>packageName</i>	<b>String</b> The name of the package in which the adapter service resides.

### See Also

[pub.art:listRegisteredAdapters](#)

---

## pub.art.service:setAdapterServiceNodeConnection

WmART. Changes the connection node used by a specified adapter service.

### Input Parameters

*serviceName*      **String** Name of an existing adapter service for which you want to change the connection node.

*connectionAlias*      **String** Name of the new connection node to use with the adapter service.

### Output Parameters

None.

### Usage Notes

The new connection node must be enabled before you call this service.

## See Also

[pub.art.connection:enableConnection](#)

---

## pub.art.transaction:commitTransaction

WmART. Commits an explicit transaction.

### Input Parameters

---

*commitTransactionInput* **Document List** Information for each commit request.

<u>Key</u>	<u>Description</u>
<i>transactionName</i>	<b>String</b> The name of an explicit transaction that you want to commit. The <i>transactionName</i> must have been previously used in a call to <a href="#">pub.art.transaction:startTransaction</a> .  This value must be mapped from the most recent <a href="#">pub.art.transaction:startTransaction</a> that has not previously been committed or rolled back.

### Output Parameters

---

None.

### Usage Notes

This service is available only if your adapter supports built-in transaction management services, which you can confirm by checking the user guide for the adapter.

This service must be used in conjunction with the [pub.art.transaction:startTransaction](#) service. If the *transactionName* parameter was not provided in a prior call to [pub.art.transaction:startTransaction](#), a run-time error will be returned.

## See Also

[pub.art.transaction:startTransaction](#)

[pub.art.transaction:rollbackTransaction](#)

---

## pub.art.transaction:rollbackTransaction

WmART. Rolls back an explicit transaction.

### Input Parameters

---

*rollbackTransactionInput*

**Document List** Information for each rollback request.

Key	Description
<i>transactionName</i>	<p><b>String</b> The name of an explicit transaction that you want to roll back. The <i>transactionName</i> must have been previously used in a call to <a href="#">pub.art.transaction:startTransaction</a>.</p> <p>This value must be mapped from the most recent <a href="#">pub.art.transaction:startTransaction</a> that has not previously been committed or rolled back.</p>

### Output Parameters

---

None.

### Usage Notes

This service is available only if your adapter supports built-in transaction management services, which you can confirm by checking the adapter's user guide.

This service must be used in conjunction with the [pub.art.transaction:startTransaction](#) service. If the given *transactionName* parameter was not provided in a prior call to [pub.art.transaction:startTransaction](#), a run-time error will be returned.

### See Also

[pub.art.transaction:startTransaction](#)

[pub.art.transaction:commitTransaction](#)

---

## pub.art.transaction:setTransactionTimeout

WmART. Manually sets a transaction timeout interval for implicit and explicit transactions.

---

## Input Parameters

*timeoutSeconds*      **Integer** The number of seconds that the implicit or explicit transaction stays open before the transaction manager marks it for rollback.

---

## Output Parameters

None.

## Usage Notes

This service is available only if your adapter supports built-in transaction management services, which you can confirm by checking the user guide for the adapter.

When you use this service, you are temporarily overriding the Integration Server transaction timeout interval.

You must call this service within a flow before the start of any implicit or explicit transactions. Implicit transactions start when you call an adapter service in a flow. Explicit transactions start when you call the [pub.art.transaction:startTransaction](#) service.

If the execution of a transaction takes longer than the transaction timeout interval, all transacted operations are rolled back.

This service only overrides the transaction timeout interval for the flow service in which you call it.

## See Also

[pub.art.transaction:startTransaction](#)

---

# pub.art.transaction:startTransaction

WmART. Starts an explicit transaction.

---

## Input Parameters

*startTransactionInput*      **Document List** Information for each start transaction request.

<u>Key</u>	<u>Description</u>
<i>transactionName</i>	<b>String</b> Optional. Specifies the name of the transaction to be started. If you leave this parameter blank, Integration Server will

generate a name for you. In most implementations it is not necessary to provide your own transaction name.

## Output Parameters

---

*startTransactionOutput*

**Document List** Information for each start transaction request.

<u>Key</u>	<u>Description</u>
<i>transactionName</i>	<b>String</b> The name of the transaction the service just started.

## Usage Notes

This service is available only if your adapter supports built-in transaction management services, which you can confirm by checking the user guide for the adapter.

This service is intended for use with the [pub.art.transaction:commitTransaction](#) or [pub.art.transaction:rollbackTransaction](#) service. The *transactionName* value returned by a call to this service can be provided to [pub.art.transaction:commitTransaction](#) (to commit the transaction) or [pub.art.transaction:rollbackTransaction](#) (to roll back the transaction).

## See Also

[pub.art.transaction:commitTransaction](#)

[pub.art.transaction:rollbackTransaction](#)

## 2 Assets Folder

---

■ Summary of Elements in this Folder .....	54
--	----

You use the service in the assets folder to get the checksum of one or more assets on Integration Server.

## Summary of Elements in this Folder

The following asset is available in this folder:

Element	Package and Description
<code>pub.assets:getChecksums</code>	WmPublic. Retrieves the checksums for assets, specifically namespace elements, on Integration Server.

### pub.assets:getChecksums

WmPublic. Retrieves the checksums for assets, specifically namespace elements, on Integration Server. You can calculate the checksum for specific assets or all of the assets in a package. A checksum is a digest for an asset that is generated using all the properties and contents of the asset. Each asset on Integration Server has a unique checksum. Two assets on different Integration Servers have the same checksum if the assets are identical.

#### Input Parameters

*packages* **String List** Optional. Names of the packages for which you want to get the checksum of each asset in the package. A package must be enabled and loaded for Integration Server to get checksums for the assets. If a package is partially loaded, Integration Server gets checksums for the loaded assets only.

You must specify a value for *packages* or *assets*. If you specify a value for both *packages* and *assets*, Integration Server uses the *packages* value and ignores *assets*. If you specify neither, the `pub.assets:getChecksums` service ends with an exception.

*assets* **String List** Optional. List of the fully qualified names of the specific assets for which you want to get the checksum.

You must specify a value for *packages* or *assets*. If you specify a value for both *packages* and *assets*, Integration Server uses the *packages* value and ignores *assets*. If you specify neither, the `pub.assets:getChecksums` service ends with an exception.

---

## Output Parameters

---

*checksums* **Document List** List of checksums for the assets in the specified packages or the specified assets.

<u>Key</u>	<u>Description</u>
<i>package</i>	<b>String</b> Name of the package containing the asset.
<i>name</i>	<b>String</b> Fully qualified name of the asset.
<i>checksum</i>	<b>String</b> Checksum for the asset.  The checksum value is null for an asset listed in <i>assets</i> if the specified asset does not exist on Integration Server.

### Usage Notes

The `pub.assets:getChecksums` service ends with an exception if a package identified in *packages* does not exist or is not enabled.

The *checksums/checksum* output parameter contains a null value if an asset specified in *assets* does not exist.

If you do not have List access to a particular asset, the service returns only the asset *name*. The *package* and *checksum* values for the particular asset are null.

You can only compare packages and assets from the same version of Integration Server. Comparison of assets across different versions of Integration Server is not supported. Internal changes in assets from one release to another might result in functionally identical assets being reported as different.



---

# 3 Cache Folder

---

■ About Checkpoint Restart .....	58
■ Summary of Elements in this Folder .....	59

You use the elements in the cache folder to place data in a cache and retrieve it again later. You can also use the services in the cache folder to perform administrative operations such as enabling, disabling, and clearing a cache, or to implement checkpoint restart in services you write. Integration Server uses Ehcache internally for all of the services in the cache folder. Before using these services, you need create a cache manager and cache using Integration Server Administrator. For more information about creating a cache manager and cache, see *webMethods Integration Server Administrator's Guide*.

**Note:** The key you use for the services in the cache folder must be an object that overrides the equals() method from java.lang.Object. Failure to use such an object can cause the service to return incorrect results.

**Note:** You cannot use an IS document object as a key for storing objects in the cache because IS document objects do not implement the equals() method.

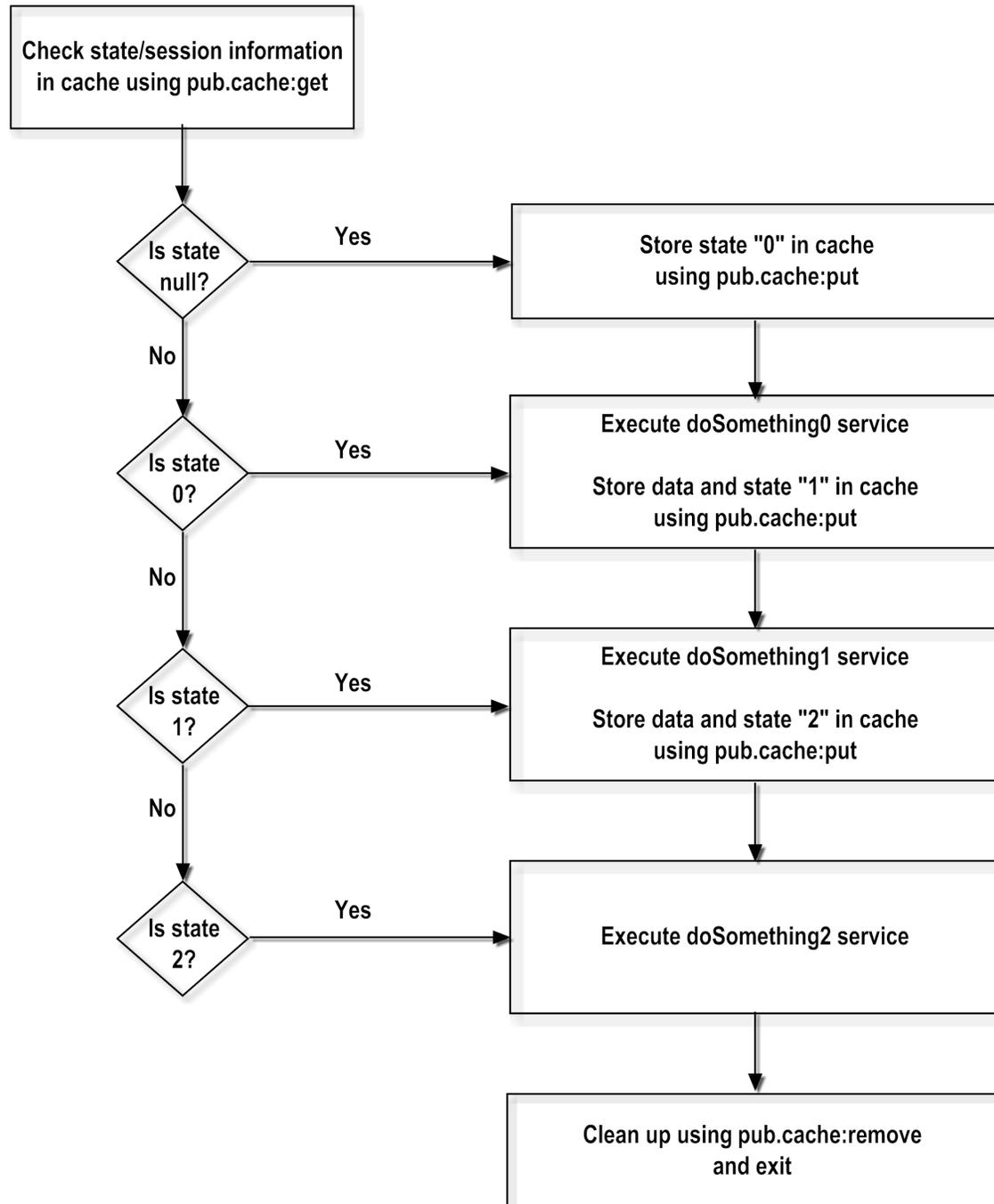
## About Checkpoint Restart

---

You can use the pub.cache services to implement checkpoint restart in services that you write to make them more robust. You use the pub.cache services to write state information and other pertinent data to the cache. If the Integration Server on which your service is executing becomes unavailable, your service will be able to check the state information in the cache and resume processing at the point where the service was interrupted.

**Note:** The pub.cache services are a tool for maintaining state information for the short term. It is up to the developer of the services to make sure they keep track of state information and correctly handle restarts.

The following diagram shows the logic of a flow service that uses checkpoint restart.



## Summary of Elements in this Folder

The following elements are available in this folder:

Element	Package and Description
<a href="#">pub.cache:containsKey</a>	WmPublic. Checks whether an element with the specified key exists in the cache.
<a href="#">pub.cache:get</a>	WmPublic. Retrieves the value of a cached element for the specified key.
<a href="#">pub.cache:getAll</a>	WmPublic. Retrieves the values of the cached elements for the specified keys.
<a href="#">pub.cache:getKeys</a>	WmPublic. Retrieves the keys of all elements available in the cache.
<a href="#">pub.cache:put</a>	WmPublic. Populates a cached element with a specified key-value pair.
<a href="#">pub.cache:putAll</a>	WmPublic. Populates a collection of elements in the cache with the specified key-value pairs.
<a href="#">pub.cache:remove</a>	WmPublic. Removes the cached element associated with the specified key.
<a href="#">pub.cache:removeAll</a>	WmPublic. Removes the cached elements associated with a list of keys or, if no keys are specified, removes all elements from the cache.
<a href="#">pub.cache:search</a>	WmPublic. Searches through an indexed cache and returns the results.
<a href="#">pub.cache.admin:clearAllCaches</a>	WmPublic. Deletes all of the elements from all caches contained in the specified cache manager.
<a href="#">pub.cache.admin:clearCache</a>	WmPublic. Deletes all elements from the specified cache.
<a href="#">pub.cache.admin:disableCache</a>	WmPublic. Disables a cache.

Element	Package and Description
<a href="#">pub.cache.admin:enableCache</a>	WmPublic. Enables a cache.
<a href="#">pub.cache.admin:evictExpiredElements</a>	WmPublic. Deletes all of the expired elements from a cache.
<a href="#">pub.cache.admin:isCacheDisabled</a>	WmPublic. Checks whether the cache is disabled.
<a href="#">pub.cache.atomic:putIfAbsent</a>	WmPublic. Adds an element to the cache if the cache does not contain an element with the specified key.
<a href="#">pub.cache.atomic:remove</a>	WmPublic. Removes the cached element associated with the specified key and value.
<a href="#">pub.cache.atomic:replace</a>	WmPublic. Replaces the cached element value with the supplied value.
<a href="#">pub.cache.atomic:replaceIfKeyExists</a>	WmPublic. Replaces the cached element if an element for the specified key exists in a cache.
<a href="#">pub.cache.bulk:isClusterBulkLoadEnabled</a>	WmPublic. Checks whether the cache on at least one Integration Server node in the Terracotta Server Array cluster is enabled for bulk loading.
<a href="#">pub.cache.bulk:isNodeBulkLoadEnabled</a>	WmPublic. Checks whether the cache of the current Integration Server node in the Terracotta Server Array cluster is bulk-load enabled.
<a href="#">pub.cache.bulk:setNodeBulkLoadEnabled</a>	WmPublic. Enables or disables bulk loading mode in the current Integration Server node for the cache.
<a href="#">pub.cache.bulk:waitUntilClusterBulkLoadComplete</a>	WmPublic. Indicates whether Integration Server delays execution of the next step in a flow service until all of the Integration Server nodes in the

Element	Package and Description
	Terracotta Server Array cluster disable bulk loading for the cache.
<a href="#">pub.cache.lock:acquireLock</a>	WmPublic. Acquires a lock on the cached element that contains the specified key.
<a href="#">pub.cache.lock:isLockedByCurrentThread</a>	WmPublic. Checks whether the current thread is holding a lock on the cached element for the specified key.
<a href="#">pub.cache.lock:releaseLock</a>	WmPublic. Releases a lock on the element for the specified key.

## pub.cache:containsKey

WmPublic. Checks whether an element with the specified key exists in the cache.

### Input Parameters

<i>cacheManagerName</i>	<b>String</b> Name of the cache manager that manages the cache. This parameter is case sensitive.
<i>cacheName</i>	<b>String</b> Name of the cache that contains the key. This parameter is case sensitive.
<i>key</i>	<b>Object</b> Key of the element for whose existence you are checking.

### Output Parameters

<i>found</i>	<b>String</b> Indicates whether the key exists in the cache. <ul style="list-style-type: none"> <li>■ <code>true</code> indicates that the key exists in the cache.</li> <li>■ <code>false</code> indicates that the key does not exist in the cache.</li> </ul>
--------------	--

### Usage Notes

The `pub.cache:containsKey` service does not verify whether the element is expired. If the element of the specified key is expired, and the element is present in the cache, `pub.cache:containsKey` returns *found* with a value of `true`.

---

Integration Server issues a `ServiceException` in the following cases:

- If you do not specify all required input parameters.
- If Integration Server cannot find the specified cache manager or cache.
- If a cache operation fails.

---

## pub.cache:get

WmPublic. Retrieves the value of a cached element for the specified key.

### Input Parameters

---

*cacheManagerName*     **String** Name of the cache manager that manages the cache. This parameter is case sensitive.

*cacheName*             **String** Name of the cache from which to retrieve the element. This parameter is case sensitive.

*key*                     **Object** Key associated with the cached value.

*useLoader*             **String** Optional. Indicates whether to use a cache loader to retrieve the element from the cache. Set to:

- `true` to use a cache loader to retrieve the element if it is not already in the cache.

**Note:** If *useLoader* is `true` and no loaders are configured for the cache, Integration Server issues a `ServiceException`.

- `false` to retrieve the element from the cache without a cache loader. This is the default setting.

### Output Parameters

---

*value*                   **Object** Value of the element retrieved from the cache.

### Usage Notes

If *useLoader* is set to `false` and `pub.cache:get` finds the key in the cache, the results contain the value of the element. If the key does not exist in the cache, `pub.cache:get` returns a null in the *value* parameter.

If *useLoader* is set to `true` and `pub.cache:get` cannot find the specified key in the cache, it uses the configured cache loader to find the key in the system-of-record (SOR). If the SOR contains the key, `pub.cache:get` returns the value of the element. If the key does not exist in the SOR, `pub.cache:get` returns a null in the *value* parameter.

The `pub.cache:get` service returns a null in the *value* parameter if:

- You run the service on a disabled cache.
- The service cannot find the specified key.
- The element associated with the specified key is expired.

Integration Server issues a `ServiceException` in the following cases:

- If *useLoader* is `true` and no loaders are configured for the cache.
- If you do not specify all required input parameters.
- If Integration Server cannot find the specified cache manager or cache.
- If a cache operation fails.

---

## pub.cache:getAll

WmPublic. Retrieves the values of the cached elements for the specified keys.

### Input Parameters

---

<i>cacheManagerName</i>	<b>String</b> Name of the cache manager that manages the cache. This parameter is case sensitive.
<i>cacheName</i>	<b>String</b> Name of the cache from which to retrieve the elements. This parameter is case sensitive.
<i>keys</i>	<b>Object List</b> Keys associated with the cached elements for which you want to retrieve values.
<i>useLoader</i>	<b>String</b> Optional. Indicates whether to use a cache loader to retrieve the elements from the cache. Set to: <ul style="list-style-type: none"> <li>■ <code>true</code> to use a cache loader to retrieve an element if the element is not already in the cache.</li> </ul>

**Note:** If *useLoader* is `true` and no loaders are configured for the cache, Integration Server issues a `ServiceException`.

- `false` to retrieve an element from the cache without a cache loader. This is the default setting.

## Output Parameters

*elements*      **Document List** An `IData[]` containing the values for the requested cached elements.

<u>Key</u>	<u>Description</u>
<i>key</i>	<b>Object</b> Key associated with the cached element.
<i>value</i>	<b>Object</b> Value of the element retrieved from the cache.

## Usage Notes

The `pub.cache:getAll` service can be memory-intensive because it loads all of the specified keys into memory.

If `useLoader` is set to `false` and `pub.cache:get` finds the key in the cache, the results contain the value of the element. If the key does not exist in the cache, `pub.cache:getAll` returns a null in the *value* parameter.

If `useLoader` is set to `true` and `pub.cache:get` cannot find the specified key in the cache, it uses the configured cache loader to find the key in the system-of-record (SOR). If the SOR contains the key, `pub.cache:get` returns the value of the element. If the key does not exist in the SOR, `pub.cache:getAll` returns a null in the *value* parameter.

The `pub.cache:getAll` service returns a null in the *elements/value* parameter if:

- You run the service on a disabled cache.
- The service cannot find the specified key.
- The element associated with the specified key is expired.

Integration Server issues a `ServiceException` in the following cases:

- If `useLoader` is `true` and no loaders are configured for the cache.
- If you do not specify all required input parameters.
- If Integration Server cannot find the specified cache manager or cache.
- If a cache operation fails.

## pub.cache:getKeys

WmPublic. Retrieves the keys of all elements available in the cache.

---

## Input Parameters

---

<i>cacheManagerName</i>	<b>String</b> Name of the cache manager that manages the cache. This parameter is case sensitive.
<i>cacheName</i>	<b>String</b> Name of the cache from which to retrieve the keys. This parameter is case sensitive.
<i>excludeExpiredKeys</i>	<b>String</b> Optional. Indicates whether the service results exclude keys for expired elements. Set to: <ul style="list-style-type: none"><li>■ <code>true</code> to exclude the keys of expired elements. This is the default.</li><li>■ <code>false</code> to include the keys of all elements.</li></ul>

---

## Output Parameters

---

<i>keys</i>	<b>Object List</b> List of keys of elements in the cache.
-------------	---

### Usage Notes

Retrieving all of the keys from a cache may negatively impact the performance of the calling service. This may be especially true when retrieving all of the keys for a large, distributed cache.

Retrieving all of the keys from a cache is a memory-intensive activity. It is possible for the list of available keys to be larger than the memory available on your system. In such cases, Integration Server might issue an `OutOfMemoryError` and become unresponsive.

Integration Server issues a `ServiceException` in the following cases:

- If you do not specify all required input parameters.
- If Integration Server cannot find the specified cache manager or cache.
- If a cache operation fails.

---

## pub.cache:put

WmPublic. Populates a cached element with a specified key-value pair.

---

### Input Parameters

---

<i>cacheManagerName</i>	<b>String</b> Name of the cache manager that manages the cache. This parameter is case sensitive.
-------------------------	---

---

<i>cacheName</i>	<b>String</b> Name of the cache in which to put the element. This parameter is case sensitive.
<i>key</i>	<b>Object</b> Key of the cached element.
<i>value</i>	<b>Object</b> Value of the cached element.
<i>useWriter</i>	<b>String</b> Optional. Indicates whether to use a cache writer to populate the value of the cached element in the system-of-record (SOR). Set to: <ul style="list-style-type: none"><li>■ <code>true</code> to use a cache writer to add the <i>value</i> to the cached element <i>and</i> the SOR.</li><li>■ <code>false</code> to add the <i>value</i> to the cached element without writing to the SOR. This is the default setting.</li></ul>

**Note:** If the *value* already exists in the cache, `pub.cache:put` replaces the value.

### Output Parameters

---

None.

### Usage Notes

If your cache is a distributed cache or is configured to use disk store or BigMemory, *key* and *value* must be Java Serializable objects.

If the element with the specified *key* does not already exist in the cache, the `pub.cache:put` service puts the element in the cache. If the element with the specified *key* already exists in the cache, `pub.cache:put` updates the element with the specified *value*.

If you run `pub.cache:put` on a disabled cache, the cache discards the call.

Integration Server issues a `ServiceException` in the following cases:

- If you do not specify all required input parameters.
- If *useWriter* is set to `true` and no writer is configured for the cache.
- If Integration Server cannot find the specified cache manager or cache.
- If a cache operation fails.

---

## pub.cache:putAll

`WmPublic`. Populates a collection of elements in the cache with the specified key-value pairs.

---

## Input Parameters

---

<i>cacheManagerName</i>	<b>String</b> Name of the cache manager that manages the cache. This parameter is case sensitive.
<i>cacheName</i>	<b>String</b> Name of the cache in which to put the elements. This parameter is case sensitive.
<i>elements</i>	<b>Document List</b> An IData[] containing the key-value pairs to put in the cache.

<u>Key</u>	<u>Description</u>
<i>key</i>	<b>Object</b> Key of the cached element.
<i>value</i>	<b>Object</b> Value of the cached element.

---

## Output Parameters

---

None.

### Usage Notes

If your cache is a distributed cache or is configured to use disk store or BigMemory, *key* and *value* must be Java Serializable objects.

The *key* and *value* input parameters cannot be null. The service might end with a NullPointerException if any *key* or *value* parameters are null.

If the element with the specified *key* does not already exist in the cache, the `pub.cache:putAll` service puts the element in the cache. If the element with the specified *key* already exists in the cache, `pub.cache:putAll` updates the element with the specified *value*.

If you run `pub.cache:putAll` on a disabled cache, the cache discards the call.

Integration Server issues a ServiceException in the following cases:

- If you do not specify all required input parameters.
- If Integration Server cannot find the specified cache manager or cache.
- If a cache operation fails.

---

## pub.cache:remove

WmPublic. Removes the cached element associated with the specified key.

### Input Parameters

---

<i>cacheManagerName</i>	<b>String</b> Name of the cache manager that manages the cache. This parameter is case sensitive.
<i>cacheName</i>	<b>String</b> Name of the cache from which to remove the element. This parameter is case sensitive.
<i>key</i>	<b>Object</b> Key of the element to remove.
<i>useWriter</i>	<b>String</b> Optional. Indicates whether to use a cache writer to remove the value of the cached element from the system-of-record (SOR). Set to: <ul style="list-style-type: none"><li>■ <code>true</code> to use a cache writer to remove the <i>value</i> from the cached element <i>and</i> the SOR.</li><li>■ <code>false</code> to remove the <i>value</i> to the cached element without using a cache writer. This is the default setting.</li></ul>

### Output Parameters

---

<i>removed</i>	<b>String</b> Indicates whether the element was removed. <ul style="list-style-type: none"><li>■ <code>true</code> indicates that the element was removed from the cache.</li><li>■ <code>false</code> indicates that the element was not removed from the cache.</li></ul>
----------------	---

### Usage Notes

If the element associated with the specified key does not exist, `pub.cache:remove` returns `false` for *removed*.

Integration Server issues a `ServiceException` in the following cases:

- If you do not specify all required input parameters.
- If Integration Server cannot find the specified cache manager or cache.
- If a cache operation fails.

---

## pub.cache:removeAll

WmPublic. Removes the cached elements associated with a list of keys or, if no keys are specified, removes all elements from the cache.

### Input Parameters

---

*cacheManagerName* **String** Name of the cache manager that manages the cache. This parameter is case sensitive.

*cacheName* **String** Name of the cache in which to put the elements. This parameter is case sensitive.

*keys* **Object List** Optional. Keys of the elements to remove from the cache. If you do not specify any keys, the service removes all of the elements from the specified cache.

### Output Parameters

---

None.

### Usage Notes

Integration Server issues a `ServiceException` in the following cases:

- If you do not specify all required input parameters.
- If Integration Server cannot find the specified cache manager or cache.
- If a cache operation fails.

---

## pub.cache:search

WmPublic. Searches through an indexed cache and returns the results. The `pub.cache:search` service accepts the selection criteria as an `IData` object and performs the search by converting the `IData` object to a search query of `Ehcache`.

**Note:** Before using this service, you must make the cache searchable. For more information about making a cache searchable, see *webMethods Integration Server Administrator's Guide*.

## Input Parameters

---

<i>cacheManagerName</i>	<b>String</b> Name of the cache manager that manages the cache. This parameter is case sensitive.
<i>cacheName</i>	<b>String</b> Name of the cache from which the search results are to be returned. This parameter is case sensitive.
<i>includeAttributes</i>	<b>String</b> Optional. List of attributes that should be included in the search results.
<i>includeKey</i>	<p><b>String</b> Optional. Indicates whether or not the keys should be returned as part of the search results.</p> <ul style="list-style-type: none"> <li>■ <code>true</code> indicates that the keys should be returned as part of the search results.</li> <li>■ <code>false</code> indicates that the keys should not be returned as part of the search results. This is the default.</li> </ul> <p><b>Note:</b> You should set this parameter to <code>true</code> only if the value is required for your programming logic.</p>
<i>includeValue</i>	<p><b>String</b> Optional. Indicates whether or not the values should be returned as part of the search results.</p> <ul style="list-style-type: none"> <li>■ <code>true</code> indicates that the values should be returned as part of the search results.</li> <li>■ <code>false</code> indicates that the values should not be returned as part of the search results. This is the default.</li> </ul> <p><b>Note:</b> You should set this parameter to <code>true</code> only if the value is required for your programming logic.</p>
<i>criteria</i>	<b>Document</b> A document (IData) that specifies the conditions based on which the data in the cache is to be queried.

<u>Value</u>	<u>Description</u>
<i>OP1</i>	<b>String</b> or <b>Document</b> A string operand or a nested document type that can be evaluated by a logical operator.
<i>OPR</i>	<b>String</b> A logical (and, or, and not) or comparison operator (between, gt (greater than), eq (equal to), and so on). For a

---

	complete list of operators that Integration Server supports as the value for the <i>criteria \OPR</i> parameter, see the Usage Notes for this service.
	<b>OP2</b> <b>String, Document, Object, or Object List</b> A string operand, nested document type, object, or object list that can be evaluated by a logical operator.
<i>maxResults</i>	<b>String</b> Optional. The maximum number of results to return. If <i>maxResults</i> is not specified, Integration Server returns all the results.
<i>groupBy</i>	<b>String List</b> Optional. The unique values of specified attributes based on which the search results are to be grouped. The attributes that you specify for grouping the results must be included in the list of attributes specified for the <i>includeAttributes</i> parameter.
<i>orderBy</i>	<p><b>String Table</b> Optional. Whether the service results are to be returned in ascending or descending order.</p> <p>Values for this parameter must be specified as a two-dimensional string array comprising the field name and whether DESCENDING or ASCENDING (case sensitive) in the following format:</p> <pre>{{"fieldName","order"}, {"fieldName","order"}}</pre> <p>Where <i>order</i> is ASCENDING or DESCENDING.</p> <p>For example, if you have configured name and age as the search attributes for the <i>personCache</i> cache that stores name, age, and address, the <i>orderBy</i> parameter can be specified as shown below:</p> <pre>{{"name","ASCENDING"}}</pre> <p>-OR-</p> <pre>{{"name","ASCENDING"}, {"age","DESCENDING"}}</pre>
<i>aggregator</i>	<p><b>String Table</b> Aggregator function such as average, count, max, min, or sum that you want to be applied on the search results. Values for this parameter should be specified as a two-dimensional string array in the following format:</p> <pre>{{"fieldName","aggregator"}, {"fieldName","aggregator"}}</pre> <p>Where <i>aggregator</i> can be average, count, max, min, or sum (case-insensitive).</p>

For example, `{{"age", "Max"}}` or `{{"income", "Average"}}`.

If you want to use the aggregator function, you must:

- Specify a value for the *groupBy* parameter.
- Set the *includeKey* and *includeValue* parameters to `false`.

## Output Parameters

*result*

**Object List** The returned results of the search.

Each object in the *result* parameter is of type `com.wm.app.b2b.client.cache.config.SearchResult`.

For more information about `com.wm.app.b2b.client.cache.config.SearchResult` type, see *webMethods Integration Server Java API Reference*.

Each object contains information about the cache element, key, value, search attribute values, and aggregator results, if any.

## Usage Notes

If you want to search a document (IData) in a cache, you must first create an attribute extractor class that can be used by Ehcache to extract the search attributes. Ehcache uses the extractor class and the provided search attribute information to search the cache when the `pub.cahce:search` service executes.

A sample IData that you can specify as the *criteria* input parameter is given below. The following input searches the `personCache` cache for entries, where age is greater than 60 and gender is female.

```

IData idata = IDataFactory.create();
//Cache information
    IDataMap map = new IDataMap(idata)        ;
    map.put("cacheManagerName", "SearchTestManager")    ;
    map.put("cacheName", "personCache")        ;
    map.put("includeKey", "true")            ;
    map.put("includeValue", "true")        ;
//Condition 1 - Age > 60
    IData input1 = IDataFactory.create()        ;
    IDataCursor cursor1 = input1.getCursor()    ;
    IDataUtil.put(cursor1, "OP1", "age")        ;
    IDataUtil.put(cursor1, "OPR", "gt")        ;
    IDataUtil.put(cursor1, "OP2", 60)        ;
//Condition 2 - Gender == Female
    IData input2 = IDataFactory.create()        ;
    IDataCursor cursor2= input2.getCursor()    ;
    IDataUtil.put(cursor2, "OP1", "gender")    ;
    IDataUtil.put(cursor2, "OPR", "eq")        ;
    IDataUtil.put(cursor2, "OP2", Person.Gender.FEMALE) ;
//Joining conditions, condition 1 && condition 2
    IData input3 = IDataFactory.create()        ;
    IDataCursor cursor3 = input3.getCursor()    ;
    IDataUtil.put(cursor3, "OP1", input1)        ;

```

```

        IDataUtil.put(cursor3, "OPR", "AND")      ;
        IDataUtil.put(cursor3, "OP2", input2)    ;
        map.put("criteria", input3) ;
//Run the search
        IData result = invoke(NSName.create("pub.cacheimpl:search"),
        idata);
        Object[] objectList = (Object[]) IDataUtil.get(result.getCursor(),
        "result");
        for (int i = 0; i < objectList.length; i++) {
        Person value = (Person) ((SearchResult) objectList[i]).getValue();
        System.out.println(value.getGender());
        System.out.println( value.getAge());
        }

```

When using the `between` operator for the `criteria \OPR` parameter, you must specify start and end values. That is, you must specify a two-element Object List with the 0 element as the start value and the first element as the end value. For example:

```

//Condition 1 < age <10
        IData input1 = IDataFactory.create() ;
        IDataCursor cursor1 = input1.getCursor() ;
        IDataUtil.put(cursor1, "OP1", "age") ;
        IDataUtil.put(cursor1, "OPR", "between") ;
        IDataUtil.put(cursor1, "OP2", {1,10}) ;

```

Integration Server supports the following operators as the value for `criteria \OPR` parameter:

Operators	Description
and	AND logical operator
between	Comparison between two values.
eq	Equals
gt	Greater than
ge	Greater than or equal to
ilike	Regular expression matcher using '?' and '*'.
in	Comparison operator meaning in the collection given as an argument.
isNull	Checks whether the value of an attribute is null.
lt	Less than.
le	Less than or equal to

Operators	Description
not	NOT logical operator
ne	Not equal to
notNull	Checks whether the value of an attribute with given name is NOT null.
or	OR logical operator

Because the state of the cache can change between search executions, include all of the aggregators to the search query at the same time, so that the returned aggregators are of the same iteration of the cache.

By default a search will return an unlimited number of results. If too many results are returned, Integration Server might issue an `OutOfMemoryError` and become unresponsive. Limit the size of the results using the `maxResults` parameter.

To ensure that Integration Server does not issue an `OutOfMemoryError` while invoking the service using Designer, do the following:

1. Copy the `ehcache-ee.jar` from `Software AG_directory/common/lib/` to the `Software AG_directory/Designer/eclipse/plugins/com.softwareag.is.runtimejars` folder.
2. Edit the `Software AG_directory/Designer/eclipse/plugins/com.softwareag.is.runtimejars/META-INF/MANIFEST.MF` file by appending `ehcache-ee.jar` to the "Bundle-ClassPath:" property. Maintain the characters per line in the MANIFEST.MF file while editing.
3. Restart Designer using the `-clean` option:

```
Software AG_directory
/Designer/eclipse/ eclipse.exe -clean
```

This cleans the folders in the following directory:

```
<workspace_location>\.metadata\.plugins\com.softwareag.is.runtimejars.
```

## pub.cache.admin:clearAllCaches

WmPublic. Deletes all of the elements from all caches contained in the specified cache manager.

### Input Parameters

*cacheManagerName* **String** Name of the cache manager that manages the cache. This parameter is case sensitive.

*cachePrefix*

**String** Optional. Prefix of the name of the caches from which to delete elements. If you do not specify a prefix, the service deletes elements from *all* caches in the cache manager.

**Note:** If `pub.cache.admin:clearAllCaches` does not find any caches with the specified prefix, the service does not clear any caches.

## Output Parameters

---

*status*

**String** The status indicating either `Error` or `Success`.

*message*

**String** The status message.

## Usage Notes

**Note:** Only users with administrator privileges can execute this service.

**Important:** The elements you delete from the caches are deleted permanently. You cannot undo this action.

If `pub.cache.admin:clearAllCaches` does not find any caches with the specified prefix, the service does not clear any caches, but returns `Success` for *status*.

You cannot clear caches managed by system cache managers.

If you have any existing locks on any of the elements in the cache, the `pub.cache.admin:clearAllCaches` service waits indefinitely until all locks are released.

Integration Server returns a *status* of `Error` in the following cases:

- If the cache is managed by a system cache manager.
- If you do not specify all required input parameters.
- If Integration Server cannot find the specified cache manager.
- If a cache operation fails.

---

## pub.cache.admin:clearCache

WmPublic. Deletes all elements from the specified cache.

### Input Parameters

---

*cacheManagerName*

**String** Name of the cache manager that manages the cache. This parameter is case sensitive.

---

*cacheName*      **String** Name of the cache from which to delete the elements. This parameter is case sensitive.

### Output Parameters

---

*status*      **String** The status indicating either `Error` or `Success`.

*message*      **String** The status message.

### Usage Notes

**Note:** Only users with administrator privileges can execute this service.

**Important:** The elements you delete from the cache are deleted permanently. You cannot undo this action.

You cannot clear caches managed by system cache managers.

If you have any existing locks on any of the elements in the cache, the `pub.cache.admin:clearCache` service waits indefinitely until all locks are released.

Integration Server returns a *status* of `Error` in the following cases:

- If the cache is managed by a system cache manager.
- If you do not specify all required input parameters.
- If Integration Server cannot find the specified cache manager or cache.
- If a cache operation fails.

---

## pub.cache.admin:disableCache

WmPublic. Disables a cache.

### Input Parameters

---

*cacheManagerName*      **String** Name of the cache manager that manages the cache. This parameter is case sensitive.

*cacheName*      **String** Name of the cache to disable. This parameter is case sensitive.

---

## Output Parameters

---

*status*                      **String** The status indicating either `Error` or `Success`.

*message*                    **String** The status message.

## Usage Notes

**Note:**     Only users with administrator privileges can execute this service.

When you disable a cache, the cached elements are unavailable for cache operations but still present in the cache. Use the [pub.cache.admin:enableCache](#) service to make the disabled cached elements available again.

You cannot disable caches managed by system cache managers.

After the `pub.cache.admin:disableCache` service disables a cache, other services cannot add, modify, delete, or retrieve its contents. For example, calls to the `pub.cache:get` service will return null.

The disabled state of a cache lasts only for the lifetime of its cache manager. When the cache manager is restarted, either by restarting Integration Server or clicking the reload icon on the **Settings > Caching** page in Integration Server Administrator, the cache is enabled.

When you disable a cache, Integration Server:

- Discards operations such as [pub.cache:get](#), [pub.cache:remove](#), and [pub.cache.atomic:replace](#) calls on the cache.
- Returns a null for [pub.cache:get](#) calls on the cache.

Integration Server returns a *status* of `Error` in the following cases:

- If the cache is managed by a system cache manager.
- If you do not specify all required input parameters.
- If Integration Server cannot find the specified cache manager or cache.
- If a cache operation fails.

---

## pub.cache.admin:enableCache

WmPublic. Enables a cache.

---

### Input Parameters

---

<i>cacheManagerName</i>	<b>String</b> Name of the cache manager that manages the cache. This parameter is case sensitive.
<i>cacheName</i>	<b>String</b> Name of the cache to enable. This parameter is case sensitive.

### Output Parameters

---

<i>status</i>	<b>String</b> The status indicating either <code>Error</code> or <code>Success</code> .
<i>message</i>	<b>String</b> The status message.

### Usage Notes

**Note:** Only users with administrator privileges can execute this service.

You cannot enable caches managed by system cache managers.

When you disable a cache, the cached elements are unavailable but still present in the cache. Use [pub.cache.admin.enableCache](#) to make the cached elements available again.

Integration Server returns a *status* of `Error` in the following cases:

- If the cache is managed by a system cache manager.
- If you do not specify all required input parameters.
- If Integration Server cannot find the specified cache manager or cache.
- If a cache operation fails.

---

## pub.cache.admin:evictExpiredElements

WmPublic. Deletes all of the expired elements from a cache.

### Input Parameters

---

<i>cacheManagerName</i>	<b>String</b> Name of the cache manager that manages the cache. This parameter is case sensitive.
<i>cacheName</i>	<b>String</b> Name of the cache from which you want to delete the expired elements. This parameter is case sensitive.

---

## Output Parameters

*status*                    **String** The status indicating either `Error` or `Success`.

*message*                    **String** The status message.

## Usage Notes

**Note:** Only users with administrator privileges can execute this service.

**Important:** The elements you delete from the cache are deleted permanently. You cannot undo this action.

The `pub.cache.admin:evictExpiredElements` service can take longer to delete expired elements depending on the number of elements contained in the cache. If the cache contains a large number of elements, `pub.cache.admin:evictExpiredElements` might take longer to process the request than when the cache contains fewer elements.

Integration Server returns a *status* of `Error` in the following cases:

- If the cache is managed by a system cache manager.
- If you do not specify all required input parameters.
- If Integration Server cannot find the specified cache manager or cache.
- If a cache operation fails.

---

## pub.cache.admin:isCacheDisabled

WmPublic. Checks whether the cache is disabled.

### Input Parameters

*cacheManagerName*        **String** Name of the cache manager that manages the cache. This parameter is case sensitive.

*cacheName*                    **String** Name of the cache for which you want to check the status (whether it is disabled or enabled). This parameter is case sensitive.

### Output Parameters

*isDisabled*                    **String** Indicates whether the cache is disabled. A value of:

- `true` indicates that the cache is disabled.
- `false` indicates that the cache is enabled.

*status* **String** Conditional. The status indicating `ERROR`. The service returns `ERROR` for *status* only if the service fails when checking whether the cache is disabled.

*message* **String** Conditional. The status message of the error. The service returns *message* with *status* only if an error occurs when checking whether the cache is disabled.

### Usage Notes

**Note:** Only users with administrator privileges can execute this service.

`pub.cache.admin.isCacheDisabled` returns `ERROR` for *status* and an accompanying *message* in the following cases:

- If you do not specify all required input parameters.
- If Integration Server cannot find the specified cache manager or cache.
- If a cache operation fails.

---

## pub.cache.atomic:putIfAbsent

`WmPublic`. Adds an element to the cache if the cache does not contain an element with the specified key.

### Input Parameters

---

*cacheManagerName* **String** Name of the cache manager that manages the cache. This parameter is case sensitive.

*cacheName* **String** Name of the cache to which to add the element. This parameter is case sensitive.

*key* **Object** Key of the element.

*value* **Object** Value to add to the element.

---

## Output Parameters

---

<i>keyExists</i>	<p><b>String</b> Indicates whether an element with the specified key exists in the cache.</p> <ul style="list-style-type: none"> <li>■ <code>true</code> indicates that an element with the key already exists in the cache.</li> <li>■ <code>false</code> indicates that an element with the key does not exist in the cache.</li> </ul>
<i>oldValue</i>	<p><b>Object</b> Conditional. Value of the element that already exists in the cache. Returned only if <i>keyExists</i> is <code>true</code>.</p>

## Usage Notes

The `pub.cache.atomic:putIfAbsent` service adds the specified *value* to the cache only if the specified *key* does not already exist in the cache. If the specified *key* already exists in the cache, `pub.cache.atomic:putIfAbsent` returns `true` for *keyExists* and the value of the cached element for *oldValue*.

Integration Server issues a `ServiceException` in the following cases:

- If you do not specify all required input parameters.
- If Integration Server cannot find the specified cache manager or cache.
- If a cache operation fails.

---

## pub.cache.atomic:remove

`WmPublic`. Removes the cached element associated with the specified key and value.

### Input Parameters

---

<i>cacheManagerName</i>	<p><b>String</b> Name of the cache manager that manages the cache. This parameter is case sensitive.</p>
<i>cacheName</i>	<p><b>String</b> Name of the cache from which to remove the element. This parameter is case sensitive.</p>
<i>key</i>	<p><b>Object</b> Key of the element to remove.</p>
<i>value</i>	<p><b>Object</b> Value of the element to remove.</p>

---

## Output Parameters

---

<i>removed</i>	<b>String</b> Indicates whether the service removed the element. <ul style="list-style-type: none"><li>■ <code>true</code> indicates that the element was removed from the cache.</li><li>■ <code>false</code> indicates that the element was not removed from the cache. The service returns <code>false</code> if the specified <i>key</i> and <i>value</i> do not exist in the cache.</li></ul>
----------------	--

## Usage Notes

Integration Server issues a `ServiceException` in the following cases:

- If you do not specify all required input parameters.
- If Integration Server cannot find the specified cache manager or cache.
- If a cache operation fails.

---

## pub.cache.atomic:replace

WmPublic. Replaces the cached element value with the supplied value.

## Input Parameters

---

<i>cacheManagerName</i>	<b>String</b> Name of the cache manager that manages the cache. This parameter is case sensitive.
<i>cacheName</i>	<b>String</b> Name of the cache that contains the value you want to replace. This parameter is case sensitive.
<i>key</i>	<b>Object</b> Key of the cache element whose value you want to replace.
<i>oldValue</i>	<b>Object</b> Value of the cached element.
<i>newValue</i>	<b>Object</b> Value with which to replace the value of the cached element.

## Output Parameters

---

<i>replaced</i>	<b>String</b> Indicates whether the service replaced <i>oldValue</i> with <i>newValue</i> in the cache.
-----------------	---

- `true` indicates that the value was replaced in the cache.
- `false` indicates that the value was not replaced in the cache.

### Usage Notes

The `pub.cache.atomic.replace` service replaces an element in the cache *only* if an element with the values of the specified `key` and `oldValue` parameters already exist in the cache. If an element exist in the cache with the specified `key` and `oldValue`, `pub.cache.atomic.replace` replaces the cached element with the specified `key` and `newValue`.

Integration Server issues a `ServiceException` in the following cases:

- If you do not specify all required input parameters.
- If Integration Server cannot find the specified cache manager or cache.
- If a cache operation fails.

---

## pub.cache.atomic:replaceIfKeyExists

`WmPublic`. Replaces the cached element if an element for the specified key exists in a cache.

### Input Parameters

---

<i>cacheManagerName</i>	<b>String</b> Name of the cache manager that manages the cache. This parameter is case sensitive.
<i>cacheName</i>	<b>String</b> Name of the cache that contains the key. This parameter is case sensitive.
<i>key</i>	<b>Object</b> Key of the element whose value you want to replace.
<i>value</i>	<b>Object</b> Value with which to replace the value of the cached element.

### Output Parameters

---

<i>keyExists</i>	<b>String</b> Indicates whether the element with the passed key exists in the cache. <ul style="list-style-type: none"> <li>■ <code>true</code> indicates that the element with the key already exists in the cache.</li> </ul>
------------------	---

- `false` indicates that the element with the key does not exist in the cache.

*oldValue*

**Object** Conditional. Value of the element the service replaced with *value*. Returned only if *keyExists* is `true`.

### Usage Notes

The `pub.cache.atomic:replaceAllKeyExists` service replaces an element in the cache *only* if the specified *key* already exists in the cache. If an element with the specified *key* exists, `pub.cache.atomic:replaceAllKeyExists` replaces value of the cached element with the one specified in *value*.

Integration Server issues a `ServiceException` in the following cases:

- If you do not specify all required input parameters.
- If Integration Server cannot find the specified cache manager or cache.
- If a cache operation fails.

---

## pub.cache.bulk:isClusterBulkLoadEnabled

`WmPublic`. Checks whether the cache on at least one Integration Server node in the Terracotta Server Array cluster is enabled for bulk loading.

### Input Parameters

---

*cacheManagerName* **String** Name of the cache manager that manages the cache. This parameter is case sensitive.

*cacheName* **String** Name of the cache to check for bulk loading. This parameter is case sensitive.

### Output Parameters

---

*enabled* **String** Indicates whether the cache on at least one Integration Server node of the Terracotta Server Array cluster is in bulk-load mode.

- `true` indicates that at least one node is in bulk-load mode.
- `false` indicates that the node is not in bulk-load mode.

### Usage Notes

The `pub.cache.bulk:isClusterBulkLoadEnabled` service applies only to distributed caches. If you run `pub.cache.bulk:isClusterBulkLoadEnabled` on a local cache, the service returns `false` for *enabled*.

Integration Server issues a `ServiceException` in the following cases:

- If you do not specify all required input parameters.
- If Integration Server cannot find the specified cache manager or cache.
- If a cache operation fails.

---

## pub.cache.bulk:isNodeBulkLoadEnabled

`WmPublic`. Checks whether the cache of the current Integration Server node in the Terracotta Server Array cluster is bulk-load enabled.

### Input Parameters

---

<i>cacheManagerName</i>	<b>String</b> Name of the cache manager that manages the cache. This parameter is case sensitive.
<i>cacheName</i>	<b>String</b> Name of the distributed cache to check for bulk loading. This parameter is case sensitive.

### Output Parameters

---

<i>enabled</i>	<b>String</b> Indicates whether the cache is enabled for bulk loading. <ul style="list-style-type: none"> <li>■ <code>true</code> indicates that the Integration Server node calling the service is the same one that enabled bulk-loading.</li> <li>■ <code>false</code> indicates one of the following: <ul style="list-style-type: none"> <li>■ The Integration Server node calling the service was not the same node that enabled bulk loading.</li> <li>■ The specified cache is a local cache.</li> </ul> </li> </ul>
----------------	---

### Usage Notes

The `pub.cache.bulk:isNodeBulkLoadEnabled` service applies only to distributed caches.

Integration Server issues a `ServiceException` in the following cases:

- If you do not specify all required input parameters.

- If Integration Server cannot find the specified cache manager or cache.
- If a cache operation fails.

---

## pub.cache.bulk:setNodeBulkLoadEnabled

WmPublic. Enables or disables bulk loading mode in the current Integration Server node for the cache.

### Input Parameters

---

<i>cacheManagerName</i>	<b>String</b> Name of the cache manager that manages the cache. This parameter is case sensitive.
<i>cacheName</i>	<b>String</b> Name of the cache on which to enable or disable bulk loading. This parameter is case sensitive.
<i>bulkMode</i>	<b>String</b> Indicates whether to enable the bulk-load mode for the cache. Set to: <ul style="list-style-type: none"><li>■ <code>true</code> to enable the bulk-load mode. This is the default.</li><li>■ <code>false</code> to disable the bulk-load mode.</li></ul>

### Output Parameters

---

None.

### Usage Notes

The `pub.cache.bulk:setNodeBulkLoadEnabled` service applies only to distributed caches.

The `pub.cache.bulk:setNodeBulkLoadEnabled` service does nothing if you:

- Try to enable bulk loading (*bulkMode* set to `true`) when the node is already in bulk-load mode.
- Try to disable bulk loading (*bulkMode* set to `false`) when the node is not already in bulk-load mode.

Integration Server issues a `ServiceException` in the following cases:

- If you do not specify all required input parameters.
- If Integration Server cannot find the specified cache manager or cache.
- If a cache operation fails.

---

## pub.cache.bulk:waitUntilClusterBulkLoadComplete

WmPublic. Indicates whether Integration Server delays execution of the next step in a flow service until all of the Integration Server nodes in the Terracotta Server Array cluster disable bulk loading for the cache.

### Input Parameters

---

*cacheManagerName*     **String** Name of the cache manager that manages the cache. This parameter is case sensitive.

*cacheName*             **String** Name of the cache on which to hold processing. This parameter is case sensitive.

### Output Parameters

---

None.

### Usage Notes

The `pub.cache.bulk:waitUntilClusterBulkLoadComplete` service applies only to distributed caches.

If none of the nodes are bulk-load enabled, Integration Server immediately executes the next step in the flow service without waiting.

Integration Server issues a `ServiceException` in the following cases:

- If you do not specify all required input parameters.
- If Integration Server cannot find the specified cache manager or cache.
- If a cache operation fails.

---

## pub.cache.lock:acquireLock

WmPublic. Acquires a lock on the cached element that contains the specified key.

### Input Parameters

---

*cacheManagerName*     **String** Name of the cache manager that manages the cache. This parameter is case sensitive.

*cacheName*             **String** Name of the cache that contains the element to lock. This parameter is case sensitive.

*key* **Object** Key of the cached element to lock.

*lockType* **String** Optional. Type of lock you want to place on the cached element.

<b>Key</b>	<b>Description</b>
<i>read</i>	Places a read lock on the key. This is the default.
<i>write</i>	Places a write lock on the key.

*lockWaitTime* **String** Optional. Amount of time in milliseconds to wait to acquire a lock before timing out.

<b>Key</b>	<b>Description</b>
Any integer less than 0	The service waits indefinitely until it acquires a lock.
0	The service fails if it is unable to acquire the lock immediately. This is the default.
Any integer greater than 0	The service fails if it is unable to acquire the lock within the specified time.

## Output Parameters

None.

## Usage Notes

If the `pub.cache.lock:acquireLock` service acquires a lock, the element remains locked until released with the `pub.cache.lock:releaseLock` service.

You must acquire and release a lock in the same thread in which `pub.cache.lock:acquireLock` is executing. Failing to do so could cause the key to remain locked indefinitely.

When using the debug flow service to step through a flow service, depending on your breakpoint settings Designer might use a new thread for each step. You cannot release a lock acquired in a previous step of the same flow if break points are triggered on Designer or if you are stepping through a flow service. To avoid orphaned locks, disable invocations of `pub.cache.lock:acquireLock` and `pub.cache.lock:releaseLock` before

stepping through the flow service or make sure that `pub.cache.lock:acquireLock` and `pub.cache.lock:releaseLock` services are called between the boundaries of two breakpoints.

Integration Server issues a `ServiceException` in the following cases:

- If you do not specify all required input parameters.
- Whether or not a cached element needs to be locked depends on the application that is using the element. For example, if an application retrieves and holds an element for a time and needs to prevent another client from changing the element during this time, the application should acquire a lock on the element. However, if the application does not hold the element, a lock might not be required.
- If Integration Server cannot find the specified cache manager or cache.
- If `pub.cache.lock:acquireLock` is unable to acquire a lock.
- If the specified `lockWaitTime` is not a valid integer.
- If a cache operation fails.

---

## pub.cache.lock:isLockedByCurrentThread

`WmPublic`. Checks whether the current thread is holding a lock on the cached element for the specified key.

### Input Parameters

---

*cacheManagerName*    **String** Name of the cache manager that manages the cache. This parameter is case sensitive.

*cacheName*    **String** Name of the cache that contains the key. This parameter is case sensitive.

*key*    **Object** Key for which you want to check locks.

*lockType*    **String** Optional. Type of lock for which to check the cached element.

Key	Description
<i>read</i>	The service checks whether the read lock is held for the current thread. This is the default.
<i>write</i>	The service checks whether the write lock is held for the current thread.

---

## Output Parameters

---

*locked* **String** Indicates whether the cached element is locked by the current thread.

- `true` indicates that the element is locked.
- `false` indicates that the element is not locked.

**Note:** If the specified cache is a local cache and the specified *lockType* is *read*, the service always returns *false*.

## Usage Notes

Integration Server issues a `ServiceException` in the following cases:

- If you do not specify all required input parameters.
- If Integration Server cannot find the specified cache manager or cache.

---

## pub.cache.lock:releaseLock

WmPublic. Releases a lock on the element for the specified key.

## Input Parameters

---

*cacheManagerName* **String** Name of the cache manager that manages the cache. This parameter is case sensitive.

*cacheName* **String** Name of the cache that contains the key. This parameter is case sensitive.

*key* **Object** Key of the cached element to unlock.

*lockType* **String** Optional. Type of lock you want to release from the cached element.

Key	Description
<i>read</i>	Releases a read lock. This is the default.
<i>write</i>	Releases a write lock.

## Output Parameters

---

None.

## Usage Notes

You must call `pub.cache.lock:releaseLock` from the same thread from which you call `pub.cache.lock:acquireLock`. Failing to do so could cause the key to remain locked indefinitely.

Integration Server issues a `ServiceException` in the following cases:

- If you do not specify all required input parameters.
- If Integration Server cannot find the specified cache manager or cache.
- If you try to release a read lock on a cached element for which a write lock is set, or release a write lock on a cached element for which a read lock is set.

# 4 Client Folder

---

- Summary of Elements in this Folder ..... 94

You use the elements in the client folder to formulate and submit requests to HTTP, FTP, SFTP, e-mail, and LDAP servers.

## Summary of Elements in this Folder

The following elements are available in this folder:

Element	Package and Description
<a href="#">pub.client.ftp</a>	WmPublic. Performs a series of FTP actions.
<a href="#">pub.client.ftp:append</a>	WmPublic. Appends data to a remote file.
<a href="#">pub.client.ftp:cd</a>	WmPublic. Changes the working directory on the FTP server. (This service corresponds to the standard FTP command <code>cd dirpath</code> .)
<a href="#">pub.client.ftp:cdls</a>	WmPublic. Changes the working directory on the FTP server and retrieves a list of file names. (This service corresponds to the standard FTP commands <code>cd dirpath</code> and <code>ls namePattern</code> .)
<a href="#">pub.client.ftp:delete</a>	WmPublic. Deletes a file in the current working directory on an FTP server. (This service corresponds to the standard FTP command <code>delete somefile</code> .)
<a href="#">pub.client.ftp:dir</a>	WmPublic. Retrieves the file list during an FTP session. (This service corresponds to the standard FTP command <code>dir namepattern</code> .)
<a href="#">pub.client.ftp:get</a>	WmPublic. Retrieves a file from a remote FTP server. (This service corresponds to the standard FTP command <code>get</code> .)
<a href="#">pub.client.ftp:getCompletedNotification</a>	WmPublic. A publishable document type that represents the document published to notify parties that an FTP get command has completed.
<a href="#">pub.client.ftp:login</a>	WmPublic. Connects to a remote FTP server and logs in with a specified user name and password.

Element	Package and Description
<a href="#">pub.client.ftp:logout</a>	WmPublic. Logs off of the FTP server and ends the current FTP session.
<a href="#">pub.client.ftp:ls</a>	WmPublic. Retrieves the file list during an FTP session. (This service corresponds to the standard FTP command <code>ls namepattern</code> .)
<a href="#">pub.client.ftp:mdelete</a>	WmPublic. Deletes multiple files in the current working directory on an FTP server. (This service corresponds to the standard FTP command <code>mdelete pattern</code> .)
<a href="#">pub.client.ftp:mget</a>	WmPublic. Transfers multiple files from the remote FTP server. (This service corresponds to the standard FTP command <code>mget</code> .)
<a href="#">pub.client.ftp:mput</a>	WmPublic. Transfers multiple files to a remote FTP server. (This service corresponds to the standard FTP command <code>input</code> .)
<a href="#">pub.client.ftp:put</a>	WmPublic. Transfers a file to a remote FTP server. (This service corresponds to the standard FTP command <code>put</code> .)
<a href="#">pub.client.ftp:putCompletedNotification</a>	WmPublic. A publishable document type that represents the document published to notify parties that an FTP put command has completed.
<a href="#">pub.client.ftp:quote</a>	WmPublic. Executes a given FTP command.
<a href="#">pub.client.ftp:rename</a>	WmPublic. Renames a file on a remote FTP server. (This service corresponds to the standard FTP command <code>rename</code> .)
<a href="#">pub.client.ftp:sessioninfo</a>	WmPublic. Returns session information for all of the FTP servers that users are currently logged into.
<a href="#">pub.client:http</a>	WmPublic. Issues an HTTP request that you specify and returns the HTTP response.
<a href="#">pub.client.ldap:add</a>	WmPublic. Inserts a new entry into the directory.

Element	Package and Description
<a href="#">pub.client.ldap:bind</a>	WmPublic. Performs an LDAP bind operation that associates the connection with the specified principal.
<a href="#">pub.client.ldap:cancelNotification</a>	WmPublic. Cancels a previously created notification request.
<a href="#">pub.client.ldap:compare</a>	WmPublic. Compares the value of an attribute in the LDAP directory with a value specified by the service.
<a href="#">pub.client.ldap:delete</a>	WmPublic. Removes an entry from the directory.
<a href="#">pub.client.ldap:modify</a>	WmPublic. Performs an LDAP modify operation that allows you to specify a list of attributes with corresponding lists of values to add to, replace, or remove from the directory entry.
<a href="#">pub.client.ldap:registerNotification</a>	WmPublic. Creates a notification (or "persistent search") that causes Integration Server to listen for LDAP events. When the notification gets an event, the specified service is called.
<a href="#">pub.client.ldap:rename</a>	WmPublic. Performs an LDAP rename (move) operation allowing you to rename an entry.
<a href="#">pub.client.ldap:search</a>	WmPublic. Performs an LDAP search operation with the specified parameters and returns the results of the search.
<a href="#">pub.client.oauth:executeRequest</a>	WmPublic. Allow clients to access protected resources using an OAuth token.
<a href="#">pub.client.sftp:cd</a>	WmPublic. Changes the working directory on the remote SFTP server.
<a href="#">pub.client.sftp:chgrp</a>	WmPublic. Changes the group ownership of one or more remote files.
<a href="#">pub.client.sftp:chmod</a>	WmPublic. Changes permissions of one or more remote files.

Element	Package and Description
<code>pub.client.sftp:chown</code>	WmPublic. Changes the user of one or more remote files.
<code>pub.client.sftp:get</code>	WmPublic. Retrieves a file from a remote SFTP server and saves it on the local machine.
<code>pub.client.sftp:login</code>	WmPublic. Connects to a remote SFTP server and logs in with the specified SFTP user alias.
<code>pub.client.sftp:logout</code>	WmPublic. Logs off the user from the SFTP server and ends the current SFTP session.
<code>pub.client.sftp:ls</code>	WmPublic. Retrieves the remote directory listing of the specified path or current remote directory if path is not specified.
<code>pub.client.sftp:mkdir</code>	WmPublic. Creates a new remote directory.
<code>pub.client.sftp:put</code>	WmPublic. Transfers a file to a remote SFTP server.
<code>pub.client.sftp:pwd</code>	WmPublic. Displays the remote working directory on the SFTP server.
<code>pub.client.sftp:rename</code>	WmPublic. Renames a file or directory on a remote SFTP server.
<code>pub.client.sftp:rm</code>	WmPublic. Deletes one or more remote files on the SFTP server.
<code>pub.client.sftp:rmdir</code>	WmPublic. Deletes one or more remote directories on the SFTP server.
<code>pub.client.sftp:symlink</code>	WmPublic. Creates a symbolic link between the old path and the new path of a file.
<code>pub.client:smtp</code>	WmPublic. Sends a MIME-type e-mail message.
<code>pub.client:soapClient</code>	WmPublic. Creates and sends SOAP 1.1 and SOAP 1.2 messages over HTTP, HTTPS, or

Element	Package and Description
	JMS transports for any style/use combination supported by Integration Server.
<a href="#">pub.client:soapHTTP</a>	WmPublic. <i>Deprecated</i> - There is no replacement service. Submits a SOAP message to a server via HTTP or HTTPS.
<a href="#">pub.client:soapRPC</a>	WmPublic. <i>Deprecated</i> - There is no replacement service. Submits a SOAP remote procedure call via HTTP or HTTPS.

## pub.client:ftp

WmPublic. Performs a series of FTP actions.

This service executes the following sequence:

1. Logs on to an FTP server.
2. Changes to a specified working directory.
3. Performs one of the following FTP commands: `ls`, `put`, or `get`.
4. Logs off the FTP server.

### Input Parameters

<i>serverhost</i>	<b>String</b> Name or IP address of the FTP server (for example, <code>ftp.netscape.com</code> ).
<i>serverport</i>	<b>String</b> Port number of the FTP server (for example, <code>4566</code> ).
<i>username</i>	<b>String</b> Valid FTP user of the remote FTP server (for example, <code>anonymous</code> ).
<i>password</i>	<b>String</b> Optional. Valid password of the FTP user.
<i>command</i>	<b>String</b> One of the following FTP commands: <code>ls</code> , <code>put</code> , or <code>get</code> .
<i>dirpath</i>	<b>String</b> Working directory of the FTP server (for example, <code>/tmp/pub</code> ). If the directory does not exist, the server throws an exception.
<i>transfermode</i>	<b>String</b> One of two FTP file transfer modes: <code>ascii</code> or <code>binary</code> . The default is <code>ascii</code> .

---

<i>transfertype</i>	<b>String</b> One of two FTP data transfer types: <code>passive</code> or <code>active</code> . The default is <code>active</code> .
<i>localfile</i>	<b>String</b> When <i>command</i> is set to <code>put</code> , this parameter specifies the name of the local file you want to transfer. (If <i>content</i> is specified, this field is ignored.)  When <i>command</i> is set to <code>get</code> , this parameter specifies the name of the local file in which you want the retrieved content saved.
<i>remotefile</i>	<b>String</b> When <i>command</i> is set to <code>put</code> , this parameter specifies the name of the remote file in which you want to save the data you are sending.  When <i>command</i> is set to <code>get</code> , this parameter specifies the name of the remote file that you want to retrieve.
<i>content</i>	<b>java.io.InputStream, byte[ ], or String</b> Data to be transferred when <i>command</i> is set to <code>put</code> .
<i>encoding</i>	<b>String</b> Optional. Character set in which the document is encoded. Specify an IANA-registered character set (for example, <code>ISO-8859-1</code> ).  This information is required to correctly convert the <code>String</code> object to bytes when performing a <code>get</code> . If parameter is null, the default JVM encoding is used.
<i>serverencoding</i>	<b>String</b> Optional. Specifies the encoding this service uses to convert the incoming FTP command string to encoded bytes that are supported by IANA and the FTP server. If the parameter is null, the service uses the 'UTF-8' character set to encode the FTP command <code>String</code> to bytes.
<i>timeout</i>	<b>String</b> Time (measured in seconds) this service waits for the FTP server response on the command channel before timing out and aborting the request. Default is to wait forever.
<i>putunique</i>	<b>String</b> Optional. Indicates whether to send a <code>STOR</code> or a <code>STOU</code> (Store as Unique File) command to the remote FTP server. Set to: <ul style="list-style-type: none"><li>■ <code>true</code> to send a <code>STOU</code> (Store as Unique File) command.</li><li>■ <code>false</code> to send a <code>STOR</code> command. This is the default.</li></ul>
<i>secure</i>	<b>Document</b> Indicates whether the FTP session is with a secure FTP server.

**Note:** Integration Server does not support FTPS (FTP over SSL) requests through FTP proxy.

Key	Description
<i>auth</i>	<p><b>String</b> The kind of authentication mechanism to use: <code>None</code>, <code>SSL</code>, <code>TLS</code>, or <code>TLS-P</code>.</p> <p><code>None</code> specifies that the FTP session is with a non-secure FTP server. This is the default. If the value of <i>auth</i> is <code>None</code>, the <i>securedata</i> variable is ignored.</p> <p><code>TLS-P</code> is a shortcut that is equivalent to the sequence <code>AUTH TLS, PBSZ 0, and PROT P</code>. If the value of <i>auth</i> is <code>TLS-P</code>, the <i>securedata</i> variable is ignored.</p>
<i>securedata</i>	<p><b>String</b> Use the value <code>false</code> for a client sending <code>PROT C</code> (Data Channel Protection Level Clear).</p> <p>Use the value <code>true</code> for a client sending <code>PROT P</code> (Data Channel Protection Level Private).</p> <p><b>Note:</b> If you do not set a value, the default is <code>false</code>.</p>
<i>cleanlinefeeds</i>	<p><b>String</b> Optional. Indicates whether the service should retain or remove carriage return characters at the end of each line of text. Set to:</p> <ul style="list-style-type: none"> <li>■ <code>true</code> to remove carriage returns. This is the default.</li> <li>■ <code>false</code> to retain carriage returns.</li> </ul>
<i>newSession</i>	<p><b>String</b> Optional. Flag indicating whether a new FTP session will be created for this FTP operation. Set to:</p> <ul style="list-style-type: none"> <li>■ <code>yes</code> to create a new session for this FTP operation.</li> <li>■ <code>no</code> to use the current session, if one is available, for this FTP operation. This is the default.</li> </ul>
<i>clientTimeout</i>	<p><b>String</b> Optional. Specifies the idle time-out, measured in seconds, for this FTP session. If <i>clientTimeout</i> is set to 0 (zero), the session will never time out. The default is 600 seconds (10 minutes).</p>

<i>proxyAlias</i>	<p><b>String</b> Optional. Name of the proxy server alias for the proxy server to which Integration Server routes the FTP request.</p> <p>If you do not specify a <i>proxyAlias</i>, Integration Server routes the FTP request through the proxy server specified in the default FTP proxy alias. If there is no default FTP proxy alias, the action taken by Integration Server depends on the value specified for the <code>watt.net.proxy.useNonDefaultProxies</code> parameter.</p> <ul style="list-style-type: none"> <li>■ If the <code>watt.net.proxy.useNonDefaultProxies</code> parameter is set to true, Integration Server routes the FTP request through the proxy server in any configured FTP proxy alias. If the Integration Server does not have any defined FTP proxy aliases, Integration Server sends the FTP request directly to the FTP server or throws an exception depending on the settings specified for the <code>watt.net.proxy.fallbackToDirectConnection</code> parameter.</li> <li>■ If the <code>watt.server.proxy.useNonDefaultProxies</code> parameter is set to false, Integration Server sends the request to the remote server using a direct connection.</li> </ul> <p>For more information about proxy server usage, refer to <i>webMethods Integration Server Administrator's Guide</i>.</p>
-------------------	--

## Output Parameters

---

<i>command</i>	<b>String</b> FTP command that was executed ( <code>ls</code> , <code>get</code> , or <code>put</code> ).
<i>dirlist</i>	<b>String List</b> File names returned by the <code>ls</code> command.
<i>localfile</i>	<b>String</b> Name of the local file used for a get or put operation.
<i>remotefile</i>	<b>String</b> Name of the remote file used for a get or put operation.
<i>content</i>	<b>byte[ ]</b> If <i>localfile</i> was not specified, this parameter contains the Content object sent to the remote server (if a put command was executed) or received from the remote server (if a get command was executed).
<i>returncode</i>	<b>String</b> Standard FTP protocol return code.
<i>returnmsg</i>	<b>String</b> Standard FTP protocol return message.
<i>logmsg</i>	<b>String</b> FTP log message.

## Usage Notes

If you set the *auth* variable in the *secure* parameter to SSL, TLS, or TLS-P, `pub.client.ftp` automatically sends the following sequence of FTP commands prior to sending the USER command:

```
AUTH <SSL | TLS | TLS-P> PBSZ 0 PROT <P | C>
```

The client FTP services will not negotiate for less security than you have specified with the *auth* parameter. However, if you set the *auth* variable to `None`, the client FTP services can operate (in a non-secure mode) with any FTP server.

The FTP services will always connect to a secure FTP server using a non-secure (SSL) socket. After getting a valid reply from the AUTH command, the FTP services will convert the connected socket to an SSL socket and initiate SSL handshaking.

---

## pub.client.ftp:append

WmPublic. Appends data to a remote file.

If the remote file does not exist, the service creates the file.

### Input Parameters

---

<i>sessionkey</i>	<b>String</b> Unique key for the current FTP session. The <i>sessionkey</i> is returned by the <code>pub.client.ftp:login</code> service.
<i>transfermode</i>	<b>String</b> FTP file transfer mode ( <code>ascii</code> or <code>binary</code> ). The default is <code>ascii</code> .
<i>content</i>	<b>java.io.InputStream, byte[ ], or String</b> Data to be transferred to the remote file.
<i>localfile</i>	<b>String</b> Optional. Name of the local file to append to the remote file. Used only when <i>content</i> is not specified.
<i>remotefile</i>	<b>String</b> Name of the remote file to which to append the data specified in <i>content</i> or <i>localfile</i> .

### Output Parameters

---

<i>returncode</i>	<b>String</b> Standard FTP protocol return code.
<i>returnmsg</i>	<b>String</b> Standard FTP protocol return message.

---

*logmsg*      **String** FTP log messages for the entire user session.

---

## pub.client.ftp:cd

WmPublic. Changes the working directory on the FTP server. (This service corresponds to the standard FTP command `cd dirpath`.)

### Input Parameters

---

*sessionkey*      **String** Unique key for the current FTP session. The *sessionkey* is returned by the [pub.client.ftp:login](#) service.

*dirpath*      **String** Directory to which you want to switch on the FTP server. For example: `pub`

### Output Parameters

---

*returncode*      **String** Standard FTP protocol return code.

*returnmsg*      **String** Standard FTP protocol return message.

*logmsg*      **String** FTP log messages for the entire user session.

---

## pub.client.ftp:cdls

WmPublic. Changes the working directory on the FTP server and retrieves a list of file names. (This service corresponds to the standard FTP commands `cd dirpath` and `ls namePattern`.)

### Input Parameters

---

*sessionkey*      **String** Unique key for the current FTP session. The *sessionkey* is returned by the [pub.client.ftp:login](#) service.

*dirpath*      **String** Directory to which you want to switch on the FTP server (for example, `pub`).

*filenamepattern*      **String** Optional. Pattern that specifies the file names to list (for example, `*.txt`).

*orderby*      **String** Optional. The order of the returned file list. Set to:

- `none` to send an NLST command to the remote FTP server. This is the default.
- `timestamp` to return the list in order of the timestamp. Sends an NLST `-t` command to the remote FTP server.

**Note:** The `-t` command is not part of the RFC959 standard. Some FTP servers may not support this command. Servers that support this command may return the results in either ascending or descending order of creation time.

### Output Parameters

---

*dirlist*      **String List** List of file names matching *filenamepattern*.

*returncode*      **String** Standard FTP protocol return code.

*returnmsg*      **String** Standard FTP protocol return message.

*logmsg*      **String** FTP log messages for the entire user session.

## pub.client.ftp:delete

WmPublic. Deletes a file in the current working directory on an FTP server. (This service corresponds to the standard FTP command `delete somefile`.)

### Input Parameters

---

*sessionkey*      **String** Unique key for the current FTP session. The *sessionkey* is returned by the [pub.client.ftp:login](#) service.

*remotefile*      **String** Name of the file to be deleted from the current working directory. For example: `text.txt`

If you specify pattern-matching characters in *remotefile*, all files matching the pattern will be deleted.

### Output Parameters

---

*returncode*      **String** Standard FTP protocol return code.

*returnmsg*      **String** Standard FTP protocol return message.

*logmsg*          **String** FTP log messages for the entire user session.

---

## pub.client.ftp:dir

WmPublic. Retrieves the file list during an FTP session. (This service corresponds to the standard FTP command `dir namepattern`.)

### Input Parameters

---

*sessionkey*      **String** Unique key for the current FTP session. The *sessionkey* is returned by the [pub.client.ftp:login](#) service.

*filenamepattern*      **String** Optional. Pattern that specifies the names of the files to include in the list (for example, `*.txt`).

### Output Parameters

---

*dirlist*          **String List** Directory listing of the files matching *filenamepattern* including the file names and timestamps.

*returncode*      **String** Standard FTP protocol return code.

*returnmsg*      **String** Standard FTP protocol return message.

*logmsg*          **String** FTP log messages for the entire user session.

---

## pub.client.ftp:get

WmPublic. Retrieves a file from a remote FTP server. (This service corresponds to the standard FTP command `get`.)

### Input Parameters

---

*sessionkey*      **String** Unique key for the current FTP session. The *sessionkey* is returned by the [pub.client.ftp:login](#) service.

<i>transfermode</i>	<b>String</b> FTP file transfer mode ( <i>ascii</i> or <i>binary</i> ). The default is <i>ascii</i> .
<i>localfile</i>	<b>String</b> Optional. Name of a local file where the retrieved file is to be saved.
<i>remotefile</i>	<b>String</b> Name of the remote file.
<i>encoding</i>	<p><b>String</b> Optional. Character set in which the file is encoded. This variable is required to convert the file to bytes correctly. Specify an IANA-registered character set (for example: <i>ISO-8859-1</i>).</p> <p>If this variable is null, the encoding currently set for the FTP session is used. If encoding was never set for this FTP session, the default JVM encoding is used.</p>
<i>largefilethreshold</i>	<b>String</b> Optional. Defines the size (in bytes) of a "large" file. For more information, see the Usage Notes.

<u>If you...</u>	<u>Then...</u>
Set to 0	<p>All files will be considered large files. This means:</p> <ul style="list-style-type: none"> <li>■ The output parameter <i>islargefile</i> will always be <i>true</i>.</li> <li>■ The file content will be returned in the output parameter <i>contentstream</i> (as a <i>java.io.InputStream</i> object).</li> <li>■ The output parameter <i>content</i> will be null.</li> </ul>
Set to any value greater than 0	<p>Any file larger than the value you specify will be considered large. This means:</p> <ul style="list-style-type: none"> <li>■ The output parameter <i>islargefile</i> will be <i>true</i>.</li> <li>■ The file content will be returned in the output parameter <i>contentstream</i> (as a <i>java.io.InputStream</i> object).</li> <li>■ The output parameter <i>content</i> will be null.</li> </ul>
Leave blank	<p>No file is considered large. This means:</p> <ul style="list-style-type: none"> <li>■ The output parameter <i>islargefile</i> will always be <i>false</i>.</li> </ul>

- The file content will be returned in the output parameter *content* .
- The output parameter *contentstream* will be null.

*cleanlinefeeds*      **String** Optional. Indicates whether the service should retain or remove carriage return characters at the end of each line of text. Set to:

- `true` to remove carriage returns.
- `false` to retain carriage returns. This is the default.

## Output Parameters

---

*content*              **byte[ ]** Data retrieved from the remote file.

*returncode*          **String** Standard FTP protocol return code.

*returnmsg*          **String** Standard FTP protocol return message.

*logmsg*              **String** FTP log messages for the entire user session.

*islargefile*         **String** Indicates whether the file is considered to be large (as specified by the input parameter *largefilethreshold* ). A value of:

- `true` indicates that the file is larger than the value of *largethreshold* .
- `false` indicates that the file is not larger than the value of *largethreshold* (or *largethreshold* is blank).

*contentstream*      **Object** An `java.io.InputStream` object.

## Usage Notes

The *largefilethreshold* parameter improves the ability of `pub.client.ftp:get` to retrieve larger files. If a retrieved file is larger than the size specified in the *largefilethreshold* parameter, and the *localfile* parameter is empty (which means the retrieved file is retrieved to memory, not to a file on disk), the Integration Server streams the large file to a temporary file. While this will improve the scalability of `pub.client.ftp:get`, it will also reduce the throughput of the operation because the retrieved file will be written to a temporary file.

**Tip:** Due to the impact to the throughput of `pub.client.ftp:get` when streaming is enabled, you should set the value for *largefilethreshold* to a sufficiently large value so that it causes only minimal degradation to throughput and yet allows

the service to retrieve large files without encountering an OutOfMemory exception.

### See Also

[pub.io:close](#)

---

## pub.client.ftp:getCompletedNotification

WmPublic. A publishable document type that represents the document published to notify parties that an FTP get command has completed on Integration Server which is acting as the FTP server.

When a user completes an FTP get command in his or her own user directory (that is, when the RETR command is completed on the server side but the server has not yet acknowledged the client with return code 226), an event is fired to notify interested parties by publishing a document. EDI packages that subscribe to this document will remove the file from the server.

### Parameters

---

*username*     **String** The login user name through the FTP Listener.

*filename*     **String** The absolute path name of the file.

*\_env*            **Document** Optional. A document reference to [pub.publish:envelope](#) which defines the structure and contents of the envelope of a published document.

### Usage Notes

Integration Server publishes an instance document of `pub.client.ftp:getCompletedNotification` when an FTP client has completed an FTP get command on Integration Server through an FTP/S port. Because the retrieved file is located on the Integration Server file system, only subscribers located on the same Integration Server would have access to the file. Consequently the `pub.client.ftp:getCompletedNotification` publishable document type is set to publish locally only and uses the `IS_LOCAL_CONNECTION` messaging connection alias. When Integration Server publishes an instance document for `pub.client.ftp:getCompletedNotification`, only subscribers located on the same Integration Server receive the document. Software AG does not recommend changing the assigned messaging connection alias.

---

## pub.client.ftp:login

WmPublic. Connects to a remote FTP server and logs in with a specified user name and password.

### Input Parameters

---

<i>serverhost</i>	<b>String</b> Name or IP address of the FTP server (for example, <code>ftp.netscape.com</code> ).
<i>serverport</i>	<b>String</b> Port number on which the FTP server listens for requests (for example, <code>4566</code> ). The default is <code>21</code> .
<i>dataport</i>	<b>String</b> Optional. Listener port number of the data transfer channel (for example, <code>3345</code> ).  If you do not specify <i>dataport</i> , the Integration Server will choose the listener port number. This value is used only when the <i>transfertype</i> value is <code>active</code> .
<i>username</i>	<b>String</b> Valid FTP user on the remote FTP server (for example, <code>anonymous</code> ).
<i>password</i>	<b>String</b> Optional. Valid password for the FTP user specified in <i>username</i> (for example, <code>someone@somewhere</code> ).
<i>account</i>	<b>String</b> Optional. The user name for an account on the FTP server. Specify <i>account</i> if your FTP host requires account information. The account is defined in the FTP protocol to further identify the user that is identified by the <i>username</i> and <i>password</i> input variables.
<i>transfertype</i>	<b>String</b> Type of the FTP data transfer mode ( <code>passive</code> or <code>active</code> ). The default is <code>active</code> .
<i>encoding</i>	<b>String</b> Optional. Default character set for encoding data transferred during this session. Specify an IANA-registered character set (for example, <code>ISO-8859-1</code> ).  If you do not set <i>encoding</i> , the default JVM encoding is used.
<i>serverencoding</i>	<b>String</b> Optional. Specifies the encoding this service uses to convert the incoming FTP command string to encoded bytes that are

supported by IANA and the FTP server. If the parameter is null, the service uses the 'UTF-8' character set to encode the FTP command String to bytes.

*timeout* **String** Optional. Time (measured in seconds) to wait for a response from the FTP server before timing out and terminating the request. The default is to wait forever.

*secure* **Document** Indicates whether the FTP session is with a secure FTP server.

**Note:** Integration Server does not support FTPS (FTP over SSL) requests through FTP proxy.

<u>Key</u>	<u>Description</u>
<i>auth</i>	<p><b>String</b> The kind of authentication mechanism to use: <i>None</i>, <i>SSL</i>, <i>TLS</i>, or <i>TLS-P</i>.</p> <p><i>None</i> specifies that the FTP session is with a non-secure FTP server. This is the default. If the value of <i>auth</i> is <i>None</i>, the <i>securedata</i> variable is ignored.</p> <p><i>TLS-P</i> is a shortcut that is equivalent to the sequence <i>AUTH TLS, PBSZ 0, and PROT P</i>. If the value of <i>auth</i> is <i>TLS-P</i>, the <i>securedata</i> variable is ignored.</p>
<i>securedata</i>	<p><b>String</b> Use the value <i>false</i> for a client sending <i>PROT C</i> (Data Channel Protection Level Clear).</p> <p>Use the value <i>true</i> for a client sending <i>PROT P</i> (Data Channel Protection Level Private).</p> <p><b>Note</b> If you do not set a value, the default is <i>false</i>.</p>

*newSession* **String** Optional. Flag indicating whether a new FTP session will be created for this FTP operation. Set to:

- *yes* to create a new session for this FTP operation.
- *no* to use the current session, if one is available, for this FTP operation. This is the default.

<i>clientTimeout</i>	<b>String</b> Optional. Specifies the idle time-out, measured in seconds, for this FTP session. If <i>clientTimeout</i> is set to 0 (zero), the session will never time out. The default is 600 seconds (10 minutes).
<i>proxyAlias</i>	<p><b>String</b> Optional. Name of the proxy alias for the proxy server through which Integration Server routes the FTP request.</p> <p>If you do not specify a <i>proxyAlias</i>, Integration Server routes the FTP request through the proxy server specified in the default FTP proxy alias. If there is no default FTP proxy alias, the action taken by Integration Server depends on the value specified for the <code>watt.net.proxy.useNonDefaultProxies</code> parameter.</p> <ul style="list-style-type: none"> <li>■ If the <code>watt.net.proxy.useNonDefaultProxies</code> parameter is set to true, Integration Server routes the FTP request through the proxy server in any configured FTP proxy alias. If the Integration Server does not have any defined FTP proxy aliases, Integration Server sends the FTP request directly to the FTP server or throws an exception depending on the settings specified for the <code>watt.net.proxy.fallbackToDirectConnection</code> parameter.</li> <li>■ If the <code>watt.server.proxy.useNonDefaultProxies</code> parameter is set to false, Integration Server sends the request to the remote server using a direct connection.</li> </ul> <p>For more information about proxy server usage, refer to <i>webMethods Integration Server Administrator's Guide</i>.</p>

## Output Parameters

<i>sessionkey</i>	<b>String</b> Unique key for the current FTP session. This session key must be provided to execute most other services in <code>pub.client.ftp</code> .
<i>returncode</i>	<b>String</b> Standard FTP protocol return code.
<i>returnmsg</i>	<b>String</b> Standard FTP protocol return message.
<i>logmsg</i>	<b>String</b> FTP log messages for the entire user session.

## Usage Notes

If you set the *auth* variable in the *secure* parameter to `SSL`, `TLS`, or `TLS-P`, `pub.client.ftp:login` automatically sends the following sequence of FTP commands prior to sending the `USER` command:

```
AUTH <SSL | TLS | TLS-P> PBSZ 0 PROT <P | C>
```

The client FTP services will not negotiate for less security than you have specified with the *auth* parameter. However, if you set the *auth* variable to `None`, the client FTP services can operate (in a non-secure mode) with any FTP server.

The FTP services will always connect to a secure FTP server using a non-secure (SSL) socket. After getting a valid reply from the AUTH command, the FTP services will convert the connected socket to an SSL socket and initiate SSL handshaking.

---

## pub.client.ftp:logout

WmPublic. Logs off of the FTP server and ends the current FTP session.

### Input Parameters

---

*sessionkey*      **String** Unique key for the current FTP session. The *sessionkey* is returned by the [pub.client.ftp:login](#) service.

### Output Parameters

---

*returncode*      **String** Standard FTP protocol return code.

*returnmsg*      **String** Standard FTP protocol return message.

*logmsg*          **String** FTP log messages for the entire user session.

---

## pub.client.ftp:ls

WmPublic. Retrieves the file list during an FTP session. (This service corresponds to the standard FTP command `ls namepattern`.)

### Input Parameters

---

*sessionkey*      **String** Unique key for the current FTP session. The *sessionkey* is returned by the [pub.client.ftp:login](#) service.

*filenamepattern*      **String** Optional. Pattern that specifies the names of the files to include in the list (for example, `*.txt`).

*orderby*          **String** Optional. The order of the returned file list.

<u>Value of <i>orderby</i></u>	<u>Description</u>
none	Default. Sends an NLST command to the remote FTP server.
timestamp	Returns the list in order of the timestamp. Sends an NLST -t command to the remote FTP server.

**Note:** The -t command is not part of the RFC959 standard. Some FTP servers may not support this command. Servers that support this command may return the results in either ascending or descending order of creation time.

### Output Parameters

<i>dirlist</i>	<b>String List</b> List of file names matching <i>filenamepattern</i> .
<i>returncode</i>	<b>String</b> Standard FTP protocol return code.
<i>returnmsg</i>	<b>String</b> Standard FTP protocol return message.
<i>logmsg</i>	<b>String</b> FTP log messages for the entire user session.

### Usage Note

During an FTP session, this service uses the character set specified in the *encoding* parameter of the [pub.client.ftp:login](#) service. If the file list this service retrieves includes characters from other languages, set the *encoding* parameter appropriately. For example, set *encoding* to `SJIS` for file names containing Japanese characters. If you do not set *encoding* in [pub.client.ftp:login](#), the default JVM encoding is used.

## pub.client.ftp:mdelete

WmPublic. Deletes multiple files in the current working directory on an FTP server. (This service corresponds to the standard FTP command `mdelete pattern`.)

---

## Input Parameters

---

<i>sessionkey</i>	<b>String</b> Unique key for the current FTP session. The <i>sessionkey</i> is returned by the <a href="#">pub.client.ftp:login</a> service.
<i>filenamepattern</i>	<b>String</b> Pattern that specifies the names of the files to be deleted from the current working directory (for example, *.txt).

**Important** If you do not specify a value for *filenamepattern*, all files in the working directory are deleted.

## Output Parameters

---

<i>returncode</i>	<b>String</b> Standard FTP protocol return code.
<i>returnmsg</i>	<b>String</b> Standard FTP protocol return message.
<i>logmsg</i>	<b>String</b> FTP log messages for the entire user session.

---

## pub.client.ftp:mget

WmPublic. Transfers multiple files from the remote FTP server. (This service corresponds to the standard FTP command `mget`.)

### Input Parameters

---

<i>sessionkey</i>	<b>String</b> Unique key for the current FTP session. The <i>sessionkey</i> is returned by the <a href="#">pub.client.ftp:login</a> service.
<i>transfermode</i>	<b>String</b> FTP file transfer mode ( <code>ascii</code> or <code>binary</code> ). The default is <code>ascii</code> .
<i>localdir</i>	<b>String</b> Directory in the local file system where the retrieved files are to be saved (for example, <code>c:\temp\ftpfiles</code> ).
<i>filenamepattern</i>	<b>String</b> Pattern that specifies the names of the files to be retrieved (for example, *.txt).
<i>encoding</i>	<b>String</b> Optional. Character set in which the files are encoded. This variable is required to convert the files to bytes correctly. Specify an IANA-registered character set (for example, <code>ISO-8859-1</code> ).

---

If you do not specify *encoding*, the encoding assigned to the FTP session is used. If the encoding was not set for the FTP session, the default JVM encoding is used.

### Output Parameters

---

<i>filenames</i>	<b>String List</b> List of files retrieved from the remote FTP server.
<i>returncode</i>	<b>String</b> Standard FTP protocol return code.
<i>returnmsg</i>	<b>String</b> Standard FTP protocol return message.
<i>logmsg</i>	<b>String</b> FTP log messages for the entire user session.

---

## pub.client.ftp:mput

WmPublic. Transfers multiple files to a remote FTP server. (This service corresponds to the standard FTP command `input`.)

### Input Parameters

---

<i>sessionkey</i>	<b>String</b> Unique key for the current FTP session. The <i>sessionkey</i> is returned by the <a href="#">pub.client.ftp:login</a> service.
<i>transfermode</i>	<b>String</b> FTP file transfer mode ( <code>ascii</code> or <code>binary</code> ). The default is <code>ascii</code> .
<i>localdir</i>	<b>String</b> Local directory containing the files you want to transfer to the remote FTP server (for example, <code>c:\temp\ftpfiles</code> ).
<i>filenamepattern</i>	<b>String</b> Pattern that specifies the names of the files to be transferred (for example, <code>*.txt</code> ).
<i>putunique</i>	<b>String</b> Optional. Indicates whether to send a <code>STOR</code> or a <code>STOU</code> (Store as Unique File) command to the remote FTP server. Set to: <ul style="list-style-type: none"><li>■ <code>true</code> to send a <code>STOU</code> (Store as Unique File) command.</li><li>■ <code>false</code> to send a <code>STOR</code> command. This is the default.</li></ul>

---

## Output Parameters

---

<i>filenames</i>	<b>String List</b> List of files transferred to the remote FTP server.
<i>returncode</i>	<b>String</b> Standard FTP protocol return code.
<i>returnmsg</i>	<b>String</b> Standard FTP protocol return message.
<i>logmsg</i>	<b>String</b> FTP log messages for the entire user session.

### Usage Note

Some FTP servers, such as the Integration Server FTP Listener, do not support "putting" a unique file. When using the `pub.client.ftp:put` or `pub.client.ftp:mput` service to put a unique file to an FTP server that does not support putting a unique file, you will encounter an error like this one:

```
com.wm.app.b2b.server.ServiceException: 500 'STOU': command not understood.
```

---

## pub.client.ftp:put

WmPublic. Transfers a file to a remote FTP server. (This service corresponds to the standard FTP command `put`.)

### Input Parameters

---

<i>sessionkey</i>	<b>String</b> Unique key for the current FTP session. The <i>sessionkey</i> is returned by the <a href="#">pub.client.ftp:login</a> service.
<i>transfermode</i>	<b>String</b> FTP file transfer mode ( <code>ascii</code> or <code>binary</code> ). The default is <code>ascii</code> .
<i>content</i>	<b>java.io.InputStream, byte[], or String</b> Data to be transferred to the remote file.
<i>localfile</i>	<b>String</b> Optional. Name of the local file to be appended to the remote file. Used only if <i>content</i> is not specified.
<i>remotefile</i>	<b>String</b> The name of the remote file.
<i>putunique</i>	<b>String</b> Optional. Indicates whether to send a <code>STOR</code> or a <code>STOU</code> (Store as Unique File) command to the remote FTP server. Set to: <ul style="list-style-type: none"> <li>■ <code>true</code> to send a <code>STOU</code> (Store as Unique File) command.</li> </ul>

- `false` to send a STOR command. This is the default.

## Output Parameters

---

<i>returncode</i>	<b>String</b> Standard FTP protocol return code.
<i>returnmsg</i>	<b>String</b> Standard FTP protocol return message.
<i>logmsg</i>	<b>String</b> FTP log messages for the entire user session.

## Usage Notes

- Some FTP servers, such as the Integration Server FTP Listener, do not support "putting" a unique file. When using the `pub.client.ftp.put` or `pub.client.ftp.mput` service to put a unique file to an FTP server that does not support putting a unique file, you will encounter an error like this one:

```
com.wm.app.b2b.server.ServiceException: 500 'STOU': command not understood.
```

- When a client invokes this service to transport a file, the FTP listener determines the content handler to use based on the file's extension. The content handler converts the file content to the input values for the service to invoke. The `Integration Server_directory\instances\instance_name\lib\mime.types` file contains the mappings of file extension to content type.

By default, if this service encounters a file that has no file extension, the default content handler is used. To override this, you can configure any content handler to handle files that have no file extension. To do this, add a line in the `Integration Server_directory\instances\instance_name\lib\mime.types` file that specifies the content type of the files with no extension, and the `ftp_no_extension` key. For example, to allow a content handler to accept text/xml files that have no extension, add this line to your mime.types file:

```
text/xml          ftp_no_extension
```

---

## pub.client.ftp:putCompletedNotification

`WmPublic`. A publishable document type that represents the document published to notify parties that an FTP put command has completed on Integration Server which is acting as the FTP server.

When a user completes an FTP put command in his or her own user directory (that is, when the STOR command is completed on the server side but the server has not yet acknowledged the client with return code 226), an event is fired to notify interested parties by publishing a document. EDI packages that subscribe to this document will retrieve the file from the server.

---

## Parameters

<i>username</i>	<b>String</b> The login user name through the FTP Listener.
<i>filename</i>	<b>String</b> The absolute path name of the file.
<i>_env</i>	<b>Document</b> Optional. A document reference to <a href="#">pub.publish:envelope</a> which defines the structure and contents of the envelope of a published document.

## Usage Notes

Integration Server publishes an instance document of `pub.client.ftp:putCompletedNotification` when an FTP client has completed an FTP put command on Integration Server through an FTP/S port. Because the put file is located on the Integration Server file system, only subscribers located on the same Integration Server would have access to the file. Consequently the `pub.client.ftp:putCompletedNotification` publishable document type is set to publish locally only and uses the `IS_LOCAL_CONNECTION` messaging connection alias. When Integration Server publishes an instance document for `pub.client.ftp:putCompletedNotification`, only subscribers located on the same Integration Server receive the document. Software AG does not recommend changing the assigned messaging connection alias.

---

## pub.client.ftp:quote

WmPublic. Executes a given FTP command.

You can use this service to execute non-standard FTP commands.

### Input Parameters

<i>sessionkey</i>	<b>String</b> Unique key for the current FTP session. The <i>sessionkey</i> is returned by the <a href="#">pub.client.ftp:login</a> service.
<i>string</i>	<b>String</b> The command to be executed on the FTP server. This service submits the command exactly as it is specified in <i>string</i> .

### Output Parameters

<i>returncode</i>	<b>String</b> Standard FTP protocol return code.
<i>returnmsg</i>	<b>String</b> Standard FTP protocol return message.

---

*logmsg*      **String** FTP log messages for the entire user session.

---

## pub.client.ftp:rename

WmPublic. Renames a file on a remote FTP server. (This service corresponds to the standard FTP command `rename`.)

### Input Parameters

---

*sessionkey*      **String** Unique key for the current FTP session. The *sessionkey* is returned by the [pub.client.ftp:login](#) service.

*oldname*      **String** Fully qualified name of the file you want to rename (for example, `temp/oldname.txt`).

*newname*      **String** New fully qualified name for the file (for example, `temp/newname.txt`).

### Output Parameters

---

*returncode*      **String** Standard FTP protocol return code.

*returnmsg*      **String** Standard FTP protocol return message.

*logmsg*      **String** FTP log messages for the entire user session.

---

## pub.client.ftp:sessioninfo

WmPublic. Returns session information for all of the FTP servers that users are currently logged into.

### Input Parameters

---

*name*      Not used. Reserved for future use.

---

## Output Parameters

---

*sessioninfo* **Document List** Information about the current FTP sessions. Each document in *sessioninfo* represents a single session and contains the following information:

<b>Key</b>	<b>Description</b>
<i>serverhost</i>	<b>String</b> Name or IP address of the FTP server.
<i>serverport</i>	<b>String</b> Port number on which the FTP server listens for requests.
<i>dataport</i>	<b>String</b> Listener port of the data transfer channel used by this session.
<i>username</i>	<b>String</b> User logged on to FTP server.
<i>password</i>	<b>String</b> Password for the FTP user specified in <i>username</i> .
<i>account</i>	<b>String</b> Conditional. The user name for an account on the FTP server. The account is defined in the FTP protocol to further identify the user that is identified by the <i>username</i> and <i>password</i> input variables.
<i>transfertype</i>	<b>String</b> Data transfer mode ( <i>passive</i> or <i>active</i> ) used by this session.
<i>encoding</i>	<b>String</b> Conditional. IANA character set used by this session. If <i>encoding</i> is not returned, the encoding was not explicitly set and the default JVM encoding is in effect.

## Usage Notes

When you start an FTP session with [pub.client.ftp:login](#), you can set the optional *dataport* parameter to specify the port number for data transfers. During the FTP session, `pub.client.ftp:sessionInfo` returns the *dataport* parameter with the port number used for data transfers.

If you do not set the *dataport* parameter in [pub.client.ftp:login](#), the server uses a random port number. During the FTP session, `pub.client.ftp:sessionInfo` will return a 0 for the *dataport* parameter to indicate that the port number used for data transfers is random.

---

## pub.client:http

WmPublic. Issues an HTTP request that you specify and returns the HTTP response.

### Input Parameters

---

<i>url</i>	<p><b>String</b> URL of the resource that you want to access. For example:</p> <pre>http://www.rubicon.com/orders/orders.html</pre> <p><b>Important</b> This string <i>must</i> begin with <code>http:</code> or <code>https:.</code></p>
<i>method</i>	<p><b>String</b> Specifies the HTTP method you want to use. Valid values are:</p> <pre>delete get head options patch post put trace</pre>
<i>loadAs</i>	<p><b>String</b> Optional. Form in which you want the http service to store the returned document. Set to:</p> <ul style="list-style-type: none"> <li>■ <code>bytes</code> to return the body of the response as a <code>byte[]</code>. Use this option if the body will be used as input to a service that operates on whole HTML or XML documents (for example, <a href="#">pub.xml:queryXMLNode</a>). This is the default.</li> <li>■ <code>stream</code> to return the body of the response as a <code>java.io.InputStream</code>. Use this option if the document will be used as input to a service that can process documents incrementally (for example, <a href="#">pub.xml:getXMLNodeIterator</a>).</li> </ul>
<i>data</i>	<p><b>Document</b> Data that you want the http service to submit with the HTTP request. Specify data using one or more of the following keys.</p> <p><b>Important</b> When you use more than one key, <i>args</i> is appended first, <i>table</i> is appended second, and <i>string</i> is appended last.</p>

Key	Description
-----	-------------

*args*

**Document** Optional. Name/value pairs that you want this service to submit to the resource in *url*. You can use *args* to submit data via the POST, PUT, GET, or HEAD method.

To specify data using *args*, create one String element for each name/value pair that you want to submit, where the element's name represents the name portion of the pair and the element's value represents the value portion of the pair.

When you use *args*, the http service will automatically:

- URL-encode name/value pair, so you do not need to URL-encode the values you specify in *args*.
- Insert the "&" character between pairs, so you do not need to include it in *args*.
- Prefix the entire query string with the "?" character if it submits the data in *args* via a GET or HEAD. You do not need to include this character in *args*.

When you submit data using *args*, Integration Server automatically sets the value of the Content-Type header to `application/x-www-form-urlencoded`.

If you want to explicitly specify a different Content-Type value, you must submit the value using the *string* or *bytes* variable.

*table*

**String Table** Optional. Data that the http service will use to construct a query string to submit to the resource specified in *url*.

*table* is similar to *args*, but it allows you to submit unnamed values in a query string, not just name/value pairs.

To specify data using *table*, create one row for each value that you want to submit, where the contents of column 0 of the String Table represents the name portion of the pair (leave this column null to submit an unnamed value) and the contents of column 1 represents the value portion of the pair.

When you use *table*, the http service will automatically:

- URL-encode name/value pair, so you do not need to URL-encode the values you specify in *table*.
- Insert the "&" character between the pairs (or unnamed values) that it constructs, so you do not need to include it in *table*.
- Prefix the entire query string with the "?" character if it submits the data in *table* via the GET method. You do not need to include this character in *table*.

When you submit data using *table*, Integration Server automatically sets the value of the Content-Type header to `application/x-www-form-urlencoded`. If you want to explicitly specify a different Content-Type, you must submit your data using the *string* or *bytes* variable.

*string*

**String** Optional. Text that you want the http service to submit to the resource in *url*. You can use *string* to submit data via the POST, PUT, GET, or HEAD method.

If you use *string* to submit data, make sure that you specify the string *exactly* as you want it presented in the HTTP request. (If you are using the GET or HEAD method, make sure you URL-encode the contents of *string*.)

**Note:** When you use *string*, the http service will automatically prefix the entire query string with "?" if it submits the data in *string* via a GET or HEAD. You do not need to include this character in *string*.

When performing a POST or PUT, *string* will be submitted to the resource defined by *url* as the body of the request message.

*bytes*

**byte[ ]** Optional. Data that you want this service to submit to the resource in *url*. You can use *bytes* to submit data via the POST or PUT methods only.

**Important:** When you use *bytes* and another element (*args*, *table*, or *string*) to specify data, the service appends the data from the *args*, *table*, or *string* element to *url*. The service appends *args* to *url* first, *table* second, and *string* last. The service encodes the data from the *bytes*

element in the body of the post. If the *stream* variable is not null, *bytes* is ignored.

*mimeStream* **Java.io.InputStream** Optional. MIME or SMIME message that you want this service to submit to the resource in *url*. A *mimeStream* is created by the [pub.mime:getEnvelopeStream](#), [pub.smime:createEncryptedData](#), [pub.smime:createSignedData](#), [pub.smime.keystore:createSignedData](#) or services. It contains both headers and content. The headers in the *mimeStream* are appended to the http headers.

You can use *mimeStream* to submit data via the POST or PUT methods only.

*stream* **java.io.InputStream** Optional. Data that you want the http service to submit to the resource in *url*. You can use *stream* to submit data via the POST or PUT methods only.

**Important!** When you use *stream* and another element (*args*, *table*, *string* or *bytes*) to specify data, the service appends the data from the *args*, *table*, or *string* element to *url*. The service appends *args* to *url* first, *table* second, and *string* last. The service encodes the data from the *stream* element in the body of the post. If the *stream* input is not null, the *bytes* input is ignored.

*encoding* **String** Optional. Character set in which the URL data parameters are encoded (*args* or *table* and/or *string*). Encoding is required to correctly convert the String object to bytes when generating the URL for a post. Specify an IANA-registered character set (for example, ISO-8859-1).

If this variable is null, the default JVM encoding is used. Because *string* is used in the body of the post and not used for building the URL, you do not need to specify encoding for the data parameter *string*.

*auth*

**Document** Optional. Authorization information that the http service will submit if the resource specified in *url* is protected.

<u>Key</u>	<u>Description</u>
------------	--------------------

<i>type</i>	<p><b>String</b> Type of authentication scheme that you want this service to use when it submits this request. Set to:</p> <ul style="list-style-type: none"> <li>■ <code>Basic</code> to submit a user name and password. This is the default.</li> <li>■ <code>Bearer</code> to submit authorization information for an OAuth resource server.</li> <li>■ <code>Digest</code> to submit a password digest for authentication.</li> <li>■ <code>NTLM</code> to use NTLM authentication.</li> </ul> <p>You do not need to specify a value for <i>type</i> if this request will specify values for <i>kerberos</i> to use Kerberos authentication. If you specify a value for <i>type</i> and provide <i>kerberos</i> values, Integration Server includes the authentication information related to the specified type and the Kerberos ticket in the outbound request.</p>
<i>user</i>	<p><b>String</b> User name that this service will submit when requesting a protected resource.</p> <div style="background-color: #f0f0f0; padding: 10px; margin: 10px 0;"> <p><b>Note:</b> If you have specified <code>NTLM</code> as <i>type</i>, you must specify <i>user</i> in the following format:</p> <p style="text-align: center;"><i>domain_name \user_name</i></p> </div>
<i>pass</i>	<p><b>String</b> Password associated with <i>user</i>.</p>
<i>token</i>	<p><b>String</b> The access token to submit to the OAuth resource server. Required only when <i>type</i> is set to <code>Bearer</code>.</p>
<i>kerberos</i>	<p><b>Document</b> Optional. Kerberos information that Integration Server uses to acquire a Kerberos ticket to include in the outbound client request. Integration Server includes the ticket in the HTTP Authorization header using the Negotiate authentication scheme. Specify <i>kerberos</i> details only if you need to use Kerberos authentication to access a secured website.</p>

KeyDescription

---

<i>jaasContext</i>	<p><b>String</b> Optional. The custom JAAS context used for Kerberos authentication. You must specify a JAAS context for the outbound request to use Kerberos authentication.</p> <p>Integration Server includes a JAAS context named <code>IS_KERBEROS_OUTBOUND</code> that can be used with outbound requests.</p> <p>If you specify a value for <i>jaasContext</i>, you must also specify a value for <i>servicePrincipal</i>.</p>
<i>clientPrincipal</i>	<p><b>String</b> Optional. The name of the client principal to use for Kerberos authentication.</p> <p>You must specify a <i>clientPrincipal</i> value if the specified <i>jaasContext</i> does not include the <code>principal</code> parameter. If you specify <i>clientPrincipal</i> and the specified <i>jaasContext</i> includes a <code>principal</code> parameter, the <code>principal</code> parameter in the <i>jaasContext</i> takes precedence.</p> <p>The <code>IS_KERBEROS_OUTBOUND</code> JAAS context does not include a <code>principal</code> parameter. You must specify <i>clientPrincipal</i> if you specified <code>IS_KERBEROS_OUTBOUND</code> as the <i>jaasContext</i> value.</p>
<i>clientPassword</i>	<p><b>String</b> Optional. The password for the specified client principal. You must specify <i>clientPassword</i> if the specified <i>jaasContext</i> does not specify a keytab file.</p>

---

<i>servicePrincipal</i>	<p><b>String</b> Optional. The name of the principal that is used with the service that the Kerberos client wants to access. The service provider provides the Service Principal Name.</p> <p>Specify the Service Principal Name in the following format:</p> <pre>principal-name.instance-name@realm-name</pre> <p>You must specify <i>servicePrincipal</i> if you supplied a <i>jaasContext</i>.</p>
<i>servicePrincipalForm</i>	<p><b>String</b> Optional. The format in which you want to specify the principal name of the service that is registered with the principal database. Set to:</p> <ul style="list-style-type: none"> <li>■ <code>host-based</code> to represent the principal name using the service name and the hostname, where hostname is the host computer. This is the default.</li> <li>■ <code>username</code> to represent the principal name as a named user in the LDAP or central user directory used for authentication to the key distribution center.</li> </ul>
<i>headers</i>	<p><b>Document</b> Optional. Fields that you want to explicitly override in the HTTP request header issued by the http service.</p> <p>Specify a key in <i>headers</i> for each header field that you want to set, where the key's name represents the name of the header field and the key's value represents the value of that header field.</p> <p>If you do not set <i>headers</i>, the http service uses its default header values.</p>
<i>timeout</i>	<p><b>String</b> Optional. Time (measured in milliseconds) to wait for a response from the remote server before timing out and</p>

terminating the request. The default value is defined by the `watt.net.timeout` server configuration parameter. The default value for the `watt.net.timeout` server configuration parameter is 300 seconds (300000 milliseconds).

For information about the `watt.net.timeout` server configuration parameter, see *webMethods Integration Server Administrator's Guide*.

#### *maxKeepAliveConnections*

**String** Optional. Number of client keep alive connections that you want Integration Server to retain in the client connection pool it uses for this HTTP connection.

Integration Server establishes a client connection pool for each protocol (i.e., HTTP or HTTPS), host, and port combination. How the service uses this input value differs based on whether a suitable client connection pool already exists for the HTTP connection.

- If a suitable pool already exists, Integration Server uses a connection from the pool for the HTTP request. If the specified *maxKeepAliveConnections* is different from what is currently being used for the pool, Integration Server updates the pool to use the value you specify.
- If a suitable pool does *not* exist, Integration Server establishes one, using the maximum connections defined by this parameter. If you do not specify a value, the service uses the default value defined by the `watt.net.maxClientKeepAliveConns` server configuration parameter. After establishing the client connection pool, the service uses a connection from the newly established connection pool to the HTTP request.

#### *keepAliveTimeout*

**String** Optional. Number of seconds that you want Integration Server to keep an idle connection in the client connection pool before closing it.

Integration Server establishes a client connection pool for each protocol (i.e., HTTP or HTTPS), host, and port combination. How the service uses this input value differs based on whether a suitable client connection pool already exists for the HTTP connection.

- If a suitable pool already exists, Integration Server uses a connection from the pool for the HTTP request. If the specified *keepAliveTimeout* is different from what is currently being used for the pool, Integration Server updates the pool to use the value you specify.

- If a suitable pool does *not* exist, Integration Server establishes one, using the timeout value defined by this parameter. If you do not specify a value, the service uses the default value defined by the `watt.net.clientKeepAliveTimeout` server configuration parameter. After establishing the client connection pool, the service uses a connection from the newly established connection pool to the HTTP request.

#### *newSession*

**String** Optional. Flag indicating whether a new session will be created for this HTTP request. This parameter overrides the `watt.server.new.http.session.context` server configuration parameter. For information about the `watt.server.new.http.session.context` server configuration parameter, see *webMethods Integration Server Administrator's Guide*.

Set to:

- `no` to use the current session, if one is available, for this HTTP request. This is the default.
- `yes` to create a new session for this HTTP request.

#### *proxyAlias*

**String** Optional. Name of the proxy alias for the proxy server to which Integration Server routes the HTTP request.

If you do not specify a *proxyAlias*, Integration Server routes the HTTP request through the proxy server specified in the default HTTP proxy alias. If there is no default HTTP proxy alias, the action taken by Integration Server depends on the value specified for the `watt.net.proxy.useNonDefaultProxies` parameter.

- If the `watt.net.proxy.useNonDefaultProxies` parameter is set to `true`, Integration Server routes the HTTP request through the proxy server in any configured HTTP proxy alias. If the Integration Server does not have any defined HTTP proxy aliases, Integration Server sends the HTTP request directly to the HTTP server or throws an exception depending on the settings specified for the `watt.net.proxy.fallbackToDirectConnection` parameter.
- If the `watt.server.proxy.useNonDefaultProxies` parameter is set to `false`, Integration Server sends the request to the remote server using a direct connection.

For more information about proxy server usage, refer to *webMethods Integration Server Administrator's Guide*.

<i>connectTimeout</i>	<p><b>String</b> Optional. Time (measured in milliseconds) the server will wait to connect to the remote server before timing out and terminating the request.</p> <p>If a value for <i>connectTimeout</i> is not specified, Integration Server uses the value specified for the <code>watt.net.timeout</code> server configuration parameter. If no value is specified for the <code>watt.net.timeout</code> server configuration parameter, the server will wait for the timeout value defined by the operating system before terminating the connection request. For more information about the <code>watt.net.timeout</code> server configuration parameter, see <i>webMethods Integration Server Administrator's Guide</i>.</p>
<i>useJSSE</i>	<p><b>String</b> Optional. Whether to enable the use of the Java Secure Socket Extension (JSSE) socket factory for creating outbound HTTPS connections. Set to:</p> <ul style="list-style-type: none"> <li>■ <code>yes</code> if the connection requires the use of TLS 1.1 or TLS 1.2. When set to <code>yes</code>, Integration Server creates the connection using the Java Secure Socket Extension (JSSE) socket factory.</li> <li>■ <code>no</code> if the connection does not require support for TLS 1.1 or TLS 1.2. When set to <code>no</code>, the connection supports only SSL 3.0 and TLS 1.0 and Entrust IAIK library is used to create the outbound HTTPS connection. This is the default.</li> </ul> <div style="background-color: #f0f0f0; padding: 10px; margin-top: 10px;"> <p><b>Note:</b> The value of the <i>useJSSE</i> input parameter overrides the value of the <code>watt.net.ssl.client.useJSSE</code> server configuration property.</p> <p><b>Note:</b> To control the cipher suites used on JSSE sockets that are used while making outbound HTTPS requests, set the server configuration property <code>watt.net.jsse.client.enabledCipherSuiteList</code>. For more information, see <i>webMethods Integration Server Administrator's Guide</i>.</p> </div>

## Output Parameters

---

<i>encodedURL</i>	<p><b>String</b> The URL that was submitted by <code>pub.client:http</code>. This will contain any argument set in <i>args</i>, <i>table</i>, or <i>string</i>.</p> <p>If the remote server redirected <code>pub.client:http</code> to a different location, <i>encodedURL</i> will contain the URL that <code>pub.client:http</code> submitted to the server to which it was redirected.</p>
<i>header</i>	<p><b>Document</b> Conditional. HTTP response headers.</p>

<u>Key</u>	<u>Description</u>
<i>lines</i>	<b>Document</b> Fields in the response header, where key names represent field names and values represent field values.
<i>status</i>	<b>String</b> HTTP status code of the response.
<i>statusMessage</i>	<b>String</b> HTTP status message of the response.
<i>body</i>	<b>Document</b> Body of the HTTP response.

<u>Key</u>	<u>Description</u>
<i>bytes</i>	<b>byte[ ]</b> Conditional. Body of the HTTP response represented as a <code>byte[ ]</code> . <i>bytes</i> is returned only when the <i>loadAs</i> input parameter is set to <code>bytes</code> .
<i>stream</i>	<b>java.io.InputStream</b> Conditional. The body of the HTTP response represented as an <code>InputStream</code> . <i>stream</i> is returned only when the <i>loadAs</i> input parameter is set to <code>stream</code> .

## Usage Notes

If *url* begins with `https:`, you can use [pub.security:setKeyAndChain](#) to specify the certificate chain. If you do not specify a certificate chain, `pub.client:http` uses the default outbound SSL certificate settings to authenticate the resources.

If `pub.client:http` does not receive a response within the time-out period specified in the server's `watt.net.timeout` server configuration parameter, it will throw an exception. For information about the `watt.net.timeout` server configuration parameter, see *webMethods Integration Server Administrator's Guide*.

For the HTTP request, the `pub.client:http` service uses a client connection from a client connection pool. When you set the *loadAs* input parameter to `stream` so that the service returns the response body as a stream, the connection remains in use and is not returned to the connection pool until you close the stream. To close the stream and return the connection to the pool, you can use the [pub.io:close](#) service or the `close()` method on the returned stream object.

When the HTTP response contains the HTTP 302 redirection status code, the `pub.client:http` service automatically redirects the request to the URL provided by the remote server. If the remote server redirected `pub.client:http` to a different location, the output parameter

*encodedURL* will contain the URL that `pub.client:http` submitted to the server to which the service as redirected.

If the server to which `pub.client:http` is redirected returns an HTTP response with a 401 status code and `watt.net.http401.throwException` is set to `false`, the `pub.client:http` service places the header and body of the response in the output pipeline. The service that invokes `pub.client:http` can examine the *header/status* field to determine if the server to which `pub.client:http` was redirected returned a 401. If it did, the service can re-issue the outbound request directly to the URL in *encodedURL* field, making sure to supply credentials with the request.

When the HTTP response contains the 401 status code, the `pub.client:http` service does one of the following depending on the value of the `watt.net.http401.throwException` server configuration parameter.

- When `watt.net.http401.throwException` is set to `true`, if the `pub.client:http` service receives a 401 error, the service throws a `NetException`.
- When `watt.net.http401.throwException` is set to `false`, if the `pub.client:http` service receives a 401 error, the service suppresses the `NetException` and places the HTTP response header and body, if one exists, into the *header* and *body* fields in the service output.

When the HTTP response contains the status code in the 501 to 599 range, the `pub.client:http` service does one of the following depending on the value of the `watt.net.http501-599.throwException` server configuration parameter:

- When set to `true`, the `pub.client:http` service throws a `ServiceException` when it receives a 501 to 599 level response from a remote HTTP server.
- When set to `false`, the `pub.client:http` service returns the status code, response headers, and response body in the service output when it receives a 501 to 599 level response from a remote HTTP server.

When using Kerberos authentication, principal name and principal password can be specified in the JAAS context file and in the `pub.client:http` service in the *clientPrincipal* and *clientPassword* fields in the *auth\kerberos* document. If the principal name and password are specified in the JAAS context file supplied to *jaasContext* and in the `pub.client:http` service, the values in the JAAS context file take precedence.

---

## pub.client:ldap:add

WmPublic. Inserts a new entry into the directory.

### Input Parameters

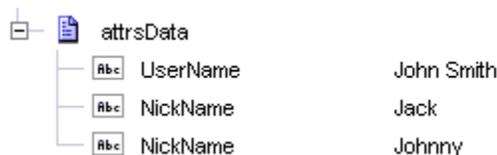
---

*url* **String** Optional. URL of the directory server to connect to. For example `ldap://servername:389`.

<i>principal</i>	<b>String</b> Optional. The principal for the directory server.
<i>credentials</i>	<b>String</b> Optional. Credentials for the directory server.
<i>timeout</i>	<b>String</b> Optional. Connection timeout in milliseconds.
<i>ldapEnv</i>	<b>Record</b> Optional. Key/value parameters to be passed to JNDI to further define the connection environment. See your JNDI provider documentation or the Oracle JNDI documentation for more information about parameters you can pass to JNDI.
<i>close</i>	<p><b>String</b> Flag that specifies whether to close the connection after the service finishes. Set to:</p> <ul style="list-style-type: none"> <li>■ <code>yes</code> to close the connection. This is the default.</li> <li>■ <code>no</code> to leave the connection open and available.</li> </ul>
<i>dn</i>	<b>String</b> The distinguished name of the new entry to add to the directory.
<i>attrs</i>	<p><b>Document List</b> Optional. LDAP attributes and their corresponding values. If an attribute is specified more than once, it will be assigned multiple values. The following example shows how to specify a user name of John Smith and one nickname.</p>



<i>attrsData</i>	<p><b>Document</b> Optional. LDAP attributes and their corresponding values. If an attribute is specified more than once, it will be assigned multiple values. The following example shows how to assign a user name of John Smith with two nicknames.</p>
------------------	--



---

## Output Parameters

---

*connectionHandle*     **Object** Optional. The returned connection object. Returned only if the *close* parameter is set to "no".

### Usage Notes

Specify only one of *attrs* or *attrsData*. If you specify both, the service uses *attrs* and ignores *attrsData*.

When *close* is set to yes, Integration Server removes the *connectionHandle* from the pipeline, but does not close the LDAP connection. To close the LDAP connection, set the `watt.server.ldap.cleanContext` server configuration parameter to `true`. For information about the `watt.server.ldap.cleanContext` server configuration parameter, see *webMethods Integration Server Administrator's Guide*.

---

## pub.client.ldap:bind

WmPublic. Performs an LDAP bind operation that associates the connection with the specified principal.

### Input Parameters

---

*url*     **String** URL of the LDAP server to connect to.

*principal*     **String** Optional. The principal for the LDAP server.

*credentials*     **String** Optional. Credentials for the LDAP server.

*timeout*     **String** Optional. Connection timeout in milliseconds.

*ldapErrv*     **Record** Optional. Key/value parameters to be passed to JNDI to further define the connection environment. See your JNDI provider documentation or the Oracle JNDI documentation for more information about parameters you can pass to JNDI.

*close*     **String** Flag that specifies whether to close the connection after the service finishes. Set to:

- `yes` to close the connection. This is the default.
- `no` to leave the connection open and available.

---

## Output Parameters

*connectionHandle*     **Object** Optional. The returned connection object. Returned only if the *close* parameter is set to "no".

## Usage Notes

When *close* is set to *yes*, Integration Server removes the *connectionHandle* from the pipeline, but does not close the LDAP connection. To close the LDAP connection, set the `watt.server.ldap.cleanContext` server configuration parameter to `true`. For information about the `watt.server.ldap.cleanContext` server configuration parameter, see *webMethods Integration Server Administrator's Guide*.

---

## pub.client.ldap:cancelNotification

WmPublic. Cancels a previously created notification request.

## Input Parameters

<i>url</i>	<b>String</b> Optional. URL of the LDAP server to connect to.
<i>principal</i>	<b>String</b> Optional. The principal for the LDAP server.
<i>credentials</i>	<b>String</b> Optional. Credentials for the LDAP server.
<i>timeout</i>	<b>String</b> Optional. Connection timeout in milliseconds.
<i>ldapEnv</i>	<b>Record</b> Optional. Key/value parameters to be passed to JNDI to further define the connection environment. See your JNDI provider documentation or the Oracle JNDI documentation for more information about parameters you can pass to JNDI.
<i>close</i>	<p><b>String</b> Flag that specifies whether to close the connection after the service finishes. Set to:</p> <ul style="list-style-type: none"> <li>■ <code>yes</code> to close the connection. This is the default.</li> <li>■ <code>no</code> to leave the connection open and available.</li> </ul>
<i>dn</i>	<b>String</b> The distinguished name of the entry.
<i>connectionHandle</i>	<b>Object</b> Optional. Connection object returned by a previously invoked LDAP service.

*scope*                    **String** The scope of the search. Must be "object" (only search the specified directory entry), "onelevel" (only search the immediate children of the specified directory entry), or "subtree" (search the directory, its children, and all of their children).

---

### Output Parameters

*connectionHandle*        **Object** Optional. The returned connection object. Returned only if the *close* parameter is set to "no".

### Usage Notes

When *close* is set to yes, Integration Server removes the *connectionHandle* from the pipeline, but does not close the LDAP connection. To close the LDAP connection, set the `watt.server.ldap.cleanContext` server configuration parameter to `true`. For information about the `watt.server.ldap.cleanContext` server configuration parameter, see *webMethods Integration Server Administrator's Guide*.

---

## pub.client.ldap:compare

WmPublic. Compares the value of an attribute in the LDAP directory with a value specified by the service.

---

### Input Parameters

*url*                        **String** Optional. URL of the LDAP server to connect to.

*principal*                **String** Optional. The principal for the LDAP server.

*credentials*              **String** Optional. Credentials for the LDAP server.

*timeout*                  **String** Optional. Connection timeout in milliseconds.

*ldapErr*                  **Record** Optional. Key/value parameters to be passed to JNDI to further define the connection environment. See your JNDI provider documentation or the Oracle JNDI documentation for more information about parameters you can pass to JNDI.

*close*                      **String** Flag that specifies whether to close the connection after the service finishes. Set to:

- `yes` to close the connection. This is the default.

- `no` to leave the connection open and available.

<i>dn</i>	<b>String</b> The distinguished name of the entry whose attribute value you want to compare to <i>attrValue</i> .
<i>connectionHandle</i>	<b>Object</b> Optional. Connection object returned by a previously invoked LDAP service.
<i>attrName</i>	<b>String</b> Name of the attribute whose value you want to compare to <i>attrValue</i> .
<i>attrValue</i>	<b>String</b> The string to compare against the value of the attribute identified by <i>attrName</i> .

### Output Parameters

---

<i>connectionHandle</i>	<b>Object</b> Optional. The returned connection object. Returned only if the <i>close</i> parameter is set to "no".
<i>result</i>	<b>String</b> The result of the compare operation. Can be "true" or "false".

### Usage Notes

When *close* is set to `yes`, Integration Server removes the *connectionHandle* from the pipeline, but does not close the LDAP connection. To close the LDAP connection, set the `watt.server.ldap.cleanContext` server configuration parameter to `true`. For information about the `watt.server.ldap.cleanContext` server configuration parameter, see *webMethods Integration Server Administrator's Guide*.

## pub.client.ldap:delete

---

WmPublic. Removes an entry from the directory.

### Input Parameters

---

<i>url</i>	<b>String</b> Optional. URL of the LDAP server to connect to.
<i>principal</i>	<b>String</b> Optional. The principal for the LDAP server.
<i>credentials</i>	<b>String</b> Optional. Credentials for the LDAP server.

---

<i>timeout</i>	<b>String</b> Optional. Connection timeout in milliseconds.
<i>ldapEnv</i>	<b>Record</b> Optional. Key/value parameters to be passed to JNDI to further define the connection environment. See your JNDI provider documentation or the Oracle JNDI documentation for more information about parameters you can pass to JNDI.
<i>close</i>	<b>String</b> Flag that specifies whether to close the connection after the service finishes. Set to: <ul style="list-style-type: none"><li>■ <code>yes</code> to close the connection. This is the default.</li><li>■ <code>no</code> to leave the connection open and available.</li></ul>
<i>dn</i>	<b>String</b> The distinguished name of the entry to delete.
<i>connectionHandle</i>	<b>Object</b> Optional. Connection object returned by a previously invoked LDAP service.

### Output Parameters

---

<i>connectionHandle</i>	<b>Object</b> Optional. The returned connection object. Returned only if the <i>close</i> parameter is set to "no".
-------------------------	---

### Usage Notes

This service does not flag an error if the entry is not deleted. One way to check is to use [pub.client.ldap:search](#) to search for the entry. If the entry is not found, you know it has been deleted.

When *close* is set to `yes`, Integration Server removes the *connectionHandle* from the pipeline, but does not close the LDAP connection. To close the LDAP connection, set the `watt.server.ldap.cleanContext` server configuration parameter to `true`. For information about the `watt.server.ldap.cleanContext` server configuration parameter, see *webMethods Integration Server Administrator's Guide*.

---

## pub.client.ldap:modify

WmPublic. Performs an LDAP modify operation that allows you to specify a list of attributes with corresponding lists of values to add to, replace, or remove from the directory entry.

## Input Parameters

---

<i>url</i>	<b>String</b> Optional. URL of the LDAP server to connect to.
<i>principal</i>	<b>String</b> Optional. The principal for the LDAP server.
<i>credentials</i>	<b>String</b> Optional. Credentials for the LDAP server.
<i>timeout</i>	<b>String</b> Optional. Connection timeout in milliseconds.
<i>ldapEnv</i>	<b>Record</b> Optional. Key/value parameters to be passed to JNDI to further define the connection environment. See your JNDI provider documentation or the Oracle JNDI documentation for more information about parameters you can pass to JNDI.
<i>close</i>	<p><b>String</b> Flag that specifies whether to close the connection after the service finishes. Set to:</p> <ul style="list-style-type: none"> <li>■ <code>yes</code> to close the connection. This is the default.</li> <li>■ <code>no</code> to leave the connection open and available.</li> </ul>
<i>dn</i>	<b>String</b> The distinguished name of the entry to modify.
<i>connectionHandle</i>	<b>Object</b> Optional. Connection object returned by a previously invoked LDAP service.
<i>attrs</i>	<p><b>Document List</b> Optional. For each LDAP attribute to change, specifies the attribute name, the values affected, and the action to perform on those values. The following example shows how to specify the removal of John Smith's nickname Johnny.</p>



## Output Parameters

---

<i>connectionHandle</i>	<b>Object</b> Optional. The returned connection object. Returned only if the <i>close</i> parameter is set to "no".
-------------------------	---

## Usage Notes

When *close* is set to yes, Integration Server removes the *connectionHandle* from the pipeline, but does not close the LDAP connection. To close the LDAP connection, set the `watt.server.ldap.cleanContext` server configuration parameter to `true`. For information about the `watt.server.ldap.cleanContext` server configuration parameter, see *webMethods Integration Server Administrator's Guide*.

---

## pub.client.ldap:registerNotification

WmPublic. Creates a notification (or "persistent search") that causes Integration Server to listen for LDAP events. When the notification gets an event, the specified service is called.

### Input Parameters

---

<i>url</i>	<b>String</b> Optional. URL of the LDAP server to connect to.
<i>principal</i>	<b>String</b> Optional. The principal for the LDAP server.
<i>credentials</i>	<b>String</b> Optional. Credentials for the LDAP server.
<i>timeout</i>	<b>String</b> Optional. Connection timeout in milliseconds.
<i>ldapEnv</i>	<b>Record</b> Optional. Key/value parameters to be passed to JNDI to further define the connection environment. See your JNDI provider documentation or the Oracle JNDI documentation for more information about parameters you can pass to JNDI.
<i>close</i>	<p><b>String</b> Flag that specifies whether to close the connection after the service finishes. Set to:</p> <ul style="list-style-type: none"> <li>■ <code>yes</code> to close the connection. This is the default.</li> <li>■ <code>no</code> to leave the connection open and available.</li> </ul>
<i>dn</i>	<b>String</b> The distinguished name of the entry to be monitored.
<i>connectionHandle</i>	<b>Object</b> Optional. Connection object returned by a previously invoked LDAP service.
<i>scope</i>	<b>String</b> The scope of the search. Must be "object" (only search the specified directory entry, "onelevel" (only search the immediate

children of the specified directory entry), or "subtree" (search the directory entry, its children, and all of their children).

<i>service</i>	<b>String</b> The target service to be invoked when the LDAP event is retrieved.
<i>user</i>	<b>String</b> Optional. Integration Server user to run <i>service</i> (the target service to be invoked when the LDAP event is retrieved). If you do not specify a user, the service runs as the Default user. Make sure <i>user</i> has the permissions necessary to run the service. Be careful when assigning the user because no password is required when invoking a service in this manner. It is recommended that you create a special account just for invoking the target service.

### Output Parameters

---

<i>connectionHandle</i>	<b>Object</b> Optional. The returned connection object. Returned only if the <i>close</i> parameter is set to "no".
-------------------------	---

### Usage Notes

When the `pub.client.ldap:registerNotification` service creates a notification, Integration Server listens for four different types of events: `objectAdded`, `objectRemoved`, `objectRenamed`, and `objectChanged`. If any one of these events is triggered, `pub.client.ldap:registerNotification` calls the specified target service and passes these inputs to it:

<u>Pipeline Input</u>	<u>Description</u>
<i>type</i>	One of the following depending on which event was triggered - "objectAdded", "objectRemoved", "objectRenamed", "objectChanged".
<i>dn</i>	Distinguished name of the entry that triggered the event.
<i>attributes</i>	Any additional LDAP attributes from the event.
<i>oldDn</i>	Applicable only for <code>objectRenamed</code> event. Distinguished name of the entry before it was renamed.

If an error occurs, `pub.client.ldap:registerNotification` places an input called "exception" in the pipeline. This input includes details on the exception that occurred.

Some LDAP servers do not support persistent searches and therefore do not support notifications.

When *close* is set to *yes*, Integration Server removes the *connectionHandle* from the pipeline, but does not close the LDAP connection. To close the LDAP connection, set the `watt.server.ldap.cleanContext` server configuration parameter to `true`. For information about the `watt.server.ldap.cleanContext` server configuration parameter, see *webMethods Integration Server Administrator's Guide*.

---

## pub.client.ldap:rename

WmPublic. Performs an LDAP rename (move) operation allowing you to rename an entry.

### Input Parameters

---

<i>url</i>	<b>String</b> Optional. URL of the LDAP server to connect to.
<i>principal</i>	<b>String</b> Optional. The principal for the LDAP server.
<i>credentials</i>	<b>String</b> Optional. Credentials for the LDAP server.
<i>timeout</i>	<b>String</b> Optional. Connection timeout in milliseconds.
<i>ldapEnv</i>	<b>Record</b> Optional. Key/value parameters to be passed to JNDI to further define the connection environment. See your JNDI provider documentation or the Oracle JNDI documentation for more information about parameters you can pass to JNDI.
<i>close</i>	<b>String</b> Flag that specifies whether to close the connection after the service finishes. Set to: <ul style="list-style-type: none"> <li>■ <code>yes</code> to close the connection. This is the default.</li> <li>■ <code>no</code> to leave the connection open and available.</li> </ul>
<i>connectionHandle</i>	<b>Object</b> Optional. Connection object returned by a previously invoked LDAP service.
<i>newDn</i>	<b>String</b> The new name for the entry.

### Output Parameters

---

<i>connectionHandle</i>	<b>Object</b> Optional. The returned connection object. Returned only if the <i>close</i> parameter is set to "no".
-------------------------	---

## Usage Notes

When *close* is set to yes, Integration Server removes the *connectionHandle* from the pipeline, but does not close the LDAP connection. To close the LDAP connection, set the `watt.server.ldap.cleanContext` server configuration parameter to `true`. For information about the `watt.server.ldap.cleanContext` server configuration parameter, see *webMethods Integration Server Administrator's Guide*.

---

## pub.client.ldap:search

WmPublic. Performs an LDAP search operation with the specified parameters and returns the results of the search.

### Input Parameters

---

<i>url</i>	<b>String</b> Optional. URL of the LDAP server to connect to.
<i>principal</i>	<b>String</b> Optional. The principal for the LDAP server.
<i>credentials</i>	<b>String</b> Optional. Credentials for the LDAP server.
<i>timeout</i>	<b>String</b> Optional. Connection timeout in milliseconds.
<i>ldapErro</i>	<b>Record</b> Optional. Key/value parameters to be passed to JNDI to further define the connection environment. See your JNDI provider documentation or the Oracle JNDI documentation for more information about parameters you can pass to JNDI.
<i>close</i>	<p><b>String</b> Flag that specifies whether to close the connection after the service finishes. Set to:</p> <ul style="list-style-type: none"> <li>■ <code>yes</code> to close the connection. This is the default. If the <i>close</i> parameter is set to "yes", the <i>connectionHandle</i> parameter must also be mapped.</li> <li>■ <code>no</code> to leave the connection open and available.</li> </ul>
<i>dn</i>	<b>String</b> The distinguished name indicating the root from to begin the search.
<i>connectionHandle</i>	<b>Object</b> Required if the <i>close</i> parameter is set to "yes", otherwise it is optional. Connection object returned by a previously invoked LDAP service.

---

<i>scope</i>	<b>String</b> The scope of the search. Must be "object" (only search the specified directory entry), "onelevel" (only search the immediate children of the specified directory entry), or "subtree" (search the directory entry, its children, and all their children).
<i>filter</i>	<b>String</b> The filter string that works with RFC 2254.
<i>countLimit</i>	<b>String</b> Optional. The maximum number of results to return (0, the default, indicates no limit).
<i>timeLimit</i>	<b>String</b> Optional. The number of milliseconds to wait for the search to complete (0, the default, indicates to wait forever).
<i>returnAttributes</i>	<b>Record</b> Optional. A list of attribute names to return (an empty array indicates that no results should be returned. A null array, the default, indicates that all attributes should be returned).
<i>returnObjects</i>	<b>String</b> Optional. Specifies whether or not objects associated with the results should be returned. Can be "yes" or "no". The default is "no".
<i>dereferenceLinks</i>	<b>String</b> Optional. Whether to return the symbolic link to the entry or the entry itself. Can be "yes"/"no". The default is "yes", which returns the entry to which the link points.
<i>isDocumentList</i>	<b>String</b> Optional. Whether to return results as a list. Set to: <ul style="list-style-type: none"> <li>■ Yes to return results as a Document List containing Documents (IData). The results are returned in the <i>resultsList</i> output parameter.</li> <li>■ No (the default) to return results as a Document containing Documents. The results are returned in the <i>results</i> output parameter.</li> </ul>

### Output Parameters

---

<i>connectionHandle</i>	<b>Object</b> Optional. The returned connection object. Returned only if the <i>close</i> parameter is set to No.
<i>results</i>	<b>Document</b> Conditional. The returned results of the search. Returned only if <i>isDocumentList</i> is set to No.
<i>resultsList</i>	<b>Document List</b> Conditional. Returned only if <i>isDocumentList</i> is set to Yes.

## Usage Notes

To see if no match was found, check for either:

- A *results* parameter that does not contain a key-value pair.
- A *resultsList* parameter with a size of 0.

When *close* is set to yes, Integration Server removes the *connectionHandle* from the pipeline, but does not close the LDAP connection. To close the LDAP connection, set the `watt.server.ldap.cleanContext` server configuration parameter to `true`. For information about the `watt.server.ldap.cleanContext` server configuration parameter, see *webMethods Integration Server Administrator's Guide*.

---

## pub.client.oauth:executeRequest

WmPublic. Allows Integration Server to access protected resources on a resource server using an existing Open Authentication (OAuth) access token.

When a client application requires the OAuth protocol to access a user's protected resources on a third-party resource server (for example, Facebook, Google, Twitter, or another Integration Server), the client application must present an access token on behalf of the user in order to gain access. This service presents the access token to the resource server on behalf of the user.

**Note:** To use this service, you must have registered the client with the provider's authorization server and received an access token. You will use the information given to you by the provider to configure this service. For information about registering a client and obtaining an access token, refer to the provider's documentation.

Integration Server uses the Scribe API, a simple open source client implementation, to connect to OAuth providers. The Scribe API provides client implementations for many providers such as Facebook, Google, and Twitter. For information about using the Scribe API, see the Usage Notes.

## Input Parameters

---

*provider*

**String** Name of the service provider to which the client will connect. Integration Server uses this parameter to determine which OAuth client implementation in the Scribe library to use when issuing requests.

Possible values are `IntegrationServer` and `Other` (case insensitive).

**Note:** If set to `Other`, you must specify a value for *providerClass*.

---

<i>providerClass</i>	<p><b>String</b> Name of a class that implements the <code>org.scribe.builder.api.Api</code> interface.</p> <p>This parameter is required only when the <i>provider</i> parameter is set to <code>Other</code>.</p> <p><b>Note:</b> The <code>org.scribe.builder.api.Api</code> interface is part of the Scribe API. It facilitates the use of the <code>pub.client.oauth.executeRequest</code> service to connect to providers other than <code>IntegrationServer</code>. For more information about <code>org.scribe.builder.api.Api</code> see the Usage Notes.</p>
<i>clientID</i>	<p><b>String</b> The client identifier assigned to the client by the provider.</p> <p>The <i>clientID</i> is used to authenticate the client to the provider. The value is assigned by the provider at registration time.</p>
<i>clientSecret</i>	<p><b>String</b> The secret assigned to the client when it registered with the provider.</p> <p>Use this parameter to specify either the client secret or the key to the client secret in the outbound password store. For information about using the outbound password store, see the Usage Notes.</p>
<i>accessToken</i>	<p><b>String</b> The access token assigned to the client application when it registered with the provider.</p> <p><b>Note:</b> The process for obtaining the <i>accessToken</i> varies depending on the provider. For information about obtaining the <i>accessToken</i>, refer to your provider's documentation.</p> <p>Use this parameter to specify either the access token or the key to the access token in the outbound password store. For information about using the outbound password store, see the Usage Notes.</p>
<i>accessTokenSecret</i>	<p><b>String</b> Optional. The access token secret assigned to the client application when it registered with the provider.</p> <p><b>Note:</b> The process for obtaining the <i>accessTokenSecret</i> varies depending on the provider. For information about obtaining the <i>accessTokenSecret</i>, refer to your provider's documentation.</p> <p>Use this parameter to specify either the access token secret or the key to the access token secret in the outbound password store.</p>

For information about using the outbound password store, see the Usage Notes.

**Note:** Not all providers use an access token secret. If the provider issued a secret with the token, you must specify it with the *accessTokenSecret* parameter.

<i>resourceUri</i>	<p><b>String</b> The URI given to the client application when it registered with the provider.</p> <p>The client application uses this URI when issuing requests to the provider. It points to the resource that the client application wants to access on the provider.</p>
<i>method</i>	<p><b>String</b> The HTTP method Integration Server will use to issue the request to the provider.</p> <p>Possible values are <i>get</i>, <i>post</i>, <i>put</i>, and <i>delete</i>.</p>
<i>headers</i>	<p><b>Document List</b> Optional. One or more name/value pairs to add to the header of the request sent to the provider.</p> <p>If the provider requires a request header, you must provide a value.</p>
<i>queryStringParameters</i>	<p><b>Document List</b> Optional. One or more name/value pairs to add to the URL of the <i>get</i> request sent to the provider.</p>
<i>bodyParameters</i>	<p><b>Document List</b> Optional. One or more name/value pairs to add to the body of the request sent to the provider.</p>
<i>requestDataType</i>	<p><b>String</b> Optional. If supplying data with the request to the provider, indicates the data type of the <i>requestData</i> parameter. If set to:</p> <ul style="list-style-type: none"> <li>■ <i>bytes</i>, <i>requestData</i> must be a <code>byte[]</code>. This is the default.</li> <li>■ <i>string</i>, <i>requestData</i> must be a <code>java.lang.String</code>.</li> </ul>
<i>requestData</i>	<p><b>Object</b> Optional. Data to include in the body of the request sent to the provider. The value can be a string or a <code>byte[]</code>.</p> <p><b>Note:</b> The data type of <i>requestData</i> must match what is specified by the <i>requestDataType</i> parameter. <i>requestData</i> is ignored if <i>bodyParameters</i> is specified.</p>
<i>responseDataType</i>	<p><b>String</b> Optional. Indicates the data type of the <i>responseData</i> output parameter. Set to:</p>

- `stream` to return the `responseData` as a `java.io.InputStream`. This is the default.
- `bytes` to return the `responseData` as a `byte[]`.
- `string` to return the `responseData` as a `java.lang.String`, constructed using the default platform encoding (indicated by the `file.encoding` Java system property, or UTF-8 if `file.encoding` is not set).

## Output Parameters

---

*responseData* **Object** The response from the provider. If the *responseDataType* is set to:

- `stream`, *responseData* is a `java.io.InputStream`.
- `bytes`, *responseData* is a `byte[]`.
- `string`, *responseData* is a `java.lang.String`, constructed using the default platform encoding (indicated by the `file.encoding` Java system property, or UTF-8 if `file.encoding` is not set).

## Usage Notes

- Since the values for the *clientSecret*, *accessToken*, and *accessTokenSecret* parameters contain sensitive data, you might want to consider storing their values in the Integration Server outbound password store. For information about services you can use to store these values in the outbound password store, see the `pub.security.outboundPasswords` in the [About the Security Elements](#) folder.

If you decide to use the outbound password store, the value for each parameter (*clientSecret*, *accessToken*, and *accessTokenSecret*) must match the key you supplied in the `pub.security.outboundPasswords` services. Integration Server uses that key to retrieve the values from the store, then uses the values to send the request to the provider.

If you decide not to use the outbound password store, Integration Server sends the request to the provider using the values you supply with the *clientSecret*, *accessToken*, and *accessTokenSecret* parameters.

- You can use this service to connect to OAuth providers other than Integration Server by setting the *provider* parameter to `Other` and the *providerClass* parameter to the name of an `org.scribe.builder.api.Api` implementation. The `org.scribe.builder.api.Api` interface is defined in the Scribe open source API. For this approach, Software AG recommends the following:
  - That you download the Scribe API source code from the GitHub website. You can either browse the code or generate the Javadoc for the Scribe classes using the following command:

```
javadoc -sourcepath your_scribe_install_dir/src/main/java -d
your_destination_dir org.scribe.builder.api
```

- That you check the Scribe OAuth library to see if an implementation for the provider that you want to use already exists. Scribe provides client implementation for many providers.

**Important:** If an implementation already exists but the provider has changed its interface and the implementation in the Scribe library no longer works, you can either modify the implementation yourself or request an update from the author of Scribe.

- If your provider supports OAuth 2.0, then extend `org.scribe.builder.api.DefaultApi20`. If your provider supports OAuth 1.0a, then extend `org.scribe.builder.api.DefaultApi10a`.

---

## pub.client.sftp:cd

WmPublic. Changes the working directory on the remote SFTP server.

### Input Parameters

---

<i>sessionKey</i>	<b>String</b> Unique key for the current SFTP session. The <i>sessionKey</i> is returned by the <a href="#">pub.client.sftp:login</a> service.
<i>path</i>	<b>String</b> Absolute or relative path of the directory that you want as the working directory on the remote SFTP server.

### Output Parameters

---

<i>returnCode</i>	<b>String</b> Standard SFTP protocol return code.
<i>returnMsg</i>	<b>String</b> Text message describing the return code.

### Usage Notes

You cannot execute SFTP commands in parallel using the same session key.

---

## pub.client.sftp:chgrp

WmPublic. Changes the group ownership of one or more remote files.

---

### Input Parameters

---

<i>sessionKey</i>	<b>String</b> Unique key for the current SFTP session. The <i>sessionKey</i> is returned by the <a href="#">pub.client.sftp:login</a> service.
<i>groupId</i>	<b>String</b> Numeric group identifier of the group to which you want to transfer ownership of the remote files.
<i>path</i>	<b>String</b> Absolute or relative path of the remote files.

### Output Parameters

---

<i>returnCode</i>	<b>String</b> Standard SFTP protocol return code.
<i>returnMsg</i>	<b>String</b> Text message describing the return code.

### Usage Notes

You cannot execute SFTP commands in parallel using the same session key.

---

## pub.client.sftp:chmod

WmPublic. Changes permissions of one or more remote files.

### Input Parameters

---

<i>sessionKey</i>	<b>String</b> Unique key for the current SFTP session. The <i>sessionKey</i> is returned by the <a href="#">pub.client.sftp:login</a> service.
<i>mode</i>	<b>String</b> The permission mode to apply to the remote file (for example, <i>777</i> ).
<i>path</i>	<b>String</b> Absolute or relative path of the remote files.

### Output Parameters

---

<i>returnCode</i>	<b>String</b> Standard SFTP protocol return code.
<i>returnMsg</i>	<b>String</b> Text message describing the return code.

---

### Usage Notes

You cannot execute SFTP commands in parallel using the same session key.

---

## pub.client.sftp:chown

WmPublic. Changes the owning user of one or more remote files.

---

### Input Parameters

<i>sessionKey</i>	<b>String</b> Unique key for the current SFTP session. The <i>sessionKey</i> is returned by the <a href="#">pub.client.sftp:login</a> service.
<i>uid</i>	<b>String</b> Numeric user ID of the new owning user of the file.
<i>path</i>	<b>String</b> Absolute or relative path of the remote files.

---

### Output Parameters

<i>returnCode</i>	<b>String</b> Standard SFTP protocol return code.
<i>returnMsg</i>	<b>String</b> Text message describing the return code.

### Usage Notes

You cannot execute SFTP commands in parallel using the same session key.

---

## pub.client.sftp:get

WmPublic. Retrieves a file from a remote SFTP server and saves it on the local machine.

---

### Input Parameters

<i>sessionKey</i>	<b>String</b> Unique key for the current SFTP session. The <i>sessionKey</i> is returned by the <a href="#">pub.client.sftp:login</a> service.
<i>remoteFile</i>	<b>String</b> Absolute or relative path of the remote file.
<i>localFile</i>	<b>String</b> Optional. Absolute or relative path of the local file.

If *localFile* is not specified, the `pub.client.sftp:get` service returns the retrieved file in the output parameter *contentStream* (as a `java.io.InputStream` object).

*mode*

**String** Optional. Specifies how the retrieved file is to be saved to the local file. Use this parameter only if you have specified a value for *localFile*. Set to:

- `overwrite` to overwrite the contents of the local file with the contents of the remote file. This is the default.
- `append` to append the entire contents of the remote file to the local file.
- `resume` to resume writing the contents of the remote file to the local file from the point at which the writing stopped during previous SFTP sessions.

## Output Parameters

*returnCode*      **String** Standard SFTP protocol return code.

*returnMsg*      **String** Text message describing the return code.

*contentStream*      **Object** Conditional. A `java.io.InputStream` object.

The `pub.client.sftp:get` service returns the retrieved file in the output parameter *contentStream* (as a `java.io.InputStream` object) if *localFile* is not specified.

## Usage Notes

You cannot execute SFTP commands in parallel using the same session key.

## pub.client.sftp:login

WmPublic. Connects to a remote SFTP server and logs in with the specified SFTP user alias.

**Important:** You must use this service to initiate an SFTP session before using most other `pub.client.sftp` services.

---

### Input Parameters

---

<i>userAlias</i>	<b>String</b> Alias containing the SFTP client configuration for an SFTP user account. Integration Server creates a new session each time it logs on to the SFTP server and returns the session key to the user.
<i>reuseSession</i>	<b>String</b> Flag indicating whether or not Integration Server reuses a session that is already open for the specified user alias. Set to: <ul style="list-style-type: none"><li>■ <code>true</code> to reuse the session that is already open for the specified user alias. If no session is open for this user alias, the <a href="#">pub.client.sftp:login</a> service will create a new session and return the key for this new session.</li><li>■ <code>false</code> to create a new session for the specified user alias. This is the default.</li></ul>

---

### Output Parameters

---

<i>sessionKey</i>	<b>String</b> Unique key for the current SFTP session. Most other <a href="#">pub.client.sftp</a> services require this session key to execute.
<i>returnCode</i>	<b>String</b> Standard SFTP protocol return code.
<i>returnMsg</i>	<b>String</b> Text message describing the return code.

### Usage Notes

You cannot execute SFTP commands in parallel using the same session key.

---

## pub.client.sftp:logout

WmPublic. Logs off the user from the SFTP server and ends the current SFTP session.

---

### Input Parameters

---

<i>sessionKey</i>	<b>String</b> Unique key for the current SFTP session. The <i>sessionKey</i> parameter is returned by the <a href="#">pub.client.sftp:login</a> service.
-------------------	--

---

## Output Parameters

---

*returnCode*     **String** Standard SFTP protocol return code.

*returnMsg*     **String** Text message describing the return code.

## Usage Notes

You cannot execute SFTP commands in parallel using the same session key.

---

## pub.client.sftp:ls

WmPublic. Retrieves the remote directory listing of the specified path. If path is not specified, the `pub.client.sftp:ls` service retrieves the file listing of the current remote directory. The `pub.client.sftp:ls` service also retrieves additional details such as permissions and ownership information.

---

## Input Parameters

---

*sessionKey*     **String** Unique key for the current SFTP session. The *sessionKey* parameter is returned by the `pub.client.sftp:login` service.

*path*            **String** Optional. Absolute or relative path of the remote directory. If no *path* is specified, the `pub.client.sftp:ls` service retrieves the directory listing of the current remote directory.

You can use the wildcard characters asterisk (\*) and question mark (?) after the last slash mark (/) to view all remote directories that match the specified path.

---

## Output Parameters

---

*returnCode*     **String** Standard SFTP protocol return code.

*returnMsg*     **String** Text message describing the return code.

*dirList*        **String List** List of directories matching the pattern specified in the *path* parameter.

## Usage Notes

You cannot execute SFTP commands in parallel using the same session key.

---

## pub.client.sftp:mkdir

WmPublic. Creates a new remote directory.

### Input Parameters

---

<i>sessionKey</i>	<b>String</b> Unique key for the current SFTP session. The <i>sessionKey</i> parameter is returned by the <a href="#">pub.client.sftp:login</a> service.
<i>path</i>	<b>String</b> Absolute or relative path of the remote directory where you want to create a new directory.

### Output Parameters

---

<i>returnCode</i>	<b>String</b> Standard SFTP protocol return code.
<i>returnMsg</i>	<b>String</b> Text message describing the return code.

### Usage Notes

You cannot execute SFTP commands in parallel using the same session key.

---

## pub.client.sftp:put

WmPublic. Transfers a file to a remote SFTP server.

### Input Parameters

---

<i>sessionKey</i>	<b>String</b> Unique key for the current SFTP session. The <i>sessionKey</i> is returned by the <a href="#">pub.client.sftp:login</a> service.
<i>contentStream</i>	<b>java.io.InputStream</b> Optional. Data to be transferred to the remote file.
<i>localFile</i>	<b>String</b> Optional. Name of the local file to be appended to the remote file. Use <i>localFile</i> only if <i>contentStream</i> is not specified.
<i>remoteFile</i>	<b>String</b> Optional. Absolute or relative path of the remote file to which the local file is to be appended.

---

<i>mode</i>	<p><b>String</b> Optional. Specifies how the local file is to be transferred to the remote SFTP server. Set to:</p> <ul style="list-style-type: none"> <li>■ <code>overwrite</code> to overwrite the contents of the remote file with the contents of the local file. This is the default.</li> <li>■ <code>append</code> to append the entire contents of the local file to the remote file.</li> <li>■ <code>resume</code> to resume writing the contents of the local file to the remote file from the point the writing was stopped during previous SFTP sessions.</li> </ul>
-------------	---

### Output Parameters

---

<i>returnCode</i>	<b>String</b> Standard SFTP protocol return code.
<i>returnMsg</i>	<b>String</b> Text message describing the return code.

### Usage Notes

If you specify *contentStream*, you must specify *remoteFile*. In this case, *localFile* is optional.

If you specify *localFile*, then *remoteFile* and *contentStream* are optional. In this case, the remote file will be given the same name as the local file.

You cannot execute SFTP commands in parallel using the same session key.

---

## pub.client.sftp:pwd

WmPublic. Displays the remote working directory in the SFTP server.

### Input Parameters

---

<i>sessionKey</i>	<b>String</b> Unique key for the current SFTP session. The <i>sessionKey</i> is returned by the <a href="#">pub.client.sftp:login</a> service.
-------------------	--

### Output Parameters

---

<i>returnCode</i>	<b>String</b> Standard SFTP protocol return code.
<i>returnMsg</i>	<b>String</b> Text message describing the return code.

---

*path*            **String** Absolute or relative path of the working directory on the remote SFTP server.

### Usage Notes

You cannot execute SFTP commands in parallel using the same session key.

---

## pub.client.sftp:rename

WmPublic. Renames a file or directory on a remote SFTP server.

### Input Parameters

---

*sessionKey*    **String** Unique key for the current SFTP session. The *sessionKey* is returned by the [pub.client.sftp:login](#) service.

*oldPath*        **String** Fully qualified name of the file you want to rename (for example, `temp/oldname.txt`).

*newPath*        **String** New fully qualified name for the file (for example, `temp/newname.txt`).

### Output Parameters

---

*returnCode*    **String** Standard SFTP protocol return code.

*returnMsg*     **String** Text message describing the return code.

### Usage Notes

You cannot execute SFTP commands in parallel using the same session key.

---

## pub.client.sftp:rm

WmPublic. Deletes one or more remote files on the SFTP server.

### Input Parameters

---

*sessionKey*    **String** Unique key for the current SFTP session. The *sessionKey* is returned by the [pub.client.sftp:login](#) service.

---

*path*            **String** Absolute or relative path of the file you want to delete.

### Output Parameters

---

*returnCode*    **String** Standard SFTP protocol return code.

*returnMsg*     **String** Text message describing the return code.

### Usage Notes

You cannot execute SFTP commands in parallel using the same session key.

---

## pub.client.sftp:rmdir

WmPublic. Deletes one or more remote directories on the SFTP server.

### Input Parameters

---

*sessionKey*    **String** Unique key for the current SFTP session. The *sessionKey* is returned by the [pub.client.sftp:login](#) service.

*path*            **String** Absolute or relative path of the directory you want to delete.

### Output Parameters

---

*returnCode*    **String** Standard SFTP protocol return code.

*returnMsg*     **String** Text message describing the return code.

### Usage Notes

The remote directories that you want to delete must be empty.

You cannot execute SFTP commands in parallel using the same session key.

---

## pub.client.sftp:symlink

WmPublic. Creates a symbolic link between the old path and the new path of a file.

---

### Input Parameters

---

<i>sessionKey</i>	<b>String</b> Unique key for the current SFTP session. The <i>sessionKey</i> is returned by the <a href="#">pub.client.sftp.login</a> service.
<i>oldPath</i>	<b>String</b> Old path of the file for which you want to create a symbolic link.
<i>newPath</i>	<b>String</b> New path of the file to which the symbolic link should point.

### Output Parameters

---

<i>returnCode</i>	<b>String</b> Standard SFTP protocol return code.
<i>returnMsg</i>	<b>String</b> Text message describing the return code.

### Usage Notes

You cannot execute SFTP commands in parallel using the same session key.

---

## pub.client:smtp

WmPublic. Sends a MIME-type e-mail message.

You may attach one or more content objects or files to the message.

### Input Parameters

---

<i>to</i>	<b>String</b> Optional. E-mail address of the receiver. If you specify multiple addresses, separate them with commas.
<i>cc</i>	<b>String</b> Optional. E-mail addresses of additional receivers. If you specify multiple addresses, separate them with commas.
<i>bcc</i>	<b>String</b> Optional. E-mail addresses of additional receivers. If you specify multiple addresses, separate them with commas.
<i>subject</i>	<b>String</b> Optional. Subject of the message.
<i>subjectCharset</i>	<b>String</b> Optional. The character set used to encode the MIME message headers (including subject). If <i>subjectCharset</i> is not specified, then <i>charset</i> is used. If <i>charset</i> is not specified, the value in the server

configuration parameter `watt.server.email.charset` is used. If that parameter is not set, the `utf-8` encoding is used.

*charset* **String** Optional. The character encoding of the body text. If you do not specify the value of *charset*, the value in the server configuration parameter `watt.server.email.charset` is used. If that parameter is not set, the `utf-8` encoding is used.

*from* **String** Optional. E-mail address of the sender. If you do not specify a *from* value, Integration Server uses the value specified for the `mail.smtp.from` JVM property. If no value is specified for that property, Integration Server uses the default value, `user@servername`, where *user* is the operating system user ID, and *servername* is the host name of the Integration Server.

*mailhost* **String** SMTP host name for outbound messages. For example:  
`smtp.webMethods.com`

If no value is provided for the *mailhost* parameter, Integration Server uses the value of the Java system property `mail.smtp.host` set for Integration Server as the *mailhost* value. For more information about setting Java system properties for Integration Server, which includes modifying the `custom_wrapper.conf` file, see *webMethods Integration Server Administrator's Guide*.

If no value is supplied for the *mailhost* parameter and the `mail.smtp.host` Java system property is not set, the `pub.client:smtp` service ends with an exception.

*mailhostPort* **String** Optional. The number of the port on which the SMTP host listens. This parameter does not need to be set if the host listens on port 25 (the standard SMTP port).

*auth* **Document** Optional. Authorization information that the SMTP service will submit.

Key	Description
<i>user</i>	<b>String</b> User name that this service will submit when requesting a protected resource.
<i>pass</i>	<b>String</b> Password associated with <i>user</i> .

*secure* **Document** Optional. Parameters specifying the security protocol and truststore information for certificate validation that Integration Server uses when communicating with the SMTP server port.

<u>Key</u>	<u>Description</u>
<i>transportLayerSecurity</i>	<p><b>String</b> Type of security protocol Integration Server uses when communicating with the SMTP server port. Set to:</p> <ul style="list-style-type: none"> <li>■ <code>none</code> to use a non-secure mode when communicating with the port on the SMTP sever. This is the default.</li> <li>■ <code>explicit</code> to use explicit security when communicating with the port on the SMTP server. With explicit security, Integration Server establishes an un-encrypted connection to the e-mail server and then switches to the secure mode.</li> <li>■ <code>implicit</code> to use implicit security when communicating with the port on the SMTP server. With implicit security, Integration Server always establishes an encrypted connection to the e-mail server.</li> </ul>
<i>truststoreAlias</i>	<p><b>String</b> Optional. Alias for the truststore that contains the list of certificates that Integration Server uses to validate the trust relationship.If you do not specify a truststore alias, the default truststore alias will be used.</p>
<i>body</i>	<p><b>String</b> Optional. The content of the message.</p>
<i>mimeStream</i>	<p><b>java.io.InputStream</b> Optional. MIME or S/MIME message that you want to send in the e-mail. A <i>mimeStream</i> is created by the <a href="#">pub.mime:getEnvelopeStream</a>, <a href="#">pub.smime:createEncryptedData</a>, <a href="#">pub.smime:createSignedData</a>, or <a href="#">pub.smime.keystore:createSignedData</a> services. It contains both headers and content. If the <i>mimeStream</i> already contains the from, to, and subject headers, you do not need to pass them as individual inputs to this service.</p>
<i>attachments</i>	<p><b>Document List</b> Optional. Attachments to the message. Each attachment defines one message part in a multi-part message.</p>
<u>Key</u>	<u>Description</u>
<i>contentType</i>	<p><b>String</b> MIME type of the message. For example:</p>

application/x-edifact

<i>content</i>	<b>byte[]</b> , <b>String</b> , or <b>java.io.InputStream</b> Content of the message.
<i>filename</i>	<p><b>String</b> Name to assign to the attachment. If you do not specify <i>attachment/content</i>, the parameter specifies the file name of a local file to attach to the message. In other words:</p> <ul style="list-style-type: none"> <li>■ If you specify <i>attachment/content</i> and <i>attachments/filename</i>, the service uses the value of <i>attachments/filename</i> as the name to assign to the attachment specified by <i>attachment/content</i>.</li> <li>■ If you specify <i>attachments/filename</i>, but not <i>attachment/content</i>, the service attaches the local file specified by <i>attachments/filename</i>.</li> </ul>
<i>encoding</i>	<b>String</b> Optional. Encoding of the message. For example: <code>base64</code> or <code>7bit</code> . If <i>encoding</i> is not specified, <code>7bit</code> is used.
<i>charset</i>	<b>String</b> Optional. Character set encoding of the attachment. This value is added to the Content-Type header for the attachment. If <i>charset</i> is not specified, the value in the server configuration parameter <code>watt.server.email.charset</code> is used. If that parameter is not set, the <code>utf-8</code> encoding is used.

*properties* **Document List** Optional. SMTP system properties to send to the SMTP server. This parameter overrides the settings on the `watt.config.systemProperties` server configuration parameter. If you omit this *properties* parameter, Integration Server uses the settings specified on the `watt.config.systemProperties` server configuration parameter.

<u>Key</u>	<u>Description</u>
<i>name</i>	Name of the SMTP system property.
<i>value</i>	Value to specify for the SMTP property.

---

## Output Parameters

---

*status*                      **String** Final status of service.

### Usage Notes

Any one of the recipient fields, that is the *to*, *cc*, or the *bcc* parameter, must be defined.

If you are using *filename* to attach a file to the message and the file is not a plain text file, you must set the *contenttype* and *encoding*. For example, to attach *IntegrationServer\_directory \instances \instance\_name \mydir \myfile.doc* to a [pub.client:smtp](#) service, you would invoke the service with the following values in *attachments*:

*contenttype* :application/msword

*filename* :instances/*instance\_name*/mydir/myfile.doc

*encoding* :base64

---

## pub.client:soapClient

WmPublic. Creates and sends SOAP 1.1 and SOAP 1.2 messages over HTTP, HTTPS, or JMS transports for any style/use combination supported by Integration Server.

### Input Parameters

---

*address*                      **String** String specifying the URI of the web service endpoint. If you are submitting the request to an Integration Server, remember to direct it to the default SOAP processor (ws) as shown in the following example:

```
http://rubicon:5555/soap/ws/example:calculator
```

*request*                      **Document** The input parameters that are to be passed to the web service.

*wsdName*                      **String** The name of the consumer web service descriptor that contains the operation you want to invoke.

*wsdBinderName*              **String** The name of a binder that contains information to use to create and send the request. This binder must be in the consumer web service descriptor specified in *wsdName*.

**Note:** The consumer web service endpoint alias assigned to a binder indicates which proxy server Integration Server uses to send the request. For more information about proxy server usage and web service endpoint

aliases, see the *webMethods Integration Server Administrator's Guide* guide.

*wsdOperationName* **String** The name of the operation that you want to invoke. This operation must be contained in the binder specified in *wsdBinderName* .

*targetInputSignature* **String** Fully qualified name of the IS document type to use to validate and encode the contents of *request* .

*targetOutputSignature* **String** Fully qualified name of the IS document type to use to validate and decode the output value returned by the web service.

*soapHeaders* **Document** Optional. Header documents included in the SOAP request as SOAP headers.

*transportHeaders* **Document** Optional. Transport header fields that you want to explicitly set in the request issued by the `pub.client:soapClient` service.

Specify a key in *transportHeaders* for each header field that you want to set, where the key's name represents the name of the header field and the key's value represents the value of that header field.

The names and values supplied to *transportHeaders* must be of type `String`. If a transport header has a name or value that is not of type `String`, the header will not be included in the message.

The headers that you pass in to *transportHeaders* vary depending on the transport used to send the SOAP message. The supplied *wsdBinderName* determines the transport.

For more information about specifying *transportHeaders* , refer to the Usage Notes.

*method* **Document** Optional. The QName of the requested procedure. The name is defined as follows:

<u>Key</u>	<u>Value</u>
<i>namespaceName</i>	<b>String</b> Namespace portion of the procedure's QName.

*localName* **String** Local portion of the procedure's QName.

**Note:** The *method* parameter applies when style is RPC.

*auth*

**Document** Optional. Parameters specifying the credentials that are to be submitted to the server specified in *address*.

Integration Server allows two levels of authorization credentials: transport level and message level.

Each element is defined as follows:

<u>Key</u>	<u>Description</u>						
<i>transport</i>	<b>Document</b> Optional. Transport level authorization parameters.						
	<table border="1"> <thead> <tr> <th><u>Key</u></th> <th><u>Description</u></th> </tr> </thead> <tbody> <tr> <td><i>type</i></td> <td> <b>String</b> Optional. Type of authentication that the service will perform. Set to: <ul style="list-style-type: none"> <li>■ Basic to submit a user name and password. This is the default.</li> <li>■ Digest to submit a password digest for authentication.</li> <li>■ NTLM to use NTLM authentication.</li> </ul> </td> </tr> <tr> <td><i>user</i></td> <td> <b>String</b> Optional. User name that this service will use if one is requested. </td> </tr> </tbody> </table>	<u>Key</u>	<u>Description</u>	<i>type</i>	<b>String</b> Optional. Type of authentication that the service will perform. Set to: <ul style="list-style-type: none"> <li>■ Basic to submit a user name and password. This is the default.</li> <li>■ Digest to submit a password digest for authentication.</li> <li>■ NTLM to use NTLM authentication.</li> </ul>	<i>user</i>	<b>String</b> Optional. User name that this service will use if one is requested.
<u>Key</u>	<u>Description</u>						
<i>type</i>	<b>String</b> Optional. Type of authentication that the service will perform. Set to: <ul style="list-style-type: none"> <li>■ Basic to submit a user name and password. This is the default.</li> <li>■ Digest to submit a password digest for authentication.</li> <li>■ NTLM to use NTLM authentication.</li> </ul>						
<i>user</i>	<b>String</b> Optional. User name that this service will use if one is requested.						

**Note:** If you have specified NTLM as *type*, you must specify *user* in the following format:

*domain\_name \user\_name*

<i>pass</i>	<b>String</b> Optional. Password that this service will submit if one is requested.								
<i>serverCerts</i>	<b>Document</b> Optional. The message signer's private key and certificate chain.  <i>privateKey</i> <b>Object</b> The SOAP message signer's private key.  <i>certChain</i> <b>Object List</b> A list containing the signer's complete certificate chain, where element 0 in the list contains the signer's certificate and element 1 contains the CA's certificate.								
<i>message</i>	<b>Document</b> Optional. Message level authorization parameters.								
	<table> <thead> <tr> <th><u>Key</u></th> <th><u>Description</u></th> </tr> </thead> <tbody> <tr> <td><i>user</i></td> <td><b>String</b> Optional. User name that this service will use if one is requested.</td> </tr> <tr> <td><i>pass</i></td> <td><b>String</b> Optional. Password that this service will submit if one is requested.</td> </tr> <tr> <td><i>serverCerts</i></td> <td><b>Document</b> Optional. The message signer's private key and certificate chain.  <i>privateKey</i> <b>Object</b> The SOAP message signer's private key.  <i>certChain</i> <b>Object List</b> A list containing the signer's complete certificate</td> </tr> </tbody> </table>	<u>Key</u>	<u>Description</u>	<i>user</i>	<b>String</b> Optional. User name that this service will use if one is requested.	<i>pass</i>	<b>String</b> Optional. Password that this service will submit if one is requested.	<i>serverCerts</i>	<b>Document</b> Optional. The message signer's private key and certificate chain.  <i>privateKey</i> <b>Object</b> The SOAP message signer's private key.  <i>certChain</i> <b>Object List</b> A list containing the signer's complete certificate
<u>Key</u>	<u>Description</u>								
<i>user</i>	<b>String</b> Optional. User name that this service will use if one is requested.								
<i>pass</i>	<b>String</b> Optional. Password that this service will submit if one is requested.								
<i>serverCerts</i>	<b>Document</b> Optional. The message signer's private key and certificate chain.  <i>privateKey</i> <b>Object</b> The SOAP message signer's private key.  <i>certChain</i> <b>Object List</b> A list containing the signer's complete certificate								

chain, where element 0 in the list contains the signer's certificate and element 1 contains the CA's certificate.

*partnerCert* **Object** Optional. The partner's complete certificate chain, where element 0 in the list contains the message signer's certificate and element 1 contains the CA's certificate.

*timeout*

**String** Optional. Time (measured in milliseconds) to wait for a response from the server hosting the web service before timing out and terminating the request.

A value of 0 means Integration Server waits for a response indefinitely. If the connection to the host or JMS provider ends before Integration Server receives a response, the service ends with an exception and a status code of 408.

If this parameter is not specified, or an invalid (non-numeric) value is specified, Integration Server uses one of the following values:

- For HTTP, Integration Server uses the value of the `watt.server.SOAP.request.timeout` server configuration parameter as the *timeout* value
- For JMS, Integration Server uses the value of the `watt.server.soapjms.request.timeout` server configuration parameter as the *timeout* value.

For more information about `watt.server.SOAP.request.timeout` and `watt.server.soapjms.request.timeout` server configuration parameter, see *webMethods Integration Server Administrator's Guide*.

Integration Server ignores timeout if the name/value pair `jms.async= true` is passed in to *transportHeaders*.

*soapAction*

**String** Optional. Specifies one of the following:

- If *soapProtocol* is set to `SOAP 1.1 Protocol`, specifies the value to which you want to set the SOAPAction HTTP header.

- If *soapProtocol* is set to SOAP 1.2 Protocol, specifies the value to which you want to set the *action* attribute in the Content-Type header.

Integration Server ignores *soapAction* in either of the following situations:

- The Content-Type header is passed in to *transportHeaders* and *soapProtocol* is set to SOAP 1.2 Protocol.
- The *soapAction* header is passed into *transportHeaders* and *soapProtocol* is set to SOAP 1.1 Protocol.

*soapProtocol*

**String** Optional. Indicates the SOAP protocol the service uses to send messages. Valid values are SOAP 1.1 Protocol or SOAP 1.2 Protocol.

*encoding*

**String** Optional. Specifies the encoding method. Default value is UTF-8.

Integration Server ignores *encoding* if the Content-Type header is passed in to *transportHeaders*.

## Output Parameters

---

*soapResponseData*

**Object** A SOAP object containing the SOAP response message returned by the server specified in *address*.

*response*

**Document** Output parameters returned from the web service.

*header*

**Document** Conditional. Headers from the response and request messages.

*header* contains the following keys:

<u>Key</u>	<u>Value</u>
<i>requestLines</i>	<p><b>Document</b> Conditional. Header fields from the request message. Each key in <i>requestLines</i> represents a line (field) in the request header. Key names represent the names of header fields. The keys' values are Strings containing the values of the fields.</p> <p>The contents of the <i>requestLines</i> document are not identical to <i>transportHeaders</i>. The</p>

transport can add, remove, or alter specific headers while processing the request.

Whether or not the web service connector returns the *requestLines* parameter depends on the success or failure of the `pub.client:soapClient` service. In the case of failure, the point at which the failure occurs determines the presence of the *requestLines* parameter. For more information, see the usage notes for this service.

For the HTTP or HTTPS transports, the *requestLines* parameter will not contain any HTTP headers that the transport mechanism added or modified when sending the request.

For the JMS transport, each key in *requestLines* represents a JMS message header. Key names represent the names of header fields. Key values are Strings containing the values of the header fields. The JMS provider populates some JMS message header fields after it successfully receives the JMS message. Additionally, the Integration Server specific run-time properties (properties that begin with the "jms." prefix) are not returned in JMS transport, each key in *requestLines*. The JMS provider uses the information in these properties to populate the JMS message header fields that correspond to the properties.

### *lines*

**Document** Header fields from the response. Each key in *lines* represents a field (line) of the response header. Key names represent the names of header fields. The keys' values are Strings containing the values of the fields.

Whether or not the `pub.client:soapClient` service returns the *lines* parameter depends on the success or failure of the service. In the case of failure, the point at which the failure occurs determines the presence of the *lines* parameter. For more information, see the usage notes for this service.

For the HTTP or HTTPS transports, the *lines* parameter contains any HTTP/HTTPS headers present in the response.

For the JMS transport, the *lines* parameter contains the JMS headers present in the response.

*status* **String** Status code from the request, returned by the underlying transport. response. For more information about status codes returned by the service, see the usage notes for this service.

*statusMessage* **String** Status message returned by the transport. For more information about status messages returned by the service, see the usage notes for this service.

*soapStatus* **String** Flag indicating whether the SOAP request message was processed successfully.

<u>A value of...</u>	<u>Indicates that...</u>
0	The remote server successfully processed the SOAP request and returned a SOAP response message.
1	The remote server returned a SOAP fault, indicating that the SOAP request was received but was not processed successfully.
2	The server returned an error that was not a SOAP fault. This indicates that some type of HTTP error occurred (often, an HTTP 404). You can check the <i>status</i> field in <i>header</i> to determine the type of HTTP error that occurred.

### Usage Notes

If the *address* begins with `https:`, you must specify a private key and certificate chain. You can use the *auth/transport/serverCerts* parameters to do so. If you do not specify them using the *auth/transport/serverCerts* parameters, `pub.client:soapClient` uses the web service endpoint alias specified in the binder. If the endpoint alias does not have an associated private key and certificate chain, then the default outbound SSL certificate settings are used to authenticate the resources.

As part of executing `pub.client:soapClient`, Integration Server executes any service handlers assigned to the consumer web service descriptor specified in *wsdName*.

When a document type contains a String variable that represents a required attribute (meaning that the variable name starts with the "@" symbol and the **Required** property is set to **True** in Designer) and the input document does not contain the required attribute, Integration Server adds an empty attribute during document encoding. For example, if the document type contains a required String variable named `@myAttribute` but `@myAttribute` is missing from the input document, Integration Server adds `myAttribute=""` to the XML document.

**Note:** Because empty xmlns attributes are invalid, if the document type contains a required String variable named `@xmlns` and the input document does not specify a value for the `@xmlns` attribute, Integration Server *does not* add `xmlns=""` to the XML document.

Keep the following points in mind when specifying *transportHeaders* for HTTP or HTTPS:

- For any header name/value pair supplied in *transportHeaders*, Integration Server simply passes through the supplied headers and does not perform any validation for the headers.
- If you do not set *transportHeaders* or do not specify the following header fields in *transportHeaders*, Integration Server adds and specifies values for the following header fields:
  - Accept
  - Authorization
  - Connection
  - Content-Type
  - Host
  - SOAPAction (Added when *soapProtocol* is SOAP 1.1 only)
  - User-Agent

**Important:** Pass in the preceding headers to *transportHeaders* only if you are an experienced web service developer. Incorrect header values can result in failure of the HTTP request.

- If you specify `Content-Type` in *transportHeaders*, Integration Server ignores the value of the *encoding* input parameter.
- If you specify `Content-Type` in *transportHeaders* and the *soapProtocol* input parameter is set to SOAP 1.2, Integration Server ignores the value of the *soapAction* input parameter.
- If you specify the `SOAPAction` header in *transportHeaders* and the *soapProtocol* input parameter is set to SOAP 1.1 Protocol, Integration Server ignores the value of the *soapAction* input parameter.

- If MTOM processing converts any portion of the SOAP request to an MTOM/XOP attachment, it will overwrite the `Content-Type` value supplied to the `transportHeaders` input.
- Integration Server sets the value of `Content-Length` automatically and overrides any value passed in to `transportHeaders`.
- Integration Server automatically adds the `Cookie` header to the HTTP header and supplies any cookies established between Integration Server and the HTTP server with which it is interacting. If you supply the `Cookie` header to `transportHeaders`, Integration Server prepends the values you supply to the already established `Cookie` header value.
- The following headers are considered to be standard and require the specified capitalization: `Accept`, `Authorization`, `Connection`, `Content-Type`, `Cookie`, `Host`, `SOAPAction`, `User-Agent`.
- Using capitalization other than that which is specified results in undefined behavior.
- Supplying duplicate entries for any standard header results in undefined behavior.

Keep the following points in mind when specifying `transportHeaders` for JMS:

- Specify a key in `transportHeaders` for each header field that you want to set, where the key's name represents the name of the header field and the key's value represents the value of that header field.
- You can specify the following JMS message header fields in `transportHeaders`:
  - `JMSCorrelationID`
  - `JMSType`

**Note:** The `JMSCorrelationID` and `JMSType` names are case-sensitive.

- You can specify the following JMS-defined properties in `transportHeaders`:
  - `JMSXGroupID`
  - `JMSXGroupSeq`

If the value of `JMSXGroupSeq` is not an integer, Integration Server ignores the name/value pair and does not place it in the message header.

**Note:** The `JMSXGroupID` and `JMSXGroupSeq` names are case-sensitive.

- The "JMSX" prefix is reserved for JMS-defined properties. If a header whose name starts with "JMSX" is passed into `transportHeaders` and it is not named `JMSXGroupID` or `JMSXGroupSeq`, Integration Server generates a fault and returns it to the service.
- You can set any provider-specific property whose name starts with "JMS\_" in `transportHeaders`. Integration Server maps a supplied name/value pair whose name starts with "JMS\_" directly to a JMS message property. Because the JMS

standard reserves the prefix “JMS\_<vendor\_name>” for provider-specific properties, Integration Server does not validate the name or value of this content.

**Note:** The JMS provider determines which provider-specific properties to accept and include in the JMS message properties. For more information about provider-specific message properties how the JMS provider handles them, review the JMS provider documentation.

- You can use *transportHeaders* to specify run-time properties that affect the values of the JMS message and JMS message headers. The following table identifies these properties and indicates the JMS message header fields affected by each property.

<u>Property Name</u>	<u>Description</u>
<code>.jms.async</code>	Indicates whether this is a synchronous or asynchronous request/reply. This run-time property does not affect a JMS message header field.
	<b>Note:</b> This property applies when <code>pub.client:soapClient</code> calls an operation with an In-Out message exchange pattern only
<u>Value</u>	<u>Description</u>
<code>true</code>	Indicates this is an asynchronous request/reply. Integration Server does not wait for a response message before executing the next step in the flow service.  If <code>.jms.async</code> is <code>true</code> , Integration Server ignores the <i>timeout</i> input parameter.
<code>false</code>	Default. Indicates this is a synchronous request/reply. Integration Server waits for a response before executing the next step in the flow service.
<code>.jms.deliveryMode</code>	Specifies the message delivery mode for the message. Integration Server uses this value to set the <code>JMSDeliveryMode</code> header.
<u>Value</u>	<u>Description</u>

Property Name	Description
	<p>PERSISTENT Indicates the request message is persistent.</p> <p>2 Default. Indicates the request message is persistent.</p> <p>NON_PERSISTENT Indicates the request message is not persistent.</p> <p>1 Indicates the request message is not persistent.</p> <p><b>Note:</b> If the <code>jms.deliveryMode</code> is not one of the above values, Integration Server ignores the name/value pair and uses the default value of 2.</p>
<code>jms.messageType</code>	<p>Message type identifier for the message. Integration Server uses this value to set the JMSType header.</p> <p>Specify one of the following:</p> <ul style="list-style-type: none"> <li>■ BytesMessage</li> <li>■ TextMessage</li> </ul> <p><b>Note:</b> If the <code>jms.messageType</code> value is not BytesMessage or TextMessage, Integration Server ignores the name/value pair and uses the default value of BytesMessage</p>
<code>jms.timeToLive</code>	<p>Length of time, in milliseconds, that the JMS provider retains the message. A value of 0 means that the message does not expire.</p> <p>The JMS provider uses this value to set the JMSExpiration header in the sent JMS message.</p> <p><b>Note:</b> If the <code>jms.timeToLive</code> value is not a valid Long, Integration Server ignores the property and uses the default value of 0.</p>
<code>jms.priority</code>	<p>Specifies the message priority. The JMS standard defines priority levels from 0 to 9, with 0 as the lowest priority and 9 as the highest.</p> <p>Integration Server uses this value to set the JMSPriority header.</p>

Property Name	Description
	If the <code>jms.priority</code> value is not a value between 0 to 9, Integration Server ignores the property and uses the default value of 4.

- The lowercase “jms.” prefix is reserved for run-time properties used by Integration Server. If a header starts with “jms.” and is not one of the “jms.” properties defined by Integration Server, Integration Server ignores the property.

The *header* information returned when `pub.client:soapClient` executes an operation in a web service descriptor created on Integration Server version 8.2 or later varies depending on the following:

- The transport used to send the SOAP message which is determined by the *wsdBinderName*
- The success or failure of the `pub.client:soapClient` service and if failure occurs, the point at which that happens
- The message exchange pattern for the operation specified in *wsdOperationName*

**Note:** The same conditions that affect the contents of *header* also determine whether the *soapResponseData* contains a SOAP response, a SOAP fault, or an exception.

The following table identifies the basic success and failure scenarios when `pub.client:soapClient` service executes an operation in a web service descriptor created on Integration Server version 8.2 or later and the *header* information that would be returned in each scenario. The table also indicates whether the scenario results in a SOAP response, SOAP fault, or exception being returned in *soapResponseData*.

**Note:** JMS status codes as well as the status code 900 are specific to Integration Server and are not derived from any standard.

## Use Case

The `pub.client:soapClient` service fails before sending the SOAP request.

Parameter	Description
<i>status</i>	900
<i>statusMessage</i>	Error occurred while preparing SOAP request

<i>requestLines</i> returned?	Yes, if the web service connector created the SOAP request successfully but execution failed before sending the request.
<i>lines</i> returned?	No
<i>soapResponseData</i>	Contains an exception.

### Use Case

---

The `pub.client:soapClient` service fails while sending the SOAP request.

Parameter	Description
<i>status</i>	For HTTP, the status code will be the value returned by the HTTP server. For JMS, the status code will be 400.
<i>statusMessage</i>	For HTTP, the status message will be the message returned by the HTTP server. For JMS, the status message will be: Error occurred while sending request to JMS provider.
<i>requestLines</i> returned?	Yes
<i>lines</i> returned?	For HTTP, <i>lines</i> may be returned. For example, when the provider returns a status code in the 300 range or 400 range, it is possible that the provider populated response headers. For JMS, <i>lines</i> will not be returned.
<i>soapResponseData</i>	Contains an exception.

### Use Case

---

The `pub.client:soapClient` service fails while sending the SOAP request because a timeout occurs.

Parameter	Description
<i>status</i>	408

<i>statusMessage</i>	Timeout
<i>requestLines</i> returned?	Yes
<i>lines</i> returned?	No
<i>soapResponseData</i>	Contains an exception.

### Use Case

---

The `pub.client:soapClient` service executes successfully.

Parameter	Description
<i>status</i>	<p>For HTTP, the status code will be the value returned by the HTTP server. The status code will typically be in the 200 range.</p> <p>For JMS and an In-Out or Robust In-Only operation, the status code will be 200.</p> <p>For JMS and an In-Only operation, the status code will be 202.</p>
<i>statusMessage</i>	<p>For HTTP, the status message will be the message returned by the HTTP server.</p> <p>For JMS, the status message will be: OK</p>
<i>requestLines</i> returned?	Yes
<i>lines</i> returned?	<p>Depends on the message exchange pattern (MEP) of the operation.</p> <p>For In-Only and Robust In-Only, <i>lines</i> is not returned.</p> <p>For In-Out, <i>lines</i> is returned.</p>
<i>soapResponseData</i>	Contains the SOAP response for In-Out MEP only.

### Use Case

---

The `pub.client:soapClient` service executes successfully but the JMS provider is not available, causing Integration Server to write the JMS message to the client side queue.

**Note:** This use case applies to JMS only. It occurs only when the client side queue is enabled for the JMS binder specified in *wsdBinderName*.

Parameter	Description
<i>status</i>	300
<i>statusMessage</i>	Message written to the client side queue.
<i>requestLines</i> returned?	Yes
<i>lines</i> returned?	No.
<i>soapResponseData</i>	Not returned.

#### Use Case

The `pub.client:soapClient` service sends the SOAP request successfully but receives a SOAP fault from the web service provider.

Parameter	Description
<i>status</i>	For HTTP, the status code will be the value returned by the HTTP server. The status code will typically be in the 500 range. For JMS, the status code will be 500.
<i>statusMessage</i>	For HTTP, the status message will be the message returned by the HTTP server. For JMS, the status message will be: SOAP Fault
<i>requestLines</i> returned?	Yes
<i>lines</i> returned?	No
<i>soapResponseData</i>	Contains an SOAP fault.

#### Use Case

The `pub.client:soapClient` service fails while processing the SOAP response.

Parameter	Description
-----------	-------------

<i>status</i>	900
<i>statusMessage</i>	Error occurred while processing SOAP request
<i>requestLines</i> returned?	Yes
<i>lines</i> returned?	Yes
<i>soapResponseData</i>	Contains an exception.

When invoking `pub.client:soapClient` to execute an In-Out operation *asynchronously* using SOAP over JMS, keep the following information in mind:

- For an asynchronous request/reply, you must pass `jms.async= true` into the `transportHeaders` input parameter.
- To instruct Integration Server to write the request message for an asynchronous request/reply to the client side queue when the JMS provider is not available, the JMS binder must be configured to use the client side queue. Specifically, in the consumer web service descriptor, the **Use CSQ** property for the JMS binder must be set to true.
- When `pub.client:soapClient` sends an asynchronous request it executes to completion without populating any response headers for the `lines` output parameter.
- Even though `pub.client:soapClient` does not wait for a SOAP response, it will execute the response handlers assigned to the consumer web service descriptor. However, the `messageContext` that is available to handler services will not contain a response message. Handler services that do not operate on the response message and instead perform activities such as clean up following a request handler invocation might still provide value for an asynchronous request/reply.
- If you want to retrieve the SOAP response from the provider, you need to receive and process the response with a custom solution. This might include using standard JMS trigger or an on-demand message consumer to receive the message and then using the `pub.soap*` services to process the SOAP message.

**Note:** Using a JMS trigger or message consumer to receive the response bypasses any response handlers or policies applied to the SOAP response, including any `WS-SecurityPolicy`. The SOAP response does not undergo any processing provided by the response handlers or policies attached to the consumer web service descriptor. Any response messages that require decryption or authentication will not be usable. Consequently, do not use an asynchronous request/reply to invoke an In-Out operation to which the `WS-SecurityPolicy` is applied.

When using `pub.client:soapClient` to execute a Robust In-only operation using SOAP over JMS, keep the following information in mind:

- For a consumer web service descriptor, Integration Server provides partial support for Robust In-Only operations with a SOAP over JMS binding. When Integration Server creates a consumer web service descriptor from a WSDL that contains a Robust In-Only operation and that operation is defined as part of a portType with a SOAP over JMS binding, Integration Server populates the reply destination in the JMS message header (the `JMSReplyTo` header field) but otherwise treats the operation as In-Only.

Specifically, `pub.client:soapClient` will not produce or wait for any output besides the `transportInfo` parameter. If an exception occurs while the provider processes the request, the web service connector does not retrieve or process the SOAP response.

- If you want to retrieve a SOAP response (which includes the SOAP fault) that the provider sends when an exception occurs during web service execution, you need to receive and process the response with a custom solution. This might include using a standard JMS trigger or an on-demand message consumer to receive the message and using the `pub.soap*` services to process the SOAP message.

**Note:** Using a JMS trigger or message consumer to receive the response bypasses any policies applied to the SOAP response and any response handlers assigned to the consumer web service descriptor. The SOAP response does not undergo any processing provided by the response handlers or policies attached to the consumer web service descriptor. Any response messages that require decryption or authentication will not be usable. Consequently, do not use an asynchronous request/reply to invoke an In-Out operation to which the `WS-SecurityPolicy` is applied.

### See Also

[pub.client:soapHTTP](#)  
[pub.client:soapRPC](#)

---

## pub.client:soapHTTP

WmPublic. *Deprecated* - There is no replacement service.

Submits a SOAP message to a server via HTTP or HTTPS.

### Input Parameters

---

*soapRequestData*      **Object** SOAP message that is to be sent. This object must be produced with the services in the soap folder. See the Usage Notes for more information.

*address*      **String** URL to which you want the SOAP message sent. For example:

```
https://servername:5555/soap/default
```

<i>auth</i>	<p><b>Document</b> Optional. Parameters specifying the credentials that are to be submitted to the server specified in <i>address</i> . Each element is defined as follows:</p> <table border="1"> <thead> <tr> <th style="text-align: left;"><b>Key</b></th> <th style="text-align: left;"><b>Description</b></th> </tr> </thead> <tbody> <tr> <td><i>type</i></td> <td><b>String</b> Type of authentication that the service will perform. Leave this field blank, as the only option currently available is basic HTTP authentication.</td> </tr> <tr> <td><i>user</i></td> <td><b>String</b> User name that this service will use if one is requested.</td> </tr> <tr> <td><i>pass</i></td> <td><b>String</b> Password that this service will submit if one is requested.</td> </tr> </tbody> </table>	<b>Key</b>	<b>Description</b>	<i>type</i>	<b>String</b> Type of authentication that the service will perform. Leave this field blank, as the only option currently available is basic HTTP authentication.	<i>user</i>	<b>String</b> User name that this service will use if one is requested.	<i>pass</i>	<b>String</b> Password that this service will submit if one is requested.
<b>Key</b>	<b>Description</b>								
<i>type</i>	<b>String</b> Type of authentication that the service will perform. Leave this field blank, as the only option currently available is basic HTTP authentication.								
<i>user</i>	<b>String</b> User name that this service will use if one is requested.								
<i>pass</i>	<b>String</b> Password that this service will submit if one is requested.								
<i>validateSOAP</i>	<p><b>String</b> Optional. Indicates whether or not the response message is to be validated against the SOAP schema. Set to:</p> <ul style="list-style-type: none"> <li>■ <code>true</code> to validate the response message and throw an exception if the response does not conform to the SOAP schema.</li> <li>■ <code>false</code> to bypass the validation process. This is the default.</li> </ul>								
<i>SOAPAction</i>	<p><b>String</b> Optional. Value to which you want to set the SOAPAction HTTP header.</p>								
<i>contentType</i>	<p><b>String</b> Optional. Specifies the value of Content-Type in the HTTP header. Set to:</p> <ul style="list-style-type: none"> <li>■ <code>text/xml; charset="utf-8"</code> to specify the content type as XML and the character encoding of the message text as UTF-8. This is the default.</li> <li>■ <code>text/xml</code> to specify the content type as XML. Since the <code>charset</code> parameter is not specified, the character encoding of the message text defaults to US-ASCII.</li> </ul>								
<i>loadAs</i>	<p><b>String</b> Optional. Specifies the format of the <code>soapResponseData</code>. Default value is <code>stream</code> for an HTTP service and <code>byteArrayStream</code> for an HTTPS service. Set to:</p> <ul style="list-style-type: none"> <li>■ <code>stream</code> to return the body of the response as a <code>java.io.InputStream</code>. Use this option when you will invoke an HTTP web service. This is the default for an HTTP service.</li> </ul>								

- `bytes` to return the body of the response as a `byte[]`. Use this option if the body will be used as input to a service that operates on whole HTML or XML documents (for example, [pub.xml:queryXMLNode](#)).
- `byteArrayStream` to have the response stream fully read and converted to `java.io.ByteArrayStream`. This prevents data loss or a truncated SOAP response if the connection closes prematurely. Use this option when you will invoke an HTTPS web service. This is the default for an HTTPS service.

*timeout* **String** Optional. Time (measured in milliseconds) to wait for a response from the remote server before timing out and terminating the request. The default value is to wait forever.

*encoding* **String** Default character set for encoding SOAP message. Specify an IANA-registered character set (for example, ISO-8859-1). The default is UTF-8.

The *encoding* parameter is used to override the encoding determined by the `watt.server.netEncoding` server configuration parameter. For more information about `watt.server.netEncoding`, see *webMethods Integration Server Administrator's Guide*.

## Output Parameters

---

*soapResponseData* **Object** The SOAP response message returned by the server specified in *address*.

*header* **Document** Conditional. Headers from the HTTP response. Will contain the following keys:

Key	Description
<i>lines</i>	<b>Document</b> Header fields from the HTTP response. Each key in <i>lines</i> represents a field (line) of the response header. Key names represent the names of header fields. The keys' values are Strings containing the values of the fields.
<i>status</i>	<b>String</b> Status code from the HTTP response.

<i>statusMessage</i>	<b>String</b> Status message from the HTTP response.
<i>soapStatus</i>	<b>String</b> Flag indicating whether the SOAP request message was processed successfully.
<u>A value of...</u>	<u>Indicates that...</u>
0	The remote server successfully processed the SOAP request and returned a SOAP response message.
1	The remote server returned a SOAP fault, indicating that the SOAP request was received but was not processed successfully.
2	The server returned an error that was not a SOAP fault. This indicates that some type of HTTP error occurred (often, an HTTP 404). You can check the <i>status</i> element in <i>header</i> to determine the type of HTTP error that occurred.

### Usage Notes

This service is deprecated. There is not a replacement service.

If *address* begins with `https:`, you can use [pub.security.keystore:setKeyAndChain](#) to specify the certificate chain. If you do not specify a certificate chain, `pub.client:soapHTTP` uses the default outbound SSL certificate settings to authenticate the resources.

To send a SOAP message with this service, you must first generate an empty SOAP object with the [pub.soap.utils:createSoapData](#) service and then populate it using services such as [pub.soap.utils:addHeaderEntry](#) and [pub.soap.utils:addBodyEntry](#).

### See Also

[pub.client:soapRPC](#)

### Examples

`sample.soap:buildMsg_sendHTTP`

---

## pub.client:soapRPC

WmPublic. *Deprecated* - There is no replacement service.

Submits a SOAP remote procedure call via HTTP or HTTPS.

## Input Parameters

*address* **String** String specifying the numeric address or name of the server on which the remote procedure resides. If you are submitting the request to an Integration Server, remember to direct it to the RPC processor as shown in the following example:

```
http://rubicon:5555/soap/rpc
```

*reqParms* **Document** The input parameters that are to be passed to the remote procedure. For example, if you wanted to pass three String parameters, *acct*, *amt*, and *org*, containing the values *Cash*, *150.00*, and *Sales*, *reqParms* would contain the following:

<u>Key</u>	Value
<i>acct</i>	Cash
<i>amt</i>	150.00
<i>org</i>	Sales

*method* **Document** The QName of the requested procedure where:

<u>Key</u>	<u>Value</u>
<i>namespaceName</i>	<b>String</b> Namespace portion of the procedure's QName.
<i>localName</i>	<b>String</b> Local portion of the procedure's QName.

*auth* **Document** Optional. User name and password that are to be submitted to the server specified in *address*.

<u>Key</u>	<u>Value</u>
<i>type</i>	<b>String</b> Type of authentication that the service will perform. Leave this field blank, as the only option currently available is basic HTTP authentication.

---

<i>user</i>	<b>String</b> User name that this service will use if one is requested.
<i>pass</i>	<b>String</b> Password that this service will submit if one is requested.
<i>targetInputSignature</i>	<b>String</b> Optional. Fully qualified name of the IS document type to use to validate and encode the contents of <i>reqParams</i> .
<i>targetOutputSignature</i>	<b>String</b> Optional. Fully qualified name of the IS document type to use to validate and decode the output value returned by the remote procedure.
<i>SOAPAction</i>	<b>String</b> Optional. Value to which you want to set the SOAPAction HTTP header.
<i>contentType</i>	<b>String</b> Optional. Specifies the value of Content-Type in the HTTP header. Set to: <ul style="list-style-type: none"><li>■ <code>text/xml; charset="utf-8"</code> to specify the content type as XML and the character encoding of the text as UTF-8. This is the default.</li><li>■ <code>text/xml</code> to specify the content type as XML. Since the <code>charset</code> parameter is not specified, the character encoding of the text defaults to US-ASCII.</li></ul>
<i>encoding</i>	<b>String</b> Optional. Specifies the encoding method. Default value is UTF-8.
<i>loadAs</i>	<b>String</b> Optional. Specifies the format of the <code>soapResponseData</code> . Default value is <code>stream</code> . Set to: <ul style="list-style-type: none"><li>■ <code>stream</code> to return the body of the response as a <code>java.io.InputStream</code>. Use this option when you will invoke an HTTP web service. This is the default.</li><li>■ <code>byteArrayStream</code> to have the response stream fully read and converted to <code>java.io.ByteArrayStream</code>. This prevents data loss or a truncated SOAP response if the connection closes prematurely. Use this option when you will invoke an HTTPS web service.</li></ul>
<i>timeout</i>	<b>String</b> Optional. Time (measured in milliseconds) to wait for a response from the server hosting the remote procedure before timing out and terminating the request. The default value is to wait forever.

## Output Parameters

---

*soapResponseData*      **Object** A SOAP object containing the SOAP response message returned by the server specified in *address* .

*respParms*      **Document** Output parameters returned by the remote procedure. For example, if the remote procedure returned two String parameters, *status* and *balance* , containing the values `closed` and `-4.95`, *respParms* would contain the following:

Key	Value
<i>status</i>	<code>closed</code>
<i>balance</i>	<code>-4.95</code>

*header*      **Document** Conditional. Headers from the HTTP response. Will contain the following keys:

Key	Value
<i>lines</i>	<b>Document</b> Header fields from the HTTP response. Each key in <i>lines</i> represents a field (line) of the response header. Key names represent the names of header fields. The keys' values are Strings containing the values of the fields.
<i>status</i>	<b>String</b> Status code from the HTTP response.
<i>statusMessage</i>	<b>String</b> Status message from the HTTP response.

*soapStatus*      **String** Flag indicating whether the SOAP request message was processed successfully.

A value of...	Indicates that...
0	The remote server successfully processed the SOAP request and returned a SOAP response message.

- 1 The remote server returned a SOAP fault, indicating that the SOAP request was received but was not processed successfully.
- 2 The server returned an error that was not a SOAP fault. This indicates that some type of HTTP error occurred (often, an HTTP 404). You can check the *status* field in *header* to determine the type of HTTP error that occurred.

### Usage Notes

This service is deprecated. There is not a replacement service.

To disable output validation for `pub.client:soapRPC`, set the `watt.server.soap.validateResponse` server configuration parameter to `false`. For more information about `watt.server.soap.validateResponse`, see *webMethods Integration Server Administrator's Guide*.

If *address* begins with `https:`, you can use [pub.security.keystore:setKeyAndChain](#) to specify the certificate chain. If you do not specify a certificate chain, `pub.client:soapRPC` uses the default outbound SSL certificate settings to authenticate the resources.

### See Also

[pub.client:soapHTTP](#)

### Examples

`sample.soap:buildRPC_SendHTTPSimple`



---

# 5 Date Folder

---

■ Pattern String Symbols .....	190
■ Time Zones .....	191
■ Examples .....	193
■ Notes on Invalid Dates .....	194
■ Summary of Elements in this Folder .....	194

You can use the elements in the date folder to generate and format date values.

## Pattern String Symbols

Many of the date services require you to specify pattern strings describing the data's current format and/or the format to which you want it converted. For services that require a pattern string, use the symbols in the following table to describe the format of your data. For example, to describe a date in the January 15, 1999 format, you would use the pattern string `MMMM dd, yyyy`. To describe the format `01/15/99`, you would use the pattern string `MM/dd/yy`. For more information about these pattern string symbols, see the Oracle Java API documentation for the `SimpleDateFormat` class.

Symbol	Meaning	Presentation	Example
G	era designator	Text	AD
y	year	Number	1996 or 96
M	month in year	Text or Number	July or Jul or 07
d	day in month	Number	10
h	hour in am/pm (1-12)	Number	12
H	hour in day (0-23)	Number	0
m	minute in hour	Number	30
s	second in minute	Number	55
S	millisecond	Number	978
E	day in week	Text	Tuesday or Tue
D	day in year	Number	189
F	day of week in month	Number	2 (2nd Wed in July)
w	week in year	Number	27
W	week in month	Number	2

Symbol	Meaning	Presentation	Example
a	am/pm marker	Text	PM
k	hour in day (1-24)	Number	24
K	hour in am/pm (0-11)	Number	0
z	time zone	Text	Pacific Standard Time or PST or GMT-08:00
Z	RFC 822 time zone (JVM 1.4 or later)	Number	-0800 (offset from GMT/UT)
'	escape for text	Delimiter	
''	single quote	Literal	'

## Time Zones

When working with date services, you can specify time zones. The Earth is divided into 24 standard time zones, one for every 15 degrees of longitude. Using the time zone including Greenwich, England (known as Greenwich Mean Time, or GMT) as the starting point, the time is increased by an hour for each time zone east of Greenwich and decreases by an hour for each time zone west of Greenwich. The time difference between a time zone and the time zone including Greenwich, England (GMT) is referred to as the *raw offset*.

The following table identifies the different time zones for the Earth and the raw offset for each zone from Greenwich, England. The effects of daylight savings time are ignored in this table.

**Note:** Greenwich Mean Time (GMT) is also known as Universal Time (UT).

ID	Raw Offset	Name
MIT	-11	Midway Islands Time
HST	-10	Hawaii Standard Time
AST	-9	Alaska Standard Time

---

<b>ID</b>	<b>Raw Offset</b>	<b>Name</b>
PST	-8	Pacific Standard Time
PNT	-7	Phoenix Standard Time
MST	-7	Mountain Standard Time
CST	-6	Central Standard Time
EST	-5	Eastern Standard Time
IET	-5	Indiana Eastern Standard Time
PRT	-4	Puerto Rico and U.S. Virgin Islands Time
CNT	-3.5	Canada Newfoundland Time
AGT	-3	Argentina Standard Time
BET	-3	Brazil Eastern Time
GMT	0	Greenwich Mean Time
ECT	+1	European Central Time
CAT	+2	Central Africa Time
EET	+2	Eastern European Time
ART	+2	(Arabic) Egypt Standard Time
EAT	+3	Eastern African Time
MET	+3.5	Middle East Time
NET	+4	Near East Time
PLT	+5	Pakistan Lahore Time
IST	+5.5	India Standard Time
BST	+6	Bangladesh Standard Time

ID	Raw Offset	Name
VST	+7	Vietnam Standard Time
CTT	+8	China Taiwan Time
JST	+9	Japan Standard Time
ACT	+9.5	Australian Central Time
AET	+10	Australian Eastern Time
SST	+11	Solomon Standard Time
NST	+12	New Zealand Standard Time

## Examples

You can specify *timezone* input parameters in the following formats:

- As a full name. For example:

`Asia/Tokyo`      `America/Los_Angeles`

You can use the `java.util.TimeZone.getAvailableIDs()` method to obtain a list of the valid full name time zone IDs that your JVM version supports.

- As a custom time zone ID, in the format `GMT[+ | -]hh[[:]mm]`. For example:

`GMT+2:00`      All time zones 2 hours east of Greenwich (that is, Central Africa Time, Eastern European Time, and Egypt Standard Time)

`GMT-3:00`      All time zones 3 hours west of Greenwich (that is, Argentina Standard Time and Brazil Eastern Time)

`GMT+9:30`      All time zones 9.5 hours east of Greenwich (that is, Australian Central Time)

- As a three-letter abbreviation from the table above. For example:

`PST`      Pacific Standard Time

**Note:** Because some three-letter abbreviations can represent multiple time zones (for example, "CST" could represent both U.S. "Central Standard Time" and "China Standard Time"), all abbreviations are deprecated. Use the full name or custom time zone ID formats instead.

## Notes on Invalid Dates

The dates you use with a date service must adhere to the `java.text.SimpleDateFormat` class.

If you use an invalid date with a date service, the date service automatically translates the date to a legal date. For example, if you specify `1999/02/30` as input, the date service interprets the date as `1999/03/02` (two days after `2/28/1999`).

If you use `00` for the month or day, the date service interprets `00` as the last month or day in the Gregorian calendar. For example, if you specify `00` for the month, the date service interprets it as `12`.

If the pattern `yy` is used for the year, the date service uses a 50-year moving window to interpret the value of `yy`. The date service establishes the window by subtracting 49 years from the current year and adding 50 years to the current year. For example, if you are running the webMethods Integration Server in the year 2000, the moving window would be from 1951 to 2050. The date service interprets 2-digit years as falling into this window (for example, `12` would be 2012, `95` would be 1995).

## Summary of Elements in this Folder

The following elements are available in this folder:

Element	Package and Description
<a href="#">pub.date:calculateDateDifference</a>	WmPublic. Calculates the difference between two dates and returns the result as seconds, minutes, hours, and days.
<a href="#">pub.date:compareDates</a>	WmPublic. Compares two dates and returns the result as an integer.
<a href="#">pub.date:currentNanoTime</a>	WmPublic. Returns the current time returned by the most precise system timer, in nanoseconds.

Element	Package and Description
<code>pub.date:dateBuild</code>	WmPublic. Builds a date String using the specified pattern and the specified date elements.
<code>pub.date:dateTimeBuild</code>	WmPublic. Builds a date/time string using the specified pattern and the specified date elements.
<code>pub.date:dateTimeFormat</code>	WmPublic. Converts date/time (represented as a String) string from one format to another.
<code>pub.date:elapsedNanoTime</code>	WmPublic. Calculates the time elapsed between the current time and the given time, in nanoseconds.
<code>pub.date:formatDate</code>	WmPublic. Formats a Date object as a string.
<code>pub.date:getCurrentDate</code>	WmPublic. Returns the current date as a Date object.
<code>pub.date:getCurrentDateString</code>	WmPublic. Returns the current date as a String in a specified format.
<code>pub.date:getWorkingDays</code>	WmPublic. Returns the number of working days between two dates.
<code>pub.date:incrementDate</code>	WmPublic. Increments a date by a specified amount of time.

## `pub.date:calculateDateDifference`

WmPublic. Calculates the difference between two dates and returns the result as seconds, minutes, hours, and days.

### Input Parameters

<code>startDate</code>	<b>String</b> Starting date and time.
<code>endDate</code>	<b>String</b> Ending date and time.

---

<i>startDatePattern</i>	<b>String</b> Format in which the <i>startDate</i> parameter is to be specified (for example, yyyyMMdd HH:mm:ss.SSS). For pattern-string notation, see <a href="#">"Pattern String Symbols" on page 190</a> .
<i>endDatePattern</i>	<b>String</b> Format in which the <i>endDate</i> parameter is to be specified (for example, yyyyMMdd HH:mm:ss.SSS). For pattern-string notation, see <a href="#">"Pattern String Symbols" on page 190</a> .

### Output Parameters

---

<i>dateDifferenceSeconds</i>	<b>String</b> The difference between the <i>startingDateTime</i> and <i>endingDateTime</i> , truncated to the nearest whole number of seconds.
<i>dateDifferenceMinutes</i>	<b>String</b> The difference between the <i>startingDateTime</i> and <i>endingDateTime</i> , truncated to the nearest whole number of minutes.
<i>dateDifferenceHours</i>	<b>String</b> The difference between the <i>startingDateTime</i> and <i>endingDateTime</i> , truncated to the nearest whole number of hours.
<i>dateDifferenceDays</i>	<b>String</b> The difference between the <i>startingDateTime</i> and <i>endingDateTime</i> , truncated to the nearest whole number of days.

### Usage Notes

Each output value represents the same date difference, but in a different scale. Do not add these values together. Make sure your subsequent flow steps use the correct output, depending on the scale required.

---

## pub.date:compareDates

WmPublic. Compares two dates and returns the result as an integer.

### Input Parameters

---

<i>startDate</i>	<b>String</b> Starting date to compare against <i>endDate</i> .
<i>endDate</i>	<b>String</b> Ending date to compare against <i>startDate</i> .

*startDatePattern* **String** Format in which the *startDate* parameter is specified (for example, yyyyMMdd HH:mm:ss.SSS). For pattern-string notation, see ["Pattern String Symbols" on page 190](#).

*endDatePattern* **String** Format in which the *endDate* parameter is specified (for example, yyyyMMdd HH:mm:ss.SSS). For pattern-string notation, see ["Pattern String Symbols" on page 190](#).

## Output Parameters

*result* **String** Checks whether *startDate* is before, the same, or after *endDate*.

<u>A value of...</u>	<u>Indicates that...</u>
+1	The <i>startDate</i> is after the <i>endDate</i> .
0	The <i>startDate</i> is the same as the <i>endDate</i> .
-1	The <i>startDate</i> is before the <i>endDate</i> .

## Usage Notes

If the formats specified in the *startDatePattern* and *endDatePattern* parameters are different, Integration Server takes the units that are not specified in the *startDate* and *endDate* values as 0.

That is, if the *startDatePattern* is yyyyMMdd HH:mm and the *startDate* is 20151030 11:11 and if the *endDatePattern* is yyyyMMdd HH:mm:ss.SSS and the *endDate* is 20151030 11:11:55:111, then the `pub.date:compareDates` service considers start date to be before the end date and will return the result as -1.

To calculate the difference between two dates, use the [pub.date:calculateDateDifference](#) service.

## pub.date:currentNanoTime

WmPublic. Returns the current time returned by the most precise system timer, in nanoseconds.

---

## Input Parameters

None.

## Output Parameters

*nanoTime*      **java.lang.Long** Current time returned by the most precise system timer, in nanoseconds.

---

## pub.date:dateBuild

WmPublic. Builds a date String using the specified pattern and the specified date elements.

---

## Input Parameters

*pattern*      **String** Pattern representing the format in which you want the date returned. For pattern-string notation, see "[Pattern String Symbols on page 190](#)". If you do not specify *pattern*, `dateBuild` returns null. If *pattern* contains a time zone and *timezone* is not specified, the default time zone of webMethods Integration Server is used.

*year*      **String** Optional. The year expressed in *yyyy* or *yy* format (for example, 01 or 2001). If you do not specify *year* or you specify an invalid value, `dateBuild` uses the current year.

*month*      **String** Optional. The month expressed as a number (for example, 1 for January, 2 for February). If you do not specify *month* or you specify an invalid value, `dateBuild` uses the current month.

*dayofmonth*      **String** Optional. The day of the month expressed as a number (for example, 1 for the first day of the month, 2 for the second day of the month). If you do not specify *dayofmonth* or you specify an invalid value, `dateBuild` uses the current day.

*timezone*      **String** Optional. Time zone in which you want the output date and time expressed. Specify a time zone code as shown in "[Time Zones on page 191](#)" (for example, EST for Eastern Standard Time).  
  
If you do not specify *timezone*, the value of the server's "user timezone" property is used. If this property has not been set, GMT is used.

*locale* **String** Optional. Locale in which the date is to be expressed. For example, if *locale* is `en` (for English), the pattern `EEE d MMM YYYY` will produce `Friday 23 August 2002`, and the *locale* of `fr` (for French) will produce `vendredi 23 août 2002`.

If *locale* is not specified, Integration Server uses the locale from the session used by the client that invoked the service.

---

## Output Parameters

*value* **String** The date specified by *year*, *month*, and *dayofmonth*, in the format of *pattern*.

## Usage Notes

The `pub.date:dateBuild` service creates a date String by using the supplied *year*, *month*, and *dayofmonth* to first create a local `java.util.Date` object. The local `java.util.Date` object uses the timezone of the machine running Integration Server. The `pub.date:dateBuild` service then applies the offset between the local timezone and the specified *timezone*. Finally, the service uses *pattern* to convert the `java.util.Date` object to the String returned in *value*. Due to the way in which the `pub.date:dateBuild` service works, if you run the `pub.date:dateBuild` service with the same set of inputs on two different machines, each located in a different timezone, you will get different results.

---

# pub.date:dateTimeBuild

WmPublic. Builds a date/time string using the specified pattern and the specified date elements.

---

## Input Parameters

*pattern* **String** Pattern representing the format in which you want the time returned. For pattern-string notation, see ["Pattern String Symbols" on page 190](#). If you do not specify *pattern*, `dateTimeBuild` returns null. If *pattern* contains a time zone and the *timezone* parameter is not set, the time zone of Integration Server is used.

*year* **String** Optional. The year expressed in `yyyy` or `yy` format (for example, `01` or `2001`). If you do not specify *year* or you specify an invalid value, `dateTimeBuild` uses the current year.

*month* **String** Optional. The month expressed as a number (for example, `1` for January, `2` for February). If you do not specify *month* or you specify an invalid value, `dateTimeBuild` uses the current month.

---

<i>dayofmonth</i>	<b>String</b> Optional. The day of the month expressed as a number (for example, 1 for the first day of the month, 2 for the second day of the month). If you do not specify <i>dayofmonth</i> or you specify an invalid value, <code>dateTimeBuild</code> uses the current day.
<i>hour</i>	<b>String</b> Optional. The hour expressed as a number based on a 24-hour clock. For example, specify 0 for midnight, 2 for 2:00 A.M., and 14 for 2:00 P.M. If you do not specify <i>hour</i> or you specify an invalid value, <code>dateTimeBuild</code> uses 0 as the <i>hour</i> value.
<i>minute</i>	<b>String</b> Optional. Minutes expressed as a number. If you do not specify <i>minute</i> or you specify an invalid value, <code>dateTimeBuild</code> uses 0 as the <i>minute</i> value.
<i>second</i>	<b>String</b> Optional. Seconds expressed as a number. If you do not specify <i>second</i> or you specify an invalid value, <code>dateTimeBuild</code> uses 0 as the <i>second</i> value.
<i>millis</i>	<b>String</b> Optional. Milliseconds expressed as a number. If you do not specify <i>millis</i> or you specify an invalid value, <code>dateTimeBuild</code> uses 0 as the <i>millis</i> value.
<i>timezone</i>	<b>String</b> Optional. Time zone in which you want the output date and time expressed. Specify a time zone code as shown in <a href="#">"Time Zones" on page 191</a> (for example, <code>EST</code> for Eastern Standard Time).  If you do not specify <i>timezone</i> , the value of the server's "user timezone" property is used. If this property has not been set, GMT is used.
<i>locale</i>	<b>String</b> Optional. Locale in which the date is to be expressed. For example, if <i>locale</i> is <code>en</code> (for English), the pattern <code>EEE d MMM YYYY</code> will produce <code>Friday 23 August 2002</code> , and the <i>locale</i> of <code>fr</code> (for French) will produce <code>vendredi 23 août 2002</code> .  If <i>locale</i> is not specified, Integration Server uses the locale from the session used by the client that invoked the service.

## Output Parameters

---

*value*            **String** Date and time in format of *pattern* .

## Usage Notes

The `pub.date:dateTimeBuild` service creates a date String by using the supplied *year*, *month*, *dayofmonth*, *hour*, *minute*, *second*, and *millis* to first create a local `java.util.Date` object.

The local `java.util.Date` object uses the timezone of the machine running Integration Server. The `pub.date:dateTimeBuild` service then applies the offset between the local timezone and the specified *timezone*. Finally, the service uses *pattern* to convert the `java.util.Date` object to the String returned in *value*. Due to the way in which the `pub.date:dateTimeBuild` service works, if you run the `pub.date:dateTimeBuild` service with the same set of inputs on two different machines, each located in a different timezone, you will get different results.

---

## pub.date:dateTimeFormat

WmPublic. Converts date/time (represented as a String) string from one format to another.

### Input Parameters

---

<i>inString</i>	<p><b>String</b> Date/time that you want to convert.</p> <div style="background-color: #f0f0f0; padding: 10px; border: 1px solid #ccc;"> <p><b>Important:</b> If <i>inString</i> contains a character in the last position, that character is interpreted as 0. This can result in an inaccurate date. For information about invalid dates, see <a href="#">"Notes on Invalid Dates" on page 194</a>.</p> </div>
<i>currentPattern</i>	<p><b>String</b> Pattern string that describes the format of <i>inString</i>. For pattern-string notation, see <a href="#">"Pattern String Symbols" on page 190</a>.</p>
<i>newPattern</i>	<p><b>String</b> Pattern string that describes the format in which you want <i>inString</i> returned. For pattern-string syntax, see <a href="#">"Pattern String Symbols" on page 190</a>.</p>
<i>locale</i>	<p><b>String</b> Optional. Locale in which the date is to be expressed. For example, if <i>locale</i> is <code>en</code> (for English), the pattern <code>EEE d MMM yyyy</code> will produce <code>Friday 23 August 2002</code>, and the <i>locale</i> of <code>fr</code> (for French) will produce <code>vendredi 23 août 2002</code>.</p> <p>If <i>locale</i> is not specified, Integration Server uses the locale from the session used by the client that invoked the service.</p>
<i>lenient</i>	<p><b>String</b> Optional. A flag indicating whether Integration Server throws an exception if the <i>inString</i> value does not adhere to the format specified in <i>currentPattern</i> parameter. Set to:</p> <ul style="list-style-type: none"> <li>■ <code>true</code> to perform a lenient check. This is the default.</li> </ul> <p>In a lenient check, if the format of the date specified in the <i>inString</i> parameter does not match the format specified in the <i>currentPattern</i> parameter, Integration Server interprets and</p>

returns the date in the format specified in the *currentPattern* parameter. If the interpretation is incorrect, the service will return an invalid date.

- `false` to perform a strict check.

In a strict check, the Integration Server throws an exception if the format of the date specified in the *inString* parameter does not match the format specified in the *currentPattern* parameter.

## Output Parameters

---

*value*                      **String** The date/time given by *inString*, in the format of *newPattern*.

## Usage Notes

As described in "[Notes on Invalid Dates](#)" on page 194, if the pattern *yy* is used for the year, `dateTimeFormat` uses a 50-year moving window to interpret the value of the year. If you need to change this behavior so that the year is interpreted as 80 years before or 20 years after the current date (as described in the Java class `java.text.SimpleDateFormat`), set the server configuration parameter `watt.server.pubDateTimeFormat.javaSlidingWindow` to `true`. For information about setting configuration parameters, see *webMethods Integration Server Administrator's Guide*.

By default, the Integration Server throws an exception if no input is passed to the service. To suppress the error message and return a null value for the *value* parameter, set the server configuration parameter `watt.server.date.suppressPatternError` to `true`. For information about setting configuration parameters, see *webMethods Integration Server Administrator's Guide*.

If *currentPattern* does not contain a time zone, the value is assumed to be in the time zone of the webMethods Integration Server.

If *newPattern* contains a time zone, the time zone of the webMethods Integration Server is used.

## pub.date:elapsedNanoTime

`WmPublic`. Calculates the time elapsed between the current time and the given time, in nanoseconds.

## Input Parameters

---

*nanoTime*                      **java.lang.Long** Time in nanoseconds. If *nanoTime* is less than zero, then the service treats it as zero.

---

## Output Parameters

---

<i>elapsedNanoTime</i>	<b>java.lang.Long</b> The difference between the current time in nanoseconds and <i>nanoTime</i> . If <i>nanoTime</i> is greater than the current nano time, the service returns zero.
<i>elapsedNanoTimeStr</i>	<p><b>String</b> The difference between the current time in nanoseconds and <i>nanoTime</i> . The difference is expressed as a String, in this format:</p> <p>[years] [days] [hours] [minutes] [seconds] [millisec] [microsec] &lt;nanosec&gt;</p> <p>If <i>nanoTime</i> is greater than the current nano time, the service returns zero.</p>

---

## pub.date:formatDate

WmPublic. Formats a Date object as a string.

### Input Parameters

---

<i>date</i>	<b>java.util.Date</b> Optional. Date/time that you want to convert.
<i>pattern</i>	<b>String</b> Pattern string that describes the format in which you want the date returned. For pattern-string notation, see " <a href="#">Pattern String Symbols</a> " on page 190.
<i>timezone</i>	<p><b>String</b> Optional. Time zone in which you want the output date and time expressed. Specify a time zone code as shown in "<a href="#">Time Zones</a>" on page 191 (for example, EST for Eastern Standard Time).</p> <p>If you do not specify <i>timezone</i> , the value of the server's "user timezone" property is used. If this property has not been set, GMT is used.</p>
<i>locale</i>	<p><b>String</b> Optional. Locale in which the date is to be expressed. For example, if <i>locale</i> is en (for English), the pattern <code>EEE d MMM yyyy</code> will produce <code>Friday 23 August 2002</code>, and the <i>locale</i> of fr (for French) will produce <code>vendredi 23 août 2002</code>.</p> <p>If <i>locale</i> is not specified, Integration Server uses the locale from the session used by the client that invoked the service.</p>

---

## Output Parameters

---

*value*            **String** The date/time given by *date* in the format specified by *pattern*.

---

## pub.date:getCurrentDate

WmPublic. Returns the current date as a Date object.

---

## Input Parameters

---

None.

---

## Output Parameters

---

*date*            **java.util.Date** Current date.

---

## pub.date:getCurrentDateString

WmPublic. Returns the current date as a String in a specified format.

---

## Input Parameters

---

*pattern*            **String** Pattern representing the format in which you want the date returned. For pattern-string notation, see "[Pattern String Symbols](#)" on page 190.

*timezone*            **String** Optional. Time zone in which you want the output date and time expressed. Specify a time zone code as shown in "[Time Zones](#)" on page 191 (for example, EST for Eastern Standard Time).

If you do not specify *timezone*, the value of the server's "user timezone" property is used. If this property has not been set, GMT is used.

*locale*            **String** Optional. Locale in which the date is to be expressed. For example, if *locale* is en (for English), the pattern `EEE d MMM yyyy` will produce `Friday 23 August 2002`, and the *locale* of fr (for French) will produce `vendredi 23 août 2002`.

If *locale* is not specified, Integration Server uses the locale from the session used by the client that invoked the service.

---

## Output Parameters

---

*value*                    **String** Current date in the format specified by *pattern* .

---

## pub.date:getWorkingDays

WmPublic. Returns the number of working days between two dates.

The number of working days returned includes the *startDate*, but excludes the *endDate*.

---

## Input Parameters

---

*startDate*                **String** Starting date and time.

*endDate*                    **String** Ending date and time.

*startDatePattern*        **String** Format in which the *startDate* parameter is to be specified (for example, yyyyMMdd HH:mm:ss.SSS). For pattern-string notation, see "[Pattern String Symbols](#)" on page 190.

*endDatePattern*         **String** Format in which the *endDate* parameter is to be specified (for example, yyyyMMdd HH:mm:ss.SSS). For pattern-string notation, see "[Pattern String Symbols](#)" on page 190.

---

## Output Parameters

---

*workingDays*              **String** Number of days between *startDate* and *endDate* excluding weekends and holidays.

## Usage Notes

The `pub.date:getWorkingDays` service requires you to configure holidays and weekend days in the `holidays.cnf` configuration file in `Integration Server_directory\instances\instance_name\packages\WmPublic\config` directory. The `pub.date:getWorkingDays` service uses this configuration file to find the number of working days between two dates.

**Note:** If you make any changes to the `holidays.cnf`, you must reload the `WmPublic` package or restart Integration Server for the changes to take effect.

### Parameter Settings for holidays.cnf file

The following table gives the parameter settings for the holidays.cnf file:

Parameter	Description
holiday.format	The date format in which holidays are to be specified. The default format is MM/dd/yyyy.
holiday.<number>	<p>Holidays in a year. No default values are set for the holiday parameter in the holidays.cnf file.</p> <p>For example, if July 4, 2009 and December 25, 2009 are holidays, then it can be specified as:</p> <pre>holiday.1=07/04/2009 holiday.2=12/25/2009</pre>
weekend.<number>	<p>Weekends days. Valid values are sunday, monday, tuesday, wednesday, thursday, friday, and saturday. No default values are set for the weekend parameter in the holidays.cnf file.</p> <p>If Sunday and Saturday are weekends, then it can be specified as:</p> <pre>weekend.1 weekend.2=saturday</pre>

**Note:** If invalid date or weekend is specified in the configuration file, the `pub.date:getWorkingDays` service will throw errors on execution.

## pub.date:incrementDate

WmPublic. Increments a date by a specified amount of time.

### Input Parameters

*startDate* **String** Starting date and time.

*startDatePattern* **String** Format in which the *startDate* parameter is specified (for example, yyyyMMdd HH:mm:ss.SSS). For pattern-string notation, see ["Pattern String Symbols" on page 190](#).

*endDatePattern* **String** Optional. Pattern representing the format in which you want the *endDate* to be returned. For pattern-string notation, see ["Pattern String Symbols" on page 190](#).

If no *endDatePattern* is specified, the *endDate* will be returned in the format specified in the *startDatePattern* parameter.

- addYears* **String** Optional. Number of years to add to *startDate*. The value must be an integer between -2147483648 and 2147483647.
- addMonths* **String** Optional. Number of months to add to *startDate*. The value must be an integer between -2147483648 and 2147483647.
- addDays* **String** Optional. Number of days to add to *startDate*. The value must be an integer between -2147483648 and 2147483647.
- addHours* **String** Optional. Number of hours to add to *startDate*. The value must be an integer between -2147483648 and 2147483647.
- addMinutes* **String** Optional. Number of minutes to add to *startDate*. The value must be an integer between -2147483648 and 2147483647.
- addSeconds* **String** Optional. Number of seconds to add to *startDate*. The value must be an integer between -2147483648 and 2147483647.
- addMilliseconds* **String** Optional. Number of milliseconds to add to *startDate*. The value must be an integer between -2147483648 and 2147483647.
- timezone* **String** Optional. Time zone in which you want the *endDate* to be expressed. Specify a time zone code as shown in ["Time Zones" on page 191](#) (for example, EST for Eastern Standard Time).  
If you do not specify *timezone*, the value of the server's "user timezone" property is used. If this property has not been set, GMT is used.
- locale* **String** Optional. Locale in which the *endDate* is to be expressed. For example, if *locale* is en (for English), the pattern EEE d MMM YYYY will produce Friday 23 August 2002, and the *locale* of fr (for French) will produce vendredi 23 août 2002.  
If *locale* is not specified, Integration Server uses the locale from the session used by the client that invoked the service.

## Output Parameters

---

- endDate* **String** The end date and time, calculated by incrementing the *startDate* with the specified years, months, days, hours, minutes, seconds, and/

or milliseconds. The *endDate* will be in the *endDatePattern* format, if specified.

If no *endDatePattern* is specified or if blank spaces are specified as the value, the *endDate* will be returned in the format specified in the *startDatePattern* parameter.

### Usage Notes

The *addYears*, *addMonths*, *addDays*, *addHours*, *addMinutes*, *addSeconds*, and *addMilliseconds* input parameters can take positive or negative values. For example, If *startDate* is 10/10/2001, *startDatePattern* is MM/dd/yyyy, *addYears* is 1, and *addMonths* is -1, *endDate* will be 09/10/2002.

If you specify only the *startDate*, *startDatePattern*, and *endDatePattern* input parameters and do not specify any of the optional input parameters to increment the period, the `pub.date:incrementDate` service just converts the format of *startDate* from *startDatePattern* to *endDatePattern* and returns it as *endDate*.

Integration Server issues an exception if the format of the date specified in the *startDate* parameter does not match the format specified in the *startDatePattern*. Integration Server issues a similar exception if the format of the date specified in the *endDate* parameter does not match the *endDatePattern* format.

# 6 Db Folder

---

- Summary of Elements in this Folder ..... 210

You use the elements in the db folder to access JDBC-enabled databases.

**Note:** The webMethods Adapter for JDBC also provides services that perform operations against JDBC-enabled databases. See the *webMethods Adapter for JDBC Installation and User's Guide* for information.

## Summary of Elements in this Folder

The following elements are available in this folder:

Element	Package and Description
<a href="#">pub.db:call</a>	WmDB. Invokes a stored procedure on a target database. As an alternative to this service, consider using the services provided with the webMethods Adapter for JDBC.
<a href="#">pub.db:clearTransaction</a>	WmDB. Clears the transactional state within a database connection. As an alternative to this service, consider using the services provided with the webMethods Adapter for JDBC.
<a href="#">pub.db:close</a>	WmDB. Closes a specified database connection. As an alternative to this service, consider using the services provided with the webMethods Adapter for JDBC.
<a href="#">pub.db:closeAll</a>	WmDB. Closes all database connections that the session has opened. As an alternative to this service, consider using the services provided with the webMethods Adapter for JDBC.
<a href="#">pub.db:commit</a>	WmDB. Commits changes to a database. As an alternative to this service, consider using the services provided with the webMethods Adapter for JDBC.
<a href="#">pub.db:connect</a>	WmDB. Creates a connection to the database using the supplied JDBC URL, user name, and password. As an alternative to this service, consider using the services provided with the webMethods Adapter for JDBC.
<a href="#">pub.db:delete</a>	WmDB. Removes all rows in the specified table that meet the given criteria. As an alternative to this service, consider using the services provided with the webMethods Adapter for JDBC.

Element	Package and Description
<a href="#">pub.db:execSQL</a>	WmDB. Executes the specified SQL statement. As an alternative to this service, consider using the services provided with the webMethods Adapter for JDBC.
<a href="#">pub.db:getProclInfo</a>	WmDB. Retrieves information about one or more stored procedures. As an alternative to this service, consider using the services provided with the webMethods Adapter for JDBC.
<a href="#">pub.db:getProcs</a>	WmDB. Retrieves the names of stored procedures for the specified database. As an alternative to this service, consider using the services provided with the webMethods Adapter for JDBC.
<a href="#">pub.db:getTableInfo</a>	WmDB. Retrieves information about columns in the specified table. As an alternative to this service, consider using the services provided with the webMethods Adapter for JDBC.
<a href="#">pub.db:getTables</a>	WmDB. Retrieves the names of tables in the specified database and schema. As an alternative to this service, consider using the services provided with the webMethods Adapter for JDBC.
<a href="#">pub.db:insert</a>	WmDB. Inserts one or more rows into the specified table. As an alternative to this service, consider using the services provided with the webMethods Adapter for JDBC.
<a href="#">pub.db:query</a>	WmDB. Retrieves all rows from the specified table that meet the given criteria. As an alternative to this service, consider using the services provided with the webMethods Adapter for JDBC.
<a href="#">pub.db:rollback</a>	WmDB. Discards changes to a database. As an alternative to this service, consider using the services provided with the webMethods Adapter for JDBC.
<a href="#">pub.db:startTransaction</a>	WmDB. Begins a transaction on a database connection. As an alternative to this service, consider using the services provided with the webMethods Adapter for JDBC.
<a href="#">pub.db:update</a>	WmDB. Updates all rows in a table that meet the given criteria. Rows are updated with the supplied new data. As

Element	Package and Description
	an alternative to this service, consider using the services provided with the webMethods Adapter for JDBC.

## pub.db:call

WmDB. Invokes a stored procedure on a target database. As an alternative to this service, consider using the services provided with the webMethods Adapter for JDBC.

### Input Parameters

You may specify the connection parameters in one of the following ways:

- *\$dbAlias*
- *\$dbURL, \$dbUser, \$dbPass, \$dbDriver*
- *\$dbConnection*

<i>\$dbAlias</i>	<b>String</b> Optional. Alias of the database on which you want to execute the stored procedure.
<i>\$dbURL</i>	<b>String</b> Optional. JDBC URL that identifies the database resource.
<i>\$dbUser</i>	<b>String</b> Optional. User name to use to log into the database.
<i>\$dbPass</i>	<b>String</b> Optional. Password for the user specified in <i>\$dbUser</i> .
<i>\$dbDriver</i>	<b>String</b> Optional. Name of the JDBC driver to use.
<i>\$dbConnection</i>	<b>com.wm.app.b2b.server.DBConnection</b> Optional. Connection object returned by <a href="#">pub.db:connect</a> .
<i>\$dbCatalog</i>	<p><b>String</b> Optional. Name of the database's system catalog. Include this parameter if your DBMS supports distributed databases and you want to invoke a stored procedure from a database other than the one to which you are connected.</p> <p>If you are not using a distributed database system, you do not need to specify this parameter.</p> <p>If you are running against DB2, use this parameter to specify the stored procedure's location.</p>

*\$dbSchemaPattern* **String** Optional. Name of the schema to which the stored procedure belongs.

If your database supports pattern-matching on schemas, you may specify the schema name with a pattern-matching string, where `_` represents a single character and `%` represents any string of characters. For example, the value of `HR%` would represent any schema beginning with characters HR.

If you are running against DB2, you use this parameter to specify the stored procedure's AuthID.

*\$dbProc* **String** The name of the stored procedure you want to invoke.

*\$dbProcSig* **Document List** Optional. Set of parameters containing information about the stored procedure you want to invoke.

Key	Description
-----	-------------

<i>name</i>	<b>String</b> Parameter name defined in the stored procedure.
-------------	---

<i>sqlType</i>	<b>String</b> Type of procedure parameter for <i>name</i> as defined in the database. Set to one of the following values:
----------------	---

BIT	TINYINT
SMALLINT	INTEGER
BIGINT	FLOAT
REAL	DOUBLE
NUMERIC	DECIMAL
CHAR	VARCHAR
LONGVARCHAR	DATE
TIME	TIMESTAMP
BINARY	VARBINARY
LONGVARBINARY	NULL

<i>direction</i>	<b>String</b> Way in which the parameter is used by the stored procedure. Set to one of the following values:
------------------	---

in
out
inout
return value

*\$dbParamsByOrder* **String** Optional. Indicates whether the contents of *\$data* should be sent to the database in order. Set to:

- `true` to send the contents of *\$data* to the database in the order they are listed in *\$data*.

- `false` to send the contents of `$data` to the database in no particular order. This is the default.

*\$data*

**Document** Parameter values for the stored procedure. Whether the `$data` parameter is mandatory or not depends on the `watt.server.jdbc.sp.mandateParams` property. If this property is set to `false`, the `$data` input parameter is optional. If this property is `true`, you must provide values for the `$data` parameter. The default value for the `watt.server.jdbc.sp.mandateParams` property is `true`.

## Output Parameters

---

*\$dbMessage*

**String** Conditional. Message indicating the success or failure of the operation.

## Usage Notes

The output will also contain output parameters and procedure return values (the return value is called `RETURN_VALUE`).

## pub.db:clearTransaction

---

WmDB. Clears the transactional state within a database connection. As an alternative to this service, consider using the services provided with the webMethods Adapter for JDBC.

## Input Parameters

---

You may specify the connection parameters in one of the following ways:

- *\$dbAlias*
- *\$dbURL, \$dbUser, \$dbPass, \$dbDriver*
- *\$dbConnection*

*\$dbAlias*

**String** Optional. Alias of the database connection on which you want to clear the transactional state. The alias is passed automatically if the database is connected.

*\$dbURL*

**String** Optional. JDBC URL that identifies the database resource.

---

<i>\$dbUser</i>	<b>String</b> Optional. User name to use to log into the database.
<i>\$dbPass</i>	<b>String</b> Optional. Password for the user.
<i>\$dbDriver</i>	<b>String</b> Optional. Name of the JDBC driver to use.
<i>\$dbConnection</i>	<b>com.wm.app.b2b.server.DBConnection</b> Optional. Connection object returned by <a href="#">pub.db:connect</a> .

### Output Parameters

---

<i>\$dbMessage</i>	<b>String</b> A message indicating the success or failure of the operation.
--------------------	---

### Usage Notes

On some databases, exceptional conditions within transactions will automatically abort the entire transaction. When this happens, the standard commit/rollback operations are meaningless because there is no current transaction. If this occurs, use the `clearTransaction` service to clear the transactional state and prepare for a new transaction. You should only use this service if you have begun a transaction and cannot end it with a standard commit or rollback.

The `clearTransaction` service *does not* involve a database operation; it is entirely internal to the webMethods Integration Server.

---

## pub.db:close

WmDB. Closes a specified database connection. As an alternative to this service, consider using the services provided with the webMethods Adapter for JDBC.

### Input Parameters

---

You may specify the connection parameters in one of the following ways:

- *\$dbAlias*
- *\$dbURL, \$dbUser, \$dbPass, \$dbDriver*
- *\$dbConnection*

<i>\$dbAlias</i>	<b>String</b> Optional. Database alias.
<i>\$dbURL</i>	<b>String</b> Optional. JDBC URL that identifies the database resource.

---

<i>\$dbUser</i>	<b>String</b> Optional. User name to use to log into the database.
<i>\$dbPass</i>	<b>String</b> Optional. Password for the user.
<i>\$dbDriver</i>	<b>String</b> Optional. Name of the JDBC driver to use.
<i>\$dbConnection</i>	<b>com.wm.app.b2b.server.DBConnection</b> Optional. Connection object returned by <a href="#">pub.db:connect</a> .
<i>\$closeDbConnection</i>	<p><b>String</b> Optional. Indicates whether to remove the database connection from the pool or return it to the pool for future use. Set to:</p> <ul style="list-style-type: none"> <li>■ <code>true</code> to close the connection and remove it from the pool.</li> <li>■ <code>false</code> to close the connection and return it to the pool for future use. This is the default.</li> </ul>

### Output Parameters

---

<i>\$dbMessage</i>	<b>String</b> Message indicating the success or failure of the operation.
--------------------	---

## pub.db:closeAll

---

WmDB. Closes all database connections that the session has opened. As an alternative to this service, consider using the services provided with the webMethods Adapter for JDBC.

### Input Parameters

---

None.

### Output Parameters

---

<i>\$dbMessage</i>	<b>String</b> Message indicating the success or failure of the operation.
--------------------	---

## pub.db:commit

---

WmDB. Commits changes to a database. As an alternative to this service, consider using the services provided with the webMethods Adapter for JDBC.

---

## Input Parameters

---

You may specify the connection parameters in one of the following ways:

- *\$dbAlias*
- *\$dbURL, \$dbUser, \$dbPass, \$dbDriver*
- *\$dbConnection*

<i>\$dbAlias</i>	<b>String</b> Optional. Alias of the database on which you want to commit changes. The alias is passed automatically if the database is connected.
<i>\$dbURL</i>	<b>String</b> Optional. JDBC URL that identifies the database resource.
<i>\$dbUser</i>	<b>String</b> Optional. User name to use to log into the database.
<i>\$dbPass</i>	<b>String</b> Optional. Password for the user.
<i>\$dbDriver</i>	<b>String</b> Optional. Name of the JDBC driver to use.
<i>\$dbConnection</i>	<b>com.wm.app.b2b.server.DBConnection</b> Optional. Connection object returned by <a href="#">pub.db:connect</a> .

---

## Output Parameters

---

<i>\$dbMessage</i>	<b>String</b> Message indicating the success or failure of the operation.
--------------------	---

### Usage Notes

This service returns an exception if an error occurs when committing changes to the database. The most common reason for this error is that no transaction has been started (see "[pub.db:startTransaction](#)" on page 236).

---

## pub.db:connect

WmDB. Creates a connection to the database using the supplied JDBC URL, user name, and password. As an alternative to this service, consider using the services provided with the webMethods Adapter for JDBC.

You can also specify a JDBC driver specific to the database.

## Input Parameters

---

You may specify the connection parameters in one of the following ways:

- *\$dbAlias*
- *\$dbURL, \$dbUser, \$dbPass, \$dbDriver*
- *\$dbURL, \$dbDriver, \$dbProperties*

<i>\$dbAlias</i>	<b>String</b> Optional. Database alias.
<i>\$dbURL</i>	<b>String</b> Optional. JDBC URL that identifies the database resource.
<i>\$dbUser</i>	<b>String</b> Optional. User name to use to log into the database.
<i>\$dbPass</i>	<b>String</b> Optional. Password for the user.
<i>\$dbDriver</i>	<b>String</b> Optional. Name of the JDBC driver to use.
<i>\$dbProperties</i>	<p><b>Document</b> Optional. Set of connection parameters that are to be used to make the database connection. Within <i>\$dbProperties</i>, key names represent the names of the connection parameters that are to be used to establish the connection, and the value of a key specifies the value of that particular parameter.</p> <p>In most cases, you will include the keys <i>user</i> and <i>password</i> in <i>\$dbProperties</i> to specify the user name and password parameters that are to be used to connect to the database. You may include additional parameters as needed.</p> <p>The following example shows how <i>\$dbProperties</i> would look if you wanted to set the <i>weblogic.codeset</i> parameter to GBK in order to extract Unicode data out of the database:</p>

<b>Key</b>	<b>Value</b>
<i>user</i>	dbu
<i>password</i>	dbu
<i>weblogic.codeset</i>	GBK

---

## Output Parameters

---

*\$dbConnection*      **com.wm.app.b2b.server.DBConnection** Connection object.

*\$dbMessage*      **String** Message indicating the success or failure of the operation.

## Usage Notes

Database connections opened by [pub.db:connect](#) are associated with the current session.

Multiple attempts to connect to the same database by the same client will result in the same connection being reused. This means that if client A and client B request connections to the same database, they each get their own new connection. If client A makes another call to [pub.db:connect](#), the previous connection is reused. Associating the database connection with the client session prevents remote clients from having to reconnect repeatedly to a target database.

Connections are not pooled or shared across sessions. Unless explicitly closed (by calling [pub.db:close](#) or [pub.db:closeAll](#)), connections associated with a session are closed when the session is flushed from memory. This happens at a regular interval, which can be configured using the Integration Server Administrator. For more information about setting the session time-out limit, see *webMethods Integration Server Administrator's Guide*.

---

## pub.db:delete

WmDB. Removes all rows in the specified table that meet the given criteria. As an alternative to this service, consider using the services provided with the webMethods Adapter for JDBC.

## Input Parameters

---

You may specify the connection parameters in one of the following ways:

- *\$dbAlias*
- *\$dbURL, \$dbUser, \$dbPass, \$dbDriver*
- *\$dbConnection*

*\$dbAlias*      **String** Optional. Database alias.

*\$dbURL*      **String** Optional. JDBC URL that identifies the database resource.

*\$dbUser*      **String** Optional. User name to use to log into the database.

<i>\$dbPass</i>	<b>String</b> Optional. Password for the user.
<i>\$dbDriver</i>	<b>String</b> Optional. Name of the JDBC driver to use.
<i>\$dbConnection</i>	<b>com.wm.app.b2b.server.DBConnection</b> Optional. Connection object returned by <a href="#">pub.db.connect</a> .
<i>\$dbCatalog</i>	<p><b>String</b> Optional. Name of the database's system catalog. Include this parameter if your DBMS supports distributed databases and you want to delete rows from a table that is not in the database to which you are connected.</p> <p>If you are not using a distributed database system or if you want to delete rows from the database to which you are connected, you do not need to specify this parameter.</p> <p>If you are running against DB2, use this parameter to specify the database location.</p>
<i>\$dbSchemaPattern</i>	<p><b>String</b> Optional. Name of the schema to which the table belongs.</p> <p>If your database supports pattern-matching on schemas, you may specify the schema name with a pattern-matching string, where <code>_</code> represents a single character and <code>%</code> represents any string of characters. For example, the value <code>HR%</code> would represent any schema beginning with the characters <code>HR</code>.</p> <p>If you are running against DB2, you use this parameter to specify the table's AuthID.</p>
<i>\$dbTable</i>	<b>String</b> Name of the table to remove rows from.
<i>\$data</i>	<p><b>Document</b> Optional. Criteria that the rows to delete must meet.</p> <p><b>Important</b> If no criteria are provided, <i>all rows</i> are deleted from the table.</p>

## Output Parameters

---

<i>\$updateCount</i>	<b>String</b> Number of rows deleted.
<i>\$dbMessage</i>	<b>String</b> Conditional. Message indicating the success or failure of the operation.

---

## pub.db:execSQL

WmDB. Executes the specified SQL statement. As an alternative to this service, consider using the services provided with the webMethods Adapter for JDBC.

The service does not perform any parsing on the SQL statement.

### Input Parameters

---

<i>\$dbAlias</i>	<b>String</b> Optional. Database alias.
<i>\$dbURL</i>	<b>String</b> Optional. JDBC URL that identifies the database resource.
<i>\$dbUser</i>	<b>String</b> Optional. User name to use to log into the database.
<i>\$dbPass</i>	<b>String</b> Optional. Password for the user.
<i>\$dbDriver</i>	<b>String</b> Optional. Name of the JDBC driver to use.
<i>\$dbConnection</i>	<b>com.wm.app.b2b.server.DBConnection</b> Optional. Connection object returned by <a href="#">pub.db:connect</a> .
<i>\$dbCatalog</i>	<b>String</b> Optional. Name of the database's system catalog. Include this parameter if your DBMS supports distributed databases and you want to retrieve information from a database to which you are not currently connected.  If you are not using a distributed database system, you do not need to specify this parameter.  If you are running against DB2, use this parameter to specify the database location.
<i>\$dbSchemaPattern</i>	<b>String</b> Optional. Name of the schema to which the table belongs.  If your database supports pattern-matching on schemas, you may specify

the schema name with a pattern-matching string, where `_` represents a single character and `%` represents any string of characters. For example, the value `HR%` would represent any schema beginning with the characters `HR`.

If you are running against DB2, you use this parameter to specify the table's AuthID.

*\$dbSQL*

**String** SQL statement to execute.

*\$dbProcessEsc*

**String** Optional. Flag that indicates whether JDBC SQL escapes will be processed. These escapes allow database-independent access to database-dependent functionality. For example, different dialects of SQL have different syntax for date literals. Using a JDBC escape, you can encode a date literal in a SQL string that should work on any database. Documentation on JDBC SQL escapes is widely available.

Set to:

- `true` to process JDBC SQL escapes. This is the default.
- `false` to skip processing JDBC SQL escapes.

*\$dbProcessReporterTokens*

**String** Optional. Flag that indicates whether reporter tags (for example, `%value xxx%`) will be processed in the SQL. Including these tokens in your SQL allows dynamic construction of complex SQL statements, at the possible expense of some execution speed.

Set to:

- `true` to process tags.
- `false` to ignore tags. This is the default.

*\$dbParamValues*

**Object List** Optional. If the "?" parameters in the SQL statement are not supplied indirectly (with the *\$dbParamNames* parameter), they can be supplied

directly via this parameter. See "[Usage Notes](#)" on page 223 below. Objects in *\$dbParamValues* can be of any type.

*\$dbParamNames*

**String List** Optional. Names of any "?" parameters in the SQL. See "[Usage Notes](#)" on page 223 below.

*\$dbParamTypes*

**String List** Optional. SQL type names for each parameter. Use type names from the JDBC 1.2 specification ("INTEGER", "VARCHAR", etc.).

## Output Parameters

---

*sql*

**String** Conditional. SQL as it was actually passed to the target database. This can be helpful in debugging calls to this service when dynamic SQL is used (that is, you are using either JDBC SQL escapes or webMethods Reporter tokens in your SQL).

*paramsAsStrings*

**String List** Conditional. Values used for each of the parameters in the SQL statement. This can be helpful in debugging calls to this service when "?" parameters are being used.

*\$rowCount*

**String** Conditional. Number of rows in *results*.

*results*

**com.wm.util.Table** Conditional. Results from the SQL statement. The Integration Server recognizes and treats this parameter as a Document List at run time.

*\$updateCount*

**String** Conditional. Number of rows updated.

*\$dbMessage*

**String** Conditional. Message indicating the success or failure of the operation.

## Usage Notes

This service does not support updates from a web browser or HTML form.

You may specify the connection parameters in one of the following ways:

- *\$dbAlias*
- *\$dbURL, \$dbUser, \$dbPass, \$dbDriver*
- *\$dbConnection*

SQL supports host variables ("?") in statements to be executed. Because the pipeline is based on named values and individual host variables are not named, *\$dbParamNames* and *\$dbParamTypes* are used to supply an index-to-name mapping for each SQL statement executed. For example, consider the following SQL query:

```
SELECT * FROM royalties WHERE pub_id = ? and roy_amt > ?
```

To execute this SQL query, you could supply the following values to the [pub.db:execSQL](#) service:

Key	Value	Description
<i>\$dbSQL</i>	SELECT * FROM royalties WHERE pub_id = ? and roy_amt > ?	SQL query to execute.
<i>\$dbParamNames</i>	pub_id roy_amt	Pipeline items to use for the host variables.
<i>\$dbParamTypes</i>	varchar integer	SQL types for the host variables.
<i>pub_id</i>	P1053	Values for the host variables.
<i>roy_amt</i>	10	Values for the host variables.

**Note:** Even if there is only one host variable in the SQL statement, both *\$dbParamNames* and *\$dbParamTypes* are String arrays.

**Example:** Consider the following SQL query, which contains an `INSERT` with three host variables:

```
INSERT INTO books VALUES (?, ?, ?)
```

To execute this SQL query, you could supply the following values to the [pub.db:execSQL](#) service:

Key	Value	Description
<i>\$dbSQL</i>	INSERT INTO books VALUES (?, ?, ?)	SQL query to execute.

Key	Value	Description
<i>\$dbParamNames</i>	book_id pub_id book_title	Pipeline items to use for the host variables.
<i>\$dbParamTypes</i>	varchar varchar varchar	SQL types for the host variables.
<i>book_id</i>	B234	Values for the host variables.
<i>pub_id</i>	P1053	Values for the host variables.
<i>book_title</i>	The Importance of Being Earnest	Values for the host variables.

**Note:** The SQL type names used in the examples are defined in the `java.sql.Types` and `SQL92`. Even if you used an Oracle database, which calls long string types "varchar2," you would call them `varchar`. The standard names from `SQL92` will be mapped into database-specific type names.

## pub.db:getProcInfo

WmDB. Retrieves information about one or more stored procedures. As an alternative to this service, consider using the services provided with the `webMethods Adapter for JDBC`.

### Input Parameters

You may specify the connection parameters in one of the following ways:

- *\$dbAlias*
- *\$dbURL, \$dbUser, \$dbPass, \$dbDriver*
- *\$dbConnection*

*\$dbAlias*                    **String** Optional. Database alias.

*\$dbURL*                    **String** Optional. JDBC URL that identifies the database resource.

---

<i>\$dbUser</i>	<b>String</b> Optional. User name to use to log into the database.
<i>\$dbPass</i>	<b>String</b> Optional. Password for the user.
<i>\$dbDriver</i>	<b>String</b> Optional. Name of the JDBC driver to use.
<i>\$dbConnection</i>	<b>com.wm.app.b2b.server.DBConnection</b> Optional. Connection object returned by <a href="#">pub.db.connect</a> .
<i>\$dbCatalog</i>	<p><b>String</b> Optional. Name of the database's system catalog. Include this parameter if your DBMS supports distributed databases and you want to retrieve information about a stored procedure that is not in the database to which you are currently connected.</p> <p>If you are not using a distributed database system, you do not need to specify this parameter.</p> <p>If you are running against DB2, use this parameter to specify the stored procedure's location.</p>
<i>\$dbSchemaPattern</i>	<p><b>String</b> Optional. Name of the schema to which the table belongs.</p> <p>If your database supports pattern-matching on schemas, you may specify the schema name with a pattern-matching string, where <code>_</code> represents a single character and <code>%</code> represents any string of characters. For example, the value <code>HR%</code> would represent any schema beginning with the characters <code>HR</code>.</p> <p>If you are running against DB2, you use this parameter to specify the stored procedure's AuthID.</p>
<i>\$dbProc</i>	<b>String</b> Name of the procedure about which you want information.

### Output Parameters

---

This service returns one document (IData object) for each item in the stored procedure's signature that matches the specified input criteria. Each document contains information about the signature item. The document's key will be the same as the signature item's name. For a description of what information is supplied by your database, see `java.sql.DatabaseMetaData.getProcedureColumns` in your JDBC documentation.

---

## pub.db:getProcs

WmDB. Retrieves the names of stored procedures for the specified database. As an alternative to this service, consider using the services provided with the webMethods Adapter for JDBC.

### Input Parameters

---

You may specify the connection parameters in one of the following ways:

- *\$dbAlias*
- *\$dbURL, \$dbUser, \$dbPass, \$dbDriver*
- *\$dbConnection*

<i>\$dbAlias</i>	<b>String</b> Optional. Database alias.
<i>\$dbURL</i>	<b>String</b> Optional. JDBC URL that identifies the database resource.
<i>\$dbUser</i>	<b>String</b> Optional. User name to use to log into the database.
<i>\$dbPass</i>	<b>String</b> Optional. Password for the user.
<i>\$dbDriver</i>	<b>String</b> Optional. Name of the JDBC driver to use.
<i>\$dbConnection</i>	<b>com.wm.app.b2b.server.DBConnection</b> Optional. Connection object returned by <a href="#">pub.db:connect</a> .
<i>\$dbCatalog</i>	<p><b>String</b> Optional. Name of the database's system catalog. Include this parameter if your DBMS supports distributed databases and you want to retrieve a list of stored procedures from a database other than the one to which you are connected.</p> <p>If you are not using a distributed database system, you do not need to specify this parameter.</p> <p>If you are running against DB2, use this parameter to specify the database location.</p>
<i>\$dbSchemaPattern</i>	<p><b>String</b> Optional. Name of the schema to which the stored procedures belong.</p> <p>If your database supports pattern-matching on schemas, you may specify the schema name with a pattern-matching</p>

string, where `_` represents a single character and `%` represents any string of characters. For example, the value `HR%` would represent any schema beginning with the characters `HR`.

If you are running against DB2, you use this parameter to specify the database's AuthID.

*\$dbProcNamePattern* **String** Optional. Pattern-matching string that specifies the procedures that you want included in the returned list, where `_` represents a single character and `%` represents any string of characters. For example, the value `DATE%` would represent any procedure beginning with the characters `DATE`.

### Output Parameters

This service returns one document (IData object) for each stored procedure that matches the specified input criteria. Each document contains information about a stored procedure. The document's key will be the same as the stored procedure name. For a description of what information is supplied by your database, see `java.sql.DatabaseMetaData.getProcedures` in your JDBC documentation.

## pub.db:getTableInfo

WmDB. Retrieves information about columns in the specified table. As an alternative to this service, consider using the services provided with the webMethods Adapter for JDBC.

### Input Parameters

You may specify the connection parameters in one of the following ways:

- *\$dbAlias*
- *\$dbURL, \$dbUser, \$dbPass, \$dbDriver*
- *\$dbConnection*

*\$dbAlias* **String** Optional. Database alias.

*\$dbURL* **String** Optional. JDBC URL that identifies the database resource.

*\$dbUser* **String** Optional. User name to use to log into the database.

*\$dbPass* **String** Optional. Password for the user.

---

<i>\$dbDriver</i>	<b>String</b> Optional. Name of the JDBC driver to use.
<i>\$dbConnection</i>	<b>com.wm.app.b2b.server.DBConnection</b> Optional. Connection object returned by <a href="#">pub.db.connect</a> .
<i>\$dbCatalog</i>	<b>String</b> Optional. Name of the database's system catalog. Include this parameter if your DBMS supports distributed databases and you want information about a table that is not in the database to which you are connected.  If you are not using a distributed database system or you want information about a table in the database to which you are connected, you do not need to specify this parameter.  If you are running against DB2, use this parameter to specify the database location.
<i>\$dbSchemaPattern</i>	<b>String</b> Optional. Name of the schema to which the table belongs.  If your database supports pattern-matching on schemas, you may specify a pattern-matching string for the schema name, where <code>_</code> represents a single character and <code>%</code> represents any string of characters. For example, the value <code>HR%</code> would represent any schema beginning with the characters <code>HR</code> .  If you are running against DB2, you use this parameter to specify the table's AuthID.
<i>\$dbTable</i>	<b>String</b> Name of table whose column names you want to retrieve.
<i>\$dbColumnNamePattern</i>	<b>String</b> Optional. Pattern-matching string that specifies the column names that you want to retrieve, where <code>_</code> represents a single character and <code>%</code> represents any string of characters. For example, the value <code>ADDR%</code> would represent any column name beginning with the characters <code>ADDR</code> .

### Output Parameters

---

This service returns one document (IData object) for each column that matches the specified input criteria. Each document contains information about a column. The document's key will be the same as the column name.

## Usage Notes

This service accepts input from a web browser or HTML form.

---

## pub.db:getTables

WmDB. Retrieves the names of tables in the specified database and schema. As an alternative to this service, consider using the services provided with the webMethods Adapter for JDBC.

## Input Parameters

---

You may specify the connection parameters in one of the following ways:

- *\$dbAlias*
- *\$dbURL, \$dbUser, \$dbPass, \$dbDriver*
- *\$dbConnection*

<i>\$dbAlias</i>	<b>String</b> Optional. Database alias.
<i>\$dbURL</i>	<b>String</b> Optional. JDBC URL that identifies the database resource.
<i>\$dbUser</i>	<b>String</b> Optional. User name to use to log into the database.
<i>\$dbPass</i>	<b>String</b> Optional. Password for the user.
<i>\$dbDriver</i>	<b>String</b> Optional. Name of the JDBC driver to use.
<i>\$dbConnection</i>	<b>com.wm.app.b2b.server.DBConnection</b> Optional. Connection object returned by <a href="#">pub.db:connect</a> .
<i>\$dbCatalog</i>	<p><b>String</b> Optional. Name of the database's system catalog. Include this parameter if your DBMS supports distributed databases and you want information from a database that is not the one to which you are connected.</p> <p>If you are not using a distributed database system or you want information about the database to which you are connected, you do not need to specify this parameter.</p> <p>If you are running against DB2, use this parameter to specify the database location.</p>

---

<i>\$dbSchemaPattern</i>	<p><b>String</b> Optional. Name of the schema for which you want the names of tables.</p> <p>If your database supports pattern-matching on schemas, you may specify a pattern-matching string for the schema name, where <code>_</code> represents a single character and <code>%</code> represents any string of characters. For example, the value <code>HR%</code> would represent any schema beginning with the characters HR.</p> <p>If you want the table names from all schemas, set <i>\$dbSchemaPattern</i> to null.</p> <p>If you are running against DB2, you use this parameter to specify the table's AuthID.</p>
<i>\$dbTableNamePattern</i>	<p><b>String</b> Optional. Pattern string describing the tables whose names you want to retrieve.</p> <p>If your database supports pattern-matching on schemas, you may specify a pattern-matching string, where <code>_</code> represents a single character and <code>%</code> represents any string of characters. For example, the value <code>HR%</code> would represent any table name beginning with the characters HR.</p> <p>If you want all table names, set <i>\$dbTableNamePattern</i> to null.</p>
<i>\$dbTableTypeList</i>	<p><b>String List</b> Optional. Set of parameters specifying the types of tables whose names you want to retrieve. Common JDBC table types include: <code>TABLE</code>, <code>VIEW</code>, <code>SYSTEM TABLE</code>, <code>ALIAS</code>, and <code>SYNONYM</code>. Check your driver documentation for others.</p>

### Output Parameters

---

This service returns one document (IData object) for each table that matches the specified input criteria. Each document contains information about a table. The document's key will be the same as the table name.

### Usage Notes

This service accepts input from a web browser or HTML form.

---

## pub.db:insert

WmDB. Inserts one or more rows into the specified table. As an alternative to this service, consider using the services provided with the webMethods Adapter for JDBC.

## Input Parameters

---

You may specify the connection parameters in one of the following ways:

- *\$dbAlias*
- *\$dbURL, \$dbUser, \$dbPass, \$dbDriver*
- *\$dbConnection*

<i>\$dbAlias</i>	<b>String</b> Optional. Database alias.
<i>\$dbURL</i>	<b>String</b> Optional. JDBC URL that identifies the database resource.
<i>\$dbUser</i>	<b>String</b> Optional. User name to use to log into the database.
<i>\$dbPass</i>	<b>String</b> Optional. Password for the user.
<i>\$dbDriver</i>	<b>String</b> Optional. Name of the JDBC driver to use.
<i>\$dbConnection</i>	<b>com.wm.app.b2b.server.DBConnection</b> Optional. Connection object returned by <a href="#">pub.db.connect</a> .
<i>\$dbCatalog</i>	<p><b>String</b> Optional. Name of the database's system catalog. Include this parameter if your DBMS supports distributed databases and you want to insert rows into a table that is not in the database to which you are connected.</p> <p>If you are not using a distributed database system or if you want to insert rows into the database to which you are connected, you do not need to specify this parameter.</p> <p>If you are running against DB2, use this parameter to specify the database location.</p>
<i>\$dbSchemaPattern</i>	<p><b>String</b> Optional. Name of the schema to which the table belongs.</p> <p>If your database supports pattern-matching on schemas, you may specify the schema name with a pattern-matching string, where <code>_</code> represents a single character and <code>%</code> represents any string of characters. For example, the value <code>HR%</code> would represent any schema beginning with the characters <code>HR</code>.</p> <p>If you are running against DB2, you use this parameter to specify the table's AuthID.</p>

---

<i>\$dbTable</i>	<b>String</b> Name of table in which you want to insert rows.
<i>\$dbRollbackOnFail</i>	<b>String</b> Optional. Flag that determine whether changes are committed if a failure occurs while processing multiple inserts. Set to: <ul style="list-style-type: none"> <li>■ <code>true</code> to undo changes on failure.</li> <li>■ <code>false</code> to commit changes on failure. This is the default.</li> </ul>
<i>\$data</i>	<b>Document</b> or <b>Document List</b> Optional. Data to insert.

### Output Parameters

---

<i>\$updateCount</i>	<b>String</b> Number of rows the service inserted.
<i>\$failCount</i>	<b>String</b> Number of rows the service failed to insert.
<i>\$errors</i>	<b>Document</b> Conditional. Error messages generated during service execution.
<i>\$dbMessage</i>	<b>String</b> Conditional. Message indicating the success or failure of the operation.

### Usage Notes

This service accepts input from a web browser or HTML form.

---

## pub.db:query

WmDB. Retrieves all rows from the specified table that meet the given criteria. As an alternative to this service, consider using the services provided with the webMethods Adapter for JDBC.

### Input Parameters

---

You may specify the connection parameters in one of the following ways:

- *\$dbAlias*
- *\$dbURL*, *\$dbUser*, *\$dbPass*, *\$dbDriver*
- *\$dbConnection*

<i>\$dbAlias</i>	<b>String</b> Optional. Database alias.
------------------	---

---

<i>\$dbURL</i>	<b>String</b> Optional. JDBC URL that identifies the database resource.
<i>\$dbUser</i>	<b>String</b> Optional. User name to use to log into the database.
<i>\$dbPass</i>	<b>String</b> Optional. Password for the user.
<i>\$dbDriver</i>	<b>String</b> Optional. Name of the JDBC driver to use.
<i>\$dbConnection</i>	<b>com.wm.app.b2b.server.DBConnection</b> Optional. Connection object returned by <a href="#">pub.db.connect</a> .
<i>\$dbCatalog</i>	<p><b>String</b> Optional. Name of the database's system catalog. Include this parameter if your DBMS supports distributed databases and you want to query a table that is not in the database to which you are connected.</p> <p>If you are not using a distributed database system or if you want to query a table in the database to which you are connected, you do not need to specify this parameter.</p> <p>If you are running against DB2, use this parameter to specify the database location.</p>
<i>\$dbSchemaPattern</i>	<p><b>String</b> Optional. Name of the schema to which the table belongs.</p> <p>If your database supports pattern-matching on schemas, you may specify the schema name with a pattern-matching string, where <code>_</code> represents a single character and <code>%</code> represents any string of characters. For example, the value <code>HR%</code> would represent any schema beginning with the characters HR.</p> <p>If you are running against DB2, you use this parameter to specify the table's AuthID.</p>
<i>\$dbTable</i>	<b>String</b> Name of table to query.
<i>\$data</i>	<b>Document</b> Optional. Criteria that the rows to retrieve must meet.

### Output Parameters

---

<i>results</i>	<b>com.wm.util.Table</b> Conditional. Results of the query. The Integration Server recognizes and treats this parameter as a Document List at run time.
----------------	---

---

<i>\$dbMessage</i>	<b>String</b> Conditional. Message indicating the success or failure of an operation.
<i>\$rowCount</i>	<b>String</b> Conditional. Number of rows for the table that meet the criteria specified in <i>\$data</i> .

### Usage Notes

This service accepts input from a web browser or HTML form.

---

## pub.db:rollback

WmDB. Discards changes to a database. As an alternative to this service, consider using the services provided with the webMethods Adapter for JDBC.

### Input Parameters

---

You may specify the connection parameters in one of the following ways:

- *\$dbAlias*
- *\$dbURL*, *\$dbUser*, *\$dbPass*, *\$dbDriver*
- *\$dbConnection*

<i>\$dbAlias</i>	<b>String</b> Optional. Alias of the database for which you want to discard changes. This information is passed automatically.
<i>\$dbURL</i>	<b>String</b> Optional. JDBC URL that identifies the database resource.
<i>\$dbUser</i>	<b>String</b> Optional. User name to use to log into the database.
<i>\$dbPass</i>	<b>String</b> Optional. Password for the user.
<i>\$dbDriver</i>	<b>String</b> Optional. Name of the JDBC driver to use.
<i>\$dbConnection</i>	<b>com.wm.app.b2b.server.DBConnection</b> Optional. Connection object returned by <a href="#">pub.db:connect</a> .

### Output Parameters

---

<i>\$dbMessage</i>	<b>String</b> Message indicating the success or failure of the operation.
--------------------	---

## Usage Notes

This service throws an exception if an error occurs when discarding changes to the database. The most common reason for this error is that no transaction has been started (see [pub.db:startTransaction](#)).

---

## pub.db:startTransaction

WmDB. Begins a transaction on a database connection. As an alternative to this service, consider using the services provided with the webMethods Adapter for JDBC.

### Input Parameters

---

You may specify the connection parameters in one of the following ways:

- *\$dbAlias*
- *\$dbURL, \$dbUser, \$dbPass, \$dbDriver*
- *\$dbConnection*

<i>\$dbAlias</i>	<b>String</b> Optional. Alias of the database for which you want to start the transaction. This information is passed automatically.
<i>\$dbURL</i>	<b>String</b> Optional. JDBC URL that identifies the database resource.
<i>\$dbUser</i>	<b>String</b> Optional. User name to use to log into the database.
<i>\$dbPass</i>	<b>String</b> Optional. Password for the user.
<i>\$dbDriver</i>	<b>String</b> Optional. Name of the JDBC driver to use.
<i>\$dbConnection</i>	<b>com.wm.app.b2b.server.DBConnection</b> Optional. Connection object returned by <a href="#">pub.db:connect</a> .

### Output Parameters

---

<i>\$dbMessage</i>	<b>String</b> Message indicating the success or failure of the operation.
--------------------	---

## Usage Notes

By default, all database connections are opened in "auto commit" mode, meaning the results of a operation are automatically committed to the database when that

operation succeeds. To use a connection in a transactional context, you must first call [pub.db:startTransaction](#) to take that connection out of "auto commit" mode.

This service returns an exception if an error occurs when starting the new transaction. Common reasons for an error when starting a new transaction are:

- A transaction is already in progress (see [pub.db:commit](#), [pub.db:rollback](#), or [pub.db:clearTransaction](#)).
- The target database does not support transactions.

After a transaction has been started, it must be terminated with a call to either [pub.db:commit](#) (to save all changes to the database) or [pub.db:rollback](#) (to discard changes).

---

## pub.db:update

WmDB. Updates all rows in a table that meet the given criteria. Rows are updated with the supplied new data. As an alternative to this service, consider using the services provided with the webMethods Adapter for JDBC.

### Input Parameters

---

You may specify the connection parameters in one of the following ways:

- *\$dbAlias*
- *\$dbURL, \$dbUser, \$dbPass, \$dbDriver*
- *\$dbConnection*

<i>\$dbAlias</i>	<b>String</b> Optional. Database alias.
<i>\$dbURL</i>	<b>String</b> Optional. JDBC URL that identifies the database resource.
<i>\$dbUser</i>	<b>String</b> Optional. User name to use to log into the database.
<i>\$dbPass</i>	<b>String</b> Optional. Password for the user.
<i>\$dbDriver</i>	<b>String</b> Optional. Name of the JDBC driver to use.
<i>\$dbConnection</i>	<b>com.wm.app.b2b.server.DBConnection</b> Optional. Connection object returned by <a href="#">pub.db:connect</a> .
<i>\$dbCatalog</i>	<b>String</b> Optional. Name of the database's system catalog. Include this parameter if your DBMS supports distributed databases and you want to update rows in a table that is not in the database to which you are connected.

If you are not using a distributed database system or if you want to update rows in the database to which you are connected, you do not need to specify this parameter.

If you are running against DB2, you use this parameter to specify the database location.

*\$dbSchemaPattern*

**String** Optional. Name of the schema to which the table belongs.

If your database supports pattern-matching on schemas, you may specify the schema name with a pattern-matching string, where `_` represents a single character and `%` represents any string of characters. For example, the value `HR%` would represent any schema beginning with the characters HR.

If you are running against DB2, you use this parameter to specify the table's AuthID.

*\$dbTable*

**String** Name of table to update.

*\$criteria*

**Document** Criteria that the rows to update must meet.

**Important** If no criteria are provided, *all rows* are updated.

*\$set*

**Document** New data with which to update rows.

## Output Parameters

---

*\$updateCount*

**String** Number of rows updated.

*\$dbMessage*

**String** Conditional. Message indicating the operation failed.

# 7 Document Folder

---

■ Summary of Elements in this Folder .....	240
--	-----

You use the elements in the document folder to perform operations on documents in the pipeline.

## Summary of Elements in this Folder

The following elements are available in this folder:

Element	Package and Description
<a href="#">pub.document:bytesToDocument</a>	WmPublic. Converts an array of bytes to a document.
<a href="#">pub.document:deleteDocuments</a>	WmPublic. Deletes the specified documents from a set of documents.
<a href="#">pub.document:documentListToDocument</a>	WmPublic. Constructs a document (an IData object) from a document list (an IData[ ]) by generating key/value pairs from the values of two elements that you specify in the document list.
<a href="#">pub.document:documentToBytes</a>	WmPublic. Converts a document to an array of bytes.
<a href="#">pub.document:documentToDocumentList</a>	WmPublic. Expands the contents of a document into a list of documents.
<a href="#">pub.document:documentToXMLValues</a>	WmPublic. Converts a document (IData object) to a String by encoding it in the webMethods XMLValues format.
<a href="#">pub.document:groupDocuments</a>	WmPublic. Groups a set of documents based on specified criteria.
<a href="#">pub.document:insertDocument</a>	WmPublic. Inserts a new document in a set of documents at a specified position.
<a href="#">pub.document:searchDocuments</a>	WmPublic. Searches a set of documents for entries matching a set of Criteria.
<a href="#">pub.document:sortDocuments</a>	WmPublic. Sorts a set of input documents based on the specified sortCriteria.

Element	Package and Description
<a href="#">pub.document:XMLValuesToDocument</a>	WmPublic. Decodes a String containing an XMLValues-encoded document and produces a document (IData object).

## pub.document:bytesToDocument

WmPublic. Converts an array of bytes to a document. This service can only be used with byte arrays created by executing the [pub.document:documentToBytes](#) service.

### Input Parameters

<i>documentBytes</i>	<p><b>Object</b> An array of bytes (byte[]) to convert to a document.</p> <ul style="list-style-type: none"> <li>■ If <i>documentBytes</i> is null, the service does not return a document or an error message.</li> <li>■ If <i>documentBytes</i> is not a byte array, the service throws a service exception.</li> <li>■ If <i>documentBytes</i> is zero-length, the service produces an empty document.</li> </ul>
----------------------	---

### Output Parameters

<i>document</i>	<b>Document</b> A document.
-----------------	-----------------------------

### Usage Notes

Use this service with the [pub.document:documentToBytes](#) service, which converts a document into a byte array. You can pass the resulting byte array to the [pub.document:bytesToDocument](#) service to convert it back into the original document.

In order for the document-to-bytes-to-document conversion to work, the entire content of the document must be serializable. Every object in the document must be of a data type known to Integration Server, or it must support the `java.io.Serializable` interface.

**Note:** If Integration Server encounters an unknown object in the document that does not support the `java.io.Serializable` interface, that object's value will be lost. It will be replaced with a string containing the object's class name.

---

## pub.document:deleteDocuments

WmPublic. Deletes the specified documents from a set of documents.

### Input Parameters

---

<i>documents</i>	<b>Document List</b> Set of documents that contain the documents you want to delete.
<i>indices</i>	<b>String List</b> Index values of documents to be deleted from the <i>documents</i> parameter document list.

### Output Parameters

---

<i>documents</i>	<b>Document List</b> List of documents whose indices do <i>not</i> match the values in <i>indices</i> parameter.
<i>deletedDocuments</i>	<b>Document List</b> List of deleted documents.

### Usage Notes

The pub.document:deleteDocuments service returns an error if the *indices* parameter value is less than zero or more than the number of documents in the *documents* input parameter.

---

## pub.document:documentListToDocument

WmPublic. Constructs a document (an IData object) from a document list (an IData[ ]) by generating key/value pairs from the values of two elements that you specify in the document list.

### Input Parameters

---

<i>documentList</i>	<b>Document List</b> Set of documents (IData[ ]) that you want to transform into a single document (IData object).
---------------------	--

**Note:** If the *documentList* parameter contains a single document instead of a Document List, the documentListToDocument service does nothing.

<i>name</i>	<b>String</b> Name of the element in the <i>documentList</i> parameter whose value provides the name of each key in the resulting document.
-------------	---

**Important!** The data type of the element that you specify in the *name* parameter must be String.

*value* **String** Name of the element in the *documentList* parameter whose values will be assigned to the keys specified in *name* . This element can be of any data type.

## Output Parameters

*document* **Document** Document (IData object) containing the key/value pairs generated from the *documentList* parameter.

## Usage Notes

The following example illustrates how the `documentListToDocument` service would convert a document list that contains three documents to a single document containing three key/value pairs. When you use the `documentListToDocument` service, you specify which two elements from the source list are to be transformed into the keys and values in the output document. In the following example, the values from the *pName* elements in the source list are transformed into key names, and the values from the *pValue* elements are transformed into the values for these keys.

A `documentList` containing these three documents:

Key	Value
<i>pName</i>	<code>cx_timeout</code>

<i>pValue</i>	1000
---------------	------

Key	Value
<i>pName</i>	<code>cx_max</code>

<i>pValue</i>	2500
---------------	------

Key	Value
<i>pName</i>	<code>cx_min</code>

<i>pValue</i>	10
---------------	----

Would be converted to a document containing these three key:

---

Key	Value
<i>cx_timeout</i>	1000
<i>cx_max</i>	2500
<i>cx_min</i>	10

---

## pub.document:documentToBytes

WmPublic. Converts a document to an array of bytes.

### Input Parameters

---

<i>document</i>	<b>Document</b> Document (IData object) to convert to bytes. <ul style="list-style-type: none"><li>■ If <i>document</i> is null, the service does not return an output or an error message.</li><li>■ If <i>document</i> is not a document (IData), the service throws a service exception.</li><li>■ If <i>document</i> contains no elements, the service produces a zero-length byte array.</li></ul>
-----------------	---

### Output Parameters

---

<i>documentBytes</i>	<b>Object</b> A serialized representation of the document as an array of bytes (byte[]).
----------------------	--

### Usage Notes

Use the [pub.document:documentToBytes](#) service with the [pub.document:bytesToDocument](#) service, which converts the byte array created by this service back into the original document.

The [pub.document:documentToBytes](#) service is useful when you want to write a document to a file (using the [pub.file:bytesToFile](#) service), an input stream (using the [pub.io:bytesToStream](#) service), or a cache (using the [pub.cache:put](#) service).

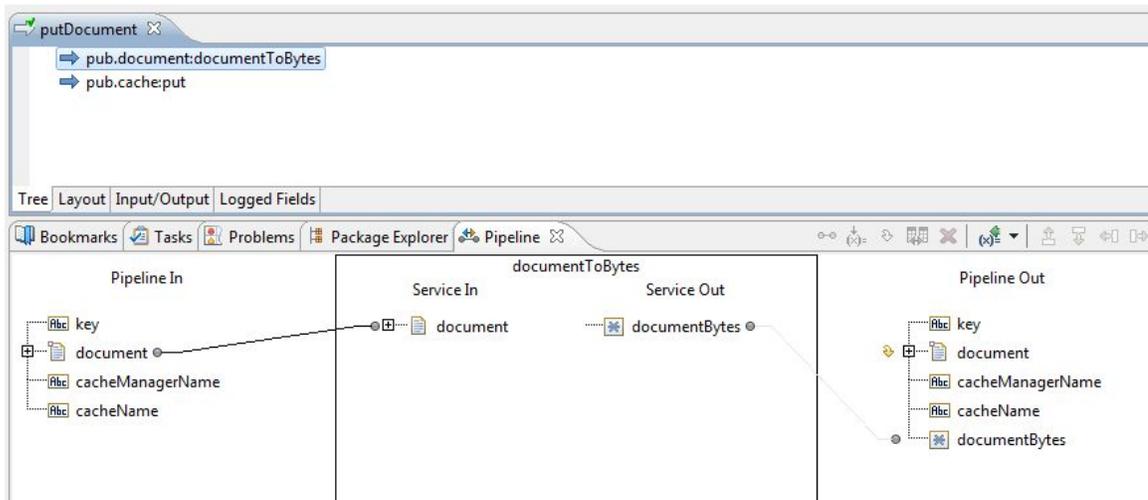
In order for the document-to-bytes-to-document conversion to work, the entire content of the document must be serializable. Every object in the document must be of a data type known to Integration Server, or it must support the `java.io.Serializable` interface. If Integration Server encounters an unknown object in the document that does not support

the `java.io.Serializable` interface, that object's value will be lost. Integration Server will replace it with a string containing the object's class name.

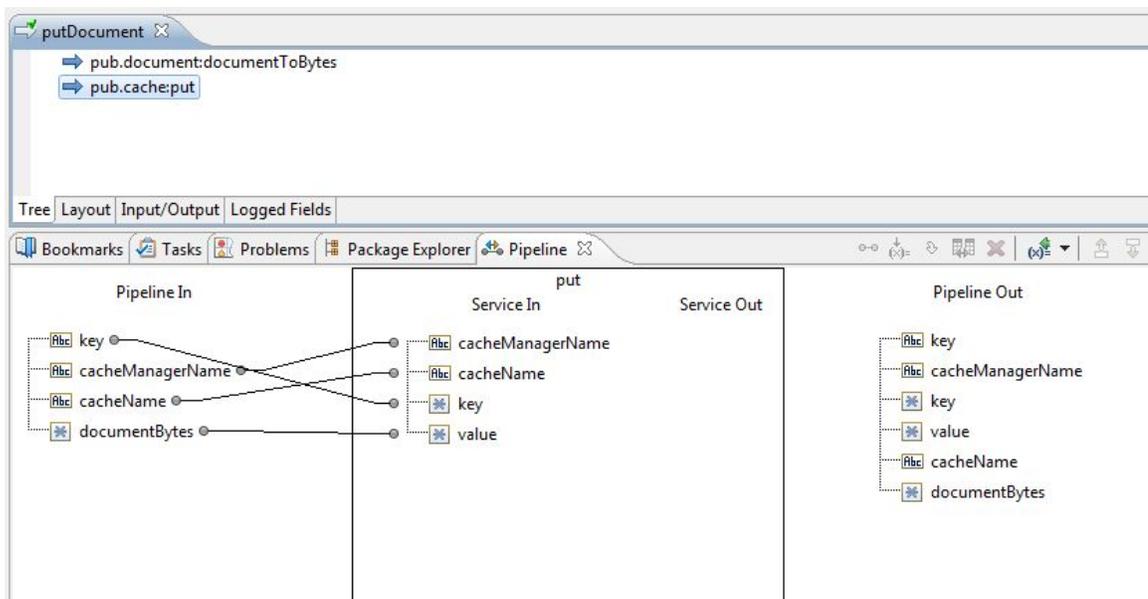
## Example

This example describes how to use the `pub.document:documentToBytes` and `pub.document:bytesToDocument` services to cache a document in the pipeline.

1. A document is not directly serializable, so you must first use the `pub.document:documentToBytes` service. Invoke the service and map the document to the `document` input parameter. This will add the output `documentBytes` to the pipeline.

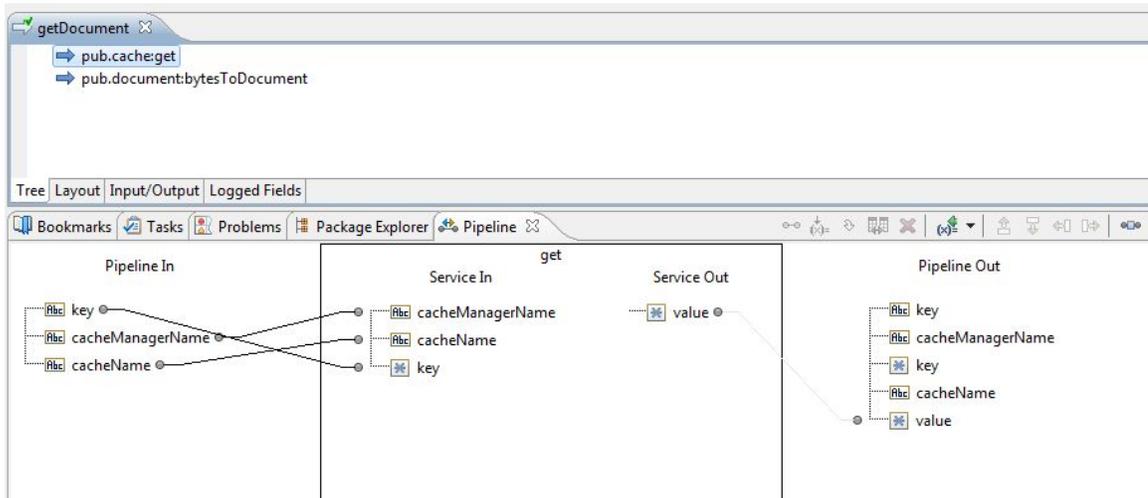


2. Invoke `pub.cache:put` and map `documentBytes` to the `value` input parameter for that service. Set the `key` input parameter to a value that is meaningful to your application.

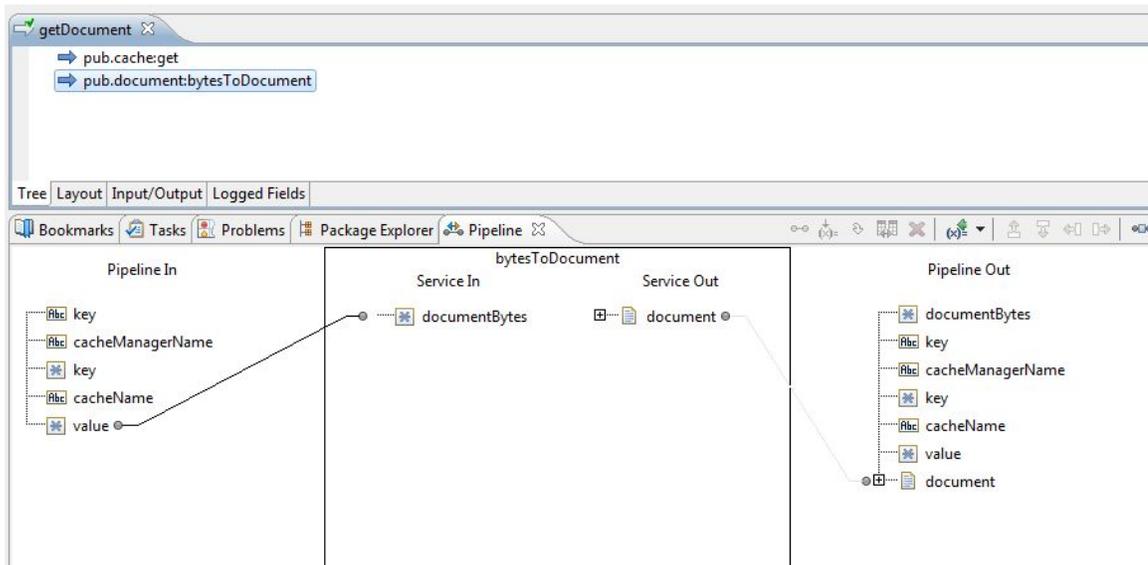


3. At another point in your application, you will need to retrieve the document you cached. You can do so by invoking the `pub.cache:get` service and supplying the same

*key* input parameter as in step 2. The *value* output parameter that was added as output to the pipeline will contain the byte array from step 1.



4. Invoke `pub.document:bytesToDocument` and map the *value* output parameter to the *documentBytes* input parameter. This will add the output *document* to the pipeline, which will match the original document in step 1.



## pub.document:documentToDocumentList

WmPublic. Expands the contents of a document into a list of documents.

Each key/value pair in the source document is transformed to a single document containing two keys (whose names you specify). These two keys will contain the key name and value of the original pair.

---

## Input Parameters

---

<i>document</i>	<b>Document</b> Document (IData object) to transform.
<i>name</i>	<b>String</b> Name to assign to the key that will receive the key name from the original key/value pair. (In the example above, this parameter was set to <i>pName</i> .)
<i>value</i>	<b>String</b> Name to assign to the key that will receive the value from the original key/value pair. (In the example above, this parameter was set to <i>pValue</i> .)

---

## Output Parameters

---

<i>documentList</i>	<b>Document List</b> List containing a document for each key/value pair in the <i>document</i> parameter. Each document in the list will contain two keys, whose names were specified by the <i>name</i> and <i>value</i> parameters. The values of these two keys will be the name and value (respectively) of the original pair.
---------------------	--

## Usage Notes

The following example shows how a document containing three keys would be converted to a document list containing three documents. In this example, the names *pName* and *pValue* are specified as names for the two new keys in the document list.

A document containing these three keys:

<b>Key</b>	<b>Value</b>
<i>cx_timeout</i>	1000
<i>cx_max</i>	2500
<i>cx_min</i>	10

Would be converted to a document list containing these three documents:

<b>Key</b>	<b>Value</b>
<i>pName</i>	<i>cx_timeout</i>
<i>pValue</i>	1000

---

Key	Value
<i>pName</i>	cx_max
<i>pValue</i>	2500

---

Key	Value
<i>pName</i>	cx_min
<i>pValue</i>	10

---

## pub.document:documentToXMLValues

WmPublic. Converts a document (IData object) to a String by encoding it in the webMethods XMLValues format.

### Input Parameters

---

*document*      **Document** Document (IData object) to convert. This document can contain any number of other fields, lists, and other documents.

### Output Parameters

---

*xmlvalues*      **String** String representation of the *document* parameter, encoded in the webMethods XMLValues format.

### Usage Notes

---

To convert the encoded String back into an IData object, use the [pub.document:XMLValuesToDocument](#) service.

---

## pub.document:groupDocuments

WmPublic. Groups a set of documents based on specified criteria.

## Input Parameters

---

<i>documents</i>	<b>Document List</b> Set of documents to be grouped based on the specified criteria.
<i>groupCriteria</i>	<p><b>Document List</b> The criteria on which the input documents are to be grouped. Valid values for the <i>groupCriteria</i> parameter are:</p> <ul style="list-style-type: none"> <li>■ <i>key</i>. Key in the pipeline. The value for <i>key</i> can be a path expression. For example, "Family/Children[0]/BirthDate" retrieves the birthday of the first child from the input Family document list.</li> <li>■ <i>compareStringsAs</i>. Optional. Valid values for <i>compareStringsAs</i> are <code>string</code>, <code>numeric</code>, and <code>datetime</code>. The default value is <code>string</code>.</li> <li>■ <i>pattern</i>. Optional. <i>pattern</i> will be considered only if the <i>compareStringsAs</i> parameter is of type <code>datetime</code>. For information about using patterns, see <a href="#">"Time Zones" on page 191</a>.</li> </ul> <p><b>Note:</b> If <i>key</i> is not found in all the input documents, the documents that do not match the <i>groupCriteria</i> are grouped together as a single group.</p>

## Output Parameters

---

<i>documentGroups</i>	<b>Document List</b> List of documents where each element represents a set of documents grouped based on the criteria specified.
-----------------------	--

## Usage Notes

The following example illustrates how to specify the values for the *groupCriteria* parameter:

<u>key</u>	<u>compareStringsAs</u>	<u>pattern</u>
name	string	
age	numeric	
birthdate	datetime	yyyy-MM-dd

The input documents will be grouped based on name, age, and birth date.

---

## pub.document:insertDocument

WmPublic. Inserts a new document in a set of documents at a specified position.

### Input Parameters

---

<i>documents</i>	<b>Document List</b> Set of documents in which a new document is to be inserted.
<i>insertDocument</i>	<b>Document</b> The new document to be inserted to the set of documents specified in the <i>documents</i> parameter.
<i>index</i>	<b>String</b> Optional. The position in the set which the document is to be inserted.  The <i>index</i> parameter is zero-based. If the value for the <i>index</i> parameter is not specified, the document will be inserted at the end of the document list specified in the <i>documents</i> parameter.

### Output Parameters

---

<i>documents</i>	<b>Document List</b> Document list after inserting the new document.
------------------	--

---

## pub.document:searchDocuments

WmPublic. Searches a set of documents for entries matching a set of Criteria.

### Input Parameters

---

<i>documents</i>	<b>Document List</b> Set of documents from which the documents meeting the search criteria are to be returned.
<i>searchCriteria</i>	<b>Document</b> Criteria on which the documents in the <i>documents</i> parameter are to be searched.  Valid values for <i>searchCriteria</i> parameters are: <ul style="list-style-type: none"><li>■ <i>key</i>. Name of the element in documentList whose value provides the value for the search text. The value for <i>key</i> can be a path expression. For example, "Family/Children[0]/BirthDate" retrieves the birthday of the first child from the input Family document list.</li></ul>

- *value* . Optional. Any search text. If no value is specified, the service searches for null in the document list.
- *compareStringsAs* . Optional. Allowed values are `string`, `numeric`, and `datetime`. The default value is `string`.
- *pattern* . Optional. *pattern* will be considered only if the *compareStringsAs* value is of type `datetime`. For information about using patterns, see ["Time Zones" on page 191](#).

*sorted*

**String** Optional. The value of the *sorted* parameter is `true` if the document list is already sorted based on the search criteria and same search key; otherwise `false`.

If the value for the *sorted* parameter is set to `true`, the required documents are searched faster.

## Output Parameters

---

<i>resultdocuments</i>	<b>Document List</b> List of documents which are matching the search criteria.
<i>documentListIndices</i>	<b>String List</b> Positions of search documents in the document list.
<i>documents</i>	<b>Document List</b> List of documents that were input.

## Usage Note

For example, if you want to search a set of documents for documents where BirthDate is 10th January 2008, the values for the *searchCriteria* parameter would be:

<u>key</u>	<u>value</u>	<u>compareStringsAs</u>	<u>pattern</u>
Birthdate	2008-01-10	datetime	yyyy-MM-dd

## pub.document:sortDocuments

WmPublic. Sorts a set of input documents based on the specified sortCriteria.

## Input Parameters

---

<i>documents</i>	<b>Document List</b> Set of documents that are to be sorted.
------------------	--

*sortCriteria* **Document List** Criteria based on which the documents in the *documents* parameter are to be sorted.

Valid values for *sortCriteria* parameters are:

- *key*. Name of the element in *documentList* whose value provides the value based on which the documents are to be sorted. The value for *key* can be a path expression. For example, "Family/Children[0]/BirthDate" retrieves the birthday of the first child from the input Family document list.
- *order*. Optional. Allowed values are *ascending* and *descending*. The default value is *ascending*.
- *compareStringsAs*. Optional. Allowed values are *string*, *numeric*, and *datetime*. Default value is *string*.
- *pattern*. Optional. The value for *pattern* will be considered only if the *compareStringsAs* value is of type *datetime*. For information about using patterns, see ["Time Zones" on page 191](#).

**Note:** If *key* is not found in all the input documents, the sorted list of documents appears at the end or start of the list based on the *order* specified. If the order is *ascending*, then all the documents that do not match the sort criteria appears at the top of the list, followed by the sorted list. If the order is *descending*, the sorted list will appear at the top, followed by the documents that do not match the sort criteria.

## Output Parameters

*documents* **Document List** The documents sorted based on the sort criteria specified in the *sortCriteria* parameter.

## Usage Notes

For example, if you want to sort a set of documents based on name, age, and then on birth date, the values for *sortCriteria* parameter would be:

<u>key</u>	<u>order</u>	<u>compareStringsAs</u>	<u>pattern</u>
Name	ascending	string	
Age	descending	numeric	
Birthdate	ascending	datetime	yyyy-MM-dd

---

## pub.document:XMLValuesToDocument

WmPublic. Decodes a String containing an XMLValues-encoded document and produces a document (IData object).

### Input Parameters

---

*xmlvalues*      **String** An XMLValues encoding of a document.

**Important:** This String must contain a webMethods XMLValues encoding of a document. No other encoding format is accepted.

### Output Parameters

---

*document*      **Document** Document (IData object) result of the decoding of *xmlvalues* .

### Usage Notes

An XMLValues-encoded document is produced using [pub.document:documentToXMLValues](#).



# 8 Event Folder

---

■ Summary of Elements in this Folder .....	256
--	-----

You use the elements in the event folder to subscribe to events, write event handlers, and work with EDA (Event Driven Architecture) events.

## Summary of Elements in this Folder

The following elements are available in this folder:

Element	Package and Description
<a href="#">pub.event:addSubscriber</a>	WmPublic. Creates a subscription for a specified event.
<a href="#">pub.event:alarm</a>	WmPublic. Specification for alarm event handlers.
<a href="#">pub.event:alarmInfo</a>	WmPublic. Document type for alarm event information.
<a href="#">pub.event:audit</a>	WmPublic. Specification for audit event handlers.
<a href="#">pub.event:auditError</a>	WmPublic. Specification for audit event errors.
<a href="#">pub.event:auditErrorInfo</a>	WmPublic. Document type for audit error event information.
<a href="#">pub.event:auditInfo</a>	WmPublic. Document type for audit event information.
<a href="#">pub.event:callstackItem</a>	WmPublic. Document type for the name of the service in the invocation path when an exception occurred and the index that indicates the step at which the exception occurred.
<a href="#">pub.event:deleteSubscriber</a>	WmPublic. Removes an event handler from the subscription list for a specified event.
<a href="#">pub.event.eda:event</a>	WmPublic. Document type that defines the structure of an event document.

Element	Package and Description
<a href="#">pub.event.eda:eventToDocument</a>	WmPublic. <i>Deprecated</i> There is no replacement service. Converts an EDA event in the form of an XML string to a document (IData object).
<a href="#">pub.event.eda:schema_event</a>	WmPublic. Schema that defines the structure and data types used for the event header in the <code>pub.event.eda:event</code> document type.
<a href="#">pub.event:error</a>	WmPublic. Specification for error event handlers.
<a href="#">pub.event:errorInfo</a>	WmPublic. Document type for error event information.
<a href="#">pub.event:exception</a>	WmPublic. Specification for exception event handlers.
<a href="#">pub.event:exceptionInfo</a>	WmPublic. Document type for exception information.
<a href="#">pub.event:gdEnd</a>	WmPublic. Specification for gdEnd event handlers.
<a href="#">pub.event:gdEndInfo</a>	WmPublic. Document type for gdEnd event information.
<a href="#">pub.event:gdStart</a>	WmPublic. Specification for gdStart event handlers.
<a href="#">pub.event:gdStartInfo</a>	WmPublic. Document type for gdStart event information.
<a href="#">pub.event:getEventTypes</a>	WmPublic. Returns the list of supported event types on Integration Server.
<a href="#">pub.event:getSubscribers</a>	WmPublic. Returns the list of subscribers for a specified event type.
<a href="#">pub.event:jmsReceiveErrorEvent</a>	WmPublic. Specification for a JMS retrieval failure event handler.

Element	Package and Description
<a href="#">pub.event:jmsSendErrorEvent</a>	WmPublic. Specification for the JMS delivery failure event handler.
<a href="#">pub.event:journal</a>	WmPublic. Specification for journal event handlers.
<a href="#">pub.event:journalInfo</a>	WmPublic. Document type for journal event information
<a href="#">pub.event:modifySubscriber</a>	WmPublic. Modifies the information about a subscription.
<a href="#">pub.event.nerv:eventToDocument</a>	WmPublic. <i>Deprecated</i> - There is no replacement service. Converts an incoming JMS message into an IS document ( <a href="#">pub.event.eda:event</a> ).
<a href="#">pub.event.nerv:send</a>	WmPublic. <i>Deprecated</i> - Replaced by <a href="#">pub.event.routing:send</a> Sends an EDA event to the Network for Event Routing and Variation (NERV).
<a href="#">pub.event.nerv:subscribe</a>	WmPublic. <i>Deprecated</i> - Replaced by <a href="#">pub.event.routing:subscribe</a> Subscribes to a particular type of event on Network for Event Routing and Variation (NERV) and identifies the service that will act as the event handler.
<a href="#">pub.event.nerv:unsubscribe</a>	WmPublic. <i>Deprecated</i> - Replaced by <a href="#">pub.event.routing:unsubscribe</a> Unsubscribes from an event type previously subscribed to using the <a href="#">pub.event.nerv:subscribe</a> service.
<a href="#">pub.event:portStatus</a>	WmPublic. Specification for a port status event.
<a href="#">pub.event:portStatusInfo</a>	WmPublic. Document type for port event information.
<a href="#">pub.event:reloadEventManagerSettings</a>	WmPublic. Reloads the settings from the event manager's configuration file ( <a href="#">eventcfg.bin</a> ) on the server.

Element	Package and Description
<a href="#">pub.event:replication</a>	WmPublic. Specification for replication event handlers.
<a href="#">pub.event:replicationInfo</a>	WmPublic. Document type for replication event information.
<a href="#">pub.event.routing:send</a>	WmPublic. Sends events to the messaging provider using Software AG Event Routing.
<a href="#">"pub.event.routing:eventAcknowledgment" on page 308</a>	WmPublic. Defines the input signature for a callback service that processes acknowledgments sent by the Event Routing framework.
<a href="#">pub.event.routing:subscribe</a>	WmPublic. Subscribes to a particular type of event using Software AG Event Routing.
<a href="#">pub.event.routing:unsubscribe</a>	WmPublic. Unsubscribes from an event type previously subscribed to using the <a href="#">pub.event.routing:subscribe</a> service.
<a href="#">pub.event:saveEventManagerSettings</a>	WmPublic. Saves the current subscriber information to the event manager's configuration file (eventcfg.bin) on the server.
<a href="#">pub.event:security</a>	WmPublic. Specification for security event handlers.
<a href="#">pub.event:securityInfo</a>	WmPublic. Document type for security event information.
<a href="#">pub.event:sessionEnd</a>	WmPublic. Specification for sessionEnd event handlers.
<a href="#">pub.event:sessionEndInfo</a>	WmPublic. Document type for sessionEnd event information.
<a href="#">pub.event:sessionExpire</a>	WmPublic. Specification for sessionExpire event handlers.
<a href="#">pub.event:sessionExpireInfo</a>	WmPublic. Document type for sessionExpire event information.

Element	Package and Description
<a href="#">pub.event:sessionStart</a>	WmPublic. Specification for sessionStart event handlers.
<a href="#">pub.event:sessionStartInfo</a>	WmPublic. Document type for sessionStart event information.
<a href="#">pub.event:stat</a>	WmPublic. Specification for stat event handlers.
<a href="#">pub.event:statInfo</a>	WmPublic. Document type for stat event information.
<a href="#">pub.event:txEnd</a>	WmPublic. Specification for txEnd event handlers.
<a href="#">pub.event:txEndInfo</a>	WmPublic. Document type for txEnd event information.
<a href="#">pub.event:txStart</a>	WmPublic. Specification for txStart event handlers.
<a href="#">pub.event:txStartInfo</a>	WmPublic. Document type for txStart event information.

## pub.event:addSubscriber

WmPublic. Creates a subscription for a specified event.

**Important:** Subscriptions that you add using this service take effect immediately; however, they are not made permanent unless you also persist them to disk with the [pub.event:saveEventManagerSettings](#) service. If you do not run [pub.event:saveEventManagerSettings](#) after adding subscribers, your changes will be lost when the server is restarted.

### Input Parameters

*EventType*      **String** Type of event to which the event handler is subscribing. Must be one of the following:

- Alarm Event
- Audit Event
- Audit Error Event

```

Error Event
Exception Event
GD End Event
AGD Start Event
JMS Delivery Failure Event
JMS Retrieval Failure Event
Journal Event
Port Status Event
Replication Event
Security Event
Session End Event
Session Expire Event
Session Start Event
Stat Event
Tx End Event
Tx Start Event

```

**Tip:** To view the current list of event types, you can execute the [pub.event:getEventTypes](#) service.

### *Filter*

**String** Selects (filters) the set of events within *EventType* to which the event handler is subscribing. `addSubscriber` uses *Filter* as a pattern string to filter a particular attribute of an event.

The pattern string can be composed of literal characters, which match a character exactly, and/or the "\*" character, which matches any sequence of characters. For example:

<u>This pattern string...</u>	<u>Would match...</u>
*	Any string
M*	Any string that starts with an uppercase "M."
M*X	Any string that starts with an uppercase "M" and ends with an uppercase "X."

The following table shows the attribute that is filtered for each event type. Note that some event types cannot be filtered.

<u>EventType</u>	<u>Filtered attribute</u>
Alarm Event	Message generated by the alarm event.

---

Audit Event	Fully qualified name of the service that generates the audit event.
Audit Error Event	Concatenated values of the <i>destination</i> and <i>errorCode</i> parameters of the audit error event.
Error Event	Error message text for the error that generates the error event.
Exception Event	Fully qualified name of the service that generates the exception event.
GD End Event	None. This event type cannot be filtered. <i>Filter</i> is ignored for this event type.
GD Start Event	Fully qualified name of the service that generates the GD Start Event.
JMS Delivery Failure Event	Name of the JMS connection alias used to send the message to the JMS provider.
JMS Retrieval Failure Event	Fully qualified name of the JMS trigger that invoked the trigger service for which the error occurred.
Journal Event	The major code and minor code of the message that causes the journal event.
Port Status Event	None. This event type cannot be filtered. <i>Filter</i> is ignored for this event type.
Replication Event	Name of the package being replicated.
Security Event	None. This event type cannot be filtered. <i>Filter</i> is ignored for this event type.
Session End Event	None. This event type cannot be filtered. <i>Filter</i> is ignored for this event type.
Session Expire Event	None. This event type cannot be filtered. <i>Filter</i> is ignored for this event type.

Session Start Event	<p>User ID of the user starting the session or the groups to which the user belongs. (The filter is applied to a space-delimited list of groups, composed of group names suffixed with the user's user ID.)</p> <p>The following examples show how you might filter session start events for various groups and/or user IDs:</p> <p>To select session starts for any user in the Administrators group, the filter would be:</p> <pre>*Administrators*</pre> <p>To select session starts for the user ID "LRMalley" in the Administrators group, the filter would be:</p> <pre>*Administrators*LRMalley</pre> <p>To select session starts for the user ID "LRMalley" in any group, the filter would be:</p> <pre>*LRMalley</pre>
Stat Event	None. This event type cannot be filtered. <i>Filter</i> is ignored for this event type.
Tx End Event	None. This event type cannot be filtered. <i>Filter</i> is ignored for this event type.
Tx Start Event	None. This event type cannot be filtered. <i>Filter</i> is ignored for this event type.
<i>Service</i>	<b>String</b> Fully qualified name of the event-handler service (the service that will execute when the event specified by <i>EventType</i> and <i>Filter</i> occurs).
<i>Comment</i>	<b>String</b> Descriptive comment for this subscription. This comment is displayed when subscriptions are viewed with Designer.
<i>Enabled</i>	<p><b>String</b> Flag specifying the status of the subscription. Must be one of the following values. Set to:</p> <ul style="list-style-type: none"> <li>■ <code>true</code> to make the subscription active.</li> <li>■ <code>false</code> to make the subscription inactive. This is the default.</li> </ul>
	<p><b>Note:</b> Although the default value is false, you will generally want to set <i>Enabled</i> to <code>true</code> to activate the subscription immediately when it is added.</p>

---

## Output Parameters

---

*Result*            **String** Flag indicating whether the subscriber was successfully added. A value of:

- `true` indicates that the subscriber was added successfully.
- `false` indicates that the subscriber was not added.

### See Also

[pub.event:deleteSubscriber](#)  
[pub.event:modifySubscriber](#)  
[pub.event:getSubscribers](#)  
[pub.event:saveEventManagerSettings](#)

---

## pub.event:alarm

WmPublic. Specification for alarm event handlers.

---

### Input Parameters

---

*time*            **String** Date and time that the event occurred, in the format *yyyy/MM/dd HH:mm:ss.SS*.

*service*        **String** Fully qualified name of the service that generated the event. A service can generate an alarm event when a client invokes a service that accesses information or a service on a remote server. If the client is not a member of an allowed group for the port on the remote server, the service will generate an alarm event.

*sessionID*     **String** Identification number for the session during which the alarm event was generated. Some alarm events are not generated during sessions. In these cases, the *sessionID* variable will not contain a value.

*msg*            **String** Text describing the alarm.

---

### Output Parameters

---

None.

### Usage Notes

Remember to register your handler with the Event Manager.

When you subscribe an event handler to an alarm event, you can create a filter for the *msg* field to specify the services whose alarm events you want to subscribe to. That is, you can specify which services' alarm events invoke the event handler.

---

## pub.event:alarmInfo

WmPublic. Document type for alarm event information.

### Parameters

---

<i>time</i>	<b>String</b> Date and time that the event occurred, in the format <i>yyyy/MM/dd HH:mm:ss.SS</i> .
<i>service</i>	<b>String</b> Fully qualified name of the service that generated the event.
<i>sessionID</i>	<b>String</b> Session ID of the service firing the alarm.
<i>msg</i>	<b>String</b> Text describing the alarm.

---

## pub.event:audit

WmPublic. Specification for audit event handlers.

### Input Parameters

---

<i>time</i>	<b>String</b> Date and time that the event occurred, in the format <i>yyyy-MM-dd HH:mm:ss z</i> (for example, "2004-10-28 14:46:39 EDT").
	<b>Note:</b> You can set the format for the <i>time</i> parameter in the <code>watt.server.dateStampFmt</code> property.
<i>TID</i>	<b>String</b> Server thread that generated the audit event.
<i>service</i>	<b>String</b> Fully qualified name of the service that generated the event.
<i>sessionID</i>	<b>String</b> Session ID of the service that generated the event.
<i>result</i>	<b>String</b> Description of the audit point. A value of: <ul style="list-style-type: none"> <li>■ <code>Started</code> indicates that this event marks the beginning of a service.</li> </ul>

- **Ended** indicates that this event marks the end of a service that executed successfully.
- **Failed** indicates that this event marks the end of a service that executed unsuccessfully (that is, threw an exception) and is not configured to retry. A failed event also marks the end of a service that executed unsuccessfully after exhausting all of its retries.
- **Retried** indicates that this event is created each time a service is retried.

*pipeline*      **Document** Optional. The pipeline that was passed to the service. This parameter is required only if the service is configured to include the pipeline when auditing.

*userName*      **String** User ID that invoked the service that generated the event.

---

### Output Parameters

None.

### Usage Notes

Events are created for a service only if auditing for that type of event is enabled for the service. For example, start events will not be created unless auditing for service start is enabled for that service.

Remember to register your handler with the Event Manager. Not all audit handlers that you code will log information.

When writing your own audit handler, be careful to not modify the *pipeline* variable within your handler.

Use the `watt.server.event.audit.async` server parameter to indicate whether event handlers for audit events are invoked synchronously or asynchronously. When this parameter is set to true, Integration Server invokes the event handlers that subscribe to audit events asynchronously. When this parameter is set to false, Integration Server invokes the event handlers that subscribe to audit events synchronously. The default is true (asynchronous).

---

## pub.event:auditError

WmPublic. Specification for audit error event handlers.

---

## Input Parameters

---

<i>time</i>	<b>String</b> Date and time at which the audit error occurred. This parameter is in the format <i>yyyy-MM-dd HH:mm:ss.SSS</i> (for example, "2004-10-28 14:46:39.505").												
<i>destination</i>	<b>String</b> Specifies the audit log destination to which the audit logging system attempted to write when the error occurred.												
	<table> <thead> <tr> <th><u>Value</u></th> <th><u>Audit logger where the error occurred...</u></th> </tr> </thead> <tbody> <tr> <td>ServiceDBDest</td> <td>Service audit logger</td> </tr> <tr> <td>ErrorDBDest</td> <td>Error audit logger</td> </tr> <tr> <td>SessionDBDest</td> <td>Session audit logger</td> </tr> <tr> <td>ProcessAuditDBDest</td> <td>Process audit logger</td> </tr> <tr> <td>CoreAuditDBDest</td> <td>All other audit loggers</td> </tr> </tbody> </table>	<u>Value</u>	<u>Audit logger where the error occurred...</u>	ServiceDBDest	Service audit logger	ErrorDBDest	Error audit logger	SessionDBDest	Session audit logger	ProcessAuditDBDest	Process audit logger	CoreAuditDBDest	All other audit loggers
<u>Value</u>	<u>Audit logger where the error occurred...</u>												
ServiceDBDest	Service audit logger												
ErrorDBDest	Error audit logger												
SessionDBDest	Session audit logger												
ProcessAuditDBDest	Process audit logger												
CoreAuditDBDest	All other audit loggers												
<i>message</i>	<b>String</b> Error message of the exception.												
<i>stackTrace</i>	<b>String</b> Stack trace for the exception.												
<i>errorCode</i>	<b>String</b> Optional. The SQL error code returned with the exception.												

---

## Output Parameters

---

None.

### Usage Notes

Remember to register your event handler using [pub.event:addSubscriber](#). Set the *EventType* input variable to `Audit Error Event`.

An audit error event is fired in the following situations:

- When a `SQLException` is encountered while trying to insert an audit record into the audit logging database.
- When Integration Server initializes and cannot connect to the audit logging database.
- When the Service logger is configured to retry failed auditing attempts, the audit error event is fired for the initial failure and each subsequent failure.

When you subscribe to an audit error event, you can supply a filter to limit the events that your event handler receives. The filter applies to the concatenated values of the *destination* and *errorCode* fields. You can use the asterisk (\*) as a wildcard character in the filter. The following table shows how you can use filters to limit the events that your event handler will receive:

This filter...	Limits the events that the event handler receives to...
<i>YourSearchTerm</i>	Events that contain <i>onlyYourSearchTerm</i> .
* <i>YourSearchTerm</i>	Events that contain <i>YourSearchTerm</i> at the end of the audit error event value.
<i>YourSearchTerm</i> *	Events that contain <i>YourSearchTerm</i> at the beginning of the audit error event value.
* <i>YourSearchTerm</i> *	Events that contain <i>YourSearchTerm</i> anywhere within the audit error event value.

Use the `watt.server.event.audit.async` server parameter to indicate whether event handlers for audit error events are invoked synchronously or asynchronously. When this parameter is set to `true`, Integration Server invokes the event handlers that subscribe to audit error events asynchronously. When this parameter is set to `false`, Integration Server invokes the event handlers that subscribe to audit error events synchronously. The default is `true` (asynchronous).

## pub.event:auditErrorInfo

WmPublic. Document type for audit error event information.

### Input Parameters

<i>time</i>	<b>String</b> Date and time at which the audit error occurred. This parameter is in the format <i>yyyy-MM-dd HH:mm:ss.SSS</i> (for example, "2004-10-28 14:46:39.505").
<i>destination</i>	<b>String</b> Specifies the audit log destination to which the audit logging system attempted to write when the error occurred.

<u>Value</u>	<u>Audit logger where the error occurred...</u>
<code>ServiceDBDest</code>	Service audit logger

---

<code>ErrorDBDest</code>	Error audit logger
<code>SessionDBDest</code>	Session audit logger
<code>ProcessAuditDBDest</code>	Process audit logger
<code>CoreAuditDBDest</code>	All other audit loggers
<i>message</i>	<b>String</b> Error message of the exception.
<i>stackTrace</i>	<b>String</b> Stack trace for the exception.
<i>errorCode</i>	<b>String</b> Optional. The SQL error code returned with the exception.

### Usage Notes

Use the `watt.server.event.audit.async` server parameter to indicate whether event handlers for audit events are invoked synchronously or asynchronously. When this parameter is set to `true`, Integration Server invokes the event handlers that subscribe to audit events asynchronously. When this parameter is set to `false`, Integration Server invokes the event handlers that subscribe to audit events synchronously. The default is `true` (asynchronous).

---

## pub.event:auditInfo

WmPublic. Document type for audit event information.

### Parameters

---

*time* **String** Date and time that the event occurred, in the format *yyyy-MM-dd HH:mm:ss z* (for example, "2004-10-28 14:46:39 EDT").

**Note:** You can set the format for the *time* parameter in the `watt.server.dateStampFmt` property.

*TID* **String** Server thread that generated the audit event in hashed format. The TID is generated by calling the Java `Thread.currentThread().hashCode()` method.

*service* **String** Fully qualified name of the service that generated the event.

*sessionID* **String** Session ID of the service that generated the event.

---

<i>result</i>	<p><b>String</b> Description of the audit point.</p> <ul style="list-style-type: none"> <li>■ <code>Started</code> indicates that this event marks the beginning of a service.</li> <li>■ <code>Ended</code> indicates that this event marks the end of a service that executed successfully.</li> <li>■ <code>Failed</code> indicates that this event marks the end of a service that executed unsuccessfully (that is, threw an exception) and is not configured to retry. A failed event also marks the end of a service that executed unsuccessfully after exhausting all of its retries.</li> <li>■ <code>Retried</code> indicates that this event is created each time a service is retried.</li> </ul>
<i>pipeline</i>	<p><b>Document</b> Optional. The pipeline that was passed to the service. This parameter is required only if the service is configured to include the pipeline when auditing.</p>
<i>userName</i>	<p><b>String</b> User ID that invoked the service that generated the event.</p>

### Usage Notes

Use the `watt.server.event.audit.async` server parameter to indicate whether event handlers for audit events are invoked synchronously or asynchronously. When this parameter is set to `true`, Integration Server invokes the event handlers that subscribe to audit events asynchronously. When this parameter is set to `false`, Integration Server invokes the event handlers that subscribe to audit events synchronously. The default is `true` (asynchronous).

---

## pub.event:callstackItem

WmPublic. Document type for the name of the service in the invocation path when an exception occurred and the index that indicates the step at which the exception occurred.

### Parameters

---

<i>service</i>	<p><b>String</b> Fully qualified name of the last service (that is, most recently called) on the call stack.</p>
<i>flowStep</i>	<p><b>String</b> Path representing the last executed flow step in the service where the exception occurred. The path takes the form <code>/n/n/n...</code>, where <code>n</code> represents a sequential index of flow steps and <code>/</code> indicates a level of nesting. The following illustrates an example of flow steps and the <i>flowStep</i> value that would be assigned to them if the exception occurred at that step:</p>

Step	flowStep Value
MAP	/0
SEQUENCE	/1
BRANCH	/1/0
INVOKE	/1/0/0
MAP	/2
SEQUENCE	/3
EXIT	/3/0

If the stack contains a Java service, the *flowStep* value for that service will be empty.

## pub.event:deleteSubscriber

WmPublic. Removes an event handler from the subscription list for a specified event.

**Important:** Deletions made using this service take effect immediately; however, they are not made permanent unless you persist them to disk with the [pub.event:saveEventManagerSettings](#) service. If you do not run [pub.event:saveEventManagerSettings](#) after deleting subscribers, your changes will be lost when the server is restarted.

### Input Parameters

<i>EventType</i>	<b>String</b> Type of event from which the event handler is unsubscribing. Must be one of the following values:
	Alarm Event
	Audit Event
	Exception Event
	GD End Event
	GD Start Event
	JMS Delivery Failure Event
	JMS Retrieval Failure Event
	Port Status Event
	Replication Event
	Security Event
	Session End Event
	Session Expire Event
	Session Start Event
	Stat Event
	Tx End Event
	Tx Start Event

**Tip:** To view the current list of event types, you can execute the [pub.event:getEventTypes](#) service.

*gID* **String** ID of the subscriber that you want to delete. To get a list of subscriber IDs, execute the [pub.event:getSubscribers](#) service.

## Output Parameters

*Result* **String** Flag indicating whether the subscriber was successfully deleted. A value of:

- `true` indicates that the subscriber was deleted successfully.
- `false` indicates that the subscriber was not deleted (typically an invalid subscriber ID was provided in *gID*).

## See Also

[pub.event:addSubscriber](#)  
[pub.event:modifySubscriber](#)  
[pub.event:getSubscribers](#)  
[pub.event:saveEventManagerSettings](#)

## pub.event.eda:event

WmPublic. Document type that defines the structure of an EDA event.

## Parameters

*evt:Header* **Document** The event header.

Key	Description
<i>evt:Start</i>	<b>String</b> Start date and time of the event, in the format <code>yyyy-MM-dd'T'HH:mm:ss.SSSZ</code> .
<i>evt:End</i>	<b>String</b> Optional. End date and time of the event, in the format <code>yyyy-MM-dd'T'HH:mm:ss.SSSZ</code> . The absence of an <i>End</i> value in an event is often interpreted as indicating that the event started and ended at the same time. While this is valid in some consumers, other consumers consider it impossible for events to start and end at precisely the same time. When an event does not specify

an *End* value, the consumer may set a default value, such as start time plus one millisecond.

<i>evt:Kind</i>	<b>String</b> Optional. Indicates whether the event is a new event (Event) or a heartbeat event (Heartbeat). A heartbeat event indicates the temporal progress of the stream.
<i>evt:Type</i>	<b>String</b> The unique identifier of the event type. Event Types use qualified names (QNames) as the mechanism for concisely identifying the particular type. The value of <i>evt:type</i> combines the URI and local name as a string. For example, <code>{http://namespaces.softwareag.com/EDA/WebM/Process/1.0}ProcessInstanceChange</code> is the event type identifier that reports changes to a process instance.
<i>evt:Version</i>	<b>String</b> Optional. The version of the event type with which the event is compatible. An event should contain a version only if the event type supports versioning. An event should not specify a version if the event type does not support versioning.
<i>evt:CorrelationID</i>	<b>String</b> Optional. Unique identifier used to associate an event with other events.
<i>evt:EventID</i>	<b>String</b> Optional. The unique identifier of the event. This element can be supplied by the user who emitted the event or can be system generated.
<i>evt:Priority</i>	<b>String</b> Optional. The priority of the event to the producer. The value can be <code>Normal</code> or <code>High</code> .
<i>evt:ProducerID</i>	<b>String</b> Optional. The identifier of the event producer. For example, this can be an application identifier or a globally unique identifier for each producer.
<i>evt:UserID</i>	<b>String</b> Optional. The identifier of the user who emitted the event.
<i>evt:CustomHeaders</i>	<b>Document</b> Optional. Custom header elements.

*evt:Body*      **Document** Optional. The event body.

### Usage Notes

The prefix “*evt*” is associated with the namespace <http://namespaces.softwareag.com/EDA/Event>. All events belong to this asset namespace.

The [pub.event.eda:schema\\_event](#) schema defines the structure and data types for an EDA event document.

---

## pub.event.eda:eventToDocument

WmPublic. *Deprecated* - There is no replacement service.

Converts an EDA event in the form of an XML string to a document instance in the form of `pub.event.eda:event`.

**Note:** The [pub.event.routing:subscribe](#) service, which replaces the deprecated [pub.event.nerv:subscribe](#) service, handles the conversion of an EDA event into an IS document (IData). Hence, you no longer have to run the `pub.event.eda:eventToDocument` service to convert an EDA event before it can be processed.

---

### Input Parameters

*xmldata*      **String** XML string containing the event to convert to a document (IData object). The XML string must conform to the Event element structure declared in the `Envelope.xsd` schema.

*documentTypeName*      **String** Optional. Fully qualified name of the IS document type that specifies the structure to impose on the body of the event. By specifying a document type, you can identify:

- The order and dimensionality of elements.
- The prefix associated with each namespace in the instance document.

The document type specified in *documentTypeName* does not need to specify every element that will appear in the resulting document. At a minimum, the document type needs to specify the elements whose structure you want to explicitly set and the elements that you want to use in pipeline mapping.

If you do not specify *documentTypeName*, the structure of the event body is determined solely by the event document.

---

## Output Parameters

---

*evt:Event*      **Document** A document reference to the [pub.event.eda:event](#) document type which specifies the structure of an event as a Document (IData object).

### Usage Notes

This service transforms each element and attribute in the EDA event to an element in an IData object.

This service always converts XML nodes to String or Document object fields. It does not generate constrained objects (for example, Floats or Integers), even if the fields in the specified document are defined as constrained objects.

The [pub.event.eda:event](#) document type determines the overall structure of the *evt:Event* IData object that this service returns.

The document type specified for *documentTypeName* determines the structure of the event body contained in the *evt:Body* field. If you do not specify *documentTypeName*, the structure of the event body is determined by the event.

The document type in *documentTypeName* identifies the namespace prefixes to use for the conversion. Integration Server determines the namespace prefix information through the association of the prefix in the field name with the URI in the **XML namespace** property of the field. For example, suppose that a field in *documentTypeName* is named `SAG:account` and the XML namespace property of that field is `http://www.softwareag.com`. In the resulting IData object for the event body, Integration Server will use the prefix SAG with any element that belongs to the `http://www.softwareag.com` namespace.

If *documentTypeName* does not specify namespace prefixes or *documentTypeName* is not specified, the prefixes used in conversion depend on the event document.

- If the event contains namespace qualified elements and uses prefixes, the resulting IData object for the event body uses the prefixes from the instance document.
- If the event contains elements qualified with a default namespace and the elements do not use prefixes, the resulting IData object for the event body does not use prefixes. Only the local name appears in the resulting event body.

In a document type that represents a namespace qualified XML document, it is considered good practice for the document type and its contents to account for all of the namespaces in the XML document. That is, the document type should include namespace prefixed fields associated with the namespaces for all of the possible namespace qualified elements in the XML document.

When *documentTypeName* is provided, the server parameter `watt.server.xml.xmlNodeToDocument.keepDuplicates` determines whether or not Integration Server keeps additional occurrences of an element in an XML document. When set to true, the document produced by the `pub.event.eda:eventToDocument` service

---

contains multiple occurrences of the element. When set to false, the document keeps only the last occurrence of the element. The default is true.

The `pub.event.eda:eventToDocumentservice` is used to convert events formatted with Integration Server version 8.2 into IData. To convert events formatted with Integration Server version 9.0 to IData, use the [pub.event.nerv:eventToDocument](#) service.

---

## pub.event.eda:schema\_event

WmPublic. Schema that defines the structure and data types used for the event header in the [pub.event.eda:event](#) document type.

---

## pub.event:error

WmPublic. Specification for error event handlers.

---

### Input Parameters

<i>stackTrace</i>	<b>String</b> Stack trace for the error.
<i>errorMessage</i>	<b>String</b> The error message, if any, generated by the error that causes the error event.
<i>serviceName</i>	<b>String</b> Fully qualified name of the service in which the error occurred.

---

### Output Parameters

None.

### Usage Notes

Remember to register your even handler using Event Manager or [pub.event:addSubscriber](#).

The `watt.server.event.exception.async` server parameter indicates whether event handlers for error events are invoked synchronously or asynchronously. When this parameter is set to true, Integration Server invokes the event handlers that subscribe to error events asynchronously. When this parameter is set to false, Integration Server invokes the services that subscribe to the error events synchronously. The default is true (asynchronous).

An error event handler can have a filter for the contents of *errorMessage*. The following filter specifies that any error event with an *errorMessage* whose value contains the word "missing" will invoke the event handler: `*missing*`

---

## pub.event:errorInfo

WmPublic. Document type for error event information.

### Parameters

---

<i>stackTrace</i>	<b>String</b> Stack trace for the error.
<i>errorMessage</i>	<b>String</b> The error message, if any, generated by the error that causes the error event.
<i>serviceName</i>	<b>String</b> Fully qualified name of the service in which the error occurred.

### Usage Note

The `watt.server.event.exception.async` server parameter indicates whether event handlers for error events are invoked synchronously or asynchronously. When this parameter is set to true, Integration Server invokes the event handlers that subscribe to error events asynchronously. When this parameter is set to false, Integration Server invokes the services that subscribe to the error events synchronously. The default is true (asynchronous).

An error event handler can have a filter for the contents of `errorMessage`. The following filter specifies that any error event with an `errorMessage` whose value contains the word "missing" will invoke the event handler: `*missing*`

---

## pub.event:exception

WmPublic. Specification for exception event handlers.

### Input Parameters

---

<i>time</i>	<b>String</b> Date and time that the event occurred, in the format <code>yyyy/MM/dd HH:mm:ss.SS</code> .
-------------	--

**Note:** You can set the format by specifying the `watt.server.dateStampFmt` property.

<i>error</i>	<b>String</b> Optional. Error message of the exception.
--------------	---

---

<i>localizedError</i>	<b>String</b> Optional. Error message in the language that corresponds to the locale of your webMethods installation.
<i>errorType</i>	<b>String</b> Exception type that was thrown.
<i>errorDump</i>	<b>String</b> More detailed information about the exception.
<i>service</i>	<b>String</b> Optional. Fully qualified name of the service that generated the event.
<i>user</i>	<b>String</b> User that requested the service that generated the event.
<i>callStack</i>	<b>Document List</b> Optional. The call stack information describing where the exception occurred, including the fully qualified name of a service on the stack and an index that identifies the last executed flow step in that service. Each document represents a service on the call stack. The first document in the list represents the service that threw the exception and the last document in the list represents the top-level service. The structure of this document is defined by <a href="#">pub.event:callstackItem</a> .
<i>pipeline</i>	<b>Document</b> Optional. State of the pipeline at the time the exception occurred.
<i>threadID</i>	<b>String</b> Thread ID identifying the thread that invoked the service.
<i>ssnid</i>	<b>String</b> Session ID during which the exception occurred.
<i>errorMsgID</i>	<b>String</b> Optional. The identification number for the error message.
<i>errorDetails</i>	<b>Document</b> Optional. Additional exception information provided by the author of the Java service. For more information about constructing exceptions to return additional information, see the <i>webMethods Integration Server Java API Reference</i> for the <code>com.wm.util.LocalizedException</code> class.
<i>nestedErrorInfo</i>	<b>Document</b> Optional. Nested errors and exceptions, if any. The structure of this document is defined by <a href="#">pub.event:exceptionInfo</a> .

### Output Parameters

---

None.

## Usage Notes

Remember to register your handler with the Event Manager.

Not all exception handlers that you code will log information.

Use the `watt.server.event.exception.async` server parameter to indicate whether event handlers for exception events are invoked synchronously or asynchronously. When this parameter is set to `true`, Integration Server invokes the event handlers that subscribe to exception events asynchronously. When this parameter is set to `false`, Integration Server invokes the services that subscribe to the exception events synchronously. The default is `true` (asynchronous).

When you subscribe an event handler to an exception event, you can create a filter for the `service` field to specify the services whose exception events you want to subscribe to. That is, you can specify which services' exception events invoke the event handler.

---

## pub.event:exceptionInfo

WmPublic. Document type for exception information.

### Parameters

---

<i>time</i>	<b>String</b> Date and time that the event occurred, in the format <code>yyyy/MM/dd HH:mm:ss.SS</code> .
<i>error</i>	<b>String</b> Optional. Error message of the exception.
<i>localizedError</i>	<b>String</b> Optional. Error message in the language that corresponds to the locale of your webMethods installation.
<i>errorType</i>	<b>String</b> Exception type that was thrown.
<i>errorDump</i>	<b>String</b> More detailed information about the exception.
<i>service</i>	<b>String</b> Optional. Fully qualified name of the service that generated the event.
<i>user</i>	<b>String</b> User that requested the service that generated the event.
<i>callStack</i>	<b>Document List</b> Optional. The call stack information describing where the exception occurred, including the fully qualified name of a service on the stack and an index that identifies the last executed flow step in that service. Each document represents a service on the call stack. The first document in the list represents the service that threw the exception and the

last document in the list represents the top-level service. The structure of this document is defined by [pub.event:callstackItem](#).

<i>pipeline</i>	<b>Document</b> Optional. State of the pipeline at the time the exception occurred.
<i>threadID</i>	<b>String</b> Thread ID identifying the thread that invoked the service.
<i>ssnid</i>	<b>String</b> Session ID during which the exception occurred.
<i>errorMsgID</i>	<b>String</b> Optional. The identification number for the error message.
<i>errorDetails</i>	<b>Document</b> Optional. Additional exception information provided by the author of the Java service. For more information about constructing exceptions to return additional information, see the <i>webMethods Integration Server Java API Reference</i> for the <code>com.wm.util.LocalizedException</code> class.
<i>nestedErrorInfo</i>	<b>Document</b> Optional. Nested errors and exceptions, if any. The structure of this document is defined by <a href="#">pub.event:exceptionInfo</a> .

### Usage Notes

Use the `watt.server.event.exception.async` server parameter to indicate whether event handlers for exception events are invoked synchronously or asynchronously. When this parameter is set to `true`, Integration Server invokes the event handlers that subscribe to exception events asynchronously. When this parameter is set to `false`, Integration Server invokes the services that subscribe to the exception events synchronously. The default is `true` (asynchronous).

---

## pub.event:gdEnd

WmPublic. Specification for gdEnd event handlers.

### Input Parameters

---

<i>time</i>	<b>String</b> Date and time that the event occurred, in the format <code>yyyy/MM/dd HH:mm:ss.SS</code> .
<i>TID</i>	<b>String</b> Transaction ID of the service that generated the event.
<i>result</i>	<b>String</b> Status of the transaction.

## Output Parameters

---

None.

## Usage Notes

Remember to register your handler with the Event Manager.

Use the `watt.server.event.gd.async` server parameter to indicate whether event handlers for all guaranteed delivery events (`gdStart` and `gdEnd`) are invoked synchronously or asynchronously. When this parameter is set to `true`, Integration Server invokes the event handlers that subscribe to the `gdStart` and/or `gdEnd` events asynchronously. When this parameter is set to `false`, Integration Server invokes the event handlers that subscribe to the `gdStart` and/or `gdEnd` events synchronously. The default is `true` (asynchronous).

---

## pub.event:gdEndInfo

WmPublic. Document type for `gdEnd` event information.

## Parameters

---

<i>time</i>	<b>String</b> Date and time that the event occurred, in the format <i>yyyy/MM/dd HH:mm:ss.SS</i> .
<i>TID</i>	<b>String</b> Transaction ID of the service that generated the event.
<i>result</i>	<b>String</b> Status of the transaction.

## Usage Notes

Use the `watt.server.event.gd.async` server parameter to indicate whether event handlers for all guaranteed delivery events (`gdStart` and `gdEnd`) are invoked synchronously or asynchronously. When this parameter is set to `true`, Integration Server invokes the event handlers that subscribe to the `gdStart` and/or `gdEnd` events asynchronously. When this parameter is set to `false`, Integration Server invokes the event handlers that subscribe to the `gdStart` and/or `gdEnd` events synchronously. The default is `true` (asynchronous).

---

## pub.event:gdStart

WmPublic. Specification for `gdStart` event handlers.

### Input Parameters

---

<i>time</i>	<b>String</b> Date and time that the event occurred, in the format <i>yyyy/MM/dd HH:mm:ss.SS</i> .
<i>TID</i>	<b>String</b> Transaction ID of the service that generated the event.
<i>svcname</i>	<b>String</b> Fully qualified name of the service that generated the event.
<i>result</i>	<b>String</b> Status of the transaction.

### Output Parameters

---

None.

### Usage Notes

Remember to register your handler with the Event Manager.

Use the `watt.server.event.gd.async` server parameter to indicate whether event handlers for all guaranteed delivery events (`gdStart` and `gdEnd`) are invoked synchronously or asynchronously. When this parameter is set to `true`, Integration Server invokes the event handlers that subscribe to the `gdStart` and/or `gdEnd` events asynchronously. When this parameter is set to `false`, Integration Server invokes the event handlers that subscribe to the `gdStart` and/or `gdEnd` events synchronously. The default is `true` (asynchronous).

When you subscribe an event handler to a `gdStart` event, you can create a filter for the `svcname` field to specify the services in a guaranteed delivery transaction that you want to subscribe to. That is, you can specify the services that when invoked using guaranteed delivery will invoke the event handler.

---

## pub.event:gdStartInfo

WmPublic. Document type for `gdStart` event information.

### Parameters

---

<i>time</i>	<b>String</b> Date and time that the event occurred, in the format <i>yyyy/MM/dd HH:mm:ss.SS</i> .
<i>TID</i>	<b>String</b> Transaction ID of the service that generated the event.
<i>svcname</i>	<b>String</b> Fully qualified name of the service that generated the event.

---

*result*      **String** Status of the transaction.

### Usage Notes

Use the `watt.server.event.gd.async` server parameter to indicate whether event handlers for all guaranteed delivery events (`gdStart` and `gdEnd`) are invoked synchronously or asynchronously. When this parameter is set to `true`, Integration Server invokes the event handlers that subscribe to the `gdStart` and/or `gdEnd` events asynchronously. When this parameter is set to `false`, Integration Server invokes the event handlers that subscribe to the `gdStart` and/or `gdEnd` events synchronously. The default is `true` (asynchronous).

---

## pub.event:getEventTypes

WmPublic. Returns the list of supported event types on Integration Server.

### Input Parameters

---

None.

### Output Parameters

---

*EventTypes*      **Document** The types of events that the server supports:

- Alarm Event
- Audit Event
- Audit Error Event
- Exception Event
- GD End Event
- GD Start Event
- JMS Delivery Failure Event
- JMS Retrieval Failure Event
- Port Status Event
- Replication Event
- Security Event
- Session End Event
- Session Expire Event
- Session Start Event
- Stat Event
- Tx End Event
- Tx Start Event

**Usage Note**

The `pub.event:getEventTypes` service returns a list of supported local event types on Integration Server. The service does not return a list of EDA event types in the event store.

**pub.event:getSubscribers**

WmPublic. Returns the list of subscribers for a specified event type.

**Input Parameters**

*EventType* **String** Type of event for which you want the list of subscribers. Must be one of the following values:

- Alarm Event
- Audit Event
- Audit Error Event
- Exception Event
- GD End Event
- GD Start Event
- JMS Delivery Failure Event
- JMS Retrieval Failure Event
- Port Status Event
- Replication Event
- Security Event
- Session End Event
- Session Expire Event
- Session Start Event
- Stat Event
- Tx End Event
- Tx Start Event

**Tip:** To view the current list of event types, you can execute the [pub.event:getEventTypes](#) service.

**Output Parameters**

*Subscribers* **Document** The list of subscribers. For each subscriber, *Subscribers* will contain a key that is the subscriber ID. The value of that key is a document containing the following information about the subscriber:

<u>Key</u>	<u>Description</u>
------------	--------------------

---

<i>Service</i>	<b>String</b> Fully qualified name of the event-handler service (that is, the service that subscribes to the event in <i>EventType</i> ).
<i>Filter</i>	<b>String</b> Filter associated with the subscription. This is a pattern string that selects (filters) an event based on a particular attribute. <i>Filter</i> is composed of literal characters, which match a character exactly, and/or the "*" character, which matches any sequence of characters. For example: <ul style="list-style-type: none"><li>■ * would math any string.</li><li>■ M* would math any string that starts with an uppercase "M."</li><li>■ M*X would math any string that starts with an uppercase "M" and ends with an uppercase "X."</li></ul> For a list of attributes to which the filter is applied, see <a href="#">pub.event:addSubscriber</a> .
<i>Comment</i>	<b>String</b> Descriptive comment associated with the description. If a comment has not been assigned to the subscription, <i>Comment</i> will be empty.
<i>gID</i>	<b>String</b> Subscriber ID.
<i>Enabled</i>	<b>String</b> Flag indicating the status of the subscription. Will be one of the following values. A value of: <ul style="list-style-type: none"><li>■ <code>true</code> indicates that the subscription is active.</li><li>■ <code>false</code> indicates that the subscription is inactive.</li></ul>

**See Also**

[pub.event:addSubscriber](#)  
[pub.event:modifySubscriber](#)  
[pub.event:deleteSubscriber](#)

---

## pub.event:jmsReceiveErrorEvent

WmPublic. Specification for a JMS retrieval failure event handler.

---

## Input Parameters

---

<i>triggerName</i>	<b>String</b> Specifies the name of the JMS trigger that executed the trigger service for which the JMS retrieval failure event occurred.
<i>triggerDestinationIndex</i>	<b>java.lang.Integer</b> Specifies the index for the destination from which the JMS trigger receives messages. A JMS trigger that specifies a join type can listen for messages from multiple destinations. The first destination listed has an index of 0, the second destination listed has an indices of 1, etc.
<i>deliveryCount</i>	<b>java.lang.Integer</b> Number of times the JMS provider delivered the message to the JMS trigger at the time the event occurred.
<i>time</i>	<b>String</b> Date and time that the event occurred, in the format <i>yyyy/MM/dd HH:mm:ss.SS</i>
<i>exceptionClass</i>	<b>String</b> Name of the class that caused the failure. This may be useful to determine programmatically why the error occurred.
<i>exceptionMessage</i>	<b>String</b> Message contained in the exception.
<i>data</i>	<b>Document</b> A document (IData) containing the JMS message being processed when the error occurred.

<u>Key</u>	<u>Description</u>
<i>JMSMessage</i>	A document reference (IData) to the <a href="#">pub.jms:JMSMessage</a> document type.

---

## Output Parameters

---

None

## Usage Notes

A JMS retrieval failure event occurs in the following situations:

- A trigger service executed by a JMS trigger throws a non-transient error and the `watt.server.jms.trigger.raiseEventOnException` property is set to `true` (the default).

- A trigger service associated with a JMS trigger ends because of a transient error, all retry attempts have been made, and the JMS trigger is configured to throw an exception on retry failure. In addition, the `watt.server.jms.trigger.raiseEventOnRetryFailure` property is set to `true` (the default).
- The maximum delivery count from the JMS provider has been met for the message and the `watt.server.jms.trigger.raiseEventOnRetryFailure` property is set to `true` (the default).

The `watt.server.jms.trigger.maxDeliveryCount` property specifies the maximum number of times the JMS provider can deliver a message to Integration Server. The default is 100. In a JMS message, the property `JMSXDeliveryCount` specifies the number of times the JMS provider delivered the message. Most JMS providers set this value.

- While performing exactly-once processing, the connection to the document history database is unavailable, and transient error handling for the JMS trigger is configured to **Throw exception** (non-transacted JMS trigger) or **Recover only** (transacted JMS trigger). In addition, the `watt.server.jms.trigger.raiseEventOnRetryFailure` property is set to `true` (the default).
- While performing exactly-once processing, the document resolver service ends with an `ISRuntimeException`, and transient error handling for the JMS trigger is configured to **Throw exception** (non-transacted JMS trigger) or **Recover only** (transacted JMS trigger). In addition, the `watt.server.jms.trigger.raiseEventOnRetryFailure` property is set to `true` (the default).
- While performing exactly-once processing, the document resolver service ends with an exception other than an `ISRuntimeException`. In addition, the `watt.server.jms.trigger.raiseEventOnRetryFailure` property is set to `true` (the default).

Remember to register your event handler with the Event Manager.

Use the `watt.server.event.jmsRetrievalError.async` server parameter to indicate whether event handlers for JMS retrieval failure events are invoked synchronously or asynchronously. When this parameter is set to `true`, Integration Server invokes the event handlers that subscribe to JMS retrieval failure events asynchronously. When this parameter is set to `false`, Integration Server invokes the event handlers that subscribe to the JMS retrieval failure events synchronously. The default is `true` (asynchronous).

#### See Also

[pub.jms:JMSMessage](#)

---

## pub.event:jmsSendErrorEvent

WmPublic. Specification for the JMS delivery failure event handler.

---

### Input Parameters

---

<i>aliasName</i>	<b>String</b> Name of the JMS connection alias used to send the message to the JMS provider.
<i>time</i>	<b>String</b> Date and time that the event occurred, in the format <i>yyyy/MM/dd HH:mm:ss.SS</i> .
<i>data</i>	<b>Document</b> Contents of the JMS message that could not be sent to the JMS provider.

### Output Parameters

---

None

### Usage Notes

Integration Server generates a JMS delivery failure event when a message written to the client side queue cannot be delivered to the JMS provider. When a transient error occurs, several delivery attempts may have been made.

You might want to create an event handler for a JMS delivery failure event to send notification or log information about the undelivered JMS message. You can also create an event handler that attempts to re-send the message to the JMS provider.

Remember to register your event handler with the Event Manager.

Use the `watt.server.event.jmsDeliveryFailureError.async` server parameter to indicate whether event handlers for JMS delivery failure events are invoked synchronously or asynchronously. When this parameter is set to `true`, Integration Server invokes the event handlers that subscribe to JMS delivery failure events asynchronously. When this parameter is set to `false`, Integration Server invokes the event handlers that subscribe to the JMS delivery failure events synchronously. The default is `true` (asynchronous).

---

## pub.event:journal

WmPublic. Specification for journal event handlers.

### Input Parameters

---

<i>time</i>	<b>String</b> Date and time that the event occurred, in the format <i>yyyy/mm/dd hh:mm:ss.ss</i> .
<i>productID</i>	<b>String</b> Name of the product that generated the journal log message and the event.

<i>majorCode</i>	<b>String</b> Major code of the message.
<i>minorCode</i>	<b>String</b> Minor code of the message.
<i>severity</i>	<b>String</b> Number indicating the severity of the message. <ul style="list-style-type: none"> <li>■ 1 Fatal</li> <li>■ 2 Error</li> <li>■ 3 Warning</li> <li>■ 4 Info</li> <li>■ 5, 6 Debut</li> <li>■ 7 - 10 Trace</li> </ul>
<i>defaultMessage</i>	<b>String</b> Default message associated with the major code and minor code. The actual message may differ from the default message. For example the default message might be "Package WmART is stopping due to {0}", where {0} is a placeholder for a run-time parameter. The actual message generated at run time might be "Package WmART is stopping due to ServiceException".

### Output Parameters

---

None.

### Usage Notes

Remember to register your even handler using Event Manager or [pub.event:addSubscriber](#).

A journal event handler can have a filter for the major code and minor code of the generated event. The format of the filter is <majorCode>.<minorCode>. For example, the following filter specifies that any journal event with major code of 28 followed by a minor code of 34 will invoke the event handler: \*28.34\*

If a journal event is created when synchronously or asynchronously. When this parameter is set to true, Integration Server writes the following log message "[ISS.0028.0034I] Package WmISExtDC is stopping " the journal event handler will be passed the following information in addition to the *time* value:

```

productID: ISS
majorCode: 28
minorCode: 34
severity: 4
defaultMessage: Package WmISExtDC is stopping

```

The `watt.server.event.exception.async` server parameter indicates whether event handlers for journal events are invoked synchronously or asynchronously. When this parameter is set to true, Integration Server invokes the event handlers that subscribe to

journal events asynchronously. When this parameter is set to false, Integration Server invokes the services that subscribe to the journal events synchronously. The default is true (asynchronous).

---

## pub.event:journalInfo

WmPublic. Document type for journal event information

---

### Parameters

<i>time</i>	<b>String</b> Date and time that the event occurred, in the format yyyy/mm/dd hh:mm:ss.ss.
<i>productID</i>	<b>String</b> Name of the product that generated the journal log message and the event.
<i>majorCode</i>	<b>String</b> Major code of the message.
<i>minorCode</i>	<b>String</b> Minor code of the message.
<i>severity</i>	<b>String</b> Number indicating the severity of the message. <ul style="list-style-type: none"> <li>■ 1 Fatal</li> <li>■ 2 Error</li> <li>■ 3 Warning</li> <li>■ 4 Info</li> <li>■ 5, 6 Debut</li> <li>■ 7 - 10 Trace</li> </ul>
<i>defaultMessage</i>	<b>String</b> Default message associated with the major code and minor code. The actual message may differ from the default message. For example the default message might be "Package WmART is stopping due to {0}", where {0} is a placeholder for a run-time parameter. The actual message generated at run time might be "Package WmART is stopping due to ServiceException".

### Usage Notes

If a journal event is created when synchronously or asynchronously. When this parameter is set to true, Integration Server writes the following log message "[ISS.0028.0034I] Package WmISExtDC is stopping " the journal event handler will be passed the following information in addition to the *time* value:

*productID*: ISS

*majorCode* : 28  
*minorCode* : 34  
*severity* : 4  
*defaultMessage* : Package WmISExtDC is stopping

The `watt.server.event.exception.async` server parameter indicates whether event handlers for journal events are invoked synchronously or asynchronously. When this parameter is set to true, Integration Server invokes the event handlers that subscribe to journal events asynchronously. When this parameter is set to false, Integration Server invokes the services that subscribe to the journal events synchronously. The default is true (asynchronous).

---

## pub.event:modifySubscriber

WmPublic. Modifies the information about a subscription.

**Important:** The changes you make with this service take effect immediately; however, they are not made permanent unless you also persist them to disk with the [pub.event:saveEventManagerSettings](#) service. If you do not run [pub.event:saveEventManagerSettings](#) after modifying subscribers, your changes will be lost when the server is restarted.

---

### Input Parameters

*EventType*      **String** Event type that you want the subscription to have:

- Alarm Event
- Audit Event
- Error Event
- Exception Event
- GD End Event
- GD Start Event
- JMS Delivery Failure Event
- JMS Retrieval Failure Event
- Journal Event
- Port Status Event
- Replication Event
- Security Event
- Session End Event
- Session Expire Event
- Session Start Event
- Stat Event
- Tx End Event
- Tx Start Event

**Tip:** To view the current list of event types, you can execute the [pub.event:getEventTypes](#) service.

*gID*

**String** ID of the subscriber that you want to modify. To get the current list of subscriber IDs, execute the [pub.event:getSubscribers](#) service.

*Filter*

**String** Filter that you want subscription to have. *Filter* is a pattern-matching string composed of literal characters, which match a character exactly, and/or the "\*" character, which matches any sequence of characters. For example:

<u>This pattern string...</u>	<u>Would match...</u>
*	Any string
M*	Any string that starts with an uppercase "M."
M*X	Any string that starts with an uppercase "M" and ends with an uppercase "X."

The following table shows the attribute that is filtered for each event type. Note that some event types cannot be filtered.

<u>EventType</u>	<u>Filtered attribute</u>
Alarm Event	Message generated by the alarm event.
Audit Event	Fully qualified name of the service that generates the audit event.
Error Event	Error message text for the error that generates the error event
Exception Event	Fully qualified name of the service that generates the exception event.
GD End Event	None. This event type cannot be filtered. <i>Filter</i> is ignored for this event type.

---

GD Start Event	Fully qualified name of the service that generates the GD Start Event.
JMS Delivery Failure Event	Name of the JMS connection alias used to send the message to the JMS provider.
JMS Retrieval Failure Event	Fully qualified name of the JMS trigger that called the trigger service for which the error occurred.
Journal Event	The major code and minor code of the message that causes the journal event.
Port Status Event	None. This event type cannot be filtered. <i>Filter</i> is ignored for this event type.
Replication Event	Name of the package being replicated.
Security Event	None. This event type cannot be filtered. <i>Filter</i> is ignored for this event type.
Session End Event	None. This event type cannot be filtered. <i>Filter</i> is ignored for this event type.
Session Expire Event	None. This event type cannot be filtered. <i>Filter</i> is ignored for this event type.
Session Start Event	<p>User ID of the user starting the session or the groups to which the user belongs. (The filter is applied to a space delimited list of groups, composed of group names suffixed with the user's user ID.)</p> <p>The following examples show how you might filter session-start events for various groups and/or user IDs:</p>

To select session starts for any user in the Administrators group, the filter would be:

```
*Administrators*
```

To select session starts for the user ID "LRMalley" in the Administrators group, the filter would be:

```
*Administrators*LRMalley
```

To select session starts for the user ID "LRMalley" in any group, the filter would be:

```
*LRMalley
```

Stat Event

None. This event type cannot be filtered. *Filter* is ignored for this event type.

Tx End Event

None. This event type cannot be filtered. *Filter* is ignored for this event type.

Tx Start Event

None. This event type cannot be filtered. *Filter* is ignored for this event type.

*Service* **String** Fully qualified name of the event-handler service that you want the subscription to specify.

*Comment* **String** Descriptive comment that you want to assign to the subscription.

*Enabled* **String** Flag specifying the status of the subscription. Must be one of the following values. Set to:

- true to make the subscription active.
- false to make the subscription inactive. This is the default.

**Note:** Although the default value is false, you will generally want to set *Enabled* to `true` to activate the subscription immediately after it is modified.

---

## Output Parameters

---

- Result*      **String** Flag indicating whether the subscriber was successfully modified. A value of:
- true indicates that the subscriber was updated successfully.
  - false indicates that the subscriber was not updated (typically because an invalid subscriber ID was provided in *gID*).

### See Also

[pub.event:addSubscriber](#)  
[pub.event:deleteSubscriber](#)  
[pub.event:getSubscribers](#)  
[pub.event:saveEventManagerSettings](#)

---

## pub.event.nerv:eventToDocument

WmPublic. *Deprecated* - There is no replacement service.

Converts an incoming JMS message into an IS document (`pub.event.eda:event`).

**Note:** Prior to deprecation, the `pub.event.nerv:eventToDocument` service had to be run to convert an incoming JMS message into an IS document before the service handler `pub.event.nerv:subscribe` could process the event. Now, the [pub.event.routing:subscribe](#) service, which replaces the deprecated `pub.event.nerv:subscribe` service, handles the conversion of the incoming JMS message into an IS document (IData).

---

## Input Parameters

---

- JMSMessage*      **Document** A document reference to the IS document of type [pub.jms:JMSMessage](#).
- documentTypeName*      **String** Optional. Fully qualified name of the IS document type that specifies the structure to impose on the body of the event. By specifying a document type, you can identify:
- The order and dimensionality of elements.
  - The prefix associated with each namespace in the instance document.
- The document type specified in *documentTypeName* does not need to specify every element that will appear in the resulting document. At a minimum, the document type

needs to specify the elements whose structure you want to explicitly set and the elements that you want to use in pipeline mapping.

If you do not specify *documentTypeName*, the structure of the event body is determined solely by the event document.

## Output Parameters

*evt:Event*      **Document** A document reference to the [pub.event.eda:event](#) document type which specifies the structure of an event as a document (IData Object).

## Usage Notes

This service maps various properties from a JMS message to an event document. It also maps the body from JMS message to the body on the event document.

## See Also

[pub.jms:JMSMessage](#)

[pub.event.eda:event](#)

## pub.event.nerv:send

WmPublic. *Deprecated* - Replaced by [pub.event.routing:send](#).

Sends an EDA event to the Network for Event Routing and Variation (NERV). Integration Server constructs an EDA event using the parameters defined in the service, and then sends the event to NERV.

**Note:** NERV is a framework that enables applications to communicate using events. It uses the Apache Camel integration framework for event routing, filtering, and variation. By default, NERV uses a Camel component that is configured for JMS as the transport layer and JNDI destinations as the endpoints. For more information about using NERV and configuring other providers, see *webMethods Event Processing Help*.

## Input Parameters

*documentType  
Name*      **String** Optional. Fully qualified name of the IS document type that specifies the structure to impose on the body of the event. By specifying a document type, you can identify:

- The order and dimensionality of elements.

- The prefix associated with each namespace in the instance document.

When field names in a document type include a prefix and specify a value for the **XML namespace** property, the `pub.event.nerv:send` service will convert a name/value pair in the *event/body* IData with the same prefix and name to an XML element with that prefix and with a local name equivalent to the XML namespace property value.

*event*

**Document** A document (IData) containing the event that you want to publish, including the event header and payload.

<u>Key</u>	<u>Description</u>
<i>header</i>	<b>Document</b> A document containing the information that you want to set in the event header.
<u>Key</u>	<u>Description</u>
<i>Start</i>	<b>java.util.date</b> Optional. Start date and time of the event. If you do not specify a <i>start</i> value, the <code>pub.event.nerv:send</code> service uses the current date and time.
<i>End</i>	<b>java.util.date</b> Optional. End date and time of the event.
<i>Kind</i>	<b>String</b> Optional. Indicates whether the event is a new event (Event) or a heartbeat event (Heartbeat). A heartbeat event indicates the temporal progress of the stream.  Possible values are Event and Heartbeat. The default is Event.
<i>Type</i>	<b>String</b> The name of the event type for the event.
<i>Version</i>	<b>String</b> Optional. Version of the event type with which the

---

	<p>event is compatible. An event should contain a version only if the event type supports versioning.</p> <p>Uses this regular expression for validation: <code>[0-9]+(.[0-9]+)*</code></p>
<i>CorrelationID</i>	<b>String</b> Optional. Unique identifier used to associate this event with other events.
<i>EventID</i>	<b>String</b> Optional. The unique identifier of the event. This element can be supplied by the event producer or can be system generated.
<i>Priority</i>	<b>String</b> Optional. The priority of the event to the producer. The value can be <code>Normal</code> or <code>High</code> .
<i>ProducerID</i>	<b>String</b> Optional. The identifier of the event producer. For example, this can be an application identifier or a globally unique identifier for each producer.
<i>UserID</i>	<b>String</b> Optional. The identifier of the user who emitted the event.
<i>CustomHeaders</i>	<b>Document List</b> Optional. Any other name/value parameters that should be included in the header field of the message.
<i>body</i>	<b>Document</b> Optional. A document (IData) containing the payload for the event. If this is a heartbeat event, do not specify a value for <i>body</i> .
<i>endpointUri</i>	<b>String</b> Optional. Endpoint URIs to which the event will be sent.

If this parameter is specified, NERV sends the event to the specified endpoint URIs.

If this parameter is not specified, NERV sends the event to the default endpoint configured for NERV.

*encode*

**String** Optional. Flag indicating whether to encode the data. Set this parameter to true if your XML data contains special characters, including the following: < > & " ' .

Set to:

- `true` to encode the data.  
For example, the string expression `5 < 6` would be converted to `<expr>5 &lt; 6</expr>`, which is valid.
- `false` to not encode the data. This is the default.  
For example, the string expression `5 < 6` would be converted to `<expr>5 < 6</expr>`, which is invalid.

## Output Parameters

---

**None.**

## Usage Notes

The `pub.event.nerv:send` service sends an event in the form of a Camel message to NERV. NERV then routes this event to all consumers that have registered a listener for the event type. You can send a regular EDA event or a heartbeat event for the specified event type in `header/Type`. The `pub.event.nerv:send` service infers whether the event is a regular event or a heartbeat event based on the presence of the `body` parameter. To send a heartbeat event, do not specify a value for the `body` parameter.

Some attribute values might be inserted into the event by NERV if the event producer does not supply one. For example, `EventID` and `Start` will be assigned if the event producer does not supply one.

## See Also

[pub.event.nerv:subscribe](#)

[pub.event.nerv:unsubscribe](#)

---

## pub.event.nerv:subscribe

WmPublic. *Deprecated* - Replaced by [pub.event.routing:subscribe](#).

Subscribes to a particular type of event on Network for Event Routing and Variation (NERV) and identifies the service that will act as the event handler.

**Note:** NERV is a framework that enables applications to communicate using events. It uses the Apache Camel integration framework for event routing, filtering, and variation. By default, NERV uses a Camel component that is configured for JMS as the transport layer and JNDI destinations as the endpoints. For more information about using NERV and configuring other providers, see *webMethods Event Processing Help*.

## Input Parameters

<i>eventTypeName</i>	<p><b>String</b> Fully qualified name of the event type to which to subscribe. For example, <code>{http://namespaces.softwareag.com/EDA/WebM/Sample/InventoryMgmt/1.0}PartInventoryLow</code> is an event type identifier used for inventory management. Integration Server maintains a list of default event types in the directory <i>Integration Server_directory/common/EventTypeStore</i> on the machine where webMethods Universal Messaging is installed.</p> <p>Integration Server does not check whether the specified event type exists in NERV.</p>
<i>serviceName</i>	<p><b>String</b> Fully qualified name of the flow service that Integration Server should invoke when an event is received from NERV.</p> <p>Integration Server does not check whether the service specified in <i>serviceName</i> exists until the service is invoked.</p>
<i>durableSubscriptionClientId</i>	<p><b>String</b> Optional. Unique client identifier that designates the subscription as a durable subscription on the messaging provider.</p> <p>If you provide a value for this parameter, also provide a value for <i>durableSubscriptionName</i>.</p> <p>Durable subscriptions enable subscribers to receive messages even when the subscriber is inactive, whereas non-durable subscriptions allow subscribers to receive messages only if those messages are published while the subscriber is active. For more information about how messages are received for durable and non-durable subscriptions, see <i>Using webMethods Integration Server to Build a Client for JMS</i> and <i>webMethods Broker Messaging Programmer's Guide</i>.</p>
<i>durableSubscriptionName</i>	<p><b>String</b> Optional. Name of the durable subscription associated with the identifier specified in <i>durableSubscriptionClientId</i>.</p>

*user*

**String** Optional. Name of the user who will invoke the service specified in *serviceName*. Integration Server validates the user name when the service specified in *serviceName* is executed.

If no user name is specified here, Integration Server uses the name specified in the server parameter `watt.server.event.nerv.subscribeService.user`. If no user is specified in that parameter, Integration Server uses the Administrator user to execute the service.

**Note:** The user specified in this parameter can be defined either internally or in an external directory but must belong to an existing group that has appropriate ACL permissions to invoke the specified service. For more information, see *webMethods Integration Server Administrator's Guide*.

*eventMessageToDocument*

**java.lang.Boolean** Optional. Flag indicating how Integration Server should pass the event message received from NERV as input to the service specified in *serviceName*.

Set to:

- `True` to extract the message header and body and pass the header and body in IData format to the service. This is the default.
- `False` to pass the message in the format it was received from NERV.

## Output Parameters

---

None.

## Usage Notes

The `pub.event.nerv:subscribe` service creates a subscription to a particular event type on NERV. When NERV receives a matching event, NERV routes the event to Integration Server in the form of a Camel message. Integration Server then receives the event and executes the service specified in *serviceName* using the credentials for the user specified in *user*. Integration Server passes either the message header and body, in IData format, or the entire message as it was received from NERV as specified in *eventMessageToDocument* for further processing. Make sure that the input signature of the service specified in *serviceName* matches the content passed to that service.

## See Also

[pub.event.nerv:send](#)

[pub.event.nerv:unsubscribe](#)

---

## pub.event.nerv:unsubscribe

WmPublic. *Deprecated* - Replaced by [pub.event.routing:unsubscribe](#).

Unsubscribes from an event type previously subscribed to using the `pub.event.nerv:subscribe` service.

**Note:** NERV is a framework that enables applications to communicate using events. It uses the Apache Camel integration framework for event routing, filtering, and variation. By default, NERV uses a Camel component that is configured for JMS as the transport layer and JNDI destinations as the endpoints. For more information about using NERV and configuring other providers, see *webMethods Event Processing Help*.

---

### Input Parameters

*eventTypeName*      **String** Fully qualified name of the event type from which to unsubscribe. For example, `{http://namespaces.softwareag.com/EDA/WebM/Sample/InventoryMgmt/1.0}PartInventoryLow` is an event type identifier used for inventory management.

Integration Server does not check whether the specified event type exists in NERV.

*serviceName*      **String** Fully qualified name of the service specified in the `pub.event.nerv:subscribe` service. Integration Server uses this parameter to identify the correct event type subscription to unsubscribe.

Integration Server does not check whether the service specified in *serviceName* exists.

---

### Output Parameters

**None.**

### See Also

[pub.event.nerv:send](#)

[pub.event.nerv:subscribe](#)

---

## pub.event:portStatus

WmPublic. Specification for a port status event.

### Input Parameters

---

*portStatusInfo* **Document List** of documents (Data[] objects) containing the following information for each port.

<b>Key</b>	<b>Description</b>
<i>time</i>	<b>String</b> Date and time that the event occurred, in the format <i>yyyy/MM/dd HH:mm:ss.SS</i> .
<i>port</i>	<b>String</b> Number for the port.
<i>status</i>	<b>String</b> Status of the port.
<i>protocol</i>	<b>String</b> Type of port (for example, <i>http</i> , <i>https</i> , <i>ftp</i> , or <i>email</i> ).
<i>primary</i>	<b>String</b> Primary port. By default, the webMethods Integration Server designates an HTTP port at port 5555 as the primary port.
<i>enabled</i>	<b>String</b> Flag indicating whether or not the port is enabled. Set to: <ul style="list-style-type: none"> <li>■ true to indicate that the port is enabled.</li> <li>■ false to indicate that the port is disabled.</li> </ul>

### Output Parameters

---

None.

---

## pub.event:portStatusInfo

WmPublic. Document type for port event information.

## Parameters

---

<i>time</i>	<b>String</b> Date and time that the event occurred, in the format <i>yyyy/MM/dd HH:mm:ss.SS</i> .
<i>port</i>	<b>String</b> Number for the port.
<i>status</i>	<b>String</b> Status of the port.
<i>protocol</i>	<b>String</b> Type of port (for example, <i>http</i> , <i>https</i> , <i>ftp</i> , or <i>email</i> ).
<i>primary</i>	<b>String</b> The primary port. By default, the webMethods Integration Server designates an HTTP port at port 5555 as the primary port.
<i>enabled</i>	<b>String</b> A flag indicating whether or not the port is enabled. A value of: <ul style="list-style-type: none"><li>■ <code>true</code> indicates that the port is enabled.</li><li>■ <code>false</code> indicates that the port is disabled.</li></ul>

---

## pub.event:reloadEventManagerSettings

WmPublic. Reloads the settings from the event manager's configuration file (`eventcfg.bin`) on the server.

### Input Parameters

---

None.

### Output Parameters

---

None.

### See Also

[pub.event:saveEventManagerSettings](#)

---

## pub.event:replication

WmPublic. Specification for replication event handlers.

---

## Input Parameters

---

<i>time</i>	<b>String</b> Date and time that the event occurred, in the format <i>yyyy/MM/dd HH:mm:ss.SS</i> .
<i>action</i>	<b>String</b> Description of the event (such as create or push). The value of <i>action</i> can be used to maintain separate logs for each action type.
<i>package</i>	<b>String</b> Name of package being replicated.
<i>service</i>	<b>String</b> Fully qualified name of the service that generated the event.

## Output Parameters

---

None.

## Usage Notes

Remember to register your handler with the Event Manager. When you subscribe an event handler to a replication event, you can create a filter to specify the package that, when replicated, will invoke the event handler.

Use the `watt.server.event.replication.async` server parameter to indicate whether event handlers for replication events are invoked synchronously or asynchronously. When this parameter is set to `true`, Integration Server invokes the event handlers that subscribe to replication events asynchronously. When this parameter is set to `false`, Integration Server invokes event handlers that subscribe to replication events synchronously. The default is `true` (asynchronous).

---

## pub.event:replicationInfo

WmPublic. Document type for replication event information.

## Parameters

---

<i>time</i>	<b>String</b> Date and time that the event occurred, in the format <i>yyyy/MM/dd HH:mm:ss.SS</i> .
<i>action</i>	<b>String</b> Description of the event (such as create or push). The value of <i>action</i> can be used to maintain separate logs for each action type.
<i>package</i>	<b>String</b> Name of package being replicated.
<i>service</i>	<b>String</b> Fully qualified name of the service that generated the event.

## Usage Notes

Use the `watt.server.event.replication.async` server parameter to indicate whether event handlers for replication events are invoked synchronously or asynchronously. When this parameter is set to `true`, Integration Server invokes the event handlers that subscribe to replication events asynchronously. When this parameter is set to `false`, Integration Server invokes event handlers that subscribe to replication events synchronously. The default is `true` (asynchronous).

## pub.event.routing:send

WmPublic. Sends events to the messaging provider using Software AG Event Routing. Integration Server constructs events using the parameters defined in this service.

**Note:** Event Routing is a framework that Software AG provides for applications to communicate using events. For more information about using Event Routing, see *Communicating Between Software AG Products Using Event Routing*.

## Input Parameters

*documentType*  
Name

**String** Optional. Fully qualified name of the IS document type that specifies the structure to impose on the body of the event. By specifying a document type, you can identify:

- The order and dimensionality of elements.
- The prefix associated with each namespace in the instance document.

When field names in a document type include a prefix and specify a value for the **XML namespace** property, the `pub.event.routing:send` service will convert a name/value pair in the `event/body` IData with the same prefix and name to an XML element with that prefix and with a local name equivalent to the XML namespace property value.

**Note:** You must specify an IS document type for the `documentTypeName` parameter if `createOptions` is set to `VALIDATE_BODY`. The XML structure of the IS document type specified in the `documentTypeName` parameter must match the XML structure passed as the value of `evt:body` in the `pub.event.eda:event` document type.

*event*

**Document** A document reference (IData) to the `pub.event.eda:event` document type containing the event that you want to publish, including the event header and payload.

*createOptions*

**String List** Optional. Options that Integration Server passes on to the Event Routing framework during the creation of an event.

Possible values are none or any one or both of the following:

- `EXTRACT_FILTERABLE_PROPS` if you want the Event Routing framework to verify that the workspace name and service name that combine to form the value of `evt:type` in the `pub.event.eda:event` document type are valid.
- `VALIDATE_BODY` if you want the Event Routing framework to verify that the XML structure passed as the value of `evt:body` in the `pub.event.eda:event` document type matches the XML structure of the IS document type specified as the `documentTypeName` parameter.

**Note:** If you select the `VALIDATE_BODY` option, you must specify an IS document type for the `documentTypeName` parameter. Integration Server will issue an error if you do not specify any value for `documentTypeName`.

*isAsync*

**Boolean** Optional. Specifies whether or not the event is sent synchronously or asynchronously. Set to:

- `true` to send the event asynchronously.

If set to `true`, after sending the event, Integration Server executes the next step in the flow service immediately. Integration Server does not wait for an acknowledgment before continuing service execution.

- `false` to send the event synchronously.

If set to `false`, after sending the event, Integration Server waits for an acknowledgment before executing the next step in the flow service. This is the default.

If the value of `isAsync` is set to `true`, you must specify a callback service in the `serviceName` input parameter.

*serviceName*

**String** Optional. The callback service that you want Integration Server to invoke upon receiving an acknowledgment from the Event Routing framework after completing the send operation.

The input signature for the callback service is defined by the document `pub.event.routing:eventAcknowledgement`. The `pub.event.routing:eventAcknowledgement` must be provided as an input document to the callback service.

**Note:** You must specify a callback service if the value of `isAsync` parameter is set to `true`.

*runAsUser* **String** Optional. The name of the user account whose credentials Integration Server uses to execute the callback service. This parameter is required only if the event is being sent asynchronously, that is, if the *isAsync* parameter is set to `true`.

You can specify a locally defined user account or a user account defined in a central or external directory. The user account you specify must have the execute ACL associated with the callback service.

If you do not specify any user account for an event that is sent asynchronously, Integration Server uses the name specified in the `watt.server.event.routing.runAsUser` server configuration parameter. If no user is specified in that parameter, Integration Server uses the Administrator user to execute the service. For more information about `watt.server.event.routing.runAsUser`, see *webMethods Integration Server Administrator's Guide*.

## Output Parameters

---

*eventID* **String** The unique identifier of the sent event. The *eventID* is the same as the *evt:EventID* in [pub.event.eda:event](#), either supplied by the user who emitted the event or generated by the Event Routing framework.

## Usage Notes

This service replaces [pub.event.nerv:send](#), which has been deprecated.

## See Also

[pub.event.routing:subscribe](#)  
[pub.event.routing:unsubscribe](#)  
[pub.event.eda:event](#)

## pub.event.routing:eventAcknowledgement

WmPublic. Document type that defines the input signature for a callback service that processes acknowledgments sent by the Event Routing framework. The Event Routing framework sends acknowledgments to the callback service in response to asynchronously sent events.

## Parameters

*eventID* **String** The unique identifier of the sent event. The *eventID* is the same as the *evt:EventID* in [pub.event.eda:event](#), either supplied

by the user who emitted the event or generated by the Event Routing framework.

*processedUsingClient* **Boolean** Optional. Specifies whether or not the Event Routing framework processes the event using the client thread that invoked the send operation, or its own thread. The Event Routing framework uses the first available thread to process the event and sets the parameter to `true` or `false` depending on the selected thread. Set to:

- `true` if the event is processed using the client thread that invoked the send operation.
- `false` if the event is processed using a thread in the Event Routing framework.

*exception* **String** Optional. Exception that occurred in the Event Routing framework while processing an event.

#### See Also

["pub.event.routing:send" on page 306](#)

---

## pub.event.routing:subscribe

WmPublic. Creates a subscription to a particular type of event using Software AG Event Routing. Integration Server identifies a subscription based on a combination of *eventTypeName* and *serviceName* or based on the *subscriberId*, if specified.

When the Event Routing framework receives a matching event, it routes the event to Integration Server. Upon receiving the event, Integration Server executes a specific service to process the event.

#### Input Parameters

---

*eventTypeName* **String** Fully qualified name of the event type to which to subscribe. For example, `{http://namespaces.softwareag.com/EDA/WebM/Sample/InventoryMgmt/1.0}PartInventoryLow` is an event type identifier used for inventory management. Integration Server maintains a list of default event types in the directory `Integration Server_directory/common/EventTypeStore` on the machine where webMethods Universal Messaging is installed.

*serviceName* **String** Fully qualified name of the service that Integration Server should invoke when the Event Routing framework notifies Integration Server about receiving an event of the type specified

in *eventTypeName*. The signature for this service must reference the document type [pub.event.eda:event](#) as its only input.

Integration Server verifies that the service specified in *serviceName* exists and issues an error if the service does not exist or is not valid.

*runAsUser*

**String** Optional. Name of the user account whose credentials Integration Server uses to invoke the service specified in *serviceName*. Integration Server validates the user name when the service specified in *serviceName* is executed.

You can specify a locally defined user account or a user account defined in a central or external directory. The user account you specify must have the execute ACL associated with the service specified in *serviceName*.

If you do not specify any user account for *runAsUser*, Integration Server uses the name specified in the `watt.server.event.routing.runAsUser` server configuration parameter. If no user is specified in that parameter, Integration Server uses the Administrator user to execute the service. For more information about `watt.server.event.routing.runAsUser`, see *webMethods Integration Server Administrator's Guide*.

*isDurable*

**String** Optional. Indicates whether or not the subscription is durable. Set to:

- `true` if you want the subscription to be a durable subscription.

**Note:** You must specify a unique value for the *subscriberID* parameter if you set *isDurable* to `true`.

In case of durable subscriptions, the messaging provider saves the messages for an inactive subscriber and delivers these saved messages when the subscriber creates a new subscription by specifying the same *subscriberID*. The messaging provider retains the messages for durable subscriptions until they are retrieved by the subscriber or until they expire.

- `false` if you do not want the subscription to be durable. This is the default.

Non-durable subscriptions allow subscribers to receive messages only if those messages are published while the subscription is active.

*subscriberID*

**String** Optional. User-specified unique identifier that designates the subscription as a durable subscription on the messaging

provider. If no *subscriberID* is specified, the subscription will be non-durable.

If you remove a durable subscription by using the [pub.event.routing:unsubscribe](#) service with the *removeDurable* parameter set to false and you want to resume the subscription, you can do so by giving the same *subscriberID* in the [pub.event.routing:subscribe](#) service. However, if *removeDurable* is set to true while unsubscribing, you will have to create a new subscription with a new *subscriberID*.

If no *subscriberID* is specified, the subscription that is created is non-durable.

**Note:** You must specify a unique value for the *subscriberID* parameter if you set *isDurable* to `true`.

## Output Parameters

---

None.

## Usage Notes

This service replaces [pub.event.nerv:subscribe](#), which has been deprecated.

Integration Server does not persist the subscriptions to event types across server restarts or WmPublic package reloads. Consequently, you need to create subscriptions each time Integration Server starts or you reload the WmPublic package. If you want these subscriptions to be created each time Integration Server starts or you reload the WmPublic package, you can assign the services that create durable subscriptions for various event types as startup services for the WmPublic package.

## See Also

[pub.event.routing:send](#)  
[pub.event.routing:unsubscribe](#)  
[pub.event.eda:event](#)

# pub.event.routing:unsubscribe

---

WmPublic. Removes a subscription to an event type previously subscribed to using the [pub.event.routing:subscribe](#) service.

## Input Parameters

---

<i>eventTypeName</i>	<b>String</b> Fully qualified name of the event type from which you want to unsubscribe. For example, <code>{http://namespaces.softwareag.com/EDA/WebM/Sample/InventoryMgmt/1.0}PartInventoryLow</code> is an event type
----------------------	--

identifier used for inventory management. Integration Server maintains a list of default event types in the directory *Integration Server\_directory/common/EventTypeStore* on the machine where webMethods Universal Messaging is installed.

<i>serviceName</i>	<p><b>String</b> Fully qualified name of the service specified in the <i>serviceName</i> parameter of the <a href="#">pub.event.routing:subscribe</a> service. Integration Server uses either a combination of the <i>eventName</i> and <i>serviceName</i> or the <i>subscriberId</i> to determine the correct event type subscription to unsubscribe.</p>
<i>subscriberId</i>	<p><b>String</b> Optional. User-specified unique identifier specified in the <i>subscriberId</i> parameter of the <a href="#">pub.event.routing:subscribe</a> service.</p> <p>Integration Server uses either the <i>subscriberId</i> or a combination of the <i>eventName</i> and <i>serviceName</i> to determine the correct event type subscription to unsubscribe.</p>
<i>removeDurable</i>	<p><b>String</b> Optional. Indicates whether or not to remove the durable subscription for the specified <i>subscriberId</i> from the messaging provider while unsubscribing from an event type. Set to:</p> <ul style="list-style-type: none"> <li>■ <code>true</code> to remove the durable subscription for an event type from the messaging provider.</li> </ul> <p>If set to <code>true</code>, the subscription for the specified event type and the corresponding <i>subscriberID</i> is removed from the messaging provider. If you want to receive notifications for the event type again, you must create new a subscription using the <a href="#">pub.event.routing:subscribe</a> service.</p> <div style="background-color: #f0f0f0; padding: 5px; margin: 5px 0;"> <p><b>Note:</b> If <i>removeDurable</i> is set to <code>true</code>, you must specify a <i>subscriberID</i>.</p> </div> <ul style="list-style-type: none"> <li>■ <code>false</code> to retain the durable subscription on the messaging provider after unsubscribing from an event type. This is the default.</li> </ul> <p>If set to <code>false</code>, you can resume the subscription by giving the same <i>subscriberID</i> in the <a href="#">pub.event.routing:subscribe</a> service.</p>

## Output Parameters

---

None.

## Usage Notes

This service replaces [pub.event.nerv:unsubscribe](#), which has been deprecated.

---

**See Also**

[pub.event.routing:subscribe](#)  
[pub.event.routing:send](#)  
[pub.event.eda:event](#)

---

## pub.event:saveEventManagerSettings

WmPublic. Saves the current subscriber information to the event manager's configuration file (eventcfg.bin) on the server.

**Important:** Always run this service after making any permanent changes to subscriber information (for example, add subscribers, modify subscribers, or delete subscribers). Otherwise, your changes will be lost the next time the server is restarted.

---

**Input Parameters**

None.

---

**Output Parameters**

None.

**See Also**

[pub.event:addSubscriber](#)  
[pub.event:deleteSubscriber](#)  
[pub.event:modifySubscriber](#)  
[pub.event:reloadEventManagerSettings](#)

---

## pub.event:security

WmPublic. Specification for security event handlers.

---

**Input Parameters**

<i>time</i>	<b>String</b> Date and time that the event occurred, in the format <i>yyyy/MM/dd HH:mm:ss.SS</i> .
<i>clientID</i>	<b>String</b> IP address of the host from which the request originated.
<i>serverID</i>	<b>String</b> IP address of the host on which Integration Server is running.

---

<i>userName</i>	<p><b>String</b> User ID that initiated or performed the security event.</p> <p>If a value is specified for the <i>clientApplication</i> field, then <i>userName</i> identifies the OAuth resource owner that permitted the OAuth client application to initiate or perform the security event.</p>
<i>securityEventType</i>	<p><b>String</b> Type of security event. Some examples are:</p> <ul style="list-style-type: none"><li>Authentication</li><li>Authorization</li><li>Certificates</li><li>Configuration</li><li>JDBC</li><li>Pools</li><li>Packages</li><li>Passwords</li><li>Ports</li><li>Remote Servers</li><li>Services</li><li>SSL Web Services</li></ul>
<i>clientApplication</i>	<p>User ID of the OAuth client application that initiated or performed the security event.</p> <p>If this field has a value, the <i>userName</i> field identifies the OAuth resource owner that permitted the OAuth client application to initiate or perform the security event.</p>
<i>result</i>	<p><b>String</b> Flag indicating whether the security action completed successfully. Set to:</p> <ul style="list-style-type: none"><li>■ <code>True</code> to indicate that the security event completed successfully.</li><li>■ <code>False</code> to indicate that the security event ended because of failure.</li></ul>
<i>message</i>	<p><b>String</b> Indicates what the security action was, irrespective of whether it was successful or unsuccessful. For example, if a user was successfully added to Integration Server, the message would say so. If the event was unsuccessful, this string would provide a reason or information about the failure, wherever possible.</p>

### Output Parameters

---

None.

## Usage Notes

Use the `watt.server.event.security.async` server parameter to indicate whether Integration Server invokes event handlers for security events synchronously or asynchronously. When this parameter is set to `true`, Integration Server invokes the event handlers that subscribe to security events asynchronously. When this parameter is set to `false`, Integration Server invokes event handlers that subscribe to security events synchronously. The default is `true` (asynchronous).

---

## pub.event:securityInfo

WmPublic. Document type for security event information.

### Input Parameters

---

<i>time</i>	<b>String</b> Date and time that the event occurred, in the format <i>yyyy/MM/dd HH:mm:ss.SS</i> .
<i>clientID</i>	<b>String</b> IP address of the host from which the request originated.
<i>serverID</i>	<b>String</b> IP address of the host on which Integration Server is running.
<i>userName</i>	<b>String</b> User ID that initiated or performed the security event.  If a value is specified for the <i>clientApplication</i> field, then <i>userName</i> identifies the OAuth resource owner that permitted the OAuth client application to initiate or perform the security event.
<i>securityEventType</i>	<b>String</b> Type of security event. Some examples are: <ul style="list-style-type: none"> <li>Authentication</li> <li>Authorization</li> <li>Certificates</li> <li>Configuration</li> <li>JDBC Pools</li> <li>Packages</li> <li>Passwords</li> <li>Ports</li> <li>Remote Servers</li> <li>Services</li> <li>SSL Web Services</li> </ul>

<i>clientApplication</i>	<p><b>String</b> User ID of the OAuth client application that initiated or performed the security event.</p> <p>If this field has a value, the <i>userName</i> field identifies the OAuth resource owner that permitted the OAuth client application to initiate or perform the security event.</p>
<i>result</i>	<p><b>String</b> Flag indicating whether the security action completed successfully.</p> <ul style="list-style-type: none"> <li>■ <code>True</code> indicates that the security event completed successfully.</li> <li>■ <code>False</code> indicates that the security event ended because of failure.</li> </ul>
<i>message</i>	<p><b>String</b> Indicates what the security action was, irrespective of whether it was successful or unsuccessful. For example, if a user was successfully added to Integration Server, the message would say so. If the event was unsuccessful, this string would provide a meaningful reason or information about the failure, wherever possible.</p>

### Output Parameters

---

None.

### Usage Notes

Use the `watt.server.event.security.async` server parameter to indicate whether Integration Server invokes event handlers for security events synchronously or asynchronously. When this parameter is set to `true`, Integration Server invokes the event handlers that subscribe to security events asynchronously. When this parameter is set to `false`, Integration Server invokes event handlers that subscribe to security events synchronously. The default is `true` (asynchronous).

## pub.event:sessionEnd

---

WmPublic. Specification for `sessionEnd` event handlers.

### Input Parameters

---

<i>time</i>	<p><b>String</b> Date and time that the event occurred, in the format <code>yyyy/MM/dd HH:mm:ss.SS</code>.</p>
<i>sessionID</i>	<p><b>String</b> Session ID of the service firing the alarm.</p>

*rpcs*           **String** Number of service calls the session has performed.

*age*           **String** Number of milliseconds the session existed before it ended.

### Output Parameters

---

None.

### Usage Notes

Remember to register your handler with the Event Manager.

Use the `watt.server.event.session.async` server parameter to indicate whether Integration Server invokes event handlers for session events (`sessionStart`, `sessionEnd`, and `sessionExpire`) synchronously or asynchronously. When this parameter is set to `true`, Integration Server invokes event handlers that subscribe to the session events asynchronously. When this parameter is set to `false`, Integration Server invokes event handlers that subscribe to the session events synchronously. The default is `true` (asynchronous).

---

## pub.event:sessionEndInfo

WmPublic. Document type for sessionEnd event information.

### Parameters

---

*time*           **String** Date and time that the event occurred. Given in the format *yyyy/MM/dd HH:mm:ss.SS*.

*sessionID*      **String** Session ID of the service firing the alarm.

*rpcs*           **String** Number of service calls the session has performed.

*age*           **String** Number of milliseconds the session existed before it ended.

### Usage Notes

Use the `watt.server.event.session.async` server parameter to indicate whether Integration Server invokes event handlers for session events (`sessionStart`, `sessionEnd`, and `sessionExpire`) synchronously or asynchronously. When this parameter is set to `true`, Integration Server invokes event handlers that subscribe to the session events asynchronously. When this parameter is set to `false`, Integration Server invokes event handlers that subscribe to the session events synchronously. The default is `true` (asynchronous).

---

## pub.event:sessionExpire

WmPublic. Specification for sessionExpire event handlers.

### Input Parameters

---

<i>time</i>	<b>String</b> Date and time that the event occurred. Given in the format <i>yyyy/MM/dd HH:mm:ss.SS</i> .
<i>sessionID</i>	<b>String</b> Session ID of the service firing the alarm.
<i>rpcs</i>	<b>String</b> Number of service calls the session has performed.
<i>age</i>	<b>String</b> Number of milliseconds the session existed before it expired.

### Output Parameters

---

None.

### Usage Notes

Remember to register your handler with the Event Manager.

Use the `watt.server.event.session.async` server parameter to indicate whether Integration Server invokes event handlers for session events (`sessionStart`, `sessionEnd`, and `sessionExpire`) synchronously or asynchronously. When this parameter is set to `true`, Integration Server invokes event handlers that subscribe to the session events asynchronously. When this parameter is set to `false`, Integration Server invokes event handlers that subscribe to the session events synchronously. The default is `true` (asynchronous).

---

## pub.event:sessionExpireInfo

WmPublic. Document type for sessionExpire event information.

### Parameters

---

<i>time</i>	<b>String</b> Date and time that the event occurred. Given in the format <i>yyyy/MM/dd HH:mm:ss.SS</i> .
<i>sessionID</i>	<b>String</b> Session ID of the service firing the alarm.

*rpcs*            **String** Number of service calls the session has performed.

*age*             **String** Number of milliseconds the session existed before it expired.

### Usage Notes

Use the `watt.server.event.session.async` server parameter to indicate whether Integration Server invokes event handlers for session events (`sessionStart`, `sessionEnd`, and `sessionExpire`) synchronously or asynchronously. When this parameter is set to `true`, Integration Server invokes event handlers that subscribe to the session events asynchronously. When this parameter is set to `false`, Integration Server invokes event handlers that subscribe to the session events synchronously. The default is `true` (asynchronous).

---

## pub.event:sessionStart

WmPublic. Specification for `sessionStart` event handlers.

### Input Parameters

---

*time*            **String** Date and time that the event occurred. Given in the format `yyyy/MM/dd HH:mm:ss.SS`.

*sessionID*      **String** ID of the new session.

*userid*          **String** User ID that the IS client or developer used to log on to the webMethods Integration Server.

*sessionName*   **String** Name of the new session.

### Output Parameters

---

None.

### Usage Notes

Remember to register your handler with the Event Manager. When you subscribe an event handler to a Session Start event, you can create a filter so that only session start events generated by a specific user or by a member of a specific group invoke the event handler.

Use the `watt.server.event.session.async` server parameter to indicate whether Integration Server invokes event handlers for session events (`sessionStart`, `sessionEnd`, and `sessionExpire`) synchronously or asynchronously. When this parameter is set to `true`, Integration Server invokes event handlers that subscribe to the session events asynchronously. When this parameter is set to `false`, Integration Server invokes

---

event handlers that subscribe to the session events synchronously. The default is true (asynchronous).

---

## pub.event:sessionStartInfo

WmPublic. Document type for sessionStart event information.

---

### Parameters

<i>time</i>	<b>String</b> Date and time that the event occurred. Given in the format <i>yyyy/MM/dd HH:mm:ss.SS</i> .
<i>sessionID</i>	<b>String</b> ID of the new session.
<i>userid</i>	<b>String</b> User ID that the IS client or developer used to log on to the webMethods Integration Server.
<i>sessionName</i>	<b>String</b> Name of the new session.

### Usage Notes

Use the `watt.server.event.session.async` server parameter to indicate whether Integration Server invokes event handlers for session events (`sessionStart`, `sessionEnd`, and `sessionExpire`) synchronously or asynchronously. When this parameter is set to `true`, Integration Server invokes event handlers that subscribe to the session events asynchronously. When this parameter is set to `false`, Integration Server invokes event handlers that subscribe to the session events synchronously. The default is true (asynchronous).

---

## pub.event:stat

WmPublic. Specification for stat event handlers.

---

### Input Parameters

<i>startTime</i>	<b>String</b> Date and time that the event occurred. Given in the format <i>yyyy/MM/dd HH:mm:ss.SS</i> .
<i>uptime</i>	<b>String</b> Amount of time the server has been up. Given in the format <i>yyyy/MM/dd HH:mm:ss.SS</i> .

---

<i>totalMem</i>	<b>String</b> Total amount of used and unused storage available to the JVM, in kilobytes. For example, a value of 65535 represents 64 megabytes of storage.
<i>freeMem</i>	<b>String</b> Amount of unused storage available to the Integration Server, in kilobytes. For example, a value of 65535 represents 64 megabytes of storage.
<i>usedMem</i>	<b>String</b> Amount of storage used by the Integration Server, in kilobytes. For example, a value of 65535 represents 64 megabytes of storage.
<i>freeMemPer</i>	<b>String</b> Percent of total memory unused.
<i>usedMemPer</i>	<b>String</b> Percent of total memory used.
<i>svrT</i>	<b>String</b> Number of services currently running.
<i>svrTMax</i>	<b>String</b> Peak number of servers ever running concurrently.
<i>sysT</i>	<b>String</b> Number of JVM threads running.
<i>sysTMax</i>	<b>String</b> Peak number of threads ever running.
<i>conn</i>	<b>String</b> Number of current sessions.
<i>connMax</i>	<b>String</b> Peak number of concurrent sessions.
<i>reqTotal</i>	<b>String</b> Cumulative total number of services processed.
<i>reqAvg</i>	<b>String</b> Average duration of service.
<i>newReqPM</i>	<b>String</b> New requests per minute.
<i>endReqPM</i>	<b>String</b> End requests per minute.
<i>errSvc</i>	<b>String</b> Number of services completed in error state.
<i>svcRate</i>	<b>String</b> Number of end/start(s) per second.
<i>ssnUsed</i>	<b>String</b> Number of licensed sessions currently active.

---

<i>ssnPeak</i>	<b>String</b> Number of licensed sessions that have ever run concurrently on the server.
<i>ssnMax</i>	<b>String</b> Maximum number of sessions for which the server is licensed.
<i>errSys</i>	<b>String</b> Number of unknown errors.

### Output Parameters

---

None.

### Usage Notes

Remember to register your handler with the Event Manager.

Use the `watt.server.event.stat.async` server parameter to indicate whether Integration Server invokes event handlers for statistics events synchronously or asynchronously. When this parameter is set to `true`, Integration Server invokes event handlers that subscribe to the `stat` event asynchronously. When this parameter is set to `false`, Integration Server invokes event handlers that subscribe to `stat` events synchronously. The default is `true` (asynchronous).

---

## pub.event:statInfo

WmPublic. Document type for stat event information.

### Parameters

---

<i>startTime</i>	<b>String</b> Date and time that the event occurred. Given in the format <code>yyyy/MM/dd HH:mm:ss.SS</code> .
<i>uptime</i>	<b>String</b> Amount of time the server has been up. Given in the format <code>yyyy/MM/dd HH:mm:ss.SS</code> .
<i>totalMem</i>	<b>String</b> Total amount of used and unused storage available to the JVM, in kilobytes. For example, a value of 65535 represents 64 megabytes of storage.
<i>freeMem</i>	<b>String</b> Amount of unused storage available to the Integration Server, in kilobytes. For example, a value of 65535 represents 64 megabytes of storage.

---

<i>usedMem</i>	<b>String</b> Amount of storage used by the Integration Server, in kilobytes. For example, a value of 65535 represents 64 megabytes of storage.
<i>freeMemPer</i>	<b>String</b> Percent of total memory unused.
<i>usedMemPer</i>	<b>String</b> Percent of total memory used.
<i>svrT</i>	<b>String</b> Number of services currently running.
<i>svrTMax</i>	<b>String</b> Peak number of servers ever running concurrently.
<i>sysT</i>	<b>String</b> Number of JVM threads running.
<i>sysTMax</i>	<b>String</b> Peak number of threads ever running.
<i>conn</i>	<b>String</b> Number of current sessions.
<i>connMax</i>	<b>String</b> Peak number of concurrent sessions.
<i>reqTotal</i>	<b>String</b> Cumulative total number of services processed.
<i>reqAvg</i>	<b>String</b> Average duration of service.
<i>newReqPM</i>	<b>String</b> New requests per minute.
<i>endReqPM</i>	<b>String</b> End requests per minute.
<i>errSvc</i>	<b>String</b> Number of services completed in error state.
<i>svcRate</i>	<b>String</b> Number of end/start(s) per second.
<i>ssnUsed</i>	<b>String</b> Number of licensed sessions currently active.
<i>ssnPeak</i>	<b>String</b> Number of licensed sessions that have ever run concurrently on the server.
<i>ssnMax</i>	<b>String</b> Maximum number of sessions for which the server is licensed.
<i>errSys</i>	<b>String</b> Number of unknown errors.

### Usage Notes

Use the `watt.server.event.stat.async` server parameter to indicate whether Integration Server invokes event handlers for statistics events synchronously or asynchronously. When this parameter is set to `true`, Integration Server invokes event handlers that subscribe to the `stat` event asynchronously. When this parameter is set to `false`, Integration Server invokes event handlers that subscribe to `stat` events synchronously. The default is `true` (asynchronous).

---

## pub.event:txEnd

WmPublic. Specification for `txEnd` event handlers.

### Input Parameters

---

<i>time</i>	<b>String</b> Date and time that the event occurred. Given in the format <i>yyyy/MM/dd HH:mm:ss.SS</i> .
<i>TID</i>	<b>String</b> Transaction ID of the service that generated the event.
<i>result</i>	<b>String</b> Status of the transaction.

### Output Parameters

---

None.

### Usage Notes

Remember to register your handler with the Event Manager.

Use the `watt.server.event.tx.async` server parameter to indicate whether Integration Server invokes event handlers for transaction events (`txStart` and `txEnd`) synchronously or asynchronously. When this parameter is set to `true`, Integration Server invokes event handlers that subscribe to the `txStart` or `txEnd` events asynchronously. When this parameter is set to `false`, Integration Server invokes event handlers that subscribe to the `txStart` or `txEnd` events synchronously. The default is `true` (asynchronous).

---

## pub.event:txEndInfo

WmPublic. Document type for `txEnd` event information.

---

## Parameters

---

<i>time</i>	<b>String</b> Date and time that the event occurred. Given in the format <i>yyyy/MM/dd HH:mm:ss.SS</i> .
<i>TID</i>	<b>String</b> Transaction ID of the service that generated the event.
<i>result</i>	<b>String</b> Status of the transaction.

## Usage Notes

Use the `watt.server.event.tx.async` server parameter to indicate whether Integration Server invokes event handlers for transaction events (`txStart` and `txEnd`) synchronously or asynchronously. When this parameter is set to `true`, Integration Server invokes event handlers that subscribe to the `txStart` or `txEnd` events asynchronously. When this parameter is set to `false`, Integration Server invokes event handlers that subscribe to the `txStart` or `txEnd` events synchronously. The default is `true` (asynchronous).

---

## pub.event:txStart

WmPublic. Specification for `txStart` event handlers.

## Input Parameters

---

<i>time</i>	<b>String</b> Date and time that the event occurred. Given in the format <i>yyyy/MM/dd HH:mm:ss.SS</i> .
<i>TID</i>	<b>String</b> Transaction ID of the service that generated the event.
<i>result</i>	<b>String</b> Status of the transaction.

## Output Parameters

---

None.

## Usage Notes

Remember to register your handler with the Event Manager.

Use the `watt.server.event.tx.async` server parameter to indicate whether Integration Server invokes event handlers for transaction events (`txStart` and `txEnd`) synchronously or asynchronously. When this parameter is set to `true`, Integration Server invokes event handlers that subscribe to the `txStart` or `txEnd` events asynchronously. When this

parameter is set to `false`, Integration Server invokes event handlers that subscribe to the `txStart` or `txEnd` events synchronously. The default is `true` (asynchronous).

---

## pub.event:txStartInfo

WmPublic. Document type for `txStart` event information.

### Parameters

---

*time*      **String** Date and time that the event occurred. Given in the format *yyyy/MM/dd HH:mm:ss.SS*.

*TID*        **String** Transaction ID of the service that generated the event.

*result*     **String** Status of the transaction.

### Usage Notes

Use the `watt.server.event.tx.async` server parameter to indicate whether Integration Server invokes event handlers for transaction events (`txStart` and `txEnd`) synchronously or asynchronously. When this parameter is set to `true`, Integration Server invokes event handlers that subscribe to the `txStart` or `txEnd` events asynchronously. When this parameter is set to `false`, Integration Server invokes event handlers that subscribe to the `txStart` or `txEnd` events synchronously. The default is `true` (asynchronous).

---

# 9 File Folder

---

■ File Access Control Configuration for the pub.file Services .....	328
■ Parameter Settings .....	328
■ Summary of Elements in this Folder .....	330

You use the elements in the file folder to perform operations on the local file system.

## File Access Control Configuration for the pub.file Services

The fileAccessControl.cnf configuration file in *Integration Server\_directory*\instances\*instance\_name*\packages\WmPublic\config directory contains parameters that Integration Server uses to provide additional validation checks to make the services in the pub.file folder secure.

**Note:** If you make any changes to the fileAccessControl.cnf, you must reload the WmPublic package or restart Integration Server for the changes to take effect.

## Parameter Settings

The following table shows the parameter settings for the fileAccessControl.cnf file:

Parameter	Description
allowedReadPaths	List of directories and/or files to which the services in the pub.file folder have read permission.
allowedWritePaths	List of directories and/or files to which the services in the pub.file folder and the pub.flow:savePipelineToFile service have write permission.
allowedDeletePaths	List of directories and/or files that the services in the pub.file folder can delete.

When modifying the parameters in the fileAccessControl.cnf file, keep the following points in mind:

- The parameters can be set to file names and/or directories.
  - If a file name is listed, access is allowed to that file only.  
For example, the following entry will allow the services in pub.file folder to write the file c:/wm8/test.txt.

```
allowedWritePaths=C:/wm8/test.txt
```

- If a directory name is listed, access is allowed to all files in that directory, but not to the subdirectories.

For example, the following entry will allow the services in pub.file folder to write to any file in the c:/wm8/test directory.

```
allowedWritePaths=C:/wm8/test
```

- Use a semicolon (;) as the delimiter when listing multiple paths. Do *not* include spaces between paths. For example:

```
allowedWritePaths=C:/wm8/test;C:/wm8/test.txt
```

- If a file or directory name has a semicolon (;), use two backslashes (\\) before the semicolon when specifying the allowed paths. For example, if the filename is: c:/temp/ab;c.txt

Specify it as: c:/temp/ab\\;c.txt

**Note:** When using a pathname with a semicolon as the value for an input parameter in a `pub.file` service, you do not need to include two backslashes before the semicolon.

- You can use the wildcard character asterisk (\*) to match multiple folders, subfolders, and files.

- A single asterisk (\*) wildcard can be used to denote single directory names.

For example, the following entry will allow the services in `pub.file` folder to write to all the `.log` files in the subdirectories whose names starts with 'fatal' in the `c:/home/` directory:

```
allowedWritePaths=C:/home/fatal*/*.log
```

- Double asterisks (\*\*) can be used to denote multiple files and subfolders in a folder.

For example, the following entry will allow the services in `pub.file` folder to write to all files in all the subdirectories of `c:/home` directory:

```
allowedWritePaths=C:/home/**
```

- The asterisk (\*) is the only wildcard character that you can use. All other characters are treated as literals.
- You cannot use a wildcard character to match any files in the root directory. Integration Server ignores the entries that use asterisks to match a file in the root directory.

For example, Integration Server will ignore the following entry:

```
allowedWritePaths = **
```

- UNIX-based operating systems are case-sensitive. When specifying paths for the parameters in `fileAccessControl.cnf`, make sure to use the appropriate case in the path names. Any path name supplied to input parameters for the `pub.file` services must match the case specified in the `fileAccessControl.cnf` parameters.
- If Integration Server runs on Windows, you can use either a forward slash (/) or two backslashes (\\) as the directory separator in paths specified in the `fileAccessControl.cnf`. When specifying values for input parameters for the `pub.file` services, you can specify a single forward slash, a single backslash, or a double backslash as the directory separator.

- If Integration Server runs on a UNIX-based operating system, you must use a forward slash as the directory separator.

## Summary of Elements in this Folder

The following elements are available in this folder:

Element	Package and Description
<a href="#">pub.file:bytesToFile</a>	WmPublic. Writes the specified byte array to a file.
<a href="#">pub.file:checkFileExistence</a>	WmPublic. Checks if a specified file exists, and if the file exists, checks whether the file name represents a file or a directory.
<a href="#">pub.file:copyFile</a>	WmPublic. Copies a file from one directory to another.
<a href="#">pub.file:deleteFile</a>	WmPublic. Deletes the specified file.
<a href="#">pub.file:getFile</a>	WmPublic. Retrieves a specified file from the local file system.
<a href="#">pub.file:listFiles</a>	WmPublic. List all the files in a specified directory.
<a href="#">pub.file:moveFile</a>	WmPublic. Moves a file from one directory to another.
<a href="#">pub.file:readerToFile</a>	WmPublic. Reads data from a java.io.Reader object and writes it to a file.
<a href="#">pub.file:streamToFile</a>	WmPublic. Writes the data in the InputStream to a file.
<a href="#">pub.file:stringToFile</a>	WmPublic. Writes text to a file.

### **pub.file:bytesToFile**

WmPublic. Writes the specified byte array to a file.

---

### Input Parameters

---

<i>fileName</i>	<b>String</b> The absolute path name of the file to which to write the byte array.
<i>bytes</i>	<b>Byte[]</b> The byte array to write to the file specified in the <i>filename</i> parameter.
<i>append</i>	<b>String</b> Optional. Specifies whether to append or overwrite if the specified file already exists. The default behavior is to create a new file if file does not exist or overwrite an existing file if file already exists.  Set to: <ul style="list-style-type: none"><li>■ <code>true</code> to append to the file if the file already exists.</li><li>■ <code>false</code> to overwrite the file if the file already exists.</li></ul>

---

### Output Parameters

---

<i>length</i>	<b>String</b> Number of bytes written to the file.
---------------	--

### Usage Notes

For security reasons, the `pub.file:bytesToFile` service checks the input *fileName* parameter against the list of `allowedWritePaths` values specified in the `fileAccessControl.cnf` file. If the input *fileName* is not on the allowed list, Integration Server throws an exception. For information about configuring the `fileAccessControl.cnf` file, refer to "[File Access Control Configuration for the pub.file Services](#)" on page 328.

---

## pub.file:checkFileExistence

WmPublic. Checks if a specified file exists, and if the file exists, checks whether the file name represents a file or a directory.

---

### Input Parameters

---

<i>fileName</i>	<b>String</b> The absolute path name of the file to be checked.
-----------------	---

---

## Output Parameters

---

<i>exists</i>	<p><b>String</b> Indicates whether the specified <i>fileName</i> exists or not. A value of:</p> <ul style="list-style-type: none"> <li>■ <code>true</code> indicates that the specified <i>fileName</i> exists.</li> <li>■ <code>false</code> indicates that the specified <i>fileName</i> does not exist.</li> </ul>
<i>isDirectory</i>	<p><b>String</b> Indicates whether the specified <i>fileName</i> is a file or a directory.</p> <ul style="list-style-type: none"> <li>■ <code>true</code> indicates that the specified <i>fileName</i> is a directory.</li> <li>■ <code>false</code> indicates that the specified <i>fileName</i> is a file.</li> </ul>

### Usage Notes

For security reasons, the `pub.file:checkFileExistence` service checks the input *fileName* parameter against the list of `allowedReadPaths` values specified in the `fileAccessControl.cnf` file. If the input *fileName* is not on the allowed list, Integration Server throws an exception. For information about configuring the `fileAccessControl.cnf` file, refer to "[File Access Control Configuration for the pub.file Services](#)" on page 328.

---

## pub.file:copyFile

WmPublic. Copies a file from one directory to another.

If a file with the same name as the file being copied exists in the *targetDirectory*, the `pub.file:copyFile` service throws an error on execution.

### Input Parameters

---

<i>fileName</i>	<p><b>String</b> The absolute path name of the file to be copied.</p>
<i>targetDirectory</i>	<p><b>String</b> Directory to which the file specified in the <i>fileName</i> parameter will be copied.</p>
<i>appendTimestamp</i>	<p><b>String</b> Optional. Specifies whether the current timestamp will be appended to the target filename. Set to:</p> <ul style="list-style-type: none"> <li>■ <code>true</code> to append a timestamp to the target <i>fileName</i> in the <code>yyyyMMddHHmmss</code> format.</li> <li>■ <code>false</code> to omit a timestamp from the target <i>fileName</i>.</li> </ul>

---

## Output Parameters

---

*targetFileName*      **String** The absolute path name of the target file to which the *fileName* parameter is copied.

### Usage Notes

For security reasons, the `pub.file:copyFile` service checks the input *fileName* against the list of `allowedReadPaths` and the input *targetDirectory* against the list of `allowedWritePaths` specified in the `fileAccessControl.cnf` file. If the file name or directory is not specified in the respective allowed lists, Integration Server throws an exception. For information about configuring the `fileAccessControl.cnf` file, refer to ["File Access Control Configuration for the pub.file Services"](#) on page 328.

---

## pub.file:deleteFile

WmPublic. Deletes the specified file.

### Input Parameters

---

*fileName*      **String** The absolute path name of the file to be deleted.

### Output Parameters

---

*status*      **String** Specifies the status of the delete operation. A value of:

- `true` indicates that the deletion succeeded.
- `false` indicates that the deletion failed.

### Usage Notes

For security reasons, the `pub.file:deleteFile` service checks the input *fileName* parameter against the list of `allowedDeletePaths` values specified in the `fileAccessControl.cnf` file. If the input *fileName* is not on the allowed list, Integration Server throws an exception. For information about configuring the `fileAccessControl.cnf` file, refer to ["File Access Control Configuration for the pub.file Services"](#) on page 328.

---

## pub.file:getFile

WmPublic. Retrieves a specified file from the local file system.

If the file contains an XML document, you can use the services in the [XML Folder](#) to convert it to an XML node.

### Input Parameters

---

<i>filename</i>	<b>String</b> The absolute path name of the file in the local file system (for example <code>c:\rubicon\document.xml</code> ).
<i>loadAs</i>	<p><b>String</b> Optional. Form in which you want the <code>getFile</code> service to make the contents of the file available to subsequent services. Set to:</p> <ul style="list-style-type: none"> <li>■ <code>bytes</code> to return the file as a byte array. Use this option if the contents of the file will be used as input to a service that operates on whole documents (for example, <code>pub.xml:queryXMLNode</code>). This is the default.</li> <li>■ <code>stream</code> to return the file as an input stream. Use this option if the contents of the file will be used as input to a service that can process a document incrementally (for example, <code>pub.xml:getXMLNodeIterator</code>).</li> <li>■ <code>string</code> to return the file as a string.</li> <li>■ <code>reader</code> to return the file as a reader.</li> </ul>
<i>encoding</i>	<b>String</b> Optional. Character set in which the file is encoded. Specify an IANA-registered character set (for example, ISO-8859-1). This information is required to correctly convert the String object to bytes when performing a <code>get</code> . If no value is specified or if the value is set to <code>autoDetect</code> , the service uses the default operating system encoding. If you specify an unsupported encoding, the system throws an exception.
<i>bufferSize</i>	<b>String</b> Optional. Buffer size (in bytes) to use if you are loading an <code>InputStream</code> (that is, <code>loadAs=stream</code> ). The default is 4096 bytes.

### Output Parameters

---

<i>body</i>	<b>Document</b> Document (IData object) containing the file as a <code>byte[]</code> , <code>InputStream</code> , <code>string</code> , or <code>reader</code> . The <i>body</i> parameter will contain one of the following keys, depending on how the <i>loadAs</i> parameter was set:
-------------	--

Key	Description
-----	-------------

<i>bytes</i>	<b>byte[ ]</b> Conditional. Returns file contents in a byte array if the <i>loadAs</i> parameter is set to <i>bytes</i> .
<i>stream</i>	<b>java.io.InputStream</b> Conditional. Returns file contents as an InputStream if the <i>loadAs</i> parameter is set to <i>stream</i> .
<i>reader</i>	<b>java.io.Reader</b> Conditional. Returns file contents as a reader if the <b>loadAs</b> parameter is set to <i>reader</i> .
<i>string</i>	<b>String</b> Conditional. Returns file contents as a string if the <i>loadAs</i> parameter is set to <i>string</i> .

### Usage Notes

The `getFile` service *does not* automatically generate an XML node from the contents of the file. To generate an XML node, the output from this service must be passed to the [pub.xml:xmlStringToXMLNode](#) service.

The `pub.file:getFile` service first checks the value of the `watt.security.pub.getFile.checkReadAllowed` server configuration parameter to determine if the service must check the `allowedReadPaths` parameter in the `fileAccessControl.cnf` to retrieve the specified file.

- If `watt.security.pub.getFile.checkReadAllowed` is set to `true`, the `pub.file:getFile` service checks the contents of the `allowReadPaths` parameter for the specified *filename*. If the *filename* appears in the `allowReadPaths` list, the `pub.file:getFile` service retrieves the file. If the specified *filename* is not on the `allowReadPaths` list, the service throws an exception.
- If `watt.security.pub.getFile.checkReadAllowed` is set to `false` the `pub.file:getFile` service retrieves the specified file without checking the `allowReadPaths` parameter in the `fileAccessControl.cnf`.

For information about configuring the `fileAccessControl.cnf`, refer to "[File Access Control Configuration for the pub.file Services](#)" on page 328.

### See Also

[pub.io:close](#)

---

## pub.file:listFiles

WmPublic. List all the files in a specified directory.

The `pub.file:listFiles` service does not list subdirectories or recursively list subdirectory contents.

---

### Input Parameters

---

<i>directory</i>	<b>String</b> [ ] Name of the directory that is to be queried.
<i>filter</i>	<b>String</b> Optional. Filter string for the file list (for example, *.java or EDI*.tx*).

### Output Parameters

---

<i>fileList</i>	<b>String</b> Names of the files in the directory specified in the <i>directory</i> parameter. Only the file names are returned; not the full path name.  <i>fileList</i> does not list subdirectories or recursively list subdirectory content.
<i>numFiles</i>	<b>String</b> Number of files in the specified directory.

### Usage Notes

For security reasons, the `pub.file:listFiles` service checks the value of the *directory* parameter against the list of `allowedReadPaths` values specified in the `fileAccessControl.cnf` file. If the specified directory is not on the allowed list, the service throws an exception. For information about configuring the `fileAccessControl.cnf` file, refer to "[File Access Control Configuration for the pub.file Services](#)" on page 328.

---

## pub.file:moveFile

WmPublic. Moves a file from one directory to another.

### Input Parameters

---

<i>fileName</i>	<b>String</b> The absolute path name of the file to be moved.
<i>targetDirectory</i>	<b>String</b> Name of the directory to which the file specified in the <i>fileName</i> parameter is to be moved.
<i>appendTimeStamp</i>	<b>String</b> Optional. Specifies whether the current timestamp is to be appended to the file name after the file is moved to the target directory. Set to: <ul style="list-style-type: none"> <li>■ <code>false</code> if you do not want to append a timestamp. This is the default.</li> </ul>

- `true` if you want to append a timestamp in the following format: `yyyyMMddHHmmss`.

### Output Parameters

---

<i>status</i>	<b>String</b> Status of the move. A value of: <ul style="list-style-type: none"> <li>■ <code>true</code> indicates that the move was successful.</li> <li>■ <code>false</code> indicates that the move failed.</li> </ul>
<i>targetFileName</i>	<b>String</b> Fully qualified path of the target file.

### Usage Notes

For security reasons, the `pub.file:moveFile` service checks the input `fileName` parameter against the list of `allowedReadPaths` and the input `targetDirectory` against the list of `allowedWritePaths` specified in the `fileAccessControl.cnf` file. If the provided file name or directory is not specified in the respective allowed lists, Integration Server throws an exception. For information about configuring the `fileAccessControl.cnf` file, see ["File Access Control Configuration for the pub.file Services" on page 328](#).

If the service cannot move the file by using the default move operation in Java, the service copies the file from the source directory to the destination directory, and then removes the file from the source directory.

If a file with the same name as the `fileName` parameter exists in the target directory, the `pub.file:moveFile` service throws an error.

---

## pub.file:readerToFile

WmPublic. Reads data from a `java.io.Reader` object and writes it to a file.

### Input Parameters

---

<i>fileName</i>	<b>String</b> Fully qualified name of the file to which the data is to be written.
<i>reader</i>	<b>java.io.Reader</b> The reader object from which the data is read.
<i>append</i>	<b>String</b> Optional. Specifies whether to append to or overwrite the file if it already exists. The default behavior is to create a new file if the file does not exist or overwrite an existing file if the file already exists.  Set to:

- `true` to append if the file already exists.
- `false` to overwrite if the file already exists.

*encoding* **String** Optional. Name of a registered, IANA character set (for example, ISO-8859-1).

If you specify an unsupported encoding, the system throws an exception. If no value is specified or if the encoding is set to `autoDetect`, the default operating system encoding is used.

### Output Parameters

---

None.

### Usage Notes

For security reasons, the `pub.file:readerToFile` service checks the input *fileName* parameter against the list of `allowedWritePaths` values specified in the `fileAccessControl.cnf` file. If the input *fileName* is not on the allowed list, the service throws an exception. For information about configuring the `fileAccessControl.cnf` file, refer to ["File Access Control Configuration for the pub.file Services" on page 328](#).

The `readerToFile` service *does not* automatically close the reader object. To close the reader, use the `pub.io:close` service.

## pub.file:streamToFile

---

WmPublic. Writes the data in the `InputStream` to a file.

### Input Parameters

---

*fileName* **String[]** Fully qualified name of the file to be written to.

*stream* **java.io.InputStream** The stream from which data is read.

*append* **String** Optional. Specifies whether to append or overwrite if the specified file already exists. The default behavior is to create a new file if the file does not exist or overwrite an existing file if the file already exists.

Set to:

- `true` to append if the file already exists.
- `false` to overwrite if the file already exists.

---

## Output Parameters

---

None.

### Usage Notes

For security reasons, the `pub.file:streamToFile` service checks the input `fileName` parameter against the list of `allowedWritePaths` values specified in the `fileAccessControl.cnf` file. If the input `fileName` is not on the allowed list, an exception is thrown. For information about configuring the `fileAccessControl.cnf` file, refer to ["File Access Control Configuration for the pub.file Services" on page 328](#).

The `streamToFile` service *does not* automatically close the stream object. To close the input stream, use the `pub.io:close` service.

---

## pub.file:stringToFile

WmPublic. Writes text to a file.

---

### Input Parameters

---

<i>fileName</i>	<b>String[]</b> Fully qualified name of the file to be written to.
<i>data</i>	<b>String</b> Text to be written.
<i>append</i>	<b>String</b> Optional. Specifies whether to append or overwrite if the specified file already exists. The default behavior is to create a new file if the file does not exist or overwrite an existing file if the file already exists.  Set to: <ul style="list-style-type: none"><li>■ <code>true</code> to append if the file already exists.</li><li>■ <code>false</code> to overwrite if the file already exists.</li></ul>
<i>encoding</i>	<b>String</b> Optional. Name of a registered, IANA character set (for example, ISO-8859-1).  If you specify an unsupported encoding, the system throws an exception. If no value is specified or if the encoding is set to <code>autoDetect</code> , the default operating system encoding is used.

---

### Output Parameters

---

None.

**Usage Notes**

For security reasons, the `pub.file:stringToFile` service checks the input `fileName` parameter against the list of `allowedWritePaths` values specified in the `fileAccessControl.cnf` file. If the input `fileName` is not on the allowed list, an exception is thrown. For information about configuring the `fileAccessControl.cnf` file, refer to ["File Access Control Configuration for the pub.file Services"](#) on page 328.

# 10 Flat File Folder

---

■ Summary of Elements in the Flat File Folder .....	342
---	-----

Use the elements in the Flat File folder to convert between Flat File documents and IS documents (IData objects), and to manage dictionary entries, entire flat file dictionaries, and flat file schemas.

## Summary of Elements in the Flat File Folder

The following elements are available in this folder:

Element	Package and Description
<a href="#">pub.flatFile:convertToString</a>	WmFlatFile. Converts an IS document (IData object) to a flat file document based on the flat file schema that you specify.
<a href="#">pub.flatFile:convertToValues</a>	WmFlatFile. Converts a flat file document to an IS document (IData object) based on the input flat file schema.
<a href="#">pub.flatFile:FormatService</a>	WmFlatFile. Service that formats the field String in a flat file schema or dictionary and ensures that the value of the String meets the format restrictions of the format service.
<a href="#">pub.flatFile:getSupportedEncodings</a>	WmFlatFile. Returns a list of supported encodings. This service will only report webMethods encodings, not Java defaults. That is, if you do not have converters.jar installed, it returns null.
<a href="#">pub.flatFile.generate:createDocumentType</a>	WmFlatFile. Creates an IS document type that defines the XML representation of a flat file schema.
<a href="#">pub.flatFile.generate:createFFDictionary</a>	WmFlatFile. Creates an empty flat file dictionary. This service throws an exception if the flat file dictionary you want to create

Element	Package and Description
	already exists when the service is invoked.
<a href="#">pub.flatFile.generate:deleteFFDictionary</a>	WmFlatFile. Deletes a flat file dictionary.
<a href="#">pub.flatFile.generate:deleteFFDictionaryEntry</a>	WmFlatFile. Deletes a single entry from a flat file dictionary.
<a href="#">pub.flatFile.generate:deleteFFSchema</a>	WmFlatFile. Deletes a flat file schema.
<a href="#">pub.flatFile.generate:FFDictionary</a>	WmFlatFile. This IS document type defines the format to use when supplying a flat file dictionary or dictionary entry (in the <i>FFXML</i> variable) and the format that services return (in the <i>FFXML</i> variable) when you are retrieving a flat file dictionary or dictionary entry.
<a href="#">pub.flatFile.generate:FFSchema</a>	WmFlatFile. This IS document type defines the format to use when supplying a flat file schema (in the <i>FFXML</i> variable) and the format that services return (in the <i>FFXML</i> variable) when you are retrieving a flat file schema.
<a href="#">pub.flatFile.generate:findDependants</a>	WmFlatFile. Returns the names of all flat file schemas and dictionaries that are dependent on a given flat file dictionary.
<a href="#">pub.flatFile.generate:findReferences</a>	WmFlatFile. Returns the names of all flat file dictionaries that are referenced by a given flat file dictionary or flat file schema.
<a href="#">pub.flatFile.generate:getFFDictionaryAsXML</a>	WmFlatFile. Returns a dictionary as an XML string.

Element	Package and Description
<a href="#">pub.flatFile.generate:getFFDictionaryEntryAsXML</a>	WmFlatFile. Returns a single dictionary entry as an XML string.
<a href="#">pub.flatFile.generate:getFFSchemaAsXML</a>	WmFlatFile. Returns the specified flat file schema as an XML string.
<a href="#">pub.flatFile.generate:listFFDictionaryEntries</a>	WmFlatFile. Lists all entries in a specified flat file dictionary that are of a specified type.
<a href="#">pub.flatFile.generate:saveXMLAsFFDictionary</a>	WmFlatFile. Creates a flat file dictionary in the Integration Server namespace by converting the specified flat file dictionary that is in XML format into a namespace flat file dictionary.
<a href="#">pub.flatFile.generate:saveXMLAsFFSchema</a>	WmFlatFile. Creates a flat file schema in the Integration Server namespace by converting the specified flat file schema that is in XML format into a namespace flat file schema.
<a href="#">pub.flatFile.generate:updateFFDictionaryEntryFromXML</a>	WmFlatFile. Updates one or more entries in a flat file dictionary in the Integration Server namespace.

## pub.flatFile:convertToString

WmFlatFile. Converts an IS document (IData object) to a flat file document based on the flat file schema that you specify.

By default, this service returns the document as a string, but you can set a flag to optionally return the document as a byte array instead.

**Note:** This service does not validate the document.

### Input Variables

*ffValues*      **Document** The IData object representing the flat file.

*ffSchema* **String** Namespace name of the flat file schema to use to convert the given IS document to a string.

*spacePad* **String** Optional. How to position the records in the flat file.

Value	Description
left	Left justify the records (add blank spaces to the right of the records) before the records are written to the output. This is the default.
right	Right justify the records (add blank spaces to the left of the records) before the records are written to the output.
none	No spaces added.

*signalError* **String** Whether to create errors in the output.

Value	Description
false	Do not create errors in output.
true	Create errors in output.

If you are upgrading from webMethods Integration Server version 4.6, to enable left or right justification you must add the following line to the *Integration Server\_directory\instances\instance\_name\packages\WmFlatFile\config\ff* file:

```
spacePadJustifies=false
```

Then, reload the WmFlatFile package so that this configuration setting will take effect. For details, see the *Flat File Schema Developer's Guide* or *webMethods Service Development Help*.

*noEmptyTrailingFields* **String** Whether trailing empty fields are to be removed from the output. Used only with records that have delimited fields.

Value	Description
true	Trailing empty fields will be removed from the output. For example, if it is set to true, the output for a record with empty trailing fields looks like

the following: AAA\*01\*02! (where ! is used as segment terminator). This is the default.

*false* A field separator remains to denote an empty field. For example, if it is set to *false*, the output for a record with empty trailing fields looks like the following: AAA\*01\*02\*\*\*\*\*! (where ! is used as segment terminator).

### *noEmptyTrailingSubFields*

**String** Whether trailing empty subfields are to be removed from the output. Used only with records that have delimited fields.

If no value is specified for the *noEmptyTrailingSubFields* parameter, Integration Server uses the value set for the *noEmptyTrailingFields* parameter.

<b>Value</b>	<b>Description</b>
<i>true</i>	Trailing empty subfields will be removed from the output.
<i>false</i>	A field separator remains to denote an empty subfield.

### *delimiters*

**Document** Optional. The separator characters used to construct the output string. To specify a delimiter, you can specify:

- One character or character representation (for example, \*, \n for line terminator, \t for tab)
- Hexidecimal value with prefix "0X" (for example, 0X09, 0X13)
- Octal value with prefix "0" or decimal value (for example, 009, 013)
- Unicode characters (for example, \uXXXX where XXXX represents the Unicode value of the character)

<b>Value</b>	<b>Description</b>
<i>record</i>	<b>String</b> Character to use to separate records. If you want to specify the two-character carriage return line feed (CRLF) characters, specify \r\n.
<i>field</i>	<b>String</b> Character to use to separate fields.
<i>subfield</i>	<b>String</b> Character to use to separate subfields.

<i>release</i>	<b>String</b> Character to use to ignore a <i>record</i> , <i>field</i> , or <i>subfield</i> delimiter in a field. If a release character occurs in a field or subfield before the delimiter, it will be prefixed with <i>release</i> before being written to the output <i>string</i> .
<i>quotedRelease</i>	<b>String</b> Character to use to ignore a <i>record</i> , <i>field</i> , or <i>subfield</i> delimiter in a field. If a quoted release character occurs in a field or subfield before the delimiter, it will be prefixed with <i>quotedRelease</i> before being written to the output <i>string</i> . The string is pre- and appended with the quoted release character.  For example, if * is a delimiter, the field value is a*b, and the quoted release character is ", the string appears as "a*b".
<i>FormatInfo</i>	<b>Document</b> Any values mapped to the <i>FormatInfo</i> variable will be passed unmodified to all format services invoked by <code>convertToString</code> and <code>convertToValues</code> .
<i>outputFileName</i>	<b>String</b> Optional. If you want the output returned in a file instead of in the <i>string</i> output variable, provide the name of the file you want created as a result of this service.
<i>Encoding</i>	<b>String</b> The type of encoding used to write data to the output file. The default encoding is UTF-8.
	<b>Note:</b> If the flat file document contains multi-byte encodings, you must install the Extended Character Set using the Software AG Installer. For more information about installing the Extended Character Set, see the "Configuring the flat file byte count parser" section of the <i>Flat File Schema Developer's Guide</i> .
<i>sortInput</i>	<b>String</b> Optional. Whether you want the service to sort the input records to match the flat file schema specified in <i>ffSchema</i> . You should specify <code>true</code> for <i>sortInput</i> if the data in <i>ffValues</i> is not in the same order as defined by <i>ffSchema</i> .

<u>Value</u>	<u>Description</u>
<code>true</code>	You want the service to sort the input records to match the flat file schema.

If you select to sort the input records, note that:

- The service will run slower.
- All undefined records will be sorted *after* the defined records.
- The order of the undefined records appear in the final document is random.

If there are multiple records at the same level with the same name, the order they appear in the final document is random.

<i>returnAsBytes</i>	<b>Value</b>	<b>Description</b>
	false	You do not want the service to sort the input records to match the flat file schema. The input records must match the order of the flat file schema. This is the default.
	false	Returns the document as a string. This is the default.
	true	Returns the document as a byte array instead of a string. This setting is useful (but optional) when parsing multi-byte encodings.

## Output Variables

<i>string</i>	<b>String</b> Data that represents the flat file document.
<i>bytes</i>	<b>Object</b> If the input variable <i>returnAsBytes</i> = <code>true</code> , returns the output as a byte array encoded using the specified encoding. The string value is not returned.
<i>errorArray</i>	<b>Object</b> String array containing messages pertaining to errors that occurred during conversion. If no errors are encountered, this contains a value of <code>null</code> .

## Usage Note

When the `pub.flatFile:convertToString` service executes, the field that is defined to start after the end of the fixed length record will not be included in the output data if the following conditions are met:

- The flat file schema uses a fixed length record delimiter.

- The flat file schema contains a fixed position field that begins beyond the defined length of the fixed length record.
- The input to the `pub.flatFile:convertToString` service contains a value for the fixed position field that begins beyond the defined length of the fixed length record.

---

## pub.flatFile:convertToValues

`WmFlatFile`. Converts a flat file document to an IS document (IData object) based on the input flat file schema.

### Input Variables

---

<i>ffData</i>	<b>Object</b> The flat file input with type of String, InputStream, or ByteArray.
<i>ffSchema</i>	<b>String</b> The full name of the flat file schema object used to parse the <i>ffData</i> object.
<i>ffIterator</i>	<b>Object</b> Optional. An object that encapsulates and keeps track of the input data during processing. It is used only when the <i>iterate</i> variable has been set to true.
<i>encoding</i>	<b>String</b> Optional. The encoding of the InputStream passed in to <i>ffData</i> . The default encoding is UTF-8.
	<p><b>Note:</b> If the flat file document contains multi-byte encodings, you must install the Extended Character Set using the Software AG Installer. For more information about installing the Extended Character Set, see the “Configuring the flat file byte count parser” section of the <i>Flat File Schema Developer’s Guide</i>.</p>
<i>delimiters</i>	<p><b>Document</b> Optional. An IData object that contains the segment terminator and the field and subfield separators. If the delimiter is <code>null</code>, it will be located using the information defined in the flat file schema. To specify a delimiter, you can specify:</p> <ul style="list-style-type: none"> <li>■ One character or character representation (for example, <code>*</code>, <code>\n</code> for line terminator, <code>\t</code> for tab)</li> <li>■ Hexidecimal value with prefix “0x” (for example, <code>0x09</code>, <code>0x13</code>)</li> <li>■ Octal value with prefix “0” or decimal value (for example, <code>011</code>, <code>023</code>)</li> </ul>

- Unicode characters (for example, `\uXXXX` where `XXXX` represents the Unicode value of the character)
- The space character.

**Important** If you specify one delimiter value, you must specify *all* values. Specifying one of these values will override any information set in the flat file schema.

Variable	Description
<i>record</i>	<b>String</b> Character used to separate records. If you want to specify the two-character carriage return line feed (CRLF) characters, specify <code>\r\n</code> .
<i>field</i>	<b>String</b> Character used to separate fields.
<i>subfield</i>	<b>String</b> Character used to separate subfields.
<i>release</i>	<b>String</b> Character used to ignore a <i>record</i> , <i>field</i> , or <i>subfield</i> delimiter in a field. If a release character occurs in a field or subfield before the delimiter, it will be prefixed with the <i>release</i> before being written to the output <i>ffValues</i> .
<i>quotedRelease</i>	<b>String</b> Character to use to ignore a <i>record</i> , <i>field</i> , or <i>subfield</i> delimiter in a field. If a quoted release character occurs in a field or subfield before the delimiter, it will be prefixed with <i>quotedRelease</i> before being written to the output <i>string</i> . The string is pre- and appended with the quoted release character.  For example, if <code>*</code> is a delimiter, the field value is <code>a*b</code> , and the quoted release character is <code>"</code> , the string appears as <code>"a*b"</code> .
<i>FormatInfo</i>	<b>Document</b> Any values mapped to the <i>FormatInfo</i> variable will be passed unmodified to all format services invoked by <code>convertToString</code> and <code>convertToValues</code> .
<i>iterate</i>	<b>String</b> Optional. Whether you want to process the input all at one time.
Value	Description

true	Processes top level records (children of the document root) in the flat file schema one at a time. After all child records of the top level record are processed, the iterator moves to the top level of the next record in the flat file schema, until all records are processed.
false	Processes all input data at one time. This is the default.

*createIfNull* **String** Optional. Whether to create the IData object if all the fields are null.

<u>Value</u>	<u>Description</u>
true	Always create IS document even though all the fields are null.
false	No IS document (IData object) will be created if all the fields are null. This is the default.

*skipWhiteSpace* **String** Optional. Whether white space at the beginning of records will be ignored.

**Note:** The fixed length record parser ignores *skipWhiteSpace* ; it preserves white space.

<u>Value</u>	<u>Description</u>
true	Ignore white spaces at the beginning of a record. This is the default.
false	Record is used as it is identified (useful for positional data record).

*keepResults* **String** Optional. Whether to return the parsed data in the *ffValues* output parameter.

<u>Value</u>	<u>Description</u>
true	The parsed <i>ffData</i> will be returned in the output <i>ffValues</i> . This is the default.

---

	<p><code>false</code> <i>ffValues</i> will not return data. Use this option when validating the structure of the <i>ffData</i> against the given flat file schema.</p>								
<i>validate</i>	<p><b>String</b> Optional. Whether to return error messages that describe how <i>ffData</i> differs from the flat file schema.</p> <table> <thead> <tr> <th><u>Value</u></th> <th><u>Description</u></th> </tr> </thead> <tbody> <tr> <td><code>true</code></td> <td>Do not return error messages describing how <i>ffData</i> differs from the specified flat file schema. This is the default.</td> </tr> <tr> <td><code>false</code></td> <td>Return errors describing how the given <i>ffData</i> violates the constraints described in the flat file schema.</td> </tr> </tbody> </table>	<u>Value</u>	<u>Description</u>	<code>true</code>	Do not return error messages describing how <i>ffData</i> differs from the specified flat file schema. This is the default.	<code>false</code>	Return errors describing how the given <i>ffData</i> violates the constraints described in the flat file schema.		
<u>Value</u>	<u>Description</u>								
<code>true</code>	Do not return error messages describing how <i>ffData</i> differs from the specified flat file schema. This is the default.								
<code>false</code>	Return errors describing how the given <i>ffData</i> violates the constraints described in the flat file schema.								
<i>returnErrors</i>	<p><b>String</b> Optional. Whether to return the validation errors. Validation errors are returned only if <i>validate</i> is set to true.</p> <table> <thead> <tr> <th><u>Value</u></th> <th><u>Description</u></th> </tr> </thead> <tbody> <tr> <td><code>asArray</code></td> <td>Return any validation errors with the <i>ffData</i> in an array called <i>errors</i> . This is the default.</td> </tr> <tr> <td><code>inResults</code></td> <td>Return validation errors in the <i>ffValues</i> object.</td> </tr> <tr> <td><code>both</code></td> <td>Return validation errors in both <i>errors</i> and <i>ffValues</i> .</td> </tr> </tbody> </table>	<u>Value</u>	<u>Description</u>	<code>asArray</code>	Return any validation errors with the <i>ffData</i> in an array called <i>errors</i> . This is the default.	<code>inResults</code>	Return validation errors in the <i>ffValues</i> object.	<code>both</code>	Return validation errors in both <i>errors</i> and <i>ffValues</i> .
<u>Value</u>	<u>Description</u>								
<code>asArray</code>	Return any validation errors with the <i>ffData</i> in an array called <i>errors</i> . This is the default.								
<code>inResults</code>	Return validation errors in the <i>ffValues</i> object.								
<code>both</code>	Return validation errors in both <i>errors</i> and <i>ffValues</i> .								
<i>maxErrors</i>	<p><b>String</b> Optional. The maximum number of errors that can be returned from one record. When the flat file parser encounters more than the maximum number of errors within a record, the parser will stop parsing and return the parsed data and errors processed up until that point. Validation errors are returned only if <i>validate</i> is set to <code>true</code>.</p>								
<i>flags</i>	<p><b>String</b> Optional. Flags that you can set to govern <code>convertToValues</code> options.</p> <table> <thead> <tr> <th><u>Variable</u></th> <th><u>Description</u></th> </tr> </thead> <tbody> <tr> <td><i>addRecordCount</i></td> <td><b>String</b> Whether you want the service to add an additional field (<i>@record-count</i> ) to each parsed record in the resulting IData object (<i>ffValues</i> ).</td> </tr> </tbody> </table>	<u>Variable</u>	<u>Description</u>	<i>addRecordCount</i>	<b>String</b> Whether you want the service to add an additional field ( <i>@record-count</i> ) to each parsed record in the resulting IData object ( <i>ffValues</i> ).				
<u>Variable</u>	<u>Description</u>								
<i>addRecordCount</i>	<b>String</b> Whether you want the service to add an additional field ( <i>@record-count</i> ) to each parsed record in the resulting IData object ( <i>ffValues</i> ).								

The *@record-count* field is used to identify the record number of each parsed record.

<b>Value</b>	<b>Description</b>
<code>true</code>	The <i>@record-count</i> field is added to each parsed record. This field contains the number of the parsed record. The first parsed record is 1, the second is 2, etc. If there are records that are undefined data, the count of the next defined record will reflect the undefined data. For example, if the <i>@record-count</i> field for a record is 2 and that record contains 5 undefined records, the <i>@record-count</i> field for the next defined record will be 8.
<code>false</code>	The <i>@record-count</i> field is not added to each parsed record. This is the default.

*detailedErrors*

**String** Whether you want detailed conditional validation error information. This flag is only used when *validate* is `true`.

<b>Value</b>	<b>Description</b>
<code>true</code>	When a conditional validation error occurs, the output <i>errors</i> variable will contain detail information about all the conditions that were violated. For more information, see <i>Flat File Schema Developer's Guide</i> .
<code>false</code>	When a conditional validation error occurs, the service does <i>not</i> provide detail error information. Conditional validators report only whether a condition failed validation with no additional information about the conditions that were violated. This is the default.

*skipToFirstRecord*

**String** Whether you want the service to wait until it finds the first valid record before reporting invalid records as errors.

<u>Value</u>	<u>Description</u>
true	The service will wait until it finds the first valid record before reporting invalid records as errors. This is the default.
false	The service will report invalid records as errors prior to locating the first valid record.

*trimWhitespace* **String** Whether you want the service to delete any blank spaces at the beginning of fields, at the end of fields, or both.

<u>Value</u>	<u>Description</u>
none	The service will not delete any blank spaces from fields. This is the default.
left	The service will delete all blank spaces at the beginning of all fields.
right	The service will delete all blank spaces at the end of all fields.
both	The service will delete all blank spaces at the beginning and end of all fields.

*resultAsArray* **String** Whether you want the service to return the *ffValues* output parameter as an `IData[]` that can be mapped to the document types generated from the schema. An `IData[]` is a document List. The *resultAsArray* parameter is used only when the *iterate* input parameter is set to `true`.

<u>Value</u>	<u>Description</u>
false	The service returns the <i>ffValues</i> output parameter as an <code>IData[]</code> that can be mapped to the document types generated from the schema. This is the default.

`true` The service returns the *ffValues* output parameter as an IData object and not as an IData[].

## Output Variables

---

<i>ffValues</i>	<b>Document</b> The IData object that represents the input flat file data.						
<i>ffIterator</i>	<b>Object</b> Optional. An object that encapsulates and keeps track of the input records during processing. It is used only when the <i>iterate</i> variable has been set to <code>true</code> . When all input data has been processed, the object becomes <code>null</code> . When the <i>ffIterator</i> variable is <code>null</code> , you should exit the LOOP to discontinue processing.						
<i>isValid</i>	<b>String</b> Whether flat file contains validation errors.						
	<table> <thead> <tr> <th>Value</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td><code>true</code></td> <td>The <i>validate</i> input variable was set to <code>true</code> and no errors were found.</td> </tr> <tr> <td><code>false</code></td> <td>The <i>validate</i> input variable was set to <code>true</code> and errors were found, or the <i>validate</i> input variable was set to <code>false</code>.</td> </tr> </tbody> </table>	Value	Description	<code>true</code>	The <i>validate</i> input variable was set to <code>true</code> and no errors were found.	<code>false</code>	The <i>validate</i> input variable was set to <code>true</code> and errors were found, or the <i>validate</i> input variable was set to <code>false</code> .
Value	Description						
<code>true</code>	The <i>validate</i> input variable was set to <code>true</code> and no errors were found.						
<code>false</code>	The <i>validate</i> input variable was set to <code>true</code> and errors were found, or the <i>validate</i> input variable was set to <code>false</code> .						
<i>errors</i>	<b>String</b> Optional. An array containing the validation errors, if any, that were found in <i>ffData</i> . For more information about validation error codes see <i>Flat File Schema Developer's Guide</i> .						

## Usage Note

---

If you specified a default record definition by which the `pub.flatFile:convertToValues` service parses the IS document (IData object), the service displays the resulting *recordWithNoID* document as a child of the document above it, in an array.

To display the *recordWithNoID* record as a child of the root, change the value of the `recWithNoIDLike46` to `true` in the *Integration Server\_directory\instances\instance\_name\packages\WmFlatFile\config\ff* file and reload the WmFlatFile package so that this configuration setting will take effect. For more information, see the *Flat File Schema Developer's Guide*.

## pub.flatFile:FormatService

WmFlatFile. Service that formats the field String in a flat file schema or dictionary and ensures that the value of the String meets the format restrictions of the format service.

Use this specification when you create format services for fields in a flat file schema or dictionary. The format service is invoked for a field when the [pub.flatFile:convertToValues](#) and [pub.flatFile:convertToString](#) services are invoked. You create a format service to format the field String and ensure that the value of the String meets the format restrictions of the format service. When creating a particular format service for use with the **Format Service** property in a flat file schema or dictionary, the service you select must implement the [pub.flatFile:FormatService](#) specification (located on its **Input/Output** tab).

**Important:** If a particular field does not have a value (that is, a value is not returned in the IS document (IData object) for the [pub.flatFile:convertToValues](#) service or is not present in the input data for the [pub.flatFile:convertToValues](#) service) the format service assigned to that field will not be executed.

### Input Variables

*value*                    **String** The field value to format.

*direction*            **String** Indicates the type of formatting to be applied to the field. Specify one of the following:

Value	Description
<code>convertToString</code>	This field is in an outbound document and needs its internal format converted to its external format.
<code>convertToValues</code>	This field is in an inbound document and needs its external format converted to its internal format.

*validate*                **String** The value of the input parameter *validate* from the [pub.flatFile:convertToValues](#) service.

<code>true</code>	The value of the <code>convertToValuesvalidate</code> parameter is <code>true</code> (validate).
<code>false</code>	The value of the <code>convertToValuesvalidate</code> parameter is <code>false</code> (do not validate). This value

is always `false` when the value of the *direction* parameter is `convertToString`.

*minLength* **String** Enables you to validate the minimum length of a field. If the field is extracted via a Fixed Position Extractor, this is the number of bytes that are extracted. If the field is not extracted via the Fixed Position Extractor and a Length Validator is associated with this field, this is the minimum length that will be considered valid. Otherwise, this parameter will not be present in the pipeline.

*maxLength* **String** Enables you to validate the maximum length of a field. If the field is extracted via a Fixed Position Extractor, this is the number of bytes that are extracted. If the field is not extracted via the Fixed Position Extractor and a Length Validator is associated with this field, this is the maximum length that will be considered valid. If the maximum length is unlimited (-1) or there is no Length Validator, this parameter will not be present in the pipeline.

*FormatInfo* **Document** Information that can be used by individual formatting services. This information can be obtained from one of 3 locations:

- `convertToString` – You can specify *FormatInfo* in addition to the delimiter information for a call to this service.
- `convertToValues` – If delimiter information is explicitly passed into the `convertToValues` service, *FormatInfo* can be specified.
- From the UNEDIFACT UNA segment – The EDI document type automatically extracts the decimal separator from the UNA segment.

The only format services that use this feature are the decimal formatting services (for implied decimal and decimal formats). The *FormatInfo* IS document should contain a string called *DecimalCharacter*. If the decimal character is `'`, the number would be formatted as 100,10 (European format) instead of 100.10, as is common in the US.

**Note:** Changes to the data in this object will be reflected in all other format services that are invoked during execution of `convertToString` and `convertToValues`.

## Output Variables

---

*formattedValue* **String** The field value with appropriate formatting applied.

*meetsFormat* **String** Whether the value could be formatted properly.

	<b>Value</b>	<b>Description</b>
	true	Indicates that the value could be properly formatted.
	false	Indicates that the value could not be properly formatted.
<i>errorMessage</i>	<b>String</b>	If <i>meetsFormat</i> is false, this parameter provides a text message describing the formatting error.
<i>valueToValidate</i>	<b>String</b>	The value that will be used by the validator for this field. If this value is not present, the value passed in the input variable <i>value</i> will be validated. This field is used only for situations in which the input variable <i>validate</i> is set to true.
<i>maxLength</i>	<b>String</b>	Enables you to validate the maximum length of a field. If the field is extracted via a Fixed Position Extractor, this is the number of bytes that are extracted. If the field is not extracted via the Fixed Position Extractor and a Length Validator is associated with this field, this is the maximum length that will be considered valid. If the maximum length is unlimited (-1) or there is no Length Validator, this parameter will not be present in the pipeline.
<i>FormatInfo</i>	<b>Document</b>	Information that can be used by individual formatting services. This information can be obtained from one of 3 locations: <ul style="list-style-type: none"> <li>■ <i>convertToString</i> – You can specify <i>FormatInfo</i> in addition to the delimiter information for a call to this service.</li> <li>■ <i>convertToValues</i> – If delimiter information is explicitly passed into the <i>convertToValues</i> service, <i>FormatInfo</i> can be specified.</li> <li>■ From the UNEDIFACT UNA segment – The EDI document type automatically extracts the decimal separator from the UNA segment.</li> </ul> <p>The only format services that use this feature are the decimal formatting services (for implied decimal and decimal formats). The <i>FormatInfo</i> IS document should contain a string called <i>DecimalCharacter</i>. If the decimal character is ',' the number would be formatted as 100,10 (European format) instead of 100.10, as is common in the US.</p> <p><b>Note:</b> Changes to the data in this object will be reflected in all other format services that are invoked during execution of <i>convertToString</i> and <i>convertToValues</i>.</p>

---

## pub.flatFile:getSupportedEncodings

WmFlatFile. Returns a list of supported encodings. This service will only report webMethods encodings, not Java defaults. That is, if you do not have converters.jar installed, it returns null.

### Input Variables

---

None.

### Output Variables

---

*encodings*                      **String List** A list of supported encodings.

---

## pub.flatFile.generate:createDocumentType

WmFlatFile. Creates an IS document type that defines the XML representation of a flat file schema.

### Input Variables

---

*FlatFileSchema*                      **String** The fully-qualified name of the flat file schema for which you want to generate an IS document type.

*PackageName*                      **String** The name of the Integration Server package in which you want the created IS document type to be placed.

*DocumentType Name*                      **String** The fully-qualified name that you want to assign to the created IS document type.

### Output Variables

---

None.

### Examples

For examples of how to use this service, see the following services in the certified samples area of the Knowledge Center on the Empower Product Support website at <https://empower.softwareag.com>.

sample.flatFile.generateFFSchema:delimited

sample.flatFile.generateFFSchema:fixedLength

---

## pub.flatFile.generate:createFFDictionary

WmFlatFile. Creates an empty flat file dictionary. This service throws an exception if the flat file dictionary you want to create already exists when the service is invoked.

### Input Variables

---

*FFDictionaryName* **String** The fully-qualified name of the flat file dictionary you want to create.

*PackageName* **String** The name of the Integration Server package in which you want the created flat file dictionary to be placed.

### Output Variables

---

None.

### Examples

For examples of how to use this service, see the following services in the certified samples area of the Knowledge Center on the Empower Product Support website at <https://empower.softwareag.com>.

sample.flatFile.generateFFSchema:delimited

sample.flatFile.generateFFSchema:fixedLength

---

## pub.flatFile.generate:deleteFFDictionary

WmFlatFile. Deletes a flat file dictionary.

Before deleting the dictionary, the Integration Server determines if other dictionaries depend on the dictionary being deleted, and gives the user the option of canceling the deletion.

### Input Variables

---

*FFDictionaryName* **String** The fully qualified name of the flat file dictionary that you want to delete.

---

## Output Variables

---

*deleted* **String** Whether the flat file dictionary was successfully deleted; *deleted* will be either `true` or `false`.

<u>Value</u>	<u>Description</u>
<code>true</code>	The flat file dictionary was successfully deleted.
<code>false</code>	The flat file dictionary was <i>not</i> successfully deleted.

### Usage Note

Before you run this service, you should run the [pub.flatFile.generate.findDependants](#) service to return the names of all flat file schemas and dictionaries that are dependent on the dictionary you are deleting.

### Examples

For examples of how to use this service, see the following services in the certified samples area of the Knowledge Center on the Empower Product Support website at <https://empower.softwareag.com>.

sample.flatFile.generateFFSchema:delimited

sample.flatFile.generateFFSchema:fixedLength

---

## pub.flatFile.generate:deleteFFDictionaryEntry

WmFlatFile. Deletes a single entry from a flat file dictionary.

### Input Variables

---

*FFDictionaryName* **String** The fully-qualified name of the flat file dictionary that contains the entry that you want to delete.

*EntryName* **String** The name of the entry that you want to delete.

*EntryType* **String** The type of entry that you are deleting. Specify `Record`, `Composite`, or `Field`.

---

## Output Variables

---

*deleted* **String** Whether the flat file dictionary entry was successfully deleted; *deleted* will be either `true` or `false`.

<u>Value</u>	<u>Description</u>
<code>true</code>	The flat file dictionary entry was successfully deleted.
<code>false</code>	The flat file dictionary entry was <i>not</i> successfully deleted.

## Examples

For examples of how to use this service, see the following services in the certified samples area of the Knowledge Center on the Empower Product Support website at <https://empower.softwareag.com>.

sample.flatFile.generateFFSchema:delimited

sample.flatFile.generateFFSchema:fixedLength

---

## pub.flatFile.generate:deleteFFSchema

WmFlatFile. Deletes a flat file schema.

### Input Variables

---

*FFSchemaName* **String** The fully-qualified name of the flat file schema that you want to delete.

### Output Variables

---

*deleted* **String** Whether the flat file schema was successfully deleted; *deleted* will be either `true` or `false`.

<u>Value</u>	<u>Description</u>
<code>true</code>	The flat file schema was successfully deleted.
<code>false</code>	The flat file schema was <i>not</i> successfully deleted.

## Examples

For examples of how to use this service, see the following services in the certified samples area of the Knowledge Center on the Empower Product Support website at <https://empower.softwareag.com>.

sample.flatFile.generateFFSchema:delimited

sample.flatFile.generateFFSchema:fixedLength

---

## pub.flatFile.generate:FFDictionary

WmFlatFile. This IS document type defines the format to use when supplying a flat file dictionary or dictionary entry (in the FFXML variable) and the format that services return (in the FFXML variable) when you are retrieving a flat file dictionary or dictionary entry.

The structure for this IS document type is defined in the following XML schema:

*Integration Server\_directory*\instances\*instance\_name* \packages\WmFlatFile\pub\FFGeneration.xsd

## Variables

*FFDictionary* **Document** The dictionary entries that you want to add or update. *FFDictionary* has the following structure:

*RecordDictionary Entry* **Document List** Optional. The dictionary entries for records that you want to add or update in the flat file dictionary. Leave this null if you do not want to add or update record entries.

<u>Variable</u>	<u>Description</u>
<i>EntryName</i>	<b>String</b> The name of the record.
<i>RecordDefinition</i>	<b>Document</b> The definition of the record. The information you specify in a record definition is the same as the information that you specify when creating a flat file dictionary using the Flat File Schema Editor. For descriptions of the fields, see <i>webMethods Service Development Help</i> .

*Composite  
DictionaryEntry*

**Document List** Optional. The dictionary entries for composites that you want to add or update in the flat file dictionary. Leave this null if you do not want to add or update composite entries.

<u>Variable</u>	<u>Description</u>
<i>EntryName</i>	<b>String</b> The name of the composite.
<i>CompositeDefinition</i>	<b>Document</b> The definition of the composite. The information you specify in a composite definition is the same as the information that you specify when creating a flat file dictionary using the Flat File Schema Editor. For descriptions of the fields, see <i>webMethods Service Development Help</i> .

*FieldDictionary  
Entry*

**Document List** Optional. The dictionary entries for fields that you want to add or update in the flat file dictionary. Leave this null if you do not want to add or update field entries.

<u>Variable</u>	<u>Description</u>
<i>EntryName</i>	<b>String</b> The name of the field.
<i>FieldDefinition</i>	<b>Document</b> The definition of the field. The information you specify in a field definition is the same as the information that you specify when creating a flat file dictionary using the Flat File Schema Editor. For descriptions of the fields, see <i>webMethods Service Development Help</i> .

### Usage Notes

If you are using this IS document type to supply a flat file dictionary as input to the [pub.flatFile.generate:saveXMLAsFFDictionary](#), be sure to supply *all* dictionary

entries. If you are using this IS document type to update an existing dictionary, provide only the entries that you want to add or update and invoke the `pub.flatFile.generate:updateFFDictionaryEntryFromXML` to update the flat file dictionary.

### Examples

For examples of how to use this service, see the following services in the certified samples area of the Knowledge Center on the Empower Product Support website at <https://empower.softwareag.com>.

sample.flatFile.generateFFSchema:delimited

sample.flatFile.generateFFSchema:fixedLength

---

## pub.flatFile.generate:FFSchema

WmFlatFile. This IS document type defines the format to use when supplying a flat file schema (in the *FFXML* variable) and the format that services return (in the *FFXML* variable) when you are retrieving a flat file schema.

The structure for this IS document type is defined in the following XML schema:

*Integration Server\_directory*\instances\*instance\_name* \packages\WmFlatFile\pub\FFGeneration.xsd

### Variables

*FFSchema* **Document** The flat file schema that you want to add or update. *FFSchema* has the following structure:

<u>Variable</u>	<u>Description</u>
<i>Delimiters</i>	<b>Document</b> The delimiters used in the flat files that adhere to this flat file schema. The information that you specify for <i>Delimiters</i> corresponds to the data you specify on the <b>Flat File Definition</b> tab in the Flat File Schema Editor. For a description of the fields, see <i>webMethods Service Development Help</i> .
<i>Document Structure</i>	<b>Document</b> The structure of the flat files that adhere to this flat file schema. The information that you specify for <i>DocumentStructure</i> corresponds to the data you specify on the <b>Flat File Structure</b> tab in the Flat File Schema Editor. For a description of the fields, see <i>webMethods Service Development Help</i> .

<u>Variable</u>	<u>Description</u>
-----------------	--------------------

*Ordered*      **String** Whether the child records appear in the flat file in the order they are defined in the flat file schema.

*RecordStructure*      **Document List** Definitions of the records within the flat file.

<u>Variable</u>	<u>Description</u>
-----------------	--------------------

<i>Ordered</i>	<b>String</b> Whether the child records appear in the flat file in the order they are defined in the flat file schema.
----------------	--

<i>RecordUsage</i>	<b>Document</b> Information about how the record is used, including either the dictionary reference for this record or the definition of the record.
--------------------	--

<i>RecordStructure</i>	<b>Document List</b> Child records of this record. This is a recursive reference to the <i>RecordStructure</i> defined in <i>FFSchema/DocumentStructure</i> .
------------------------	---

*RecordParser*      **Document** The type of record parser. In this IS document, specify only the one variable that corresponds to the type of record parser to use. That is, specify one of *FixedLengthParser* , *DelimitedParser* , *VariableLengthParser* , or *EDIParser* . For *DelimitedParser* , *VariableLengthParser* , and

*EDIParser*, you do not need to specify a value; just have the variable in the pipeline.

<i>DefaultRecord Reference</i>	<b>Document</b> Optional. The dictionary name and entry name that identifies the default record for the flat file schema. If you specify a default record, when using the flat file schema to parse a flat file schema, the default record is used for any record that cannot be recognized.
<i>Record Identifier</i>	<p><b>Document</b> Where to locate the identifier to use to correlate a record in the flat file to a record definition in the flat file schema. Specify either the <i>NthFieldIdentifier</i> variable or the <i>FixedPositionIdentifier</i> variable:</p> <ul style="list-style-type: none"> <li>■ Use <i>NthFieldIdentifier</i> to identify the field in the record (counting from zero) that contains the identifier.</li> <li>■ Use <i>FixedPositionIdentifier</i> to identify the character position in the record (counting from zero) where the record identifier is located.</li> </ul>
<i>UndefinedData Allowed</i>	<p><b>String</b> Whether you want the <a href="#">pub.flatFile:convertToValues</a> service to generate undefined data errors when you use this flat file schema to convert a flat file to an IData object.</p> <ul style="list-style-type: none"> <li>■ Specify <code>true</code> if you want to allow undefined data and do not want the <a href="#">pub.flatFile:convertToValues</a> service to flag undefined data errors.</li> <li>■ Specify <code>false</code> if you do not want to allow undefined data and you do want the <a href="#">pub.flatFile:convertToValues</a> service to flag undefined data errors.</li> </ul>
<i>Document Areas</i>	<b>String List</b> Areas for this flat file schema. An area is a way to associate an arbitrary string with a given record.
<i>FloatingRecord</i>	<b>String</b> Optional. The name of the record that is defined in the schema as a floating record.
<p><b>Note:</b> If the floating record has an alternate name, specify the alternate name.</p>	

---

*Description*      **String** Description of the flat file schema.

### Examples

For examples of how to use this service, see the following services in the certified samples area of the Knowledge Center on the Empower Product Support website at <https://empower.softwareag.com>.

sample.flatFile.generateFFSchema:delimited

sample.flatFile.generateFFSchema:fixedLength

---

## pub.flatFile.generate:findDependants

WmFlatFile. Returns the names of all flat file schemas and dictionaries that are dependent on a given flat file dictionary.

### Input Variables

---

*ffDictionaryName*    **String** The name of the flat file dictionary whose dependents you want to find.

### Output Variables

---

*dependants*            **Document List** The dependent objects and the packages that contain them.

<u>Variable</u>	<u>Description</u>
-----------------	--------------------

<i>packageName</i>	<b>String</b> The name of the package that contains the dependent object.
--------------------	---

<i>name</i>	<b>String</b> The name of the dependent object.
-------------	---

### Examples

For examples of how to use this service, see the following services in the certified samples area of the Knowledge Center on the Empower Product Support website at <https://empower.softwareag.com>.

sample.flatFile.generateFFSchema:delimited

sample.flatFile.generateFFSchema:fixedLength

---

## pub.flatFile.generate:findReferences

WmFlatFile. Returns the names of all flat file dictionaries that are referenced by a given flat file dictionary or flat file schema.

### Input Variables

---

*name* **String** The name of the flat file dictionary or flat file schema whose references you want to find.

### Output Variables

---

*references* **Document List** The referenced objects and the packages that contain them.

Variable	Description
----------	-------------

<i>packageName</i>	<b>String</b> The name of the package that contains the referenced object.
--------------------	--

<i>name</i>	<b>String</b> The name of the referenced object.
-------------	--

### Examples

For examples of how to use this service, see the following services in the certified samples area of the Knowledge Center on the Empower Product Support website at <https://empower.softwareag.com>.

sample.flatFile.generateFFSchema:delimited

sample.flatFile.generateFFSchema:fixedLength

---

## pub.flatFile.generate:getFFDictionaryAsXML

WmFlatFile. Returns a dictionary as an XML string.

### Input Variables

---

*FFDictionaryName* **String** The fully-qualified name of the flat file dictionary that you want returned as XML.

---

## Output Variables

*FFXML*            **String** The returned flat file dictionary as an XML string. The returned XML string conforms to the [pub.flatFile.generate:FFDictionary](#) IS document type.

### Examples

For examples of how to use this service, see the following services in the certified samples area of the Knowledge Center on the Empower Product Support website at <https://empower.softwareag.com>.

sample.flatFile.generateFFSchema:delimited

sample.flatFile.generateFFSchema:fixedLength

---

## pub.flatFile.generate:getFFDictionaryEntryAsXML

WmFlatFile. Returns a single dictionary entry as an XML string.

### Input Variables

*FFDictionaryName*   **String** The fully-qualified name of the flat file dictionary that contains the entry that you want returned as XML.

*EntryName*            **String** The name of the entry that you want to returned as XML.

*EntryType*            **String** The type of entry that you want returned. Specify `Record`, `Composite`, or `Field`.

---

### Output Variables

*FFXML*            **String** The returned flat file dictionary entry as an XML string. The returned XML string conforms to the [pub.flatFile.generate:FFDictionary](#) IS document type.

### Examples

For examples of how to use this service, see the following services in the certified samples area of the Knowledge Center on the Empower Product Support website at <https://empower.softwareag.com>.

sample.flatFile.generateFFSchema:delimited

---

sample.flatFile.generateFFSchema:fixedLength

---

## pub.flatFile.generate:getFFSchemaAsXML

WmFlatFile. Returns the specified flat file schema as an XML string.

### Input Variables

---

*FFSchemaName* **String** The fully-qualified name of the flat file schema that you want returned as XML.

### Output Variables

---

*FFXML* **String** The returned flat file schema as an XML string. The returned XML string conforms to the [pub.flatFile.generate:FFDictionary](#) IS document type.

### Examples

For examples of how to use this service, see the following services in the certified samples area of the Knowledge Center on the Empower Product Support website at <https://empower.softwareag.com>.

sample.flatFile.generateFFSchema:delimited

sample.flatFile.generateFFSchema:fixedLength

---

## pub.flatFile.generate:listFFDictionaryEntries

WmFlatFile. Lists all entries in a specified flat file dictionary that are of a specified type.

### Input Variables

---

*FFDictionaryName* **String** The fully-qualified name of the flat file dictionary that contains the entries that you want listed.

*EntryType* **String** The type of entries that you want listed. Specify `Record`, `Composite`, or `Field`.

---

## Output Variables

---

*EntryName*      **String List** The list of returned flat file dictionary entries.

## Examples

For examples of how to use this service, see the following services in the certified samples area of the Knowledge Center on the Empower Product Support website at <https://empower.softwareag.com>.

sample.flatFile.generateFFSchema:delimited

sample.flatFile.generateFFSchema:fixedLength

---

## pub.flatFile.generate:saveXMLAsFFDictionary

WmFlatFile. Creates a flat file dictionary in the Integration Server namespace by converting the specified flat file dictionary that is in XML format into a namespace flat file dictionary.

If a flat file dictionary with the same name already exists in the Integration Server namespace, use the [pub.flatFile.generate:deleteFFDictionary](#) service to delete the flat file dictionary before invoking this service. This service throws an exception if a flat file dictionary with the same name already exists when it is invoked.

---

## Input Variables

---

*FFDictionaryName*      **String** The fully-qualified name of the flat file dictionary that you want to create in the Integration Server namespace.

*PackageName*      **String** The name of the Integration Server package in which to save the flat file dictionary.

*FFXML*      **String** The flat file dictionary (as an XML string) that you want to create in the Integration Server namespace. The XML string must conform to the [pub.flatFile.generate:FFDictionary](#) IS document type.

**Note:** To see examples of how to supply the XML string in *FFXML* by mapping data from another file, see the samples provided in the WmFlatFileSamples package. For sample code that shows how to retrieve the data for *FFXML* from an XML file in the local file system, see *Flat File Schema Developer's Guide*.

*maxNumOfErrors* **String** Optional. The maximum number of errors that you want returned. The default is 100.

The service ensures the flat file dictionary is valid before saving it in the Integration Server namespace. The validation occurs in two stages.

1. Structural validation of the XML.
2. Logical validation of the XML contents.

If structural validation errors occur, the service reports the structural validation errors, but does not proceed with logical validation. When the XML string contains no structural validation errors, the service proceeds with logical validation and reports any logical validation errors.

## Output Variables

---

*saved* **String** Whether the flat file dictionary was saved successfully. It will have one of the following values.

Value	Description
true	The flat file dictionary was successfully saved.
false	The flat file dictionary was <i>not</i> successfully saved.

*Errors* **String List** Optional. Errors that occurred while attempting to save the flat file dictionary to the Integration Server namespace.

*Warnings* **String List** Optional. Warnings about the flat file dictionary that was created.

## Usage Note

Use this service to add a new flat file dictionary. Use the [pub.flatFile.generate:updateFFDictionaryEntryFromXML](#) if you want to update one or more entries in a flat file dictionary rather than creating a new flat file dictionary.

## Examples

For examples of how to use this service, see the following services in the certified samples area of the Knowledge Center on the Empower Product Support website at <https://empower.softwareag.com>.

sample.flatFile.generateFFSchema:delimited

sample.flatFile.generateFFSchema:fixedLength

---

## pub.flatFile.generate:saveXMLAsFFSchema

WmFlatFile. Creates a flat file schema in the Integration Server namespace by converting the specified flat file schema that is in XML format into a namespace flat file schema.

If a flat file schema with the same name already exists in the Integration Server namespace, use the [pub.flatFile.generate:deleteFFSchema](#) service to delete the flat file schema *before* invoking this service. This service throws an exception if a flat file schema with the same name already exists when it is invoked.

### Input Variables

---

<i>FFSchemaName</i>	<b>String</b> The fully-qualified name of the flat file schema that you want to create in the Integration Server namespace.
<i>PackageName</i>	<b>String</b> The name of the Integration Server package in which to save the flat file schema.
<i>FFXML</i>	<b>String</b> The flat file schema (as an XML string) that you want to create in the Integration Server namespace. The XML string must conform to the <a href="#">pub.flatFile.generate:FFDictionary</a> IS document type.

**Note:** To see examples of how to supply the XML string in *FFXML* by mapping data from another file, see the samples provided in the *WmFlatFileSamples* package. For sample code that shows how to retrieve the data for *FFXML* from an XML file in the local file system, see *Flat File Schema Developer's Guide*.

<i>maxNumOfErrors</i>	<b>String</b> Optional. The maximum number of errors that you want returned. The default is 100.
-----------------------	--

The service ensures the flat file schema is valid before saving it in the Integration Server namespace. The validation occurs in two stages.

1. Structural validation of the XML.
2. Logical validation of the XML contents.

If structural validation errors occur, the service reports the structural validation errors, but does not proceed with logical validation. When the XML string contains no structural validation errors, the service proceeds with logical validation and reports any logical validation errors.

---

## Output Variables

---

*saved* **String** Whether the flat file schema was saved successfully. It will have one of the following values.

<u>Value</u>	<u>Description</u>
true	The flat file schema was successfully saved.
false	The flat file schema was <i>not</i> successfully saved.

*Errors* **String List** Optional. Errors that occurred while attempting to save the flat file schema to the Integration Server namespace.

*Warnings* **String List** Optional. Warnings about the flat file schema that was created.

## Examples

For examples of how to use this service, see the following services in the certified samples area of the Knowledge Center on the Empower Product Support website at <https://empower.softwareag.com>.

sample.flatFile.generateFFSchema:delimited

sample.flatFile.generateFFSchema:fixedLength

---

## pub.flatFile.generate:updateFFDictionaryEntryFromXML

WmFlatFile. Updates one or more entries in a flat file dictionary in the Integration Server namespace.

This service goes through all entries that you specify in the *FFXML* variable. If an entry with the same name and type already exists in the flat file dictionary, this service overwrites the existing entry. If the entry does *not* already exist, this service creates the entry in the specified flat file dictionary.

## Input Variables

---

*FFDictionaryName* **String** The fully-qualified name of the flat file dictionary that contains the entries that you are replacing, adding, or both.

*FFXML* **String** The dictionary entries (as an XML string) that you want to use to replace an existing entry or that you want to add to the flat

file dictionary. The XML string in *FFXML* must conform to the [pub.flatFile.generate:FFDictionary](#) IS document type.

**Note:** To see examples of how to supply the XML string in *FFXML* by mapping data from another file, see the samples provided in the *WmFlatFileSamples* package. For sample code that shows how to retrieve the data for *FFXML* from an XML file in the local file system, see *Flat File Schema Developer's Guide*.

*maxNumOfErrors* **String** Optional. The maximum number of errors that you want returned. The default is 100.

The service ensures the flat file schema is valid before saving them in the flat file dictionary. The validation occurs in two stages.

1. Structural validation of the XML.
2. Logical validation of the XML contents.

If structural validation errors occur, the service reports the structural validation errors, but does not proceed with logical validation. When the XML string contains no structural validation errors, the service proceeds with logical validation and reports any logical validation errors.

## Output Variables

---

*saved* **String** Whether the dictionary entry was saved successfully. It will have one of the following values.

Value	Description
true	The dictionary entry was successfully saved.
false	The dictionary entry was <i>not</i> successfully saved.

*Errors* **String List** Optional. Errors that occurred while attempting to save the entry to the flat file dictionary.

*Warnings* **String List** Optional. Warnings about the dictionary entry that was updated or added.

**Examples**

For examples of how to use this service, see the following services in the certified samples area of the Knowledge Center on the Empower Product Support website at <https://empower.softwareag.com>.

sample.flatFile.generateFFSchema:delimited

sample.flatFile.generateFFSchema:fixedLength



# 11 Flow Folder

---

■ Summary of Elements in this Folder .....	380
--	-----

You use the elements in the flow folder to perform debugging and utility-type tasks in a flow service.

## Summary of Elements in this Folder

The following elements are available in this folder:

Element	Package and Description
<a href="#">pub.flow:clearPipeline</a>	WmPublic. Removes all fields from the pipeline. You may optionally specify fields that should not be cleared by this service.
<a href="#">pub.flow:debugLog</a>	WmPublic. Writes a message to the server log.
<a href="#">pub.flow:getCallingService</a>	WmPublic. Retrieves the service name and package name of the parent of the calling child service.
<a href="#">pub.flow:getLastError</a>	WmPublic. Obtains detailed information about the last exception that was trapped within a flow.
<a href="#">pub.flow:getRetryCount</a>	WmPublic. Retrieves the retry count and the maximum retry count for a service.
<a href="#">pub.flow:getSession</a>	WmPublic. Inserts the Session object into the pipeline as a document named <i>\$session</i> .
<a href="#">pub.flow:getTransportInfo</a>	WmPublic. Retrieves information about the protocol from which the current service was invoked.
<a href="#">pub.flow:invokeService</a>	WmPublic. Dynamically invokes any Integration Server public service and optionally returns the output from the invoked service in the pipeline for <a href="#">pub.flow:invokeService</a> .
<a href="#">pub.flow:iterator</a>	WmPublic. Returns IData arrays as batches with a specified batch size.
<a href="#">pub.flow:restorePipeline</a>	WmPublic. Restores a pipeline previously saved by <a href="#">pub.flow:savePipeline</a> .

Element	Package and Description
<a href="#">pub.flow:restorePipelineFromFile</a>	WmPublic. Restores a pipeline that was previously saved to a file.
<a href="#">pub.flow:savePipeline</a>	WmPublic. Saves a pipeline into memory, for later retrieval with <a href="#">pub.flow:restorePipeline</a> .
<a href="#">pub.flow:savePipelineToFile</a>	WmPublic. Saves the current pipeline to a file on the machine running webMethods Integration Server.
<a href="#">pub.flow:setCustomContextID</a>	WmPublic. Associates a custom value with an auditing context. This custom value can be used to search for service audit records in the webMethods Monitor.
<a href="#">pub.flow:setResponse</a>	WmPublic. <i>Deprecated</i> - Replaced by <a href="#">pub.flow:setResponse2</a> .
<a href="#">pub.flow:setResponse2</a>	WmPublic. Forces a specified response to be returned by the webMethods Integration Server to a calling process (such as a browser or application server).
<a href="#">pub.flow:setResponseCode</a>	WmPublic. Specifies the HTTP response code to be returned by Integration Server to a calling process (such as a browser or application server).
<a href="#">pub.flow:setResponseHeader</a>	WmPublic. Sets a header field in the HTTP response to a calling process (such as a browser or application server) or in the JMS message that contains the SOAP response from a web service invocation.
<a href="#">pub.flow:setResponseHeaders</a>	WmPublic. Sets one or more header fields in the HTTP response to a calling process (such as a browser or application server) or in the JMS message that contains the SOAP response from a web service invocation.
<a href="#">pub.flow:throwExceptionForRetry</a>	WmPublic. Throws an <code>ISRuntimeException</code> and instructs the Integration Server to re-execute a service using the original service input.

Element	Package and Description
<a href="#">pub.flow:tracePipeline</a>	WmPublic. Writes the names and values of all fields in the pipeline to the server log.
<a href="#">pub.flow:transportInfo</a>	WmPublic. Document type used to return information about the protocol through which a service was invoked.

## pub.flow:clearPipeline

WmPublic. Removes all fields from the pipeline. You may optionally specify fields that should not be cleared by this service.

### Input Parameters

*preserve* **String List** Optional. Field names that should not be cleared from the pipeline.

### Output Parameters

None.

## pub.flow:debugLog

WmPublic. Writes a message to the server log.

Each log message contains a timestamp, a message ID, the function name field, and message field. The following is an example:

```

                function _____ message
2009-08-16 11:01:23 EDT [ISP.0004C] My function - My message

```

### Input Parameters

*message* **String** Optional. Text of the message to write to the log.

*function* **String** Optional. Function name, typically an abbreviation used to identify the source of the message.

*level* **String** Optional. Debug level at which to display this message.

Whether or not Integration Server displays this message depends on the logging level setting for the 0090 pub Flow services facility. For example, if you specify Error for this message, but 0090 pub Flow services facility is configured to display only Fatal errors, this message will not be displayed. However, if the 0090 pub Flow services logging facility logging level is set to Warn, this message will be displayed (the Warn setting displays warning, error, *and* fatal messages).

Specify one of the following values:

<u>Specify...</u>	<u>To display the message with these types of messages...</u>
Off	No messages.
Fatal	Fatal messages only. This is the default
Error	Error and fatal messages.
Warn	Warning, error, and fatal messages.
Info	Informational, warning, error, and fatal messages.
Debug	Debug, informational, warning, error, and fatal messages.
Trace	Trace, debug, informational, warning, error, and fatal messages.

### **Output Parameters**

---

None.

### **Usage Notes**

You can control the logging level for flow messages independent of log messages for other facilities. On the **Settings > Logging > Edit** screen in Integration Server Administrator, navigate to facility **0090 pub Flow Services** and specify the level of messages that you want Integration Server to display for services in the pub.flow folder.

Prior to Integration Server 7.1, Integration Server used a number-based system to set the level of debug information written to the server log. Integration Server maintains backward compatibility with this system.

---

## pub.flow:getCallingService

WmPublic. Retrieves the service name and package name of the parent of the calling child service.

### Input Parameters

---

None.

### Output Parameters

---

*svcName*            **String** Fully qualified namespace name of the parent service.

*pkgName*            **String** Package name of the parent service.

### Usage Notes

If `pub.flow:getCallingService` is invoked from a service that does not have a parent service (for example, if the service is a top level service), Integration Server does not return the output parameters.

---

## pub.flow:getLastError

WmPublic. Obtains detailed information about the last exception that was trapped within a flow.

An exception is trapped in a flow when a service failure occurs inside a SEQUENCE step that executes until DONE, or when a service failure occurs inside a REPEAT step that repeats on FAILURE.

### Input Parameters

---

None.

### Output Parameters

---

*lastError*            **Document** Information about the last error, translated into the language used by the Integration Server. The structure of this document is defined by [pub.event:exceptionInfo](#).

### Usage Notes

If this service is not invoked from within a flow service, an exception is thrown.

Each execution of a service (whether the service succeeds or fails) updates the value returned by `getLastError`. Consequently, `getLastError` itself resets the value of `lastError`. Therefore, if the results of `getLastError` will be used as input to subsequent services, map the value of `lastError` to a variable in the pipeline.

If a map has multiple transformers, then a subsequent call to `getLastError` will return the error associated with the last failed transformer in the map, even if it is followed by successful transformers.

---

## pub.flow:getRetryCount

WmPublic. Retrieves the retry count and the maximum retry count for a service.

The retry count indicates the number of times the Integration Server has re-executed a service. For example, a retry count of 1 indicates that the Integration Server tried to execute the service twice (the initial attempt and then one retry). The maximum retry count indicates the maximum number of times the Integration Server can re-execute the service if it continues to fail because of an `ISRuntimeException`.

### Input Parameters

---

None.

### Output Parameters

---

<i>retryCount</i>	<b>String</b> The number of times the Integration Server has re-executed the service.
<i>maxRetryCount</i>	<b>String</b> The maximum number of times the Integration Server can re-execute the service. A value of -1 indicates that the service is being invoked by a trigger configured to retry until success.

### Usage Notes

Although the `pub.flow:getRetryCount` service can be invoked at any point in a flow service, the `pub.flow:getRetryCount` service retrieves retry information for the service within which it is invoked. That is, you can use the `pub.flow:getRetryCount` service to retrieve retry information for top-level services or services invoked by a trigger only. The `pub.flow:getRetryCount` service does not retrieve retry information for a nested service (a service that is invoked by another service).

The Integration Server retries a service that is configured to retry if the service uses the `pub.flow:throwISRuntimeException` service to catch a transient error and re-throw it as an `ISRuntimeException`. The Integration Server will also retry a service written in Java if the service throws an exception using `com.wm.app.b2b.server.ISRuntimeException()`. For more information about constructing `com.wm.app.b2b.server.ISRuntimeExceptions`

---

in Java services, see the *webMethods Integration Server Java API Reference* for the `com.wm.app.b2b.server.ISRuntimeException` class.

The maximum number of times the Integration Server retries a service depends on the value of the **Max attempts** property for the service. If the service is invoked by a trigger, the retry behavior is determined by the trigger retry properties.

#### See Also

[pub.flow:throwExceptionForRetry](#)

---

## pub.flow:getSession

WmPublic. Inserts the Session object into the pipeline as a document named *\$session*.

Session is useful for associating values with particular clients or users. Once *\$session* is added to the pipeline, it can be used like any other document in a flow. This permits more powerful flows that perform work spanning several user requests.

#### Input Parameters

---

None.

#### Output Parameters

---

*\$session*     **Document** Information for the current user session. Setting, copying, or dropping fields within *\$session* is effectively manipulating the Session object on the server.

---

## pub.flow:getTransportInfo

WmPublic. Retrieves information about the protocol from which the current service was invoked.

#### Input Parameters

---

None.

#### Output Parameters

---

*transport*     **Document** Information about the protocol that invoked the service. The structure of this document is defined by [pub.flow:transportInfo](#).

## Usage Notes

The `pub.flow:getTransportInfo` service can be used to retrieve protocol information for a top-level service only. A top-level service is one that is invoked by a client request or a trigger.

The value of the `protocol` key in `transport` indicates which protocol was used to invoke the top-level service. For example, if the service was invoked through SMTP, `protocol` would be set to `email`. `transport` will also contain a document (whose key is protocol-dependent) that holds protocol-specific details.

To use this service, first check the value of the `transport/protocol` parameter to determine which protocol was used. Then, depending on the value of `protocol`, extract the appropriate protocol information from `transport`. See [pub.flow:transportInfo](#) for the structure of the document that holds the protocol details.

---

## pub.flow:invokeService

WmPublic. Dynamically invokes any Integration Server public service and optionally returns the output from the invoked service in the pipeline for `pub.flow:invokeService`.

### Input Parameters

<i>ifcname</i>	<b>String</b> The flow interface name of the service to be invoked. For example, <code>pub.math</code> .
<i>svcname</i>	<b>String</b> The name of the service to invoke. For example, <code>addInts</code> .
<i>pipeline</i>	<b>Document</b> Optional. The name of the pipeline for the <code>pub.flow:invokeService</code> service.

**Note:** If *pipeline* is not specified, the invoked service returns the output parameters in its own pipeline.

### Output Parameters

If you specify a value for *pipeline*, the `pub.flow:invokeService` service returns the output parameters defined for the service specified by *ifcname* and *svcname*. For example, `pub.flow:invokeService` invokes `pub.math:addInts` as follows:

```
ifcname = "pub.math"
svcname = "addInts"
pipeline = "mypipeline"
mypipeline.num1 = "100"
mypipeline.num2 = "200"
```

In this case, `pub.flow:invokeService` returns the output parameter as follows:

```
mypipeline.value = "300"
```

If you do not specify a value for *pipeline*, the invoked service returns the output, *not* `pub.flow:invokeService`. For example, if you use `pub.flow:invokeService` to invoke `pub.math.addInts`, the output is returned in the pipeline for `pub.math.addInts`.

### Usage Notes

If the parameters specified by the service invoked by `pub.flow:invokeService` are not present or not properly specified, Integration Server issues an exception appropriate for the invoked service.

Integration Server issues a `ServiceException` for `pub.flow:invokeService` if either of the following is not present in the flow service:

- The interface specified by *ifcname*
- The service specified by *svcname*

---

## pub.flow:iterator

WmPublic. Returns IData arrays as batches with a specified batch size.

### Input Parameters

---

*\$internal*      **IData** A document that specifies the information required to batch output.

<u>Key</u>	<u>Description</u>
<i>batchContext</i>	<b>String</b> Optional. Unique handle for this specific batch request. For the first call, this parameter must be null or empty. On subsequent calls, it will contain a universally unique identifier (UUID) that Integration Server uses as the key to retrieve the current state of the batching request.
<i>batchSize</i>	<b>Integer</b> Optional. Number of records (IData) to return in the result set array. The value must be greater than 0. The default is 100.
<i>timeToLive</i>	<b>Integer</b> Length of time in seconds that the request remains in the cache. The value must be greater than 0. The default is 60.
<i>resultSetKey</i>	<b>String</b> Pipeline key name of the IData array that should be batched. If not specified, the service creates batches from the first IData array in the

pipeline. If no `IData` array exists in the service signature, the service will not batch results, and Integration Server returns all records at once.

*serviceName* **String** Fully qualified service name from which `pub.flow.iterator` receives the `IData` array. This parameter is required for the first invocation, and is optional for subsequent calls.

## Output Parameters

*\$internal* **IData** A document that specifies the results of the batch request.

Key	Description
<i>batchContext</i>	<b>String</b> Conditional. Unique handle for this specific request. Returned only if <i>hasMore</i> is <code>true</code> .
<i>hasMore</i>	<b>Boolean</b> Indicates whether there are still records to process in the result set ( <code>true</code> ), or if there are no more to process ( <code>false</code> ).
<i>resultSetKey</i>	<b>String</b> Indicates the pipeline key name of the <code>IData</code> array that was batched. If <i>resultSetKey</i> is specified in the input, Integration Server returns that value. If <i>resultSetKey</i> is not specified in the input, this parameter contains the key of the first <code>IData</code> array that was processed.

**Note:** In addition to the output parameters listed above, `pub.flow.iterator` returns any parameters returned in the output signature of the service specified in the *serviceName* input parameter.

## Usage Notes

The output service signature of the service specified by the *serviceName* input parameter must contain at least one `IData` array. If there are multiple `IData` arrays in the output signature and the *resultSetKey* input parameter is not specified, `pub.flow.iterator` batches the first `IData` array in the pipeline. If there are no `IData` arrays in the pipeline, `pub.flow.iterator` returns the output all at once instead of in batches.

The `pub.flow.iterator` service sends the rest of the pipeline to the service specified by the *serviceName* parameter in the case that multiple calls are required to get all of the data in the result set.

---

## pub.flow:restorePipeline

WmPublic. Restores a pipeline previously saved by [pub.flow:savePipeline](#).

### Input Parameters

---

- \$name*      **String** Name of the saved pipeline. Because multiple pipelines can be saved, this parameter is necessary to identify the pipeline in memory. If this value is left null or the name is unknown, an exception will be thrown.
- \$merge*      **String** Optional. Flag that indicates whether or not to merge the values in the existing pipeline with the values in the saved pipeline. Set to:
- `false` to clear the existing pipeline before restoring the saved pipeline. This is the default.
  - `true` to merge the existing pipeline with the saved pipeline. If a field exists in the saved pipeline and the existing pipeline, the saved field takes precedence.
- \$remove*      **String** Optional. Flag that indicates whether or not the saved pipeline will remain in memory after this service is executed. Set to:
- `false` to retain the saved pipeline in memory so that future calls to `restorePipeline` with the same *\$name* will still return the saved pipeline. This is the default.
  - `true` to remove the saved pipeline from memory after the service executes.

### Output Parameters

---

The output is dynamic, based on the contents of the saved and existing pipelines.

### Usage Notes

After a successful invocation of `restorePipeline`, the pipeline will contain all fields that were present immediately before [pub.flow:savePipeline](#) was invoked. `restorePipeline` clears existing pipeline values unless the optional *\$merge* field is specified.

When using MTOM streaming for SOAP attachments, *messageContext* variables and/or *XOObject* fields will not be available in the saved pipeline. A *messageContext* variable is used by many `pub.soap` services to hold the SOAP message on which the service acts. *XOObject* fields are Objects that use the `com.wm.util.XOObject` Java wrapper type. For more information about MTOM Streaming, see the *Web Services Developer's Guide*.

This service is helpful in the interactive development or debugging of an application.

---

**See Also**

[pub.flow:savePipeline](#)  
[pub.flow:restorePipelineFromFile](#)

---

## pub.flow:restorePipelineFromFile

WmPublic. Restores a pipeline that was previously saved to a file.

---

**Input Parameters**

- fileName* **String** Relative path and file name of a file containing a saved pipeline on the Integration Server. If the file is not found at run time, an exception is thrown.
- merge* **String** Optional. Flag that determines whether or not to merge the saved values into the existing pipeline. Set to:
- `false` to replace the existing pipeline with the saved values. This is the default.
  - `true` to merge the saved values into the existing pipeline.

---

**Output Parameters**

The output is dynamic, based on the contents of the saved and existing pipelines.

**Usage Notes**

This service is helpful in the interactive development or debugging of an application. In some cases, however, using the Pipeline debug property for the debugging of an application is more efficient. For more information about the Pipeline debug property, see *webMethods Service Development Help*.

Be aware that variables that exist in the saved pipeline but are not defined in the flow will not appear on the **Pipeline** tab and, therefore, will not be available for explicit mapping.

When using MTOM streaming for SOAP attachments, *messageContext* variables and/or XOObject fields will not be available in the saved pipeline. A *messageContext* variable is used by many `pub.soap` services to hold the SOAP message on which the service acts. XOObject fields are Objects that use the `com.wm.util.XOObject` Java wrapper type. For more information about MTOM Streaming, see the *Web Services Developer's Guide*.

**See Also**

[pub.flow:savePipelineToFile](#)  
[pub.flow:restorePipeline](#)

---

## pub.flow:savePipeline

WmPublic. Saves a pipeline into memory, for later retrieval with [pub.flow:restorePipeline](#).

### Input Parameters

---

*\$name*     **String** Name that will identify the pipeline in memory. An exception will be thrown if this value is not specified.

### Output Parameters

---

None.

### Usage Notes

After a successful invocation of `savePipeline`, a snapshot of pipeline fields will be saved in memory under the key provided by *\$name*. Note that because the pipeline is saved to memory, it will not be available after a server restart.

When using MTOM streaming for SOAP attachments, *messageContext* variables and/or *XOPObject* fields will not be available in the saved pipeline. A *messageContext* variable is used by many `pub.soap` services to hold the SOAP message on which the service acts. *XOPObject* fields are Objects that use the `com.wm.util.XOPObject` Java wrapper type. For more information about MTOM Streaming, see the *Web Services Developer's Guide*.

This service is helpful in the interactive development or debugging of an application.

### See Also

[pub.flow:restorePipeline](#)

[pub.flow:savePipelineToFile](#)

---

## pub.flow:savePipelineToFile

WmPublic. Saves the current pipeline to a file on the machine running Integration Server.

### Input Parameters

---

*fileName*     **String** Relative or absolute path to a file on the server file system in which Integration Server saves the contents of the pipeline.

When you specify a path for the *fileName* input parameter, Integration Server verifies whether it is specified in the `allowedWritePaths` parameter of the file access control configuration file (`fileAccessControl.cnf`). If the file path is not specified in the `allowedWritePaths` parameter, Integration

Server issues an exception. Otherwise, Integration Server either creates the file if it does not yet exist or overwrites the existing file if one already exists.

## Output Parameters

---

None.

## Usage Notes

When using MTOM streaming for SOAP attachments, *messageContext* variables and/or *XOPObject* fields will not be available in the saved pipeline. A *messageContext* variable is used by many *pub.soap* services to hold the SOAP message on which the service acts. *XOPObject* fields are Objects that use the *com.wm.util.XOPObject* Java wrapper type. For more information about MTOM Streaming, see the *Web Services Developer's Guide*.

This service is helpful in the interactive development or debugging of an application. In some cases, however, using the Pipeline debug property for the debugging of an application is more efficient. For more information about the Pipeline debug property, see *webMethods Service Development Help*.

The following table shows the data types and classes that this service can write to the output file if they are included in the pipeline:

For...	<a href="#">pub.flow:savePipelineToFile</a> supports...
Java data types	<ul style="list-style-type: none"> <li>■ byte[]</li> <li>■ Date</li> <li>■ GregorianCalendar</li> <li>■ IData</li> <li>■ IData[] (IData list)</li> <li>■ String</li> <li>■ String[] (String list)</li> <li>■ String[][] (String table)</li> <li>■ Vector</li> </ul>
Java wrapper classes	<ul style="list-style-type: none"> <li>■ Boolean</li> <li>■ Byte</li> <li>■ Character</li> <li>■ Double</li> <li>■ Float</li> <li>■ Integer</li> </ul>

For...	<a href="#">pub.flow:savePipelineToFile</a> supports...
	<ul style="list-style-type: none"> <li>■ Long</li> <li>■ Short</li> <li>■ Single dimension arrays of any of the above</li> </ul>
webMethods classes	<ul style="list-style-type: none"> <li>■ MBoolean</li> <li>■ MByte</li> <li>■ MDouble</li> <li>■ MFloat</li> <li>■ MInteger</li> <li>■ MLong</li> <li>■ MShort</li> <li>■ Single dimension arrays of any of the above</li> </ul>
Object arrays	Any non-array item listed in this table.

### See Also

[pub.flow:restorePipelineFromFile](#)  
[pub.flow:savePipeline](#)

---

## pub.flow:setCustomContextID

WmPublic. Associates a custom value with an auditing context. This custom value can be used to search for service audit records in the webMethods Monitor.

### Input Parameters

---

*id* **String** Optional. The custom value for the current auditing context. Specify a value that you want to associate with the auditing context.

### Output Parameters

---

None.

### Usage Notes

- Each client request creates a new auditing context. The auditing context is the lifetime of the top-level service. Once the custom context identifier is set, Integration

Server includes that value in each service audit record it logs in the current context. Calls to this service affect audit logging only for the current request.

- This service is useful when Integration Server is configured to log to a database. When the server logs information about a service to the database, it includes the custom context identifier in the service log. Using the webMethods Monitor, you can use the custom value as search criteria to locate and view all corresponding service audit records.
- If Integration Server is configured to log to a file system, the server writes the custom context identifier with the service audit records to a file. This file is not accessible with the webMethods Monitor. You cannot use the webMethods Monitor to query service records when logging to a file.
- If this service is invoked without a specified value for *id*, Integration Server writes a null value for the custom context identifier field for all subsequent service audit records that it logs in the current context.

---

## pub.flow:setResponse

WmPublic. *Deprecated* - Replaced by [pub.flow:setResponse2](#).

Forces a specified response to be returned by the webMethods Integration Server to a calling process (such as a browser or application server).

Formatting of the response is normally handled by templates, which format values from the pipeline. If templates are not appropriate for a particular integration scenario, a response message can be created within the flow and then returned to the caller using this service.

### Input Parameters

---

<i>responseString</i>	<b>String</b> Optional. Response to be returned to the caller, specified as a string.
<i>responseBytes</i>	<b>byte[ ]</b> Optional. Response to be returned to the caller, specified as a byte array.
<i>string</i>	<i>Deprecated</i> - Replaced by <i>responseString</i> . <b>String</b> Optional. Response to be returned to the caller, specified as a string.
<i>bytes</i>	<i>Deprecated</i> - Replaced by <i>responseBytes</i> . <b>byte[ ]</b> Optional. Response to be returned to the caller, specified as a byte array.
<i>response</i>	<i>Deprecated</i> - Replaced by <i>responseString</i> . <b>String</b> Optional. Response to be returned to the caller, specified as a string.

*contentType* **String** Optional. MIME type of the response data. By default, the server's response will match the MIME type of the request. This field allows this behavior to be overridden.

**Note:** If you explicitly set this value with Designer, you will see two choices: `text/XML` and `text/HTML`. You are not limited to these two values. You may either select one of these or type a new value.

*encoding* **String** Optional. Character set in which the response is encoded.

## Output Parameters

None.

## Usage Notes

Specify *responseString* or *responseBytes*, but not both. If you specify both, the `pub.flow:setResponse` service uses *responseString* and ignores *responseBytes*.

If neither *responseString* or *responseBytes* are specified, Integration Server uses the value of the server configuration parameter `watt.server:setResponse.pre82Mode` to determine the order in which to look for and use the deprecated fields.

- When `watt.server:setResponse.pre82Mode` is set to “true”, Integration Server follows a precedence order similar to what was available in Integration Server 7.1x and 8.0x. Specifically, Integration Server looks for the deprecated parameters in the following order and uses the value of the first parameter that it finds: *response*, *string*, *bytes*
- When `watt.server:setResponse.pre82Mode` is set to “false”, Integration Server follows a precedence order similar to what was available in Integration Server 8.2 and later. Specifically, Integration Server looks for the deprecated parameters in the following order and uses the value of the first parameter that it finds: *string*, *bytes*, *response*

One possible usage of this service is to create an XML response to an XML request. A flow that creates an XML document by calling `pub.xml:documentToXMLString` can use `pub.flow:setResponse` to return the XML document to the caller. In your flow, you would map *xmldata* (output of `pub.xml:documentToXMLString`) to *responseString* and set *contentType* to `text/xml` (inputs to `pub.flow:setResponse`). Calling `pub.flow:setResponse` will cause the server to return the XML document that you've mapped to *responseString* instead of processing the pipeline through a template.

Your client might be expecting binary data in the response, such as a JPEG image. In this case, map a byte array that represents the image to *responseBytes* and set *contentType* to `image/jpeg`.

Integration Server detects the type of request and sets the Content-Type value to `text/XML` (for requests in XML format) or `text/HTML` (for requests in all other formats). Be aware that if you specify a value for *contentType*, Designer will not be able to decode or display output from flows that include this service. This is because your *contentType*

setting will override the Content-Type value that the Integration Server uses to return output to Designer. If you use **Run** to test the flow, Designer will not display any results.

If you specify a *contentType* value that is not supported, Integration Server sets the Content-Type of the response to `text/XML`.

If the *contentType* input parameter and the Content-Type header of the request are not set, Integration Server uses the value of the request Accept header to set the Content-Type header of the response. For more information about how Integration Server handles Accept headers to set Content-Types, see *webMethods Integration Server Administrator's Guide*.

Keep in mind that when returning the processed XML document to the client that originally submitted it, you may need to modify the encoding. Java String objects are always stored using a Unicode encoding. If your original XML document used an encoding other than UTF-8 or UTF-16, it will still contain an encoding tag that indicates what this encoding was. However, if you did not modify the encoding during document processing, you need to set the encoding parameter when you invoke the `pub.flow:setResponse` service. Specifically, do one of the following:

- Set the *encoding* parameter to match the tag in the file, or
- Set the *encoding* parameter to "autoDetect" to use the encoding specified in the XML string encoding tag.

---

## pub.flow:setResponse2

WmPublic. Forces a specified response to be returned by the webMethods Integration Server to a calling process (such as a browser or application server).

Formatting of the response is normally handled by templates, which format values from the pipeline. If templates are not appropriate for a particular integration scenario, a response message can be created within the flow and then returned to the caller using this service.

### Input Parameters

---

<i>responseString</i>	<b>String</b> Optional. Response to be returned to the caller, specified as a string.
<i>responseBytes</i>	<b>byte[]</b> Optional. Response to be returned to the caller, specified as a byte array.
<i>responseStream</i>	<b>java.io.InputStream</b> Optional. Response to be returned to the caller, specified as an InputStream.

*contentType* **String** Optional. MIME type of the response data. By default, the server's response will match the MIME type of the request. This field allows this behavior to be overridden.

**Note:** If you explicitly set this value with Designer, you will see two choices: `text/XML` and `text/HTML`. You are not limited to these two values. You may either select one of these or type a new value.

*encoding* **String** Optional. Character set in which the response is encoded.

## Output Parameters

None.

## Usage Notes

This service replaces [pub.flow:setResponse](#), which is deprecated.

Specify *responseString*, *responseBytes*, or *responseStream*. If you specify more than one, the `pub.flow:setResponse2` service looks for the parameters in the following order and uses the first one that it finds: *responseString*, *responseBytes*, *responseStream*.

One possible usage of this service is to create an XML response to an XML request. A flow that creates an XML document by calling [pub.xml:documentToXMLString](#) can use `pub.flow:setResponse2` to return the XML document to the caller. In your flow, you would map *xmlData* (output of [pub.xml:documentToXMLString](#)) to *responseString* and set *contentType* to `text/xml` (inputs to `pub.flow:setResponse2`). Calling `pub.flow:setResponse2` will cause the server to return the XML document that you've mapped to *responseString* instead of processing the pipeline through a template.

Your client might be expecting binary data in the response, such as a JPEG image. In this case, map a byte array that represents the image to *responseBytes* and set *contentType* to `image/jpeg`.

Integration Server detects the type of request and sets the Content-Type value to `text/XML` (for requests in XML format) or `text/HTML` (for requests in all other formats). Be aware that if you specify a value for *contentType*, Designer will not be able to decode or display output from flows that include this service. This is because your *contentType* setting will override the Content-Type value that the Integration Server uses to return output to Designer. If you use **Run** to test the flow, Designer will not display any results.

If you specify a *contentType* value that is not supported, Integration Server sets the Content-Type of the response to `text/XML`.

If the *contentType* input parameter and the Content-Type header of the request are not set, Integration Server uses the value of the request Accept header to set the Content-Type header of the response. For more information about how Integration Server handles Accept headers to set Content-Types, see *webMethods Integration Server Administrator's Guide*.

Keep in mind that when returning the processed XML document to the client that originally submitted it, you may need to modify the encoding. Java String objects are always stored using a Unicode encoding. If your original XML document used an encoding other than UTF-8 or UTF-16, it will still contain an encoding tag that indicates what this encoding was. However, if you did not modify the encoding during document processing, you need to set the encoding parameter when you invoke the `pub.flow:setResponse2` service. Specifically, do one of the following:

- Set the *encoding* parameter to match the tag in the file, or
- Set the *encoding* parameter to "autoDetect" to use the encoding specified in the XML string encoding tag.

---

## pub.flow:setResponseCode

WmPublic. Specifies the HTTP response code to be returned by Integration Server to a calling process (such as a browser or application server).

### Input Parameters

---

- responseCode*     **String** HTTP status code to be returned to the caller.
- The response codes and phrases are defined in [RFC 2616, Section 10](#). If you provide a value for *responseCode* that is not listed in RFC 2616, Section 10, you must also provide a value for *reasonPhrase*.
- reasonPhrase*     **String** Optional. HTTP reason phrase to be returned to the caller. If no reason is provided, the default reason phrase associated with *responseCode* will be used. You must provide a *reasonPhrase* for any *responseCode* that is not listed in RFC 2616, Section 10

### Output Parameters

---

None.

---

## pub.flow:setResponseHeader

WmPublic. Sets a header field in the HTTP response to a calling process (such as a browser or application server) or in the JMS message that contains the SOAP response from a web service invocation.

### Input Parameters

---

- fieldName*     **String** Name of the header field to set.

*fieldValue*     **String** Value of the header field to set.

## Output Parameters

---

None.

## Usage Notes

`pub.flow:setResponseHeader` sets a single field in the response header. To set multiple response header fields, use `pub.flow:setResponseHeaders`.

You can use `pub.flow:setResponse` to set the Content-Type of the HTTP header field. Content-Type specifies the format of the service response. For example, to specify a JSON response, set Content-Type to `application/json`. For more information about content types Integration Server supports, see *webMethods Integration Server Administrator's Guide*.

The following HTTP header fields cannot be set by calling this service or by Integration Server applications:

- Allow
- Connection
- Content-Length
- WWW-Authenticate
- Transfer-Encoding
- Upgrade

**Note:**     Content-Length can be set with the `pub.flow:setResponse` service.

Keep the following points in mind when adding headers for a JMS message that contains a SOAP response:

- You can specify custom headers.
- You can set some JMS message header fields directly and set others using run-time properties specific to Integration Server.
  - You can set JMSType directly. This header name is case-sensitive.
  - You can set the following headers indirectly using run-time properties: JMSDeliveryMode, JMSExpiration, and JMSPriority. The following table identifies these properties and indicates the JMS message header fields affected by each property.

Property Name	Description										
<code>jms.deliveryMode</code>	Specifies the message delivery mode for the message. Integration Server uses this value to set the <code>JMSDeliveryMode</code> header.										
	<table border="1"> <thead> <tr> <th>Value</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td><code>PERSISTENT</code></td> <td>Indicates the request message is persistent.</td> </tr> <tr> <td>2</td> <td>Default. Indicates the request message is persistent.</td> </tr> <tr> <td><code>NON_PERSISTENT</code></td> <td>Indicates the request message is not persistent.</td> </tr> <tr> <td>1</td> <td>Indicates the request message is not persistent.</td> </tr> </tbody> </table> <p><b>Note:</b> If the <code>jms.deliveryMode</code> is not one of the above values, Integration Server ignores the name/value pair and uses the default value of 2.</p>	Value	Description	<code>PERSISTENT</code>	Indicates the request message is persistent.	2	Default. Indicates the request message is persistent.	<code>NON_PERSISTENT</code>	Indicates the request message is not persistent.	1	Indicates the request message is not persistent.
Value	Description										
<code>PERSISTENT</code>	Indicates the request message is persistent.										
2	Default. Indicates the request message is persistent.										
<code>NON_PERSISTENT</code>	Indicates the request message is not persistent.										
1	Indicates the request message is not persistent.										
<code>jms.timeToLive</code>	<p>Length of time, in milliseconds, that the JMS provider retains the message. A value of 0 means that the message does not expire.</p> <p>The JMS provider uses this value to set the <code>JMSExpiration</code> header in the sent JMS message.</p> <p><b>Note:</b> If the <code>jms.timeToLive</code> value is not a valid Long, Integration Server ignores the property and uses the default value of 0.</p>										
<code>jms.priority</code>	<p>Specifies the message priority. The JMS standard defines priority levels from 0 to 9, with 0 as the lowest priority and 9 as the highest.</p> <p>Integration Server uses this value to set the <code>JMSPriority</code> header.</p> <p>If the <code>jms.priority</code> value is not a value between 0 to 9, Integration Server ignores the property and uses the default value of 4.</p>										

- You can specify the following JMS-defined properties:

- JMSXGroupID
- JMSXGroupSeq

If the value of JMSXGroupSeq is not a String that contains a number, Integration Server ignores the name/value pair and does not place it in the message header.

**Note:** The JMSXGroupID and JMSXGroupSeq names are case-sensitive.

- You can set any provider-specific property whose name starts with “JMS\_” in *fieldName*. Because the JMS standard reserves the prefix “JMS\_<vendor\_name>” for provider-specific properties, Integration Server does not validate the name or value of this content.

**Note:** The JMS provider determines which provider-specific properties to accept and include in the JMS message properties. For more information about provider-specific message properties how the JMS provider handles them, review the JMS provider documentation.

- The lowercase “jms.” prefix is reserved for run-time properties used by Integration Server. If a header starts with “jms.” and is not one of the “jms.” properties defined by Integration Server, Integration Server ignores the property.
- The “JMSX” prefix is reserved for JMS-defined properties. If a header whose name starts with “JMSX” is passed into *fieldName* and it is not named JMSXGroupID or JMSXGroupSeq, Integration Server throws a ServiceException.
- You cannot set any of the SOAP over JMS message header properties. These header names start with “SOAPJMS”.

---

## pub.flow:setResponseHeaders

WmPublic. Sets one or more header fields in the HTTP response to a calling process (such as a browser or application server) or in the JMS message that contains the SOAP response from a web service invocation.

### Input Parameters

---

*headers*      **Document List** Contains the header fields to set. Specify the following for each header that you want to set.

*fieldName*                      **String** Name of the header field to set.

*fieldValue*                      **String** Value of the header field to set.

---

## Output Parameters

---

None.

## Usage Notes

`pub.flow:setResponseHeaders` sets one or more fields in the response header. If any of the fields specified by *fieldName* have already been set, they will be overwritten.

See the Usage Notes for [pub.flow:setResponseHeader](#) for more information.

---

## pub.flow:throwExceptionForRetry

WmPublic. Throws an `ISRuntimeException` and instructs the Integration Server to re-execute a service using the original service input.

---

## Input Parameters

---

*wrappedException*     **Object** Optional. Any exception that you want to include as part of this `ISRuntimeException`. This might be the exception that causes the `pub.flow:throwExceptionForRetry` service to execute. For example, if the service attempts to connect to a database and the connection attempt fails, you might map the exception generated by the database connection failure to the *wrappedException* parameter.

*message*                 **String** Optional. A message to be logged as part of this exception.

---

## Output Parameters

---

None.

## Usage Notes

Use the `pub.flow:throwExceptionForRetry` service to handle transient errors that might occur during service execution. A transient error is an error that arises from a condition that might be resolved quickly, such as the unavailability of a resource due to network issues or failure to connect to a database. The service might execute successfully if the Integration Server waits and then retries the service. If a transient error occurs, the service can catch this error and invoke `pub.flow:throwExceptionForRetry` to instruct the Integration Server to retry the service.

The `pub.flow:throwExceptionForRetry` service should be used for transient errors only.

Only top-level services or trigger services can be retried. That is, a service can be retried only when it is invoked directly by a client request or by a trigger. The service cannot be retried when it is invoked by another service (that is, when it is a nested service).

You can invoke the [pub.flow:getRetryCount](#) service to retrieve the current retry count and the maximum specified retry attempts for a service.

If the trigger service is written in Java, the service can use `ISRuntimeException()` to throw an exception and retry the service. For more information about constructing `ISRuntimeExceptions` in Java services, see the *webMethods Integration Server Java API Reference* for the `com.wm.app.b2b.server.ISRuntimeException` class.

For information about configuring retry for services or triggers, see *webMethods Service Development Help*.

### See Also

[pub.flow:getRetryCount](#)

---

## pub.flow:tracePipeline

WmPublic. Writes the names and values of all fields in the pipeline to the server log.

### Input Parameters

---

*level*     **String** Optional. Debug level at which to write the pipeline. Defaults to `Fatal`. If the debug level on the webMethods Integration Server is set to a value less than this parameter, the pipeline will not be written to the server log.

### Output Parameters

---

None.

### Usage Notes

Prior to Integration Server 7.1, Integration Server used a number-based system to set the level of debug information written to the server log. Integration Server maintains backward compatibility with this system. For more information about logging levels, see the description of the `watt.debug.level` parameter in *webMethods Integration Server Administrator's Guide*.

---

## pub.flow:transportInfo

WmPublic. Document type used to return information about the protocol through which a service was invoked.

---

## Parameters

---

*protocol* **String** Name of protocol about which *transportInfo* contains information. Will be one of the following values

<u>A value of...</u>	<u>Indicates that...</u>
email	SMTP was used to invoke the service. Detailed information is contained in the <i>email</i> parameter.
filePolling	File polling protocol was used to invoke this service. Detailed information is contained in the <i>filePolling</i> parameter.
ftp	FTP was used to invoke the service. Detailed information is contained in the <i>ftp</i> parameter.
http	HTTP was used to invoke the service. Detailed information is contained in the <i>http</i> parameter.
jms	JMS was used to invoke the service. When a standard JMS trigger or a SOAP-JMS trigger invokes a service, the protocol is JMS. Detailed information is contained in the <i>jms</i> parameter.
messaging	webMethods messaging was used to invoke the service. Specifically, a webMethods messaging trigger invoked the service. Detailed information is contained in the <i>messaging</i> parameter.

*subprotocol* **String** Conditional. Subprotocol used to invoke the service.

<u>A value of...</u>	<u>Indicates that...</u>
broker	The service was invoked by a webMethods messaging trigger that received a document from the Broker or a locally published document.

---

	<p>This subprotocol can only occur if <i>protocol</i> is messaging.</p>
HTTP	<p>The service was invoked through HTTP.</p> <p>This subprotocol can only occur if <i>protocol</i> is http.</p>
HTTPS	<p>The service was invoked through HTTPS.</p> <p>This subprotocol can only occur if <i>protocol</i> is http.</p>
jndi	<p>Integration Server uses JNDI to connect to the JMS provider. The JMS connection alias assigned to the JMS trigger specifies how Integration Server connects to the JMS provider.</p> <p>A value of <code>jndi</code> also indicates that the <i>jms/destinationName</i> value is a lookup name.</p> <p>This subprotocol can only occur if <i>protocol</i> is jms.</p>
native	<p>Integration Server uses the native webMethods API to connect to the webMethods Broker being used as the JMS provider directly. The JMS connection alias assigned to the JMS trigger specifies how Integration Server connects to the JMS provider.</p> <p>A value of <code>native</code> also indicates that the <i>jms/destinationName</i> value is the name of the actual destination.</p> <p>This subprotocol can only occur if <i>protocol</i> is jms.</p>
um	<p>The service was invoked by a webMethods messaging trigger that received a document from the Universal Messaging.</p> <p>This subprotocol can only occur if <i>protocol</i> is messaging.</p>

The *subprotocol* parameter is returned only when the returned *protocol* is http, jms, or messaging.

*email*

**Document** Conditional. Information about the SMTP transport. This parameter is returned only if the service was invoked via SMTP.

<u>Key</u>	<u>Description</u>
<i>to</i>	<b>String List</b> E-mail addresses for the recipients of the e-mail.
<i>from</i>	<b>String List</b> E-mail addresses for the senders of the e-mail.
<i>cc</i>	<b>String List</b> Conditional. E-mail addresses receiving a copy of the e-mail.
<i>bcc</i>	<b>String List</b> Conditional. E-mail addresses receiving a blind copy of the e-mail.
<i>replyto</i>	<b>String List</b> Conditional. E-mail address to which replies of this e-mail should be sent
<i>subject</i>	<b>String</b> Subject of the e-mail.
<i>filename</i>	<b>String</b> Conditional. Name of the attached file.
<i>contenttype</i>	<b>String</b> Conditional. Content-Type of the attached file.
<i>content</i>	<b>java.io.InputStream</b> Conditional. Contents of the attached file.
<i>recvdate</i>	<b>String</b> Conditional. Date the e-mail was received in String format. <i>recvdate</i> may be passed as parameter for the <code>java.util.Date</code> constructor.
<i>sentdate</i>	<b>String</b> Conditional. Date the e-mail was sent in String format. <i>sentdate</i> may be passed as parameter for the <code>java.util.Date</code> constructor.

*http*

**Document** Conditional. Information about the http transport. This parameter is returned only if the service was invoked via http.

<u>Key</u>	<u>Description</u>
------------	--------------------

<i>requestUrl</i>	<b>String</b> URL used by the client to invoke the service.
<i>query</i>	<b>String</b> Conditional. Query portion of request URL.
<i>method</i>	<b>String</b> HTTP method used by the client to request the top-level service. Possible values are GET, PUT, POST, and DELETE.
<i>requestHdrs</i>	<b>Document</b> Fields in the request header, where key names represent header field names and values represent the header field values.
<i>ipInfo</i>	<b>Document</b> Information about the http socket connection. Contains the following information:

<u>Key</u>	<u>Description</u>
<i>clientIp</i>	<b>String</b> Optional. IP address of the client connecting to this socket. If null, <i>clientIP</i> is not included in the output.
<i>clientPort</i>	<b>String</b> Optional. Port number used by the client connecting to this socket. If null, <i>clientPort</i> is not included in the output.
<i>localIp</i>	<b>String</b> Local IP address for this socket connection to client.
<i>localPort</i>	<b>String</b> Local port number for this socket connection to client.
<i>remoteIp</i>	<b>String</b> Remote IP address for this socket connection to client.

*remotePort* **String** Remote port number for this socket connection to client.

*ftp* **Document** Conditional. Information about the ftp transport. This parameter is returned only if the ftp transport invoked the service.

<u>Key</u>	<u>Description</u>
------------	--------------------

<i>filename</i>	<b>String</b> Name of file that was put into the service directory.
-----------------	---

<i>mimetype</i>	<b>String</b> Conditional. Content type of the file (for example, <code>text/xml</code> , <code>text/plain</code> , or <code>image/jpeg</code> ). The server determines content type based on the extension of the file and the extension's corresponding content type defined in <code>Integration Server_directory\instances\instance_name\lib\mime.types</code> .
-----------------	--

*ipInfo* **Document** Information about the FTP socket connection. Contains the following information:

<u>Key</u>	<u>Description</u>
------------	--------------------

<i>localIp</i>	<b>String</b> Local IP address for this socket connection to client.
----------------	--

<i>localPort</i>	<b>String</b> Local port number for this socket connection to client.
------------------	---

<i>remoteIp</i>	<b>String</b> Remote IP address for this socket connection to client.
-----------------	---

<i>remotePort</i>	<b>String</b> Remote port number for this socket connection to client.
-------------------	--

*filePolling* **Document** Conditional. Information about the file polling transport. Returned only if the file polling transport invoked the service.

<u>Key</u>	<u>Description</u>
<i>filename</i>	<b>String</b> Fully qualified name of the file submitted to the file polling listener.
<i>originalFilename</i>	<b>String</b> Name of the file when it was submitted to the file polling listener.
<i>contentType</i>	<b>String</b> Conditional. Content type of the file (for example, <code>text/xml</code> , <code>text/plain</code> , or <code>image/jpeg</code> ). The server determines content type based on the extension of the file and the extension's corresponding content type defined in <code>Integration Server_directory\instances\instance_name\lib\mime.types</code> .
<i>length</i>	<b>String</b> The original file length in bytes.
<i>lastModified</i>	<b>java.util.Date</b> Java date object indicating when the original file was last modified.
<i>jms</i>	<b>Document</b> Conditional. Information about the JMS transport. This parameter is returned only if the JMS transport invoked the service.
<u>Key</u>	<u>Description</u>
<i>connectionAliasName</i>	<b>String</b> Name of the JMS connection alias used by the JMS trigger to retrieve the message.
<i>triggerName</i>	<b>String</b> Fully qualified name of the JMS trigger that retrieved the JMS message that resulted in invocation of the service.
<i>destinationName</i>	<p><b>String</b> Name or lookup name of the destination from which the message that invoke the service was received.</p> <p>When the <i>subprotocol</i> is <code>jndi</code>, the <i>destinationName</i> value is the lookup name of the destination on the JNDI provider. When the <i>subprotocol</i> is <code>native</code>, the <i>destinationName</i> value is the name of the destination at the webMethods Broker.</p>

---

<i>destinationType</i>	<p><b>String</b> Type of destination from which the JMS trigger received the message. The destination type for the JMS trigger specified in <i>triggerName</i> subscribes determines the <i>destinationType</i> value.</p> <p><i>destinationType</i> will be one of the following:</p> <p>QUEUE indicates the destination is a queue.</p> <p>TOPIC indicates the destination is a topic.</p>
<i>requestHdrs</i>	<p><b>Document</b> Fields in the request header of the JMS message, where key names represent header field names and values represent header field values. The <i>requestHdrs</i> includes all of the JMS message headers and JMS message properties.</p> <p>When a SOAP-JMS trigger receives a message, Integration Server passes the body of the message to the web services stack for processing. The body of the JMS message is the SOAP message. Integration Server does not pass the JMS headers on to the web services stack. If you want to review the contents of the JMS request header, you must use <code>pub.flow:getTransportInfo</code> to obtain the headers.</p> <p>For a standard JMS trigger, Integration Server passes the entire JMS message, including request headers, on to the trigger service for processing.</p>
<i>messaging</i>	<p><b>Document</b> Conditional. Information about the messaging transport. This parameter is returned only if the messaging transport invoked the service.</p>
<i>connectionAliasName</i>	<p><b>String</b> Name of the <code>webMethodsMessaging</code> connection alias used by the <code>webMethodsMessaging</code> trigger to retrieve the document. If the <code>webMethodsMessaging</code> trigger received a locally published document and the publishable document type is not associated with a messaging connection alias, the <i>connectionAliasName</i> field is blank.</p>

*triggerName* **String** Fully qualified name of the webMethods messaging trigger that retrieved the document that resulted in invocation of the service.

*documentTypeName* **String** Fully qualified name of the publishable document type for which the received document is an instance.

If the trigger services executes because an AND join condition in the webMethods messaging trigger was satisfied, the *documentTypeName* contains the name of the publishable document type that completed the AND join. For example, suppose that an AND join condition specifies folder.subfolder:documentA and folder.subfolder:documentB. If the trigger receives an instance of documentB first and then an instance of documentA, the *documentTypeName* parameter contains folder.subfolder:documentA.

*redeliveryCount* **String** Number of times the messaging provider has redelivered the document to the trigger.

**A value of...**

**Indicates...**

-1

The transport used to send the document does not maintain a document redelivery count. For example, a document received from a webMethods Broker version 6.0.1 has a redelivery count of -1. (webMethods Brokers that are version 6.0.1 or earlier do not maintain document redelivery counts.)

Integration Server may or may not have received the document before.

0	The document has been received only once.
> 0	The number of times document has been redelivered.

**Usage Notes**

A document with this structure is produced by the [pub.flow:getTransportInfo](#) service.



# 12 Hashtable Folder

---

■ Summary of Elements in this Folder .....	416
--	-----

This folder contains services that you can use to create, update, and obtain information about the hashtable.

## Summary of Elements in this Folder

The following elements are available in this folder:

Element	Package and Description
<a href="#">pub.hashtable:containsKey</a>	WmPublic. Checks for the existence of a hashtable element.
<a href="#">pub.hashtable:createHashtable</a>	WmPublic. Creates a hashtable object.
<a href="#">pub.hashtable:get</a>	WmPublic. Gets the value for a specified key in the hashtable.
<a href="#">pub.hashtable:listKeys</a>	WmPublic. Lists all the keys stored in the hashtable.
<a href="#">pub.hashtable:put</a>	WmPublic. Adds a key/value pair in the hashtable.
<a href="#">pub.hashtable:remove</a>	WmPublic. Removes a key/value pair from the hashtable.
<a href="#">pub.hashtable:size</a>	WmPublic. Gets the number of elements in the hashtable.

### pub.hashtable:containsKey

WmPublic. Checks for the existence of a hashtable element.

#### Input Parameters

*hashtable*      **java.util.Hashtable** Hashtable in which to check for the existence of a hashtable element.

*key*              **String** Hashtable element to be checked for.

### Output Parameters

---

*containsKey* **String** Indicates whether the specified hashtable element exists. A value of:

- `true` indicates that the element exists.
- `false` indicates that the element does not exist.

---

## pub.hashtable:createHashtable

WmPublic. Creates a hashtable object.

### Input Parameters

---

None.

### Output Parameters

---

*hashtable* **java.util.Hashtable** The new hashtable object.

---

## pub.hashtable:get

WmPublic. Gets the value for a specified key in the hashtable.

### Input Parameters

---

*hashtable* **java.util.Hashtable** Hashtable from which to retrieve the specified value.

*key* **String** Key of the hashtable element whose value is to be retrieved.

### Output Parameters

---

*value* **Object** Value of the input hashtable element.

---

## pub.hashtable:listKeys

WmPublic. Lists all the keys stored in the hashtable.

### Input Parameters

---

*hashtable*      **java.util.Hashtable** Hashtable from which the keys are to be listed.

### Output Parameters

---

*keys*            **String[]** List of keys stored in the input hashtable.

---

## pub.hashtable:put

WmPublic. Adds a key/value pair in the hashtable.

### Input Parameters

---

*hashtable*      **java.util.Hashtable** Hashtable to which the key/value pair is to be added.

*key*              **String** Key of the element to be added to the hashtable.

*value*            **Object** Value of the element to be inserted into the hashtable.

### Output Parameters

---

*hashtable*      **java.util.Hashtable** Hashtable object after the insertion of the key/value pair.

---

## pub.hashtable:remove

WmPublic. Removes a key/value pair from the hashtable.

### Input Parameters

---

*hashtable*      **java.util.Hashtable** Hashtable from which to remove the key/value pair.

*key*              **String** Key of the hashtable element to be removed.

*value*            **Object** Value of the hashtable element to be removed.

### Output Parameters

---

*hashtable*      **java.util.Hashtable** Hashtable object after the key/value pair is removed.

*value*            **Object** Value of the hashtable element that was removed. Returns `null` if the input *key* is not found in the hashtable.

---

## pub.hashtable:size

WmPublic. Gets the number of elements in the hashtable.

### Input Parameters

---

*hashtable*      **java.util.Hashtable** Hashtable from which the number of elements stored in it is to be retrieved.

### Output Parameters

---

*size*             **String** Number of elements in the hashtable.



# 13 IO Folder

---

■ Summary of Elements in this Folder .....	422
--	-----

You use the elements in the `io` folder to convert data between `byte[ ]`, characters, and `InputStream` representations. The `io` folder contains services for reading and writing bytes, characters, and streamed data to the file system.

These services behave like the corresponding methods in the `java.io.InputStream` class. For more information about `InputStreams`, see the Java documentation.

## Summary of Elements in this Folder

**Note:** The services in this folder may only be invoked by other services on Integration Server. Streams cannot be passed between clients and Integration Server, so these services will not execute if they are invoked from a client.

The following elements are available in this folder:

Element	Package and Description
<a href="#">pub.io:bytesToStream</a>	WmPublic. Converts a <code>byte[ ]</code> to <code>java.io.ByteArrayInputStream</code> .
<a href="#">pub.io:close</a>	WmPublic. Closes an <code>InputStream</code> or a reader object and releases the resources.
<a href="#">pub.io:createByteArray</a>	WmPublic. Creates a byte array of the specified length.
<a href="#">pub.io:mark</a>	WmPublic. Marks the current position in the <code>InputStream</code> or reader object.
<a href="#">pub.io:markSupported</a>	WmPublic. Enables you to test whether your <code>InputStream</code> or reader object supports the mark and reset operations.
<a href="#">pub.io:read</a>	WmPublic. Reads a specified number of bytes from the <code>InputStream</code> and stores them into a buffer.
<a href="#">pub.io:readAsString</a>	WmPublic. Reads the data from a reader object and returns the contents as a string.
<a href="#">pub.io:readerToString</a>	WmPublic. Reads the data from a reader object and converts it to a string.

Element	Package and Description
<a href="#">pub.io:reset</a>	WmPublic. Repositions the InputStream or the reader object to the position at the time the pub.io:mark service was last invoked on the stream.
<a href="#">pub.io:skip</a>	WmPublic. Skips over and discards the specified number of bytes or characters from the input stream or a reader object.
<a href="#">pub.io:streamToBytes</a>	WmPublic. Creates a byte[ ] from data that is read from an InputStream.
<a href="#">pub.io:streamToReader</a>	WmPublic. Converts a java.io.InputStream to a java.io.Reader object.
<a href="#">pub.io:streamToString</a>	WmPublic. Creates a string from data that is read from an InputStream.
<a href="#">pub.io:stringToReader</a>	WmPublic. Converts a string object to a java.io.StringReader object.
<a href="#">pub.io:stringToStream</a>	WmPublic. Converts a string to a byte stream.

## pub.io:bytesToStream

WmPublic. Converts a byte[ ] to java.io.ByteArrayInputStream.

### Input Parameters

<i>bytes</i>	<b>byte[ ]</b> The byte array to convert.
<i>length</i>	<b>String</b> Optional. The maximum number of bytes to read and convert. If <i>length</i> is not specified, the default value for this parameter is the length of the input byte array.
<i>offset</i>	<b>String</b> Optional. The offset into the input byte array from which to start converting. If no value specified, the default value is zero.

---

## Output Parameters

---

*stream* **java.io.ByteArrayInputStream** An open `InputStream` created from the contents of the input *bytes* parameter.

### Usage Notes

This service constructs *stream* from the byte array using the constructor `ByteArrayInputStream(byte[] )`. This constructor does not make a copy of the byte array, so any changes to *bytes* will be reflected in the data read from the stream.

---

## pub.io:close

WmPublic. Closes an `InputStream` or a reader object and releases the resources.

---

### Input Parameters

---

*inputStream* **java.io.InputStream** Optional. An open `InputStream`.

**Note:** You can use either *inputStream* or *reader* to specify the input object. If both the input parameters are provided, then both the objects will be closed.

*reader* **java.io.Reader** Optional. An open reader object.

---

### Output Parameters

---

None.

### Usage Notes

If the `InputStream` is already closed, invoking this service has no effect. However, leaving an `InputStream` open may cause errors that are not recoverable until Integration Server is shut down. Use the `pub.io:close` service to explicitly close the input stream when a service leaves it open. For example, `pub.file:getFile` and `pub.client.ftp:get` leave the input stream open in the pipeline.

---

## pub.io:createByteArray

WmPublic. Creates a byte array of the specified length.

---

### Input Parameters

*length*                      **String** The length of the byte array to be created.

### Output Parameters

*bytes*                        **Object** The new byte array.

### Usage Notes

The `pub.io:read` service reads data from an `InputStream` into a byte array. You can use this service to create the byte array. Invoking this service is the equivalent of the Java code `new byte[length]`.

---

## pub.io:mark

`WmPublic`. Marks the current position in the `InputStream` or reader object.

A subsequent call to `pub.io:reset` repositions this stream at the last marked position. Marking and repositioning the input stream allows subsequent service calls to re-read the same bytes.

---

### Input Parameters

*stream*                        **java.io.InputStream** Optional. The `InputStream`.

**Note:** You can use either *stream* or *reader* to specify the input object. If both *stream* and *reader* input parameters are provided, then both the objects will be marked.

*reader*                        **java.io.Reader** Optional. The reader object.

*limit*                         **String** The maximum number of bytes that can be read before the mark position becomes invalid. If more than this number of bytes are read from the stream after the `pub.io:mark` service is invoked, the `pub.io:reset` service will have no effect.

---

### Output Parameters

*stream*                        **java.io.InputStream** Conditional. The `InputStream`. Returned only if the input parameter is *stream*.

*reader* **java.io.Reader** Conditional. The reader object. Returned only if the input parameter is *reader* .

### Usage Notes

If the `InputStream` does not support the mark operation, invoking this service has no effect.

Either of the optional input parameters, *stream* or *reader* , is required.

---

## pub.io:markSupported

WmPublic. Enables you to test whether your `InputStream` or reader object supports the mark and reset operations.

### Input Parameters

---

*stream* **java.io.InputStream** Optional. The `InputStream`.

**Note:** You can use either *stream* or *reader* to specify the input object. If both *stream* and *reader* input parameters are provided, then the *stream* input parameter is ignored.

*reader* **java.io.Reader** Optional. The reader object.

### Output Parameters

---

*stream* **java.io.InputStream** Conditional. The `InputStream`. Returned only if the input parameter is *stream* .

*supported* **String** Indicates whether the stream supports the mark and reset operations. A value of:

- `true` indicates that the `InputStream` supports the mark and reset operations.
- `false` indicates that the `InputStream` does not support the mark and reset operations.

*reader* **java.io.Reader** Conditional. The reader object. Returned only if the input parameter is *reader* .

### Usage Notes

Either of the input parameters, *stream* or *reader* , is required.

---

## pub.io:read

WmPublic. Reads a specified number of bytes from the InputStream and stores them into a buffer.

### Input Parameters

---

<i>stream</i>	<b>Object</b> The InputStream. Object from which the service is to read bytes.
<i>offset</i>	<b>String</b> Optional The offset into the byte array in the buffer to which the data is written. If no value is supplied, this defaults to 0.
<i>length</i>	<b>String</b> Optional. The maximum number of bytes to read from the InputStream. If no value is supplied, the default is the length of <i>buffer</i> .  If the value supplied for <i>length</i> is greater than the length of <i>buffer</i> , an exception will be thrown.
<i>buffer</i>	<b>Object</b> The buffer into which data is written. This is a byte array, which can be created from a Flow service by invoking <code>pub.io:createByteArray</code> .

### Output Parameters

---

<i>stream</i>	<b>Object</b> The InputStream. If any bytes were read from the stream, the stream is repositioned after the last byte read.
<i>buffer</i>	<b>Object</b> The buffer into which data was written.
<i>bytesRead</i>	<b>String</b> The number of bytes read from the InputStream and copied to <i>buffer</i> . If there is no more data because the end of the stream has been reached, <i>bytesRead</i> will be -1.

---

## pub.io:readAsString

WmPublic. Reads the data from a reader object and returns the contents as a string.

---

### Input Parameters

---

<i>reader</i>	<b>java.io.Reader</b> The reader object.
<i>length</i>	<b>String</b> The maximum number of characters to read from the input reader object.

### Output Parameters

---

<i>reader</i>	<b>java.io.Reader</b> The reader object.
<i>lengthRead</i>	<b>String</b> The number of characters read from the input reader object. If there is no more data because the end of the stream has been reached, <i>lengthRead</i> will be -1.
<i>value</i>	<b>String</b> The data read from the reader object, or null if end of stream has been reached.

### Usage Notes

The `readAsString` service *does not* automatically close the reader object. To close the reader, use the `pub.io:close` service.

---

## pub.io:readerToString

WmPublic. Reads the data from a reader object and converts it to a string.

### Input Parameters

---

<i>reader</i>	<b>java.io.Reader</b> The reader object.
---------------	--

### Output Parameters

---

<i>string</i>	<b>String</b> Data read from the reader object.
---------------	---

### Usage Notes

The `readerToString` service *does not* automatically close the reader object. To close the reader, use the `pub.io:close` service.

---

## pub.io:reset

WmPublic. Repositions the `InputStream` or the reader object to the position at the time the `pub.io:mark` service was last invoked on the stream.

### Input Parameters

---

*stream* **java.io.InputStream** Optional. The `InputStream`.

**Note:** You can use either *stream* or *reader* to specify the input object. If both *stream* and *reader* input parameters are provided, then both the objects will be reset.

*reader* **java.io.Reader** Optional. The reader object.

### Output Parameters

---

*stream* **java.io.InputStream** Conditional. The `InputStream`. Returned only if the input parameter is *stream* .

*reader* **java.io.Reader** Conditional. The reader object. Returned only if the input parameter is *reader* .

### Usage Notes

If the `InputStream` does not support the reset operation, invoking this service has no effect.

Either of the input parameters, *stream* or *reader* , is required.

---

## pub.io:skip

WmPublic. Skips over and discards the specified number of bytes or characters from the input stream or a reader object.

### Input Parameters

---

*stream* **java.io.InputStream** Optional. The `InputStream`.

**Note:** You can use either *stream* or *reader* to specify the input object. If both *stream* and *reader* input parameters are

provided, then the stream and reader object data will be skipped.

<i>reader</i>	<b>java.io.Reader</b> Optional. The reader object.
<i>length</i>	<b>String</b> The number of bytes or characters to skip.

### Output Parameters

<i>stream</i>	<b>java.io.InputStream</b> Conditional. The InputStream. Returned only if the input parameter is <i>stream</i> .
<i>reader</i>	<b>java.io.Reader</b> Conditional. The reader object. Returned only if the input parameter is <i>reader</i> .
<i>bytesSkipped</i>	<b>String</b> Conditional. The actual number of bytes that were skipped. Returned only if the input parameter is <i>stream</i> .
<i>charactersSkipped</i>	<b>String</b> Conditional. The number of characters that were skipped. Returned only if the input parameter is <i>reader</i> .

### Usage Notes

The `pub.io:skip` service uses the Java method `InputStream.skip`, which might skip some smaller number of bytes, possibly zero (0). This happens due to conditions such as reaching the end of file before  $n$  bytes have been skipped. For more information about the `InputStream.skip` method, see the Java documentation on the `InputStream` class.

Either of the optional input parameters, *stream* or *reader* , is required.

If both *stream* and *reader* input parameters are specified and if an exception occurs during the stream object usage, then the operations are not performed on the reader object also.

## pub.io:streamToBytes

WmPublic. Creates a `byte[]` from data that is read from an `InputStream`.

### Input Parameters

<i>stream</i>	<b>java.io.InputStream</b> The <code>InputStream</code> that you want to convert.
---------------	---

---

## Output Parameters

---

*bytes*                      **byte[ ]** The bytes read from *stream* .

### Usage Notes

This service reads all of the bytes from *stream* until the end of file is reached, and then it closes the `InputStream`.

---

## pub.io:streamToReader

WmPublic. Converts a `java.io.InputStream` to a `java.io.Reader` object.

### Input Parameters

---

*inputStream*              **java.io.InputStream** The `InputStream` to convert to a reader object.

*encoding*                      **String** Optional. Name of a registered, IANA character set (for example, ISO-8859-1). If you specify an unsupported encoding, the system throws an exception. If no value is specified or if the encoding is set to `autoDetect`, the default operating system encoding is used.

### Output Parameters

---

*reader*                      **java.io.Reader** The reader object read from *inputStream* .

---

## pub.io:streamToString

WmPublic. Creates a string from data that is read from an `InputStream`.

### Input Parameters

---

*inputStream*              **java.io.InputStream** The `InputStream` to convert to a string.

*encoding*                      **String** Optional. Name of a registered, IANA character set (for example, ISO-8859-1). If you specify an unsupported encoding, the system throws an exception. If no value is

---

specified or if the encoding is set to `autoDetect`, the default operating system encoding is used.

### Output Parameters

---

*string*                      **String** Data read from *inputStream* and converted to a string.

---

## pub.io:stringToReader

WmPublic. Converts a string object to a `java.io.StringReader` object.

### Input Parameters

---

*string*                      **String** The string to convert to a `StringReader` object.

### Output Parameters

---

*reader*                      **java.io.StringReader** The `StringReader` object.

---

## pub.io:stringToStream

WmPublic. Converts a string to a byte stream.

### Input Parameters

---

*string*                      **String** The string object to be converted.

*encoding*                      **String** Optional. Name of a registered, IANA character set (for example, ISO-8859-1). If you specify an unsupported encoding, the system throws an exception. If no value is specified or if the encoding is set to `autoDetect`, the default operating system encoding is used.

### Output Parameters

---

*inputStream*                      **java.io.ByteArrayInputStream** An open `InputStream` created from the contents of *string*.

# 14 JDBC Folder

---

■ Summary of Elements in this Folder .....	434
--	-----

You use the elements in the JDBC folder to obtain information about Integration Server JDBC pools.

## Summary of Elements in this Folder

The following elements are available in this folder:

Element	Package and Description
<a href="#">pub.jdbc:getPoolInfo</a>	WmPublic. Returns run-time information about the JDBC pool associated with a specified functional alias.

### pub.jdbc:getPoolInfo

WmPublic. Returns run-time information about the JDBC pool associated with a specified functional alias.

#### Input Parameters

<i>functionalAlias</i>	<b>String</b> Name of the Integration Server functional alias for which you want to obtain JDBC information.
<i>getConnectionWait</i>	<b>String</b> Optional. Number of milliseconds the service will wait to obtain a connection to a JDBC connection pool. The default is 10 seconds (10000 milliseconds). If a connection is not obtained within this time, the service ends with a status of fail.

#### Output Parameters

<i>minConnections</i>	<b>String</b> The minimum number of connections the pool can have.
<i>maxConnections</i>	<b>String</b> The maximum number of connections the pool can have.
<i>totalConnections</i>	<b>String</b> The total number of connections currently in the pool. This number includes connections that are already in use and connections that are available for use.
<i>availableConnections</i>	<b>String</b> The number of connections that are available for use.

---

<i>status</i>	<p><b>String</b> A status indicating whether the service was able to connect to the pool. A value of:</p> <ul style="list-style-type: none"><li>■ <code>Active</code> indicates that the service was able to connect to the JDBC pool.</li><li>■ <code>Inactive</code> indicates that the service was not able to connect to the JDBC pool or an invalid functional alias was specified. If the attempt to connect to the JDBC pool failed with a <code>SQLException</code>, the service might throw an exception. If the service does not throw an exception, it will return a message in the <i>message</i> variable.</li></ul>
<i>message</i>	<p><b>String</b> Conditional. Returned only if the service was not able to connect to the JDBC pool. Contains one of the following explanations:</p> <p><code>Invalid functional alias.</code></p> <p>An invalid or non-existent functional alias was specified with the <i>functionalAlias</i> input parameter.</p> <p><code>No pool is associated with this function or the pool may have failed during initialization.</code></p> <p>A valid functional alias was specified, but no pool is associated with the functional alias, or the pool did not properly initialize.</p>



# 15 JMS Folder

---

■ Summary of Elements in This Folder .....	438
--	-----

You can use the services in the JMS folder to send and receive JMS messages.

## Summary of Elements in This Folder

The following elements are available in this folder:

Element	Package and Description
<a href="#">pub.jms:acknowledge</a>	WmPublic. Sends an acknowledgment for a message to the JMS provider.
<a href="#">pub.jms:batchTriggerSpec</a>	WmPublic. Specification for the signature of a JMS trigger that processes a batch of messages at one time.
<a href="#">pub.jms:createConsumer</a>	WmPublic. Creates a message consumer to receive messages from destinations on the JMS provider.
<a href="#">pub.jms:documentResolverSpec</a>	WmPublic. Specification for the signature of a document resolver service that determines whether a JMS message has a status of New, Duplicate, or In Doubt.
<a href="#">pub.jms:JMSMessage</a>	WmPublic. Document type that represents the structure and content of a JMS message received by a JMS trigger, received by the service <a href="#">pub.jms:receive</a> , or as the output of <a href="#">pub.jms:send</a> or <a href="#">pub.jms:sendAndWait</a> .
<a href="#">pub.jms:receive</a>	WmPublic. Receives a message from a queue or topic on the JMS provider.
<a href="#">pub.jms:reply</a>	WmPublic. Sends a reply message to a requesting client.
<a href="#">pub.jms:send</a>	WmPublic. Sends a JMS message to the JMS provider.
<a href="#">pub.jms:sendAndWait</a>	WmPublic. Sends a request in the form of a JMS message to the JMS provider and optionally, waits for a reply.

Element	Package and Description
<a href="#">pub.jms:sendBatch</a>	WmPublic. Sends multiple JMS messages to the same destination on the JMS provider.
<a href="#">pub.jms:triggerSpec</a>	WmPublic. Specification for the input signature of a JMS trigger that processes one message at a time.
<a href="#">pub.jms:waitForReply</a>	WmPublic. Retrieves the reply message for an asynchronous request.
<a href="#">pub.jms.wmjms:receiveStream</a>	WmPublic. Receives a large message stream from a queue or topic on the webMethods Broker.
<a href="#">pub.jms.wmjms:sendStream</a>	WmPublic. Sends a large message stream to the webMethods Broker.

## pub.jms:acknowledge

WmPublic. Sends an acknowledgment for a message to the JMS provider.

### Input Parameters

*message*      **Object** A `javax.jms.Message` object that identifies the message for which you want Integration Server to send an acknowledgment to the JMS provider.

### Output Parameters

None.

### Usage Notes

Use this service to acknowledge a message retrieved from the JMS provider if:

- The message was received using the [pub.jms:receive](#) service, and
- The message consumer used to retrieve the message has an `acknowledgmentMode` set to `CLIENT_ACKNOWLEDGE` or `DUPS_OK_ACKNOWLEDGE`. For more information about creating a message consumer, see [pub.jms:createConsumer](#).

A message is not considered to be successfully consumed until it is acknowledged.

**Note:** Acknowledging a message automatically acknowledges the receipt of all messages received in the same session. That is, all messages received by the

---

same consumer will be acknowledged when just one of the received messages is acknowledged.

### See Also

[pub.jms:createConsumer](#)  
[pub.jms:receive](#)

---

## pub.jms:batchTriggerSpec

WmPublic. Specification for the signature of a JMS trigger that processes a batch of messages at one time.

### Input Parameters

---

*JMSMessage*      **Document List** A document list where each document references the [pub.jms:JMSMessage](#) document type.

### Output Parameters

---

None.

### Usage Notes

Use this specification as the signature for JMS trigger services that will retrieve and process a batch of messages.

If you want to use a JMS trigger to retrieve and process one message at a time, use [pub.jms:triggerSpec](#) to declare the signature of the JMS trigger service.

### See Also

[pub.jms:triggerSpec](#)  
[pub.jms:JMSMessage](#)

---

## pub.jms:createConsumer

WmPublic. Creates a message consumer to receive messages from destinations on the JMS provider.

### Input Parameters

---

*connectionAliasName*      **String** Name of the JMS connection alias that you want the message consumer to receive messages.

The JMS connection alias indicates how Integration Server connects to the JMS provider. A JMS connection alias can specify that Integration Server use a JNDI provider to look up administered objects (connection factories and destinations) and then use the connection factory to create a connection. Alternatively, a JMS connection alias can specify that Integration Server uses the native webMethods API to create the connection directly on the webMethods Broker.

*destinationName*

**String** Name or lookup name of the Destination from which you want the message consumer to receive messages. Specify the lookup name of the Destination object when the JMS connection alias uses JNDI to retrieve administered objects. Specify the provider-specific name of the Destination when the JMS connection alias uses the native webMethods API to connect directly to the webMethods Broker.

*destinationType*

**String** Optional. Type of destination from which the message consumer receives messages. Specify one of the following:

- `QUEUE` to receive messages sent to a particular queue. This is the default.
- `TOPIC` to receive messages sent to a particular topic.

**Note:** You need to specify a *destinationType* only if you specified a *connectionAliasName* that uses the native webMethods API.

*acknowledgmentMode*

**String** Optional. Specifies the acknowledgment mode. Specify one of the following:

- `AUTO_ACKNOWLEDGE` to automatically acknowledge the message when it is received by the message consumer. The message consumer will acknowledge the message before the message processing completes. The JMS provider cannot redeliver the message if Integration Server becomes unavailable before message processing completes. This is the default.
- `CLIENT_ACKNOWLEDGE` to acknowledge the receipt of a message when the JMS client (Integration Server) invokes [pub.jms:acknowledge](#) service.
- `DUPS_OK_ACKNOWLEDGE` to automatically, lazily acknowledge the receipt of messages, which reduces

---

system overhead but may result in duplicate messages being sent.

*messageSelector*

**String** Optional Specifies a filter used to receive a subset of messages from the specified destination. A *message selector* allows a client to filter the messages it wants to receive by use of a SQL92 string expression in the message header. That expression is applied to properties in the message header (not to the message body content) containing the value to be filtered.

If the SQL expression evaluates to true, the JMS provider sends the message to the message consumer; if the SQL expression evaluates to false, the JMS provider does not send the message.

*durableSubscriberName*

**String** Optional. Name of the durable subscriber that you want this service to create on the JMS provider. A durable subscriber creates a durable subscription on the JMS provider. If a durable subscriber of this name already exists on the JMS provider, this service resumes the previously established subscription.

**Note:** This parameter only applies when the *destinationType* is set to `TOPIC`. If you select `TOPIC`, but do not specify a *durableSubscriberName*, this service creates a nondurable subscriber. If *destinationType* is set to `QUEUE`, this parameter is ignored.

*noLocal*

**java.lang.Boolean** Optional. Flag indicating whether the message consumer can receive locally published messages. Integration Server considers a message to be local if the message was:

- Sent by the same Integration Server, and
- Sent using the same JMS connection alias.

Specify one of the following values:

- `True` to indicate that the consumer will not receive locally published messages.
- `False` to indicate that the consumer can receive locally published messages. This is the default.

**Note:** This parameter only applies when the *destinationType* is set to `TOPIC`.

---

## Output Parameters

---

*consumer*

**Object** An on demand message consumer object used to receive messages for the specified destination.

### Usage Notes

A *message consumer* is a `webMethods` object that encapsulates the actual `javax.jms.MessageConsumer` and `javax.jms.Session`.

Any message consumers created during the execution of a service will be closed automatically when the top-level service completes. If the consumer closes without acknowledging messages, messages are implicitly recovered back to the JMS provider.

The use of [pub.jms:createConsumer](#) in conjunction with [pub.jms:receive](#) is an alternative to using JMS triggers. Use the [pub.jms:createConsumer](#) service to create a message consumer. Use the [pub.jms:receive](#) to actively receive messages from a destination on the JMS provider.

To create a durable subscriber, set the *destinationType* to `TOPIC` and specify a *durableSubscriberName*. If you select `TOPIC`, but do not specify a *durableSubscriberName*, Integration Server creates a nondurable subscriber.

A durable subscription allows subscribers to receive all the messages published on a topic, including those published while the subscriber is inactive.

If a durable subscription already exists for the specified durable subscriber on the JMS provider, this service resumes the subscription.

A non-durable subscription allows subscribers to receive messages on their chosen topic, only if the messages are published while the subscriber is active. A non-durable subscription lasts the lifetime of its message consumer.

If the *acknowledgment Mode* field is set to `CLIENT_ACKNOWLEDGE`, you must acknowledge messages received by this consumer to the JMS provider using the [pub.jms:acknowledge](#) service.

If the message consumer created by this service will be used to receive large message streams from the `webMethods` Broker, make sure to specify an *acknowledgmentMode* of `AUTO_ACKNOWLEDGE` or `CLIENT_ACKNOWLEDGE`. If the *acknowledgmentMode* is `DUPS_OK_ACKNOWLEDGE`, the message consumer cannot be used to receive large message streams.

### See Also

- [pub.jms:acknowledge](#)
- [pub.jms:receive](#)
- [pub.jms:reply](#)
- [pub.jms:send](#)
- [pub.jms:sendAndWait](#)

---

## pub.jms:documentResolverSpec

WmPublic. Specification for the signature of a document resolver service that determines whether a JMS message has a status of New, Duplicate, or In Doubt.

### Input Parameters

---

<i>uuid</i>	<b>String</b> Universally unique identifier for the message. If the sending client assigned a value to the <i>uuid</i> field in the message, Integration Server uses the <i>uuid</i> value to identify the message. If the <i>uuid</i> field is empty, Integration Server uses the value of the <i>JMSMessageID</i> field in the message header as the UUID.
<i>triggerName</i>	<b>String</b> The name of the JMS trigger that received the message whose status needs to be resolved.
<i>JMSMessage</i>	<b>Document</b> The message whose status needs to be resolved. This is a document reference (IData) to the <a href="#">pub.jms:JMSMessage</a> document type, which defines the structure of a JMS message.

### Output Parameters

---

<i>status</i>	<b>String</b> Indicates the status of the message. The value of this field determines whether the Integration Server processes or rejects the message. The <i>status</i> field will have one of the following values: <ul style="list-style-type: none"><li>■ <b>NEW</b> indicates that the message is new and has not been processed by the JMS trigger. Integration Server instructs the JMS trigger to process the message.</li><li>■ <b>DUPLICATE</b> indicates that the message is a duplicate of one already processed by the JMS trigger. Integration Server acknowledges the message, but does not execute the trigger service.</li><li>■ <b>IN_DOUBT</b> indicates that the status of the message is still in doubt. The document resolver service could not conclusively determine whether the JMS trigger already processed the message. Integration Server acknowledges the message, but does not execute the trigger service.</li></ul>
<i>message</i>	<b>String</b> Conditional. A user-specified string that indicates why the message status is <b>DUPLICATE</b> or <b>IN_DOUBT</b> . Integration Server

writes this message to the journal log when the message has a status of `DUPLICATE` or `IN_DOUBT`.

### Usage Notes

The `pub.jms:documentResolverSpec` must be used as the signature for a document resolver service used to determine the processing status of a JMS message received by a JMS trigger. For information about building a document resolver service and enabling exactly once processing for a JMS message, see *Using webMethods Integration Server to Build a Client for JMS*.

Use `pub.publish:documentResolverSpec` as the signature for a document resolver service used to determine the status of document received a webMethods messaging trigger.

### See Also

[pub.jms:JMSMessage](#)

[pub.publish:documentResolverSpec](#)

---

## pub.jms:JMSMessage

WmPublic. Document type that represents the structure and content of a JMS message received by a JMS trigger, received by the service `pub.jms:receive`, or as the output of `pub.jms:send` or `pub.jms:sendAndWait`.

### Parameters

---

*header*      **Document** Optional. Document (IData object) containing the header of the JMS message.

Key	Description
<i>JMSCorrelationID</i>	<b>String</b> Optional. A unique identifier used to link multiple messages together. Often, a <i>JMSCorrelationID</i> is used to link a reply message with its requesting message.
<i>JMSDeliveryMode</i>	<b>java.lang.Integer</b> Optional. Delivery mode specified at the time the message was sent. Delivery mode can be one of the following: <ul style="list-style-type: none"> <li>■ <code>PERSISTENT</code> to indicate that the JMS provider places the message in a persistent message store, allowing the message to be recovered in the event of a resource failure. This is the default.</li> </ul>

- `NON-PERSISTENT` to indicate that the JMS provider does not place the message in a persistent store. The message has no guarantee of being delivered if the JMS provider fails.

**Note**When sending a message, this value is obtained from the `JMSMessage/header/deliveryMode` input parameter.

*JMSDestination*

**Object** Optional. Destination (queue or topic) to which the message was sent.

*JMSExpiration*

**java.lang.Long** Optional. Time at which this message expires. If the message producer did not specify a time-to-live, the *JMSExpiration* value is zero, indicating the message does not expire.

**Note**When sending a message, this value is obtained from the `JMSMessage/header/timeToLive` input parameter.

*JMSMessageID*

**String**. Optional. Unique identifier assigned to this message by the JMS provider.

*JMSPriority*

**java.lang.Integer** Optional. Defines the message priority. The JMS standard defines priority levels from 0 to 9, with 0 as the lowest priority and 9 as the highest.

**Note**When sending a message, this value is obtained from the `JMSMessage/header/priority` input parameter.

*JMSRedelivered*

**java.lang.Boolean** Optional. Flag indicating the JMS provider delivered this message to the JMS client previously. A value of:

- `True` indicates that the message may have been delivered in the past.
- `False` indicates that the JMS provider has not delivered this message previously.

*JMSReplyTo*

**Object** Optional. Destination to which a reply to this message should be sent.

	<i>JMSTimestamp</i>	<b>java.lang.Long</b> Optional. Time at which the message was given to the JMS provider.
	<i>JMSType</i>	<b>String</b> Optional. Message type identifier specified by the client when sending the message.
<i>properties</i>		<b>Document.</b> Optional. A document containing optional fields added to the message header. Integration Server may add the following properties to JMS messages it sends or receives.
	<b>Key</b>	<b>Description</b>
	<i>JMSXDeliveryCount</i>	<b>java.lang.Integer</b> Optional. Specifies the number of times the JMS provider delivered the message. Most JMS providers set this value.
	<i>JMS_WMClusterNodes</i>	<b>String</b> Optional. Contains the name of the Broker in a Broker cluster that will receive the message or the name of the Broker or Brokers in the Broker cluster that received the JMS message.
	<i>activation</i>	<b>String</b> Optional. A unique identifier assigned by the sender. An <i>activation</i> is used to group together messages that will be received by a JMS trigger with a join. A JMS trigger can join together messages with the same <i>activation</i> .
	<i>uuid</i>	<b>String</b> Optional. A universally unique identifier for the message assigned by the sender. Integration Server can use the <i>uuid</i> for exactly-once processing or for request/reply.
<i>body</i>		<b>Document</b> Optional A Document (IData) contenting the JMS message body. Integration Server supports the following formats for the JMS message body:
	<b>Key</b>	<b>Description</b>
	<i>string</i>	<b>String</b> Optional. Message body in the form of a String.

---

<i>bytes</i>	<b>primitive type</b> Optional Message body in the form of a one-dimensional byte array.
<i>object</i>	<b>Object.</b> Optional. Message body in the form of a Serializable Java object.
<i>data</i>	<b>Document</b> Optional. Message body in the form of a document (IData object).  <b>Note</b> This message format can only be used when sending a JMS message from one Integration Server to another. When the JMS message is sent, the sending Integration Server encodes the IData into a byte array. When the receiving Integration Server receives the message, it decodes the byte array into IData.
<i>message</i>	<b>Object</b> Optional. Message body in the form of an actual javax.jms.Message.  <b>Note</b> When a JMS message is received using the <a href="#">pub.jms:receive</a> service this field will always be populated because javax.jms.Message is required for acknowledging the message.  <b>Note</b> When receiving a javax.jms.MapMessage or javax.jms.StreamMessage this field will be populated. The data can then be processed using a Java service. A flow service cannot process the message in its current state.

## Output Parameters

---

None.

## See Also

[pub.jms:receive](#)  
[pub.jms:send](#)  
[pub.jms:sendAndWait](#)

---

## pub.jms:receive

WmPublic. Receives a message from a queue or topic on the JMS provider.

### Input Parameters

---

<i>consumer</i>	<b>Object</b> A message consumer object that the session uses to receive messages sent to the specified destination.
<i>timeout</i>	<b>java.lang.Long</b> Specifies the time to wait, in milliseconds, for a message to be received from the JMS provider. If you specify 0 (zero), the service will not wait. The default is 0 (zero).

### Output Parameters

---

*JMSMessage* **Document** A document (IData) containing the JMS message received by the consumer.

<u>Key</u>	<u>Description</u>
<i>header</i>	<b>Document</b> Conditional. A Document containing the header fields for the received message.
<u>Key</u>	<u>Description</u>
<i>JMSCorrelationID</i>	<b>String</b> Conditional. A unique identifier used to link multiple messages together. Often, a <i>JMSCorrelationID</i> is used to link a reply message with its requesting message.
<i>JMSDeliveryMode</i>	<b>java.lang.Integer</b> Conditional. Delivery mode specified at the time the message was sent.  <b>PERSISTENT</b> indicates the JMS provider places the message in a persistent message store, allowing

---

the message to be recovered in the event of a resource failure.

`NON-PERSISTENT` indicates the JMS provider does not place the message in a persistent store. The message has no guarantee of being delivered if the JMS provider fails.

<i>JMSDestination</i>	<b>Object</b> Conditional. Destination (queue or topic) to which the message was sent.
<i>JMSExpiration</i>	<b>java.lang.Long</b> Conditional. Time at which this message expires. If the message producer did not specify a time-to-live, the <i>JMSExpiration</i> value is zero, indicating the message does not expire.
<i>JMSMessageID</i>	<b>String</b> Conditional. Unique identifier assigned to this message by the JMS provider.
<i>JMSPriority</i>	<b>java.lang.Integer</b> Conditional. Defines the message priority. The JMS standard defines priority levels from 0 to 9, with 0 as the lowest priority and 9 as the highest.
<i>JMSRedelivered</i>	<b>java.lang.Boolean</b> Conditional. Flag indicating the JMS provider delivered this message to the JMS client previously.  <code>True</code> indicates the message may have been delivered in the past.  <code>False</code> indicates the JMS provider has not delivered this message previously.
<i>JMSReplyTo</i>	<b>Object</b> Conditional. Destination to which a reply to this message should be sent.

<i>JMSTimestamp</i>	<b>java.lang.Long</b> Conditional. Time at which the message was given to the JMS provider.
<i>JMSType</i>	<b>String</b> Conditional. Message type identifier specified by the client when sending the message.
<i>properties</i>	<b>Document</b> Conditional. A Document containing optional fields added to the message header. Integration Server may add the following properties to JMS messages it receives.

<u>Key</u>	<u>Description</u>
<i>JMSXDeliveryCount</i>	<b>java.lang.Integer</b> Conditional. Specifies the number of times the JMS provider delivered the message. Most JMS providers set this value.
<i>JMS_WMClusterNodes</i>	<p><b>String</b> Conditional. Name of the Broker or Brokers in the Broker cluster that received the JMS message.</p> <p>The Broker Server acting as the JMS provider populates the <i>JMS_WMClusterNodes</i> parameter after it distributes the JMS message to the Broker or Brokers in the Broker cluster.</p> <p>The <i>JMS_WMClusterNodes</i> value will be null when:</p> <ul style="list-style-type: none"> <li>■ The JMS provider is not the Broker Server.</li> <li>■ The JMS connection alias used to send the JMS message does not use a cluster connection factory to obtain the connection to the Broker Server.</li> <li>■ The cluster connection factory does not permit a policy to be overridden.</li> </ul>

<i>activation</i>	<b>String</b> Conditional. A unique identifier assigned by the sending service. A JMS trigger uses the <i>activation</i> value to determine if a message satisfies a join.
<i>uuid</i>	<b>String</b> Conditional. A universally unique identifier for the message assigned by the sender. Integration Server can use the <i>uuid</i> for exactly-once processing or for request/reply.
<i>body</i>	<b>Document</b> Conditional. A Document (IData) containing the JMS message body. Integration Server supports the following formats for the JMS message body:

<u>Key</u>	<u>Description</u>
<i>string</i>	<b>String</b> Conditional. Message body in the form of a String.
<i>bytes</i>	<b>primitive type</b> Conditional. Message body in the form of a one-dimensional byte array.
<i>object</i>	<b>Object.</b> Conditional. Message body in the form of a Serializable Java object.
<i>data</i>	<b>Document</b> Conditional. Message body in the form of a document (IData object).

**Note:** This message format can only be used when sending a JMS message from one Integration Server to another. When the JMS message is sent, the sending Integration Server encodes the IData into a byte array. When the receiving Integration Server receives the message, it decodes the byte array into IData.

*message*

**Object** Conditional. Message body in the form of an actual `javax.jms.Message`.

**Note** When the JMS message is received using the `pub.jms:receive` service this field will always be populated because `javax.jms.Message` is required for acknowledging the message.

**Note** When receiving a `javax.jms:MapMessage` or `javax.jms:StreamMessage` this field will be populated. The data can then be processed using a Java service. A flow service cannot process the message in its current state.

## Usage Notes

Use this service to receive a message from the JMS provider on demand. Receiving a message on demand provides more control over when and how Integration Server receives a message; however, it may not be as efficient or practical as using a JMS trigger to listen for and then receive the message.

To listen for messages and receive them when they are available, create a JMS trigger that listens to the destination. For more information about creating a JMS trigger, see the *webMethods Service Development Help*.

If the *timeout* period elapses before a message is received, the value of *JMSMessage* is null.

The message consumer that you use to receive the message determines the destination from which this services receives messages and the JMS connection alias used to receive the messages. You can create a message consumer object using the `pub.jms:createConsumer` service.

After you receive a message, you need to invoke a service that processes the message.

If the acknowledgment mode of the consumer is set to `CLIENT_ACKNOWLEDGE`, use the `pub.jms:acknowledge` service to acknowledge the message to the JMS provider.

## See Also

[pub.jms:acknowledge](#)  
[pub.jms:createConsumer](#)

---

## pub.jms:reply

WmPublic. Sends a reply message to a requesting client.

### Input Parameters

---

*JMSReplyMessage* **Document** A document representing the JMS message reply.

<u>Key</u>	<u>Description</u>
<i>header</i>	<b>Document</b> Optional. A document containing the header of the replying JMS message.
<u>Key</u>	<u>Description</u>
<i>deliveryMode</i>	<p><b>String</b> Optional. Specifies the message delivery mode for the reply message. Specify one of the following:</p> <p><b>PERSISTENT</b> Default. Provide once-and-only-once delivery for the message. The message will not be lost if a JMS provider failure occurs.</p> <p><b>NON_PERSISTENT</b> Provide at-most-once delivery for the message. The message has no guarantee of being saved if a JMS provider failure occurs.</p>
<i>priority</i>	<p><b>java.lang.Integer</b> Optional. Specifies the message priority. The JMS standard defines priority levels from 0 to 9, with 0 as the lowest priority and 9 as the highest.</p> <p>The default is 4.</p>
<i>timeToLive</i>	<p><b>java.lang.Long</b> Optional. Length of time, in milliseconds, that the JMS provider system retains the reply message. The default is 0,</p>

meaning that the message does not expire.

*JMSType* **String** Optional. Message type identifier for the message.

*properties* **Document** Optional. A Document containing optional fields added to the message header.

<u>Key</u>	<u>Description</u>
<i>activation</i>	<b>String</b> Optional. A unique identifier that you want to assign to the message. JMS triggers use the <i>activation</i> value to determine if a message satisfies a join.
<i>uuid</i>	<b>String</b> Optional. A universally unique identifier for the message. Integration Server can use the <i>uuid</i> for exactly-once processing or for request/reply.

*body* **Document** Optional. A Document containing the JMS message body. Integration Server supports the following formats for the JMS message body:

<u>Key</u>	<u>Description</u>
<i>string</i>	<b>String</b> Optional. Message body in the form of a String.
<i>bytes</i>	<b>primitive type</b> Optional Message body in the form of a one-dimensional byte array.
<i>object</i>	<b>Object.</b> Optional. Message body in the form of a Serializable Java object.
<i>data</i>	<b>Document</b> Optional. Message body in the form of a document (IData object).

**Note:** This message format can only be used when sending a JMS

message from one Integration Server to another. When the JMS message is sent, the sending Integration Server encodes the IData into a byte array. When the receiving Integration Server receives the message, it decodes the byte array into IData.

*message*

**Object** Optional. Message body in the form of a `javax.jms.Message`.

*consumer*

**Object** Optional. The message consumer object used to receive the request message from the JMS provider. Integration Server uses information from the *consumer* to create a message producer that will send the reply message.

You only need to specify a *consumer* when replying to a message received using `pub.jms:receive`.

*message*

**Object** Optional. A `javax.jms.Message` object that contains the request message. You can map the `JMSMessage/body/message` field in the request message to the `pub.jms:replymessage` input parameter. The `pub.jms:replyservice` uses the request message to determine the *replyTo* destination.

You only need to specify a *message* when replying to a message received using `pub.jms:receive`.

## Output Parameters

*JMSReplyMessage*

**Document.** A Document containing the reply message the JMS provider sent to the client. After it sends a message, the JMS provider populates some fields in the JMS reply message.

**Key**

**Description**

*header*

**Document** Conditional. A Document containing the header fields for the reply message.

*JMSCorrelation ID*

**String** Conditional. A unique identifier used to link the reply message with the initial request message.

---

<i>JMSDeliveryMode</i>	<p>The replying Integration Server automatically sets this value when it executes the <a href="#">pub.jms:reply</a> service.</p>
	<p><b>java.lang.Integer</b> Delivery mode used to send the message.</p>
	<p><code>PERSISTENT</code> indicates that the JMS provider provides once-and-only-once delivery for the message. The message will not be lost if a JMS provider failure occurs.</p>
	<p><code>NON_PERSISTENT</code> indicates that the JMS provider provides at-most-once delivery for the message. The message has no guarantee of being saved if a JMS provider failure occurs.</p>
	<p><b>Note</b> When sending a reply message, this value is obtained from the <code>JMSMessage/header/deliveryMode</code> input parameter.</p>
<i>JMSDestination</i>	<p><b>Object</b> Conditional. Destination (queue or topic) to which the message was sent. The <code>JMSReplyTo</code> value of the request message determines the destination of the reply message.</p>
<i>JMSExpiration</i>	<p><b>java.lang.Long</b> Conditional. Time at which this message expires. If the message producer did not specify a time-to-live, the <code>JMSExpiration</code> value is zero, indicating the message does not expire.</p>
	<p><b>Note</b> When sending a message, this value is obtained from</p>

		the <i>JMSReplyMessage/header/timeToLive</i> input parameter.
<i>JMSMessageID</i>	<b>String</b> Conditional. Unique identifier assigned to this message by the JMS provider.	
<i>JMSPriority</i>	<b>java.lang.Integer</b> Conditional. Defines the message priority. The JMS standard defines priority levels from 0 to 9, with 0 as the lowest priority and 9 as the highest.	
		<b>Note</b> When sending a reply message, this value is obtained from the <i>JMSMessage/header/priority</i> input parameter.
<i>JMSReplyTo</i>	<b>Object</b> Conditional. Specifies the destination to which a response to this message should be sent.	
<i>JMSTimestamp</i>	<b>java.lang.Long</b> Time at which the message was given to the JMS provider.	
<i>JMSType</i>	<b>String</b> Conditional. Message type identifier specified by the client when sending the message.	
<i>properties</i>	<b>Document</b> Conditional. A Document containing optional fields added to the message header. Integration Server may add the following properties to JMS messages it receives.	

<u>Key</u>	<u>Description</u>
<i>activation</i>	<b>String</b> Conditional. A unique identifier assigned by the sending service. A JMS trigger can join together messages with the same <i>activation</i> .

*uuid* **String** Conditional. A universally unique identifier for the message assigned by the sender. Integration Server can use the *uuid* for exactly-once processing or for request/reply.

*body* **Document** Conditional. A Document containing the JMS message body. Integration Server supports the following formats for the JMS message body:

<u>Key</u>	<u>Description</u>
<i>string</i>	<b>String</b> Conditional. Message body in the form of a String.
<i>bytes</i>	<b>primitive type</b> Conditional. Message body in the form of a one-dimensional byte array.
<i>object</i>	<b>Object.</b> Conditional. Message body in the form of a Serializable Java object.
<i>data</i>	<b>Document</b> Conditional. Message body in the form of a document (IData object).

**Note** This message format can only be used when sending a JMS message from one Integration Server to another. When the JMS message is sent, the sending Integration Server encodes the IData into a byte array. When the receiving Integration Server receives the message, it decodes the byte array into IData.

*message* **Object** Conditional. Message body in the form of an actual `javax.jms.Message`.

## Usage Notes

The `pub.jms:reply` service creates a JMS message (`javax.jms.Message`) based on input provided to the service or takes an existing JMS message and sends it to the JMS provider as a reply to a requesting client.

The `JMSReplyTo` field in the request message is set by the sending client and indicates the destination to which the reply will be sent.

The replying Integration Server automatically sets this value when it executes the `pub.jms:reply` service.

When executing the `pub.jms:replyservice`, Integration Server automatically sets the value of the `JMSCorrelationID` field in the `JMSReplyMessage`. Integration Server uses the value of the `wm_tag` field, `uuid` field, or the `JMSMessageID` field in the requesting JMS message to correlate the request and the response. If the `wm_tag` field is populated in the request message, Integration Server uses the `wm_tag` value as the `JMSCorrelationID`. If the request message does not specify a `wm_tag` value and you specify the `uuid` when sending the request, the replying Integration Server will use the `uuid` as the `JMSCorrelationID` of the reply message. If the request message does not specify a `wm_tag` value and you do not specify a `uuid`, the replying Integration Server uses the `JMSMessageID` of the request message as the `JMSCorrelationID` of the reply message.

When replying to a message received using `pub.jms:receive`, you need to specify the input parameters `consumer` and `message`.

If a transaction has not yet been started, the transaction manager starts a transaction context for an implicit transaction when Integration Server executes a `pub.jms:reply` service that uses a transacted JMS connection alias. A JMS connection alias is considered to be transacted when it has a transaction type of XA TRANSACTION or LOCAL TRANSACTION.

If you want more control over the actual `javax.jms.Message` that Integration Server sends to the JMS provider, you can create a Java service that calls the `com.wm.app.b2b.server.jms.producer.ProducerFacade` class, which will create a `javax.jms.Message`. See:

- `com.wm.app.b2b.server.jms.producer.ProducerFacade.createBytesMessage(String)`
- `com.wm.app.b2b.server.jms.producer.ProducerFacade.createMapMessage(String)`
- `com.wm.app.b2b.server.jms.producer.ProducerFacade.createObjectMessage(String)`
- `com.wm.app.b2b.server.jms.producer.ProducerFacade.createStreamMessage(String)`
- `com.wm.app.b2b.server.jms.producer.ProducerFacade.createTextMessage(String)`

The Java service calling this API must return an Object of type `javax.jms.Message`, which can then be mapped to the `JMSMessage/body/message` input parameter of the `pub.jms:reply` service.

When creating the `javax.jms.Message` with the `com.wm.app.b2b.server.jms.producer.ProducerFacade`, you can use the `javax.jms.Message` setter methods to set the values of the message headers and

properties directly. You can also set the value of message headers and properties using the input parameters of the `pub.jms:replyservice` that you use to send the message. If you set the message headers and properties both ways, the values provided to the `pub.jms:replyservice` take precedence.

Software AG recommends that you use a `pub.jms:reply` service to create and send the JMS message. This may provide better performance on average. However, if you want to send a `StreamMessage` or a `MapMessage`, you need to use the appropriate `com.wm.app.b2b.server.jms.producer.ProducerFacade` API.

When using Universal Messaging as the JMS provider, the JMS client can use synchronous or asynchronous publishing. To ensure delivery of a persistent JMS message (*deliveryMode* is set to `PERSISTENT`), Integration Server always uses synchronous publishing to send a persistent JMS message to Universal Messaging.

Message priority is not supported when Universal Messaging is the JMS provider. Any value specified in the *priority* field will be ignored.

#### See Also

[pub.jms:createConsumer](#)  
[pub.jms:receive](#)

---

## pub.jms:send

WmPublic. Sends a JMS message to the JMS provider.

### Input Parameters

---

*connectionAliasName* **String** Name of the JMS connection alias that you want to use to send the message.

The JMS connection alias indicates how Integration Server connects to the JMS provider. A JMS connection alias can specify that Integration Server use a JNDI provider to look up administered objects (connection factories and destinations) and then use the connection factory to create a connection. Alternatively, a JMS connection alias can specify that Integration Server uses the native `webMethods` API to create the connection directly on the `webMethods` Broker.

*destinationName* **String** Name or lookup name of the Destination to which you want to send the message. Specify the lookup name of the Destination object when the JMS connection alias uses JNDI to retrieve administered objects. Specify the provider-specific name of the Destination when the JMS connection alias uses the native `webMethods` API to connect directly to the `webMethods` Broker.

*destinationType* **String** Optional. Type of destination to which you want to send the message. Specify one of the following:

- `QUEUE` to send the message to a particular queue. This is the default.
- `TOPIC` to send the message to a topic.

**Note:** You need to specify *destinationType* only if you specified a *connectionAliasName* that uses the native webMethods API.

*JMSMessage* **Document** A document representing the JMS message you want to send.

<u>Key</u>	<u>Description</u>
------------	--------------------

<i>header</i>	<b>Document</b> Optional. A document containing the header of the JMS message.
---------------	--

<u>Key</u>	<u>Description</u>
------------	--------------------

<i>deliveryMode</i>	<p><b>String</b> Optional. Specifies the message delivery mode for the message. Specify one of the following:</p> <p><b>PERSISTENT</b> Default. Provide once-and-only-once delivery for the message. The message will not be lost if a JMS provider failure occurs.</p> <p><b>NON_PERSISTENT</b> Provide at-most-once delivery for the message. The message has no guarantee of being saved if a JMS provider failure occurs.</p>
---------------------	---

<i>priority</i>	<p><b>java.lang.Integer</b> Optional. Specifies the message priority. The JMS standard defines priority levels from 0 to 9, with 0 as the lowest priority and 9 as the highest.</p> <p>The default is 4.</p>
-----------------	--

---

<i>replyTo</i>	<p><b>String</b> Optional. Name or lookup name of the destination to which you want a reply message sent.</p> <p>If the JMS connection alias used by the <code>pub.jms:send</code> service connects to the JMS provider using JNDI, set <i>replyTo</i> to be the lookup name of the destination lookup object name.</p> <p>If the JMS connection alias used by the <code>pub.jms:send</code> service connects to the JMS provider using a native Broker connection, set <i>replyTo</i> to the Broker queue name. That is, if the JMS connection alias specifies the Broker as the JMS provider and uses the native <code>webMethods</code> API to connect directly to the <code>webMethods</code> Broker, specify the name of the queue on the Broker that should receive replies to the message.</p> <div data-bbox="943 1203 1367 1404"><p><b>Note</b> When using the native <code>webMethods</code> API to connect to the Broker, the <i>replyTo</i> destination must be a queue. Topics are not supported</p></div>
<i>timeToLive</i>	<p><b>java.lang.Long</b> Optional. Length of time, in milliseconds, that the JMS provider retains the message. The default is 0, meaning that the message does not expire.</p>
<i>JMSType</i>	<p>Optional. Message type identifier for the message.</p>
<i>properties</i>	<p><b>Document</b> Optional. A Document containing optional fields added to the message header.</p>

Integration Server adds the following properties to JMS messages it sends.

Key	Description
<i>JMS_WMClusterNodes</i>	<p><b>String</b> Optional. Name of the Broker in a Broker cluster that you want to receive the message. The specified Broker effectively overrides the policy applied to the cluster connection factory used by the JMS connection alias. If the applied policy is multisend guaranteed or multisend best effort, the <i>JMS_WMClusterNodes</i> value should contain multiple Brokers.</p>

**Important:** If you are using the *are AG* requires that you specify the value for *JMS\_WMClusterNodes* by mapping the contents of the service output parameter *JMS\_WMClusterNodes* produced by a previous invocation of `pub.jms:send` or `pub.jms:sendAndWait`.

Use this field to override a Broker cluster policy when all of the following are true:

- The Broker Server is the JMS provider.
- The JMS connection alias used to send the message (*connectionAliasName*) uses a connection from a cluster connection factory.
- The cluster connection factory permits the applied policy to be overridden.

Leave this field blank if the above conditions are

not met or if you want the JMS message to be distributed according to the policy applied to the cluster connection factory.

*activation* **String** Optional. A unique identifier used to group together messages that will be received by a JMS trigger with a join. A JMS trigger can join together messages with the same *activation* .

*uuid* **String** Optional. A universally unique identifier for the message. Integration Server can use the *uuid* for exactly-once processing or for request/reply.

*body* **Document** Optional. A Document containing the JMS message body. Integration Server supports the following formats for the JMS message body:

<b>Key</b>	<b>Description</b>
<i>string</i>	<b>String</b> Optional. Message body in the form of a String.
<i>bytes</i>	<b>primitive type</b> Optional Message body in the form of a one-dimensional byte array.
<i>object</i>	<b>Object.</b> Optional. Message body in the form of a Serializable Java object.
<i>data</i>	<b>Document</b> Optional. Message body in the form of a document (IData object).

**Note** This message format can only be used when sending a JMS message from one Integration Server to another. When the

JMS message is sent, the sending Integration Server encodes the `IData` into a byte array. When the receiving Integration Server receives the message, it decodes the byte array into `IData`.

*message*

**Object** Optional. Message body in the form of an actual `javax.jms.Message`.

*useCSQ*

**java.lang.Boolean** Optional. Flag indicating whether Integration Server places sent messages in the client side queue if the JMS provider is not available at the time the messages are sent. Set to:

- `True` to write messages to the client side queue if the JMS provider is not available at the time this service executes. When the JMS provider becomes available, Integration Server sends messages from the client side queue to the JMS provider.

**Note:** If you want to use the client side queue with the `pub.jms:send` service, the JMS connection alias specified for *connectionAliasName* must be configured to have a client side queue. A JMS connection alias has a client side queue if the **Maximum CSQ Size** property for the alias is set to a value other than 0 (zero).

- `False` to throw an `ISRuntimeException` if the JMS provider is not available at the time this service executes. This is the default.

**Note:** If the specified *connectionAliasName* uses a cluster connection factory to which the multisend guaranteed policy is applied, set *useCSQ* to `False`.

## Output Parameters

*JMSMessage*

**Document.** A Document containing the message sent to the JMS provider.

Key	Description
<i>header</i>	<b>Document</b> Conditional. A Document containing the header fields for the sent message. The JMS provider populates these fields after it has successfully received the message from Integration Server.

Key	Description
<i>JMSCorrelationID</i>	<p><b>String</b> Conditional. A unique identifier used to link messages together.</p>
<i>JMSDeliveryMode</i>	<p><b>java.lang.Integer</b> Delivery mode used to send the message.</p> <p><code>PERSISTENT</code> indicates that the JMS provider provides once-and-only-once delivery for the message. The message will not be lost if a JMS provider failure occurs.</p> <p><code>NON_PERSISTENT</code> indicates that the JMS provider provides at-most-once delivery for the message. The message has no guarantee of being saved if a JMS provider failure occurs.</p> <p><b>Note</b> When sending a message, this value is obtained from the <i>JMSMessage/header/deliveryMode</i> input parameter.</p>
<i>JMSDestination</i>	<p><b>Object</b> Conditional. Destination (queue or topic) to which the message was sent.</p>
<i>JMSExpiration</i>	<p><b>java.lang.Long</b> Conditional. Time at which this message expires. If the message producer did not specify a time-to-live, the <i>JMSExpiration</i> value is zero, indicating the message does not expire.</p> <p><b>Note</b> When sending a message, this value is obtained from the <i>JMSMessage/header/timeToLive</i> input parameter.</p>

<i>JMSMessageID</i>	<b>String</b> Conditional. Unique identifier assigned to this message by the JMS provider.
<i>JMSPriority</i>	<b>java.lang.Integer</b> Conditional. Defines the message priority. The JMS standard defines priority levels from 0 to 9, with 0 as the lowest priority and 9 as the highest.
	<b>Note</b> When sending a message, this value is obtained from the <i>JMSMessage/header/priority</i> input parameter.
<i>JMSReplyTo</i>	<b>Object</b> Conditional. Specifies the destination to which a response to this message should be sent.
<i>JMSTimestamp</i>	<b>java.lang.Long</b> Time at which the message was given to the JMS provider.
<i>JMSType</i>	<b>String</b> Conditional. Message type identifier specified by the client when sending the message.
<i>properties</i>	<b>Document</b> Conditional. A Document containing optional fields added to the message header. Integration Server adds the following properties to JMS messages it sends.

<u>Key</u>	<u>Description</u>
<i>JMS_WMClusterNodes</i>	<p><b>String</b> Conditional. Name of the Broker or Brokers in the Broker cluster that received the JMS message.</p> <p>The Broker Server acting as the JMS provider populates the <i>JMS_WMClusterNodes</i> parameter after it distributes the</p>

JMS message to the Broker or Brokers in the Broker cluster.

The *JMS\_WMClusterNodes* value will be null when:

- The JMS provider is not the Broker Server.
- The JMS connection alias used to send the JMS message does not use a cluster connection factory to obtain the connection to the Broker Server.
- The cluster connection factory does not permit a policy to be overridden.

*activation*

**String** Conditional. A unique identifier assigned by the sender. A JMS trigger can join together messages with the same *activation* .

*uuid*

**String** Conditional. A universally unique identifier for the message assigned by the sender. Integration Server can use the *uuid* for exactly-once processing or for request/reply.

*body*

**Document** Conditional. A Document containing the JMS message body. Integration Server supports the following formats for the JMS message body:

Key	Description
<i>string</i>	<b>String</b> Conditional. Message body in the form of a String.
<i>bytes</i>	<b>primitive type</b> Conditional Message body in the form of a one-dimensional byte array.

<i>object</i>	<b>Object.</b> Conditional. Message body in the form of a Serializable Java object.
<i>data</i>	<b>Document</b> Conditional. Message body in the form of a document (IData object).

**Note** This message format can only be used when sending a JMS message from one Integration Server to another. When the JMS message is sent, the sending Integration Server encodes the IData into a byte array. When the receiving Integration Server receives the message, it decodes the byte array into IData.

<i>message</i>	<b>Object</b> Conditional. Message body in the form of an actual javax.jms.Message.
----------------	---

## Usage Notes

The `pub.jms:send` service creates a JMS message (`javax.jms.Message`) based on input provided to the service or takes an existing JMS message and sends it to the JMS provider.

If a transaction has not yet been started, the transaction manager starts a transaction context for an implicit transaction when Integration Server executes a `pub.jms:send` service that uses a transacted JMS connection alias. A JMS connection alias is considered to be transacted when it has a transaction type of XA TRANSACTION or LOCAL TRANSACTION.

You can add properties to a JMS message when building a flow service that invokes this service. In Designer, use the Pipeline view to add a new variable to *JMSMessage/properties* document.

If the JMS connection alias specified for *connectionAliasName* uses the native webMethods API, you need to specify *destinationName* and *destinationType* to indicate where the webMethods Broker should send the message.

Integration Server creates the output parameter *JMSMessage* because some of the *header* fields in a JMS message are populated by the JMS provider after the message is sent. For example, the *header* field *JMSMessageID* is not in the JMS message sent by Integration Server, but *JMSMessageID* is in the *header* after the JMS provider receives the message.

Each JMS connection alias can be configured to have its own client side queue. A JMS connection alias has a client side queue if the **Maximum CSQ Size** property for the alias is set to a value other than 0 (zero). If you want to use the client side queue with the `pub.jms:send` service, the JMS connection alias specified for *connectionAliasName* must be configured to have a client side queue. If the JMS connection alias is configured to use a client side queue and *useCSQ* is set to true, Integration Server places messages in the client side queue if the JMS provider is not available at the time the `pub.jms:send` service executes. When the JMS provider becomes available, Integration Server sends messages from the client side queue to the JMS provider.

The JMS provider populates the header fields in the *JMSMessage* output parameter after it successfully receives the sent message from Integration Server. If the JMS provider is not available at the time `pub.jms:send` executes and *useCSQ* is set to true, the *header* fields in the output *JMSMessage* will not be populated. Instead these fields will be blank or be set to 0 (zero).

If the client side queue is not in use (*useCSQ* is set to false and/or the JMS connection alias is not configured to use a client side queue), Integration Server throws an `ISRuntimeException` if the JMS provider is not available when this service executes. Make sure to code your service to handle this situation.

A JMS connection alias can be configured so that Integration Server retries the `pub.jms:send` service automatically when the service fails because of a transient error. For more information about configuring a JMS connection alias for automatic retry, see the *webMethods Integration Server Administrator's Guide*.

When sending a message as part of a transaction the client side queue cannot be used. That is, the *useCSQ* field should be set to false. If *useCSQ* is set to true, Integration Server throws a `JMSSubsystemException` when the `pub.jms:send` service executes. A JMS message is sent as part of a transaction if the JMS connection alias specified in *connectionAliasName* :

- Uses a transaction type of `LOCAL_TRANSACTION` or `XA_TRANSACTION`.
- Connects to the webMethods Broker using a cluster connection factory to which the multisend guaranteed policy is applied. Integration Server uses an XA transaction to perform a two-phase commit when sending JMS messages.

If you want more control over the actual `javax.jms.Message` that Integration Server sends to the JMS provider, you can create a Java service that calls the `com.wm.app.b2b.server.jms.producer.ProducerFacade` class, which will create a `javax.jms.Message`. See:

- `com.wm.app.b2b.server.jms.producer.ProducerFacade.createBytesMessage(String)`
- `com.wm.app.b2b.server.jms.producer.ProducerFacade.createMapMessage(String)`
- `com.wm.app.b2b.server.jms.producer.ProducerFacade.createObjectMessage(String)`
- `com.wm.app.b2b.server.jms.producer.ProducerFacade.createStreamMessage(String)`
- `com.wm.app.b2b.server.jms.producer.ProducerFacade.createTextMessage(String)`

The Java service calling this API must return an Object of type `javax.jms.Message`, which can then be mapped to the `JMSMessage/body/message` input parameter of the `pub.jms:send` service.

When creating the `javax.jms.Message` with the `com.wm.app.b2b.server.jms.producer.ProducerFacade`, you can use the `javax.jms.Message` setter methods to set the values of the message headers and properties directly. You can also set the value of message headers and properties using the input parameters of the `pub.jms:send` service that you use to send the message. If you set the message headers and properties both ways, the values provided to the `pub.jms:send` service take precedence.

Software AG recommends that you use a `pub.jms:send` service to create and send the JMS message. This may provide better performance on average. However, if you want to send a `StreamMessage` or a `MapMessage`, you need to use the appropriate `com.wm.app.b2b.server.jms.producer.ProducerFacade` API.

To send a `StreamMessage`, create a Java service that calls `com.wm.app.b2b.server.jms.producer.ProducerFacade.createStreamMessage(String)`. The Java service calling this API must return an Object of type `javax.jms.Message`. Map the `javax.jms.Message` object to the `JMSMessage/body/message` input parameter of the `pub.jms:send` service.

To send a `MapMessage`, create a Java service that calls `com.wm.app.b2b.server.jms.producer.ProducerFacade.createMapMessage(String)`. The Java service calling this API must return an Object of type `javax.jms.Message`. Map the `javax.jms.Message` object to the `JMSMessage/body/message` input parameter of the `pub.jms:send` service.

If you use the input parameter `JMS_WMClusterNodes` to override the policy applied to the cluster connection factory, make sure to code the invoking service to handle any exception that the Broker Server throws when policy requirements are not or cannot be met. For more information about policy override scenarios that might result in an exception from Broker Server, see *Using webMethods Integration Server to Build a Client for JMS*.

You can use the `pub.jms:send` service to specify a destination for response messages when you do not need to wait for the response. The act of waiting for a response message comes with extra overhead for Integration Server which is unnecessary if you merely want to specify a `JMSReplyTo` destination but do not want the sending service to wait for a reply.

When executing the `pub.jms:send` service with a valid value for the `JMSMessage/header/replyTo` parameter, Integration Server creates the `javax.jms.Destination` and maps it to the `JMSReplyTo` field within the message header. Integration Server sends the message and returns immediately. The service does not wait for the response message. If `JMSMessage/header/replyTo` is empty, then Integration Server does not set the `JMSReplyTo` header for the JMS message. If `JMSMessage/header/replyTo` is invalid, then Integration Server throws a `ServiceException`.

---

## pub.jms:sendAndWait

WmPublic. Sends a request in the form of a JMS message to the JMS provider and optionally, waits for a reply.

### Input Parameters

---

<i>connectionAliasName</i>	<p><b>String</b> Name of the JMS connection alias that you want to use to send the message.</p> <p>The JMS connection alias indicates how Integration Server connects to the JMS provider. A JMS connection alias can specify that Integration Server use a JNDI provider to look up administered objects (connection factories and destinations) and then use the connection factory to create a connection. Alternatively, a JMS connection alias can specify that Integration Server uses the native webMethods API to create the connection directly on the webMethods Broker.</p>
<i>destinationName</i>	<p><b>String</b> Name or lookup name of the Destination to which you want to send the message. Specify the lookup name of the Destination object when the JMS connection alias uses JNDI to retrieve administered objects. Specify the provider-specific name of the Destination when the JMS connection alias uses the native webMethods API to connect directly to the webMethods Broker.</p>
<i>destinationType</i>	<p><b>String</b> Optional. Type of destination to which you want to send the message. Specify one of the following:</p> <ul style="list-style-type: none"><li>■ <code>QUEUE</code> to send the message to a particular queue. This is the default.</li><li>■ <code>TOPIC</code> to send the message to a topic.</li></ul> <p><b>Note:</b> You need to specify a <i>destinationType</i> only if you specified a <i>connectionAliasName</i> that uses the native webMethods API.</p>
<i>destinationName</i> <i>ReplyTo</i>	<p><b>String</b> Optional. Name or lookup name of the Destination to which you want the reply message sent. Specify the lookup name of the Destination object when the JMS connection alias uses JNDI to retrieve administered objects. Specify the provider-specific name of the Destination when the JMS connection alias uses the native webMethods API to connect directly to the webMethods Broker.</p>

If you do not specify a destination for reply messages, Integration Server uses a temporaryQueue to receive the reply. A *temporaryQueue* is a queue object created for the duration of a particular connection. It can only be consumed by the connection from which it was created.

If you want to use a dedicated listener (MessageConsumer) to retrieve replies to all requests sent using a particular JMS connection alias, do not specify a value for *destinationNameReplyTo*. A dedicated MessageConsumer can retrieve replies for a synchronous request/reply only.

If you want to use the client side queue with an asynchronous request-reply, you must specify a queue that is not temporary as the *destinationNameReplyTo* value.

*destinationTypeReplyTo*

**String** Optional. Type of destination to which you want the reply to be sent. Specify one of the following:

- `QUEUE` to send the reply message to a particular queue. This is the default.
- `TOPIC` to send the reply message to a specific topic.

*timeout*

**java.lang.Long** Optional. Time to wait (in milliseconds) for the response to arrive. If no value is specified, the service does not wait for a reply and returns a null document. You must specify a value greater than zero.

The *timeout* value only applies for a synchronous request/reply. If *async* is set to `true`, Integration Server ignores the *timeout* value.

*JMSMessage*

**Document** A document representing the JMS message you want to send.

Key	Description
<i>header</i>	<b>Document</b> Optional. A document containing the header of the JMS message.
Key	Description
<i>deliveryMode</i>	<b>String</b> Optional. Specifies the message delivery mode for the message. Specify one of the following:

	<b>PERSISTENT</b>
	Default. Provide once-and-only-once delivery for the message. The message will not be lost if a JMS provider failure occurs.
	<b>NON_PERSISTENT</b>
	Provide at-most-once delivery for the message. The message has no guarantee of being saved if a JMS provider failure occurs.
<i>priority</i>	<b>java.lang.Integer</b> Optional. Specifies the message priority. The JMS standard defines priority levels from 0 to 9, with 0 as the lowest priority and 9 as the highest.  The default is 4.
<i>timeToLive</i>	<b>java.lang.Long</b> Optional. Length of time, in milliseconds, that the JMS provider retains the message. The default is 0, meaning that the message does not expire.
<i>JMSType</i>	<b>String</b> Optional. Message type identifier for the message. Integration Server expects the reply message to be of this type.
<i>properties</i>	<b>Document</b> Optional. A Document containing optional fields added to the message header.  You can add a custom property to a JMS. Click  on the Pipeline view. Select a data type for the property and assign it a name. Assign a value to any custom properties that you add.

**Note:** Integration Server reserves use of a message property named *wm\_tag* for internal purposes. Do not add a property named *wm\_tag* to the JMS message. Integration Server may overwrite a supplied value for *wm\_tag*.

Integration Server adds the following properties to JMS messages it sends.

Key	Description
<i>JMS_WMClusterNodes</i>	<p><b>String</b> Optional. Name of the Broker in a Broker cluster that you want to receive the message. The specified Broker effectively overrides the policy applied to the cluster connection factory used by the JMS connection alias. If the applied policy is multiseed guaranteed or multiseed best effort, the <i>JMS_WMClusterNodes</i> value should contain multiple Brokers.</p>

**Important** Software AG requires that you specify the value for *JMS\_WMClusterNodes* by mapping the contents of the service output parameter *JMS\_WMClusterNodes* produced by a previous invocation of `pub.jms:send` or `pub.jms:sendAndWait`.

Use this field to override a Broker cluster policy when all of the following are true:

- The Broker Server is the JMS provider.
- The JMS connection alias used to send the message (*connectionAliasName*) uses a connection from a cluster connection factory.
- The cluster connection factory permits the applied policy to be overridden.

Leave this field blank if the above conditions are

not met or if you want the JMS message to be distributed according to the policy applied to the cluster connection factory.

*activation* **String** Optional. A unique identifier used to group together messages that will be received by a JMS trigger with a join. A JMS trigger can join together messages with the same *activation* .

*uuid* **String** Optional. A universally unique identifier for the message. Integration Server can use the *uuid* for exactly-once processing or for request/reply.

*body* **Document** Optional. A Document containing the JMS message body. Integration Server supports the following formats for the JMS message body:

<u>Key</u>	<u>Description</u>
<i>string</i>	<b>String</b> Optional. Message body in the form of a String.
<i>bytes</i>	<b>primitive type</b> Optional. Message body in the form of a one-dimensional byte array.
<i>object</i>	<b>Object</b> . Optional. Message body in the form of a Serializable Java object.
<i>data</i>	<b>Document</b> Optional. Message body in the form of a document (IData object).

**Note** This message format can only be used when sending a JMS message from one Integration Server

to another. When the JMS message is sent, the sending Integration Server encodes the `IData` into a byte array. When the receiving Integration Server receives the message, it decodes the byte array into `IData`.

*message*

**Object** Optional. Message body in the form of an actual `javax.jms.Message`.

*async*

**java.lang.Boolean** Optional. Flag specifying whether this is an asynchronous or synchronous request/reply. Set to:

- `True` to indicate that this is an asynchronous request/reply. After sending the message, Integration Server executes the next step in the flow service immediately. The Integration Server does not wait for a reply before continuing service execution.

**Note:** To retrieve the reply to an asynchronous send, invoke the [pub.jms:waitForReply](#) service.

- `False` to indicate that this is a synchronous request/reply. After sending the message, the Integration Server waits for a reply before executing the next step in the flow service. This is the default.

*useCSQ*

**java.lang.Boolean** Optional. Flag indicating whether Integration Server places sent messages in the client side queue if the JMS provider is not available at the time the messages are sent. Set to:

- `True` to write messages to the client side queue if the JMS provider is not available at the time this service executes. When the JMS provider becomes available, Integration Server sends messages from the client side queue to the JMS provider.

**Note:** If you want to use the client side queue, the JMS connection alias specified for *connectionAliasName* must be configured to have a client side queue. A JMS connection alias has a client side queue if the **Maximum CSQ Size** property for the alias is set to a value other than 0 (zero).

- `False` to throw an `ISRuntimeException` if the JMS provider is not available at the time this service executes. This is the default.

**Note:** Integration Server can write messages to the client side queue only for messages sent as part of an asynchronous request/reply. That is, if `async` is set to `true` (the default) and the JMS provider is not available at the time this service executes, Integration Server places the message in the client side queue.

**Note:** The client side queue cannot be used if the reply destination is a temporary queue. Set `useCSQ` to `False` if `destinationNameReplyTo` is not specified or is a temporary queue.

**Note:** If the specified `connectionAliasName` uses a cluster connection factory to which the multisend guaranteed policy is applied, set `useCSQ` to `False`.

## Output Parameters

*JMSMessage*

**Document.** A Document containing the message sent to the JMS provider.

<u>Key</u>	<u>Description</u>						
<i>header</i>	<b>Document</b> Conditional. A Document containing the header fields for the sent message. The JMS provider populates these fields after it has successfully received the message from Integration Server.						
	<table border="1"> <thead> <tr> <th><u>Key</u></th> <th><u>Description</u></th> </tr> </thead> <tbody> <tr> <td><i>JMSCorrelationID</i></td> <td><b>String</b> Conditional. A unique identifier used to link messages together.</td> </tr> <tr> <td><i>JMSDeliveryMode</i></td> <td><b>java.lang.Integer</b> Delivery mode used to send the message.  <code>PERSISTENT</code> indicates that the JMS provider provides once-and-only-once delivery</td> </tr> </tbody> </table>	<u>Key</u>	<u>Description</u>	<i>JMSCorrelationID</i>	<b>String</b> Conditional. A unique identifier used to link messages together.	<i>JMSDeliveryMode</i>	<b>java.lang.Integer</b> Delivery mode used to send the message.  <code>PERSISTENT</code> indicates that the JMS provider provides once-and-only-once delivery
<u>Key</u>	<u>Description</u>						
<i>JMSCorrelationID</i>	<b>String</b> Conditional. A unique identifier used to link messages together.						
<i>JMSDeliveryMode</i>	<b>java.lang.Integer</b> Delivery mode used to send the message.  <code>PERSISTENT</code> indicates that the JMS provider provides once-and-only-once delivery						

for the message. The message will not be lost if a JMS provider failure occurs.

`NON_PERSISTENT` indicates that the JMS provider provides at-most-once delivery for the message. The message has no guarantee of being saved if a JMS provider failure occurs.

**Note** When sending a message, this value is obtained from the `JMSMessage/header/deliveryMode` input parameter.

*JMSDestination*

**Object** Conditional. Destination (queue or topic) to which the message was sent.

*JMSExpiration*

**java.lang.Long** Optional. Time at which this message expires. If the message producer did not specify a time-to-live, the *JMSExpiration* value is zero, indicating the message does not expire.

**Note** When sending a message, this value is obtained from the `JMSMessage/header/timeToLive` input parameter.

*JMSMessageID*

**String** Conditional. Unique identifier assigned to this message by the JMS provider.

*JMSPriority*

**java.lang.Integer** Optional. Defines the message priority. The JMS standard defines priority levels from

0 to 9, with 0 as the lowest priority and 9 as the highest.

**Note** When sending a message, this value is obtained from the *JMSMessage/header/priority* input parameter.

*JMSReplyTo*

**Object** Conditional. Specifies the destination to which a reply to this message should be sent. The *destinationNameReplyTo* value determines the value of *JMSReplyTo*.

*JMSTimestamp*

**java.lang.Long** Time at which the message was given to the JMS provider.

*JMSType*

**String** Conditional. Message type identifier specified by the client when sending the message.

*properties*

**Document** Conditional. A Document containing optional fields added to the message header. Integration Server adds the following properties to JMS messages it sends.

**Key**

**Description**

*JMS\_WMClusterNodes*

**String** Conditional. Name of the Broker or Brokers in the Broker cluster that received the JMS message.

The Broker Server acting as the JMS provider populates the *JMS\_WMClusterNodes* parameter after it distributes the JMS message to the Broker or Brokers in the Broker cluster.

The *JMS\_WMClusterNodes* value will be null when:

- The JMS provider is not the Broker Server.
- The JMS connection alias used to send the JMS message does not use a cluster connection factory to obtain the connection to the Broker Server.
- The cluster connection factory does not permit a policy to be overridden.

*activation*      **String** Conditional. A unique identifier assigned by the sender. A JMS trigger can join together messages with the same *activation*.

*uuid*      **String** Conditional. A universally unique identifier for the message assigned by the sender. Integration Server can use the *uuid* for exactly-once processing or for request/reply.

**Note:** Integration Server adds a *wm\_tag* property to a synchronous request message sent using a JMS connection alias that uses a dedicated MessageConsumer to retrieve replies. Integration Server reserves the use of a JMS message property named *wm\_tag* for internal purposes. Integration Server may overwrite any user-supplied value for *wm\_tag*.

*body*      **Document** Conditional. A Document containing the JMS message body. Integration Server supports the following formats for the JMS message body:

Key	Description
<i>string</i>	<b>String</b> Conditional. Message body in the form of a String.

<i>bytes</i>	<b>primitive type</b> Conditional Message body in the form of a one-dimensional byte array.
<i>object</i>	<b>Object.</b> Conditional. Message body in the form of a Serializable Java object.
<i>data</i>	<b>Document</b> Conditional. Message body in the form of a document (IData object).
	<b>Note</b> This message format can only be used when sending a JMS message from one Integration Server to another. When the JMS message is sent, the sending Integration Server encodes the IData into a byte array. When the receiving Integration Server receives the message, it decodes the byte array into IData.
<i>message</i>	<b>Object</b> Conditional. Message body in the form of an actual javax.jms.Message.

*JMSReplyMessage*

**Document** Conditional. Document containing the JMS message received as a reply.

If this is a synchronous request/reply and Integration Server does not receive a reply before the specified *timeout* value elapses or if *timeout* was not set, the *JMSReplyMessage* is null.

If this is an asynchronous reply, the *JMSReplyMessage* is null.

**Key****Description***header*

**Document** Conditional. A Document containing the header fields for the reply message.

Key	Description
<i>JMSCorrelationID</i>	<p><b>String</b> Conditional. A unique identifier used to link the reply message with the initial request message.</p> <p>The replying Integration Server automatically sets this value when it executes the <a href="#">pub.jms:reply</a> service.</p>
<i>JMSDeliveryMode</i>	<p><b>java.lang.Integer</b> Conditional. Delivery mode used to send the message.</p> <p><code>PERSISTENT</code> indicates that the JMS provider provides once-and-only-once delivery for the message. The message will not be lost if a JMS provider failure occurs.</p> <p><code>NON_PERSISTENT</code> indicates that the JMS provider provides at-most-once delivery for the message. The message has no guarantee of being saved if a JMS provider failure occurs.</p>
<i>JMSDestination</i>	<p><b>Object</b> Conditional. Destination (queue or topic) to which the message was sent.</p>
<i>JMSExpiration</i>	<p><b>java.lang.Long</b> Conditional. Time at which this message expires. If the message producer did not specify a time-to-live, the <i>JMSExpiration</i> value is zero, indicating the message does not expire.</p>
<i>JMSMessageID</i>	<p><b>String</b> Conditional. Unique identifier assigned to</p>

this message by the JMS provider.

*JMSPriority*      **java.lang.Integer** Conditional. Defines the message priority. The JMS standard defines priority levels from 0 to 9, with 0 as the lowest priority and 9 as the highest.

*JMSRedelivered*      **java.lang.Boolean** Conditional. Flag indicating the JMS provider delivered this message to the JMS client previously.

`True` indicates the message may have been delivered in the past.

`False` indicates the JMS provider has not delivered this message previously.

*JMSReplyTo*      **Object** Conditional. Specifies the destination to which a response to this message should be sent.

*JMSTimestamp*      **java.lang.Long** Conditional. Time at which the message was given to the JMS provider.

*JMSType*      **String** Conditional. Message type identifier specified by the client when sending the message.

*properties*      **Document** Conditional. A Document containing optional fields added to the message header. Integration Server adds the following proprieties to JMS messages it receives.

<u>Key</u>	<u>Description</u>
------------	--------------------

---

<i>JMSXDeliveryCount</i>	<b>java.lang.Integer</b> Conditional. Specifies the number of times the JMS provider delivered the message. Most JMS providers set this value.
<i>JMS_WMClusterNodes</i>	<b>String</b> Conditional. Name of the Broker or Brokers in the Broker cluster that received the JMS message.  The Broker Server acting as the JMS provider populates the <i>JMS_WMClusterNodes</i> parameter after it distributes the JMS message to the Broker or Brokers in the Broker cluster.  The <i>JMS_WMClusterNodes</i> value will be null when: <ul style="list-style-type: none"><li>■ The JMS provider is not the Broker Server.</li><li>■ The JMS connection alias used to send the JMS message does not use a cluster connection factory to obtain the connection to the Broker Server.</li><li>■ The cluster connection factory does not permit a policy to be overridden.</li></ul>
<i>activation</i>	<b>String</b> Conditional. A unique identifier assigned by the sender. A JMS trigger uses the <i>activation</i> value to determine whether a message satisfies a join.
<i>uuid</i>	<b>String</b> Conditional. A universally unique identifier for the message assigned by the sender. Integration Server can use the <i>uuid</i> for

exactly-once processing or for request/reply.

*body*

**Document** Conditional. A Document containing the JMS message body. Integration Server supports the following formats for the JMS message body:

<u>Key</u>	<u>Description</u>
<i>string</i>	<b>String</b> Conditional. Message body in the form of a String.
<i>bytes</i>	<b>primitive type</b> Conditional. Message body in the form of a one-dimensional byte array.
<i>object</i>	<b>Object.</b> Conditional. Message body in the form of a Serializable Java object.
<i>data</i>	<b>Document</b> Conditional. Message body in the form of a document (IData object).
	<p><b>Note</b> This message format can only be used when sending a JMS message from one Integration Server to another. When the JMS message is sent, the sending Integration Server encodes the IData into a byte array. When the receiving Integration Server receives the message, it decodes the byte array into IData.</p>
<i>message</i>	<b>Object</b> Conditional. Message body in the form of an actual javax.jms.Message.

## Usage Notes

The `pub.jms:sendAndWait` service creates a JMS message (`javax.jms.Message`) based on input provided to the service or takes an existing JMS message, sends it to the JMS provider and optionally, waits for a reply.

If a transaction has not been started, the transaction manager starts a transaction context for an implicit transaction when Integration Server executes a `pub.jms:sendAndWait` service that uses a transacted JMS connection alias. A JMS connection alias is considered to be transacted when it has a transaction type of XA TRANSACTION or LOCAL TRANSACTION.

You can add properties to a JMS message when building a flow service that invokes this service. In Designer, use the Pipeline view to add a new variable to *JMSMessage/properties* document.

If the JMS connection alias specified for *connectionAliasName* uses the native `webMethods` API, you need to specify *destinationName* and *destinationType* to indicate where the `webMethods` Broker should send the message.

Integration Server creates the output parameter *JMSMessage* because some of the *header* fields in a JMS message are populated by the JMS provider after the message is sent. For example, the *header* field *JMSMessageID* is not in the JMS message sent by Integration Server, but *JMSMessageID* is in the *header* after the JMS provider receives the message.

You can use the `pub.jms:sendAndWait` service to initiate a request/reply. The sending client sends a request for information to either a topic or queue. Clients that subscribe to the destination compose and send a reply document that contains the information requested by the sender.

A single request might receive many reply messages. Integration Server that sent the request uses only the first reply document it receives from the JMS provider. Integration Server discards all other replies. First is arbitrarily defined. There is no guarantee provided for the order in which the JMS provider processes incoming replies.

The `pub.jms:sendAndWait` service can be useful in situations where multiple sources contain the response data. For example, suppose that an enterprise uses one application for managing customer data, another for storing master customer records, and a mainframe system for saving customer lists. Each of these applications could answer a request for customer data. The requesting service will use the first reply message it receives.

The `pub.jms:sendAndWait` service can issue a request/reply in a synchronous or asynchronous manner.

- In a synchronous request/reply, the service that sends the request stops executing while it waits for a reply. When the service receives a reply message, the service resumes execution. If the *timeout* elapses before the service receives a reply, Integration Server ends the request, and the service returns a null message that indicates that the request timed out. Integration Server then executes the next step in the flow service.
- In an asynchronous request/reply, the service that sends the request continues executing the steps in the service after sending the message. To retrieve the reply,

the requesting flow service must invoke the `pub.jms:waitForReply` service. If the *timeout* value specified in `pub.jms:waitForReply` elapses before the `pub.jms:waitForReply` service receives a reply, the `pub.jms:waitForReply` service returns a null document indicating that the request timed out.

When using `pub.jms:sendAndWait` to issue a request/reply, you must specify a queue as the value of the *destinationNameReplyTo* parameter. In a request/reply scenario, it is possible that the message consumer created to receive the reply might be created after the reply message is sent. (In a synchronous request/reply, the `pub.jms:sendAndWait` service creates the message consumer. In an asynchronous request/reply, the `pub.jms:waitForReply` service or a custom solution, such as a JMS trigger, creates the message consumer.) If the reply destination is a queue, a message consumer can receive messages published to the queue regardless of whether the message consumer was active at the time the message was published. If the destination is a topic, a message consumer can receive only messages published when the message consumer was active. If the reply is sent to a topic before the message consumer is created, the message consumer will not receive the reply. Consequently, when creating a request/reply, the *destinationNameReplyTo* parameter should specify the name or lookup name of a queue.

**Note:** If you are using a dedicated listener (`MessageConsumer`) to retrieve replies to all of the requests sent using a particular JMS connection alias, do not specify a value for *destinationNameReplyTo*.

A service that contains multiple asynchronous send and wait invocations allows the service to send all the requests before collecting the replies. This approach can be more efficient than sending a request, waiting for a reply, and then sending the next request.

The replying Integration Server uses the value of the *wm\_tag*, *uuid* or *JMSMessageID* in the requesting JMS message to correlate the request and the response. For more information, see [pub.jms:reply](#).

If you create a service that contains multiple asynchronous requests, make sure to link the *JMSMessage* field (*uuid* or *JMSMessageID*) whose value will be used as the reply message's *JMSCorrelationID* to another field in the pipeline. Each asynchronous request produces a *JMSMessage* document in the pipeline. If you do not link the *uuid* or *JMSMessageID* field from the *JMSMessage* document to another field, the next asynchronous request (that is, the next execution of the `pub.jms:sendAndWait` service), will overwrite the previous *JMSMessage* document. When you invoke the `pub.jms:waitForReply` service, the pipeline will contain only the input needed to retrieve the reply to the last request. The pipeline will not contain the information needed to retrieve replies to the previous requests. (That is, there will be nothing to map to the *correlationID* input parameter of the `pub.jms:waitForReply` service.)

Each JMS connection alias can be configured to have its own client side queue. A JMS connection alias has a client side queue if the **Maximum CSQ Size** property for the alias is set to a value other than 0 (zero). If you want to use the client side queue with the `pub.jms:sendAndWait` service, the JMS connection alias specified for *connectionAliasName* must be configured to have a client side queue. If the JMS connection alias is configured to use a client side queue and *useCSQ* is set to true, Integration Server places messages in the client side queue if the JMS provider is not available at the time

the `pub.jms:sendAndWait` service executes. When the JMS provider becomes available, Integration Server sends messages from the client side queue to the JMS provider.

If the client side queue is not used (`useCSQ` is set to false or the JMS connection alias is not configured to have a client side queue), Integration Server throws an `ISRuntimeException` if the JMS provider is not available when this service executes. Make sure to code your service to handle this situation.

Integration Server can write messages to the client side queue only for messages sent as part of an asynchronous request/reply. That is, if `async` is set to true (the default) and the JMS provider is not available at the time this service executes, Integration Server places the message in the client side queue. The client side queue cannot be used for a synchronous request/reply.

The client side queue cannot be used if the reply destination is a temporary queue. Consequently, if `useCSQ` is set to true, values must be specified for the `destinationNameReplyTo` and `destinationTypeReplyTo` input parameters. If these parameters are not specified, Integration Server throws the following `ServiceException` when it executes the `pub.jms:sendAndWait` service: [ISS.0134.9082] The client side queue cannot be used with a send and wait request if the reply destination is a temporary queue.

The JMS provider populates the header fields in the `JMSMessage` output parameter after it successfully receives the sent message from Integration Server. If the JMS provider is not available at the time the `pub.jms:sendAndWait` executes and `useCSQ` is set to true, the `header` fields in the output `JMSMessage` will not be populated. Instead these fields will be blank or be set to 0 (zero).

The `pub.jms:waitForReply` service cannot be used to retrieve response to requests that were routed through the client side queue. To retrieve the response, create a JMS trigger that subscribes to the reply to queue.

If the `pub.jms:sendAndWait` service executes and the message is sent directly to the JMS provider (i.e., it is not sent to the client side queue), the `JMSMessage \header \JMSMessageID` contains a unique identifier assigned by the JMS provider. If the `JMSMessageID` field is null after the service executes, the JMS provider was not available at the time the service executed. Integration Server wrote the message to the client side queue.

When sending a message as part of a transaction client side queuing cannot be used. That is, the `useCSQ` field should be set to false. If `useCSQ` is set to true, Integration Server throws a `JMSSubsystemException` when the `pub.jms:sendAndWait` service executes. A JMS message is sent as part of a transaction if the JMS connection alias specified in `connectionAliasName` :

- Uses a transaction type of `LOCAL_TRANSACTION` or `XA_TRANSACTION`.
- Connects to the webMethods Broker using a cluster connection factory to which the multisend guaranteed policy is applied. Integration Server uses an XA transaction to perform a two-phase commit when sending JMS messages.

If you do not specify a destination for reply messages, Integration Server uses a `temporaryQueue` to receive the reply. A `temporaryQueue` is a queue object created for

the duration of a particular connection. It can only be consumed by the connection from which it was created.

To use a dedicated listener (`MessageConsumer`) to retrieve replies for a request, the `pub.jms:sendAndWait` invocation must specify the following:

- The `connectionAliasName` input parameter must specify a JMS connection alias that is configured to use a dedicated message consumer. Specifically, the **Create Temporary Queue** and **Enable Request-Reply Listener for Temporary Queue** check boxes must be selected for the alias.
- The request must be asynchronous. The `async` input parameter must be set to false.
- There must not be a value specified for the `destinationNameReplyTo` input parameter.

If you want more control over the actual `javax.jms.Message` that Integration Server sends to the JMS provider, you can create a Java service that calls the `com.wm.app.b2b.server.jms.producer.ProducerFacade` class, which will create a `javax.jms.Message`. See:

- `com.wm.app.b2b.server.jms.producer.ProducerFacade.createBytesMessage(String)`
- `com.wm.app.b2b.server.jms.producer.ProducerFacade.createMapMessage(String)`
- `com.wm.app.b2b.server.jms.producer.ProducerFacade.createObjectMessage(String)`
- `com.wm.app.b2b.server.jms.producer.ProducerFacade.createStreamMessage(String)`
- `com.wm.app.b2b.server.jms.producer.ProducerFacade.createTextMessage(String)`

The Java service calling this API must return an Object of type `javax.jms.Message`, which can then be mapped to the `JMSMessage/body/message` input parameter of the `pub.jms:sendAndWait` service.

When creating the `javax.jms.Message` with the `com.wm.app.b2b.server.jms.producer.ProducerFacade`, you can use the `javax.jms.Message` setter methods to set the values of the message headers and properties directly. You can also set the value of message headers and properties using the input parameters of the `pub.jms:sendAndWait` service that you use to send the message. If you set the message headers and properties both ways, the values provided to the `pub.jms:sendAndWait` service take precedence.

Software AG recommends that you use a `pub.jms:sendAndWait` service to create and send the JMS message. This method may provide better performance on average. However, if you want to send a `StreamMessage` or a `MapMessage`, you need to use the appropriate `com.wm.app.b2b.server.jms.producer.ProducerFacade` API.

To send a `StreamMessage`, create a Java service that calls `com.wm.app.b2b.server.jms.producer.ProducerFacade.createStreamMessage(String)`. The Java service calling this API must return an Object of type `javax.jms.Message`. Map the `javax.jms.Message` object to the `JMSMessage/body/message` input parameter of the `pub.jms:sendAndWait` service.

To send a `MapMessage`, create a Java service that calls `com.wm.app.b2b.server.jms.producer.ProducerFacade.createMapMessage(String)`. The Java

---

service calling this API must return an Object of type `javax.jms.Message`. Map the `javax.jms.Message` object to the `JMSMessage/body/message` input parameter of the `pub.jms:sendAndWait` service.

If you use the input parameter `JMS_WMClusterNodes` to override the policy applied to the cluster connection factory, make sure to code the invoking service to handle any exception that the Broker Server throws when policy requirements are not or cannot be met. For more information about policy override scenarios that might result in an exception from Broker Server, see *Using webMethods Integration Server to Build a Client for JMS*.

When using Universal Messaging as the JMS provider, the JMS client can use synchronous or asynchronous publishing. To ensure delivery of a persistent JMS message (`deliveryMode` is set to `PERSISTENT`), Integration Server always uses synchronous publishing to send a persistent JMS message to Universal Messaging.

Message priority is not supported when Universal Messaging is the JMS provider. Any value specified in the `priority` field will be ignored.

You can use the `pub.jms:send` service to specify a destination for response messages when you do not need to wait for the response. The act of waiting for a response message comes with extra overhead for Integration Server which is unnecessary if you merely want to specify a `JMSReplyTo` destination but do not want the sending service to wait for a reply. For more information, see the `JMSMessage/header/replyTo` input parameter description and Usage Notes in [pub.jms:JMSMessage](#).

#### See Also

[pub.jms:reply](#)

[pub.jms:waitForReply](#)

---

## pub.jms:sendBatch

`WmPublic`. Sends a group of JMS messages to the same destination on the webMethods Broker.

#### Input Parameters

---

*connectionAliasName*

**String** Name of the JMS connection alias that you want to use to send the messages.

The JMS connection alias indicates how Integration Server connects to the webMethods Broker. A JMS connection alias can specify that Integration Server use a JNDI provider to look up administered objects (connection factories and destinations) and then use the connection factory to create a connection. Alternatively, a JMS connection alias can specify that Integration

Server uses the native webMethods API to create the connection directly on the webMethods Broker.

**Note:** *connectionAliasName* must specify a JMS connection alias that uses the webMethods Broker as the JMS provider.

*destinationName*

**String** Name or lookup name of the Destination to which you want to send the messages. Specify the lookup name of the Destination object when the JMS connection alias uses JNDI to retrieve administered objects. Specify the provider-specific name of the Destination when the JMS connection alias uses the native webMethods API to connect directly to the webMethods Broker.

**Note:** When using the `pub.jms:batchSend` service, Integration Server sends all messages to the same destination.

*destinationType*

**String** Optional. Type of destination to which you want to send the message. Specify one of the following:

- `QUEUE` to send the message to a particular queue. This is the default.
- `TOPIC` to send the message to a topic.

**Note:** You need to specify *destinationType* only if you specified a *connectionAliasName* that uses the native webMethods API.

*deliveryMode*

**String** Optional. Specifies the message delivery mode for the messages. Specify one of the following:

- `PERSISTENT` Default. Provide once-and-only-once delivery for the messages. The messages will not be lost if a webMethods Broker failure occurs.
- `NON_PERSISTENT` Provide at-most-once delivery for the messages. The messages have no guarantee of being saved if a webMethods Broker failure occurs.

**Note:** The specified delivery mode applies to all of the messages sent by the service.

*priority*

**java.lang.Integer** Optional. Specifies the message priority for all of the messages. The JMS standard defines

priority levels from 0 to 9, with 0 as the lowest priority and 9 as the highest.

The default is 4.

**Note:** The specified priority applies to all of the messages sent by the service.

*timeToLive*

**java.lang.Long** Optional. Length of time, in milliseconds, that the webMethods Broker retains each message. The default is 0, meaning that the messages do not expire.

**Note:** The specified time to live applies to all of the messages sent by the service.

*JMSMessages*

**Document List** A document list representing the JMS messages to send to the destination. Specify the following for each JMS message that you want to send:

<u>Key</u>	<u>Description</u>				
<i>header</i>	<b>Document</b> Optional. A document containing the header of the JMS message.				
	<table> <thead> <tr> <th><u>Key</u></th> <th><u>Description</u></th> </tr> </thead> <tbody> <tr> <td><i>JMSType</i></td> <td><b>String</b> Optional. Message type identifier for the message.</td> </tr> </tbody> </table>	<u>Key</u>	<u>Description</u>	<i>JMSType</i>	<b>String</b> Optional. Message type identifier for the message.
<u>Key</u>	<u>Description</u>				
<i>JMSType</i>	<b>String</b> Optional. Message type identifier for the message.				
<i>properties</i>	<b>Document</b> Optional. A Document containing optional fields added to the message header. Integration Server adds the following properties to JMS messages it sends.				
	<table> <thead> <tr> <th><u>Key</u></th> <th><u>Description</u></th> </tr> </thead> <tbody> <tr> <td><i>activation</i></td> <td><b>String</b> Optional. A unique identifier used to group together messages that will be received by a JMS trigger with a join. A JMS trigger can join together messages with the same <i>activation</i>.</td> </tr> </tbody> </table>	<u>Key</u>	<u>Description</u>	<i>activation</i>	<b>String</b> Optional. A unique identifier used to group together messages that will be received by a JMS trigger with a join. A JMS trigger can join together messages with the same <i>activation</i> .
<u>Key</u>	<u>Description</u>				
<i>activation</i>	<b>String</b> Optional. A unique identifier used to group together messages that will be received by a JMS trigger with a join. A JMS trigger can join together messages with the same <i>activation</i> .				

*uuid* **String** Optional. A universally unique identifier for the message. Integration Server can use the *uuid* for exactly-once processing or for request/reply.

*body* **Document** Optional. A Document containing the JMS message body. Integration Server supports the following formats for the JMS message body:

<u>Key</u>	<u>Description</u>
<i>string</i>	<b>String</b> Optional. Message body in the form of a String.
<i>bytes</i>	<b>primitive type</b> Optional. Message body in the form of a one-dimensional byte array.
<i>object</i>	<b>Object.</b> Optional. Message body in the form of a Serializable Java object.
<i>data</i>	<b>Document</b> Optional. Message body in the form of a document (IData object).

**Note** This message format can only be used when sending a JMS message from one Integration Server to another. When the JMS message is sent, the sending Integration Server encodes the IData into a byte array. When the receiving Integration Server receives the message, it decodes the byte array into IData.

*message*      **Object** Optional. Message body in the form of an actual `javax.jms.Message`.

**Note:** When you send a batch of messages, you can specify a different format for each JMS message body.

*useCSQ*

**java.lang.Boolean** Optional. Flag indicating whether Integration Server places sent messages in the client side queue if the webMethods Broker is not available at the time the messages are sent. Set to:

- `True` to write messages to the client side queue if the webMethods Broker is not available at the time this service executes. When the webMethods Broker becomes available, Integration Server sends messages from the client side queue to the webMethods Broker.

**Note:** If you want to use the client side queue, the JMS connection alias specified for *connectionAliasName* must be configured to have a client side queue. A JMS connection alias has a client side queue if the **Maximum CSQ Size** property for the alias is set to a value other than 0 (zero).

- `False` to throw an `ISRuntimeException` if the webMethods Broker is not available at the time this service executes. This is the default.

## Output Parameters

*JMSMessages*

**Document List** A Document list containing the messages sent to the webMethods Broker. Each document contains the following information:

<u>Key</u>	<u>Description</u>
<i>header</i>	<b>Document</b> Conditional. A Document containing the header fields for the sent message. The webMethods Broker populates these fields after it has successfully received the message from Integration Server.

<u>Key</u>	<u>Description</u>
------------	--------------------

*JMSCorrelationID* **String** Conditional. A unique identifier used to link messages together.

*JMSDeliveryMode* **java.lang.Integer** Delivery mode used to send the message.

`PERSISTENT` indicates that the JMS provider provides once-and-only-once delivery for the message. The message will not be lost if a JMS provider failure occurs.

`NON_PERSISTENT` indicates that the JMS provider provides at-most-once delivery for the message. The message has no guarantee of being saved if a JMS provider failure occurs.

**Note** When sending a message, this value is obtained from the *JMSMessage/header/deliveryMode* input parameter.

*JMSDestination* **Object** Conditional. Destination (queue or topic) to which the message was sent.

*JMSExpiration* **java.lang.Long** Conditional. Time at which this message expires. If the message producer did not specify a time-to-live, the *JMSExpiration* value is zero, indicating the message does not expire.

**Note** When sending a message, this value is obtained from the *JMSMessage/header/timeToLive* input parameter.

*JMSMessageID* **String** Conditional. Unique identifier assigned to this message by the JMS provider.

*JMSPriority* **java.lang.Integer** Conditional. Defines the message priority. The JMS standard defines priority levels from 0 to 9, with 0 as the lowest priority and 9 as the highest.

**Note** When sending a message, this value is obtained from the *JMSMessage/header/priority* input parameter.

*JMSReplyTo* **Object** Conditional. Specifies the destination to which a response to this message should be sent.

*JMSTimestamp* **java.lang.Long** Time at which the message was given to the JMS provider.

*JMSType* **String** Conditional. Message type identifier specified by the client when sending the message.

*properties* **Document** Conditional. A Document containing optional fields added to the message header. Integration Server adds the following properties to JMS messages it sends.

<b>Key</b>	<b>Description</b>
<i>activation</i>	<b>String</b> Conditional. A unique identifier assigned by the sender. A JMS trigger can join together messages with the same <i>activation</i> .
<i>uuid</i>	<b>String</b> Conditional. A universally unique identifier for the message assigned by the sender. Integration Server can use the <i>uuid</i> for exactly-once processing or for request/reply.

*body* **Document** Conditional. A Document containing the JMS message body. Integration Server supports the following formats for the JMS message body:

<b>Key</b>	<b>Description</b>
<i>string</i>	<b>String</b> Conditional. Message body in the form of a String.
<i>bytes</i>	<b>primitive type</b> Conditional. Message body in the form of a one-dimensional byte array.
<i>object</i>	<b>Object.</b> Conditional. Message body in the form of a Serializable Java object.
<i>data</i>	<b>Document</b> Conditional. Message body in the form of a document (IData object).

**Note** This message format can only be used when sending a JMS message from one Integration Server to another. When the JMS message is sent, the sending Integration Server encodes the IData into a byte array. When the receiving Integration Server receives the message, it decodes the byte array into IData.

*message* **Object** Conditional. Message body in the form of an actual `javax.jms.Message`.

### Usage Notes

The `pub.jms:sendBatch` service can be used only with the webMethods Broker. If you set the `connectionAliasName` parameter to a JMS connection alias that uses a different JMS provider, the `pub.jms:sendBatch` service ends with an exception.

The `pub.jms:sendBatch` service creates multiple JMS messages (`javax.jms.Message`) based on input provided to the service or takes existing JMS messages and sends them to the webMethods Broker.

Sending a batch of messages using the `pub.jms:sendBatch` service is an all or nothing operation. If Integration Server or the webMethods Broker determine that one of the messages is not valid during a pre-processing check, none of the messages will be sent. Make sure to code your service to handle this possibility.

The `pub.jms:sendBatch` service can be used to send messages in accordance with a supported cluster policy. A cluster policy, which is applied to a connection factory used by a JMS connection alias, determines the Broker to which the JMS messages are sent. The `pub.jms:sendBatch` service works with the round robin, sticky, random, and weighted round robin cluster policies. The `pub.jms:sendBatch` service does not work with the multisend guaranteed or multisend best effort cluster policies.

The `pub.jms:sendBatch` service cannot be used to override the cluster policy assigned to the connection factory used by the JMS connection alias.

When Integration Server executes a `pub.jms:sendBatch` service that uses a transacted JMS connection alias, Integration Server sends the messages as part of a transaction. If a transaction has not yet been started, the transaction manager starts a transaction context for an implicit transaction. A JMS connection alias is considered to be transacted when it has a transaction type of XA TRANSACTION or LOCAL TRANSACTION.

You can add properties to a JMS message when building a flow service that invokes this service. To add a new property, use the Pipeline to add a new variable to *JMSMessages/properties* document.

If the JMS connection alias specified for *connectionAliasName* uses the native webMethods API, you need to specify *destinationName* and *destinationType* to indicate where the webMethods Broker should send the message.

Integration Server creates the output parameter *JMSMessages* because some of the *header* fields in a JMS message are populated by the JMS provider after the message is sent. For example, the *header* field *JMSMessageID* is not in the JMS message sent by Integration Server, but *JMSMessageID* is in the *header* after the JMS provider receives the message.

Each JMS connection alias can be configured to have its own client side queue. A JMS connection alias has a client side queue if the **Maximum CSQ Size** property for the alias is set to a value other than 0 (zero). If you want to use the client side queue with the `pub.jms:sendBatch` service, the JMS connection alias specified for *connectionAliasName* must be configured to have a client side queue. If the JMS connection alias is configured to use a client side queue and *useCSQ* is set to true, Integration Server places messages in the client side queue if the JMS provider is not available at the time the `pub.jms:sendBatch` service executes. When the JMS provider becomes available, Integration Server sends messages from the client side queue to the JMS provider.

If the client side queue is not used (*useCSQ* is set to false or the JMS connection alias is not configured to have a client side queue), Integration Server throws an `ISRuntimeException` if the JMS provider is not available when this service executes. Make sure to code your service to handle this situation.

When sending a message as part of a transaction, the client side queue cannot be used. The *useCSQ* field should be set to false. If *useCSQ* is set to true, Integration Server throws a `JMSSubsystemException` when the `pub.jms:send` service executes.

A JMS message is sent as part of a transaction if the JMS connection alias specified in *connectionAliasName* uses a transaction type of LOCAL\_TRANSACTION or XA\_TRANSACTION.

The JMS provider populates the header fields in the *JMSMessages* output parameter after it successfully receives the sent message from Integration Server. If the JMS provider is not available at the time `pub.jms:sendBatch` executes and *useCSQ* is set to true, the *header* fields in the output *JMSMessages* will not be populated. Instead these fields will be blank or be set to 0 (zero).

If you want more control over the actual `javax.jms.Message` that Integration Server sends to the JMS provider, you can create a Java service that calls the `com.wm.app.b2b.server.jms.producer.ProducerFacade` class, which will create a `javax.jms.Message`. See:

- `com.wm.app.b2b.server.jms.producer.ProducerFacade.createBytesMessage(String)`
- `com.wm.app.b2b.server.jms.producer.ProducerFacade.createMapMessage(String)`
- `com.wm.app.b2b.server.jms.producer.ProducerFacade.createObjectMessage(String)`
- `com.wm.app.b2b.server.jms.producer.ProducerFacade.createStreamMessage(String)`
- `com.wm.app.b2b.server.jms.producer.ProducerFacade.createTextMessage(String)`

The Java service calling this API must return an Object of type `javax.jms.Message`, which can then be mapped to the *JMSMessage/body/message* input parameter of the `pub.jms:send` service.

When creating the `javax.jms.Message` with the `com.wm.app.b2b.server.jms.producer.ProducerFacade`, you can use the `javax.jms.Message` setter methods to set the values of the message headers and properties directly. You can also set the value of message headers and properties using the input parameters of the `pub.jms:send` service that you use to send the message. If you set the message headers and properties both ways, the values provided to the `pub.jms:send` service take precedence.

Software AG recommends that you use a `pub.jms:sendBatch` service to create and send the JMS message. This may provide better performance on average. However, if you want to send a `StreamMessage` or a `MapMessage`, you need to use the appropriate `com.wm.app.b2b.server.jms.producer.ProducerFacade` API.

To send a `StreamMessage`, create a Java service that calls `com.wm.app.b2b.server.jms.producer.ProducerFacade.createStreamMessage(String)`. The Java service calling this API must return an Object of type `javax.jms.Message`. Map the `javax.jms.Message` object to the *JMSMessage/body/message* input parameter of the `pub.jms:send` service.

To send a `MapMessage`, create a Java service that calls `com.wm.app.b2b.server.jms.producer.ProducerFacade.createMapMessage(String)`. The Java service calling this API must return an Object of type `javax.jms.Message`. Map the `javax.jms.Message` object to the *JMSMessage/body/message* input parameter of the `pub.jms:send` service.

When using Universal Messaging as the JMS provider, the JMS client can use synchronous or asynchronous publishing. To ensure delivery of a persistent JMS message (*deliveryMode* is set to PERSISTENT), Integration Server always uses synchronous publishing to send a persistent JMS message to Universal Messaging.

Message priority is not supported when Universal Messaging is the JMS provider. Any value specified in the *priority* field will be ignored.

---

## pub.jms:triggerSpec

WmPublic. Specification for the input signature of a JMS trigger that processes one message at a time.

### Input Parameters

---

*JMSMessage*      **Document** A document reference (IData) to the [pub.jms:JMSMessage](#) document type, which defines the structure of a JMS message.

### Output Parameters

---

None.

### Usage Notes

If you want to use a JMS trigger to retrieve and process multiple messages in one batch, use [pub.jms:batchTriggerSpec](#) to declare the inputs and outputs of the JMS trigger service.

### See Also

[pub.jms:batchTriggerSpec](#)  
[pub.jms:JMSMessage](#)

---

## pub.jms:waitForReply

WmPublic. Retrieves the reply message for an asynchronous request.

### Input Parameters

---

*correlationID*      **String** Unique identifier used to associate the reply message with the initial request.

*timeout*      **java.lang.Long** Optional. Time to wait (in milliseconds) for the reply to arrive.

If *timeout* is greater than 0 (zero) and a reply is not available at the time the `pub.jms:waitForReply` service executes, the service continues to wait for the document until the time specified in the *timeout* parameter elapses. If the service does not receive a reply by the time the timeout interval elapses, the `pub.jms:waitForReply` service returns a null document.

If *timeout* is set to 0 (zero), the `pub.jms:waitForReply` service waits indefinitely for a response. Software AG does not recommend setting *timeout* to 0 (zero).

If *timeout* is not set, the `pub.jms:waitForReply` service does not wait for a reply. The `pub.jms:waitForReply` service always returns a null document. The service returns a null document even if the reply queue contains a response for the request.

## Output Parameters

### *JMSReplyMessage*

**Document** Conditional. Document containing the JMS message received as a reply.

If the `pub.jms:waitForReply` service does not receive a a reply before the specified *timeout* value elapses of the *timeout* value was not specified, the *JMSReplyMessage* is null.

<u>Key</u>	<u>Description</u>						
<i>header</i>	<b>Document</b> Conditional. A Document containing the header fields for the reply message.						
	<table border="1"> <thead> <tr> <th><u>Key</u></th> <th><u>Description</u></th> </tr> </thead> <tbody> <tr> <td><i>JMSCorrelationID</i></td> <td><b>String</b> Conditional. A unique identifier used to link the reply message with the initial request message.</td> </tr> <tr> <td><i>JMSDeliveryMode</i></td> <td><b>java.lang.Integer</b> Conditional. Delivery mode used to send the message.  PERSISTENT indicates that the JMS provider provides once-and-only-once delivery for the message. The message will not be lost if a JMS provider failure occurs.</td> </tr> </tbody> </table>	<u>Key</u>	<u>Description</u>	<i>JMSCorrelationID</i>	<b>String</b> Conditional. A unique identifier used to link the reply message with the initial request message.	<i>JMSDeliveryMode</i>	<b>java.lang.Integer</b> Conditional. Delivery mode used to send the message.  PERSISTENT indicates that the JMS provider provides once-and-only-once delivery for the message. The message will not be lost if a JMS provider failure occurs.
<u>Key</u>	<u>Description</u>						
<i>JMSCorrelationID</i>	<b>String</b> Conditional. A unique identifier used to link the reply message with the initial request message.						
<i>JMSDeliveryMode</i>	<b>java.lang.Integer</b> Conditional. Delivery mode used to send the message.  PERSISTENT indicates that the JMS provider provides once-and-only-once delivery for the message. The message will not be lost if a JMS provider failure occurs.						

---

	<p><code>NON_PERSISTENT</code> indicates that the JMS provider provides at-most-once delivery for the message. The message has no guarantee of being saved if a JMS provider failure occurs.</p>
<i>JMSDestination</i>	<p><b>Object</b> Conditional. Destination (queue or topic) to which the message was sent.</p>
<i>JMSExpiration</i>	<p><b>java.lang.Long</b> Conditional. Time at which this message expires. If the message producer did not specify a time-to-live, the <i>JMSExpiration</i> value is zero, indicating the message does not expire.</p>
<i>JMSMessageID</i>	<p><b>String</b> Conditional. Unique identifier assigned to this message by the JMS provider.</p>
<i>JMSPriority</i>	<p><b>java.lang.Integer</b> Conditional. Defines the message priority. The JMS standard defines priority levels from 0 to 9, with 0 as the lowest priority and 9 as the highest.</p>
<i>JMSRedelivered</i>	<p><b>java.lang.Boolean</b> Conditional. Flag indicating the JMS provider delivered this message to the JMS client previously.</p> <p><code>True</code> indicates the message may have been delivered in the past.</p> <p><code>False</code> indicates the JMS provider has not delivered this message previously.</p>

<i>JMSReplyTo</i>	<b>Object</b> Conditional. Specifies the destination to which a response to this message should be sent.
<i>JMSTimestamp</i>	<b>java.lang.Long</b> Conditional. Time at which the message was given to the JMS provider.
<i>JMSType</i>	<b>String</b> Conditional. Message type identifier specified by the client when sending the message.
<i>properties</i>	<b>Document</b> Conditional. A document containing optional fields added to the message header. Integration Server may add the following properties to JMS messages it sends or receives.

<u>Key</u>	<u>Description</u>
<i>JMSXDeliveryCount</i>	<b>java.lang.Integer</b> Conditional. Specifies the number of times the JMS provider delivered the message to the requesting client. Most JMS providers set this value.
<i>JMS_WMClusterNodes</i>	<p><b>String</b> Conditional. Name of the Broker or Brokers in the Broker cluster that received the JMS message.</p> <p>The Broker Server acting as the JMS provider populates the <i>JMS_WMClusterNodes</i> parameter after it distributes the JMS message to the Broker or Brokers in the Broker cluster.</p> <p>The <i>JMS_WMClusterNodes</i> value will be null when:</p> <ul style="list-style-type: none"> <li>■ The JMS provider is not the Broker Server.</li> </ul>

- The JMS connection alias used to send the JMS message does not use a cluster connection factory to obtain the connection to the Broker Server.
- The cluster connection factory does not permit a policy to be overridden.

*activation* **String** Conditional. A unique identifier assigned by the sending service. A JMS trigger uses the *activation* to determine whether a message is part of a join.

*uuid* **String** Conditional. A universally unique identifier for the message assigned by the sender. Integration Server can use the *uuid* for exactly-once processing or for request/reply.

*body* **Document** Conditional. A Document containing the JMS message body. Integration Server supports the following formats for the JMS message body:

Key	Description
<i>string</i>	<b>String</b> Conditional. Message body in the form of a String.
<i>bytes</i>	<b>primitive type</b> Conditional Message body in the form of a one-dimensional byte array.
<i>object</i>	<b>Object.</b> Conditional. Message body in the form of a Serializable Java object.

*data*

**Document** Optional. Message body in the form of a document (IData object).

**Note** This message format can only be used when sending a JMS message from one Integration Server to another. When the JMS message is sent, the sending Integration Server encodes the IData into a byte array. When the receiving Integration Server receives the message, it decodes the byte array into IData.

*message*

**Object** Optional. Message body in the form of an actual `javax.jms.Message`.

## Usage Notes

Integration Server uses the value of the *uuid* or *JMSMessageID* fields in the requesting JMS message to correlate the response to the request. If you specify the *uuid* when sending the request, the replying Integration Server will use the *uuid* as the *JMSCorrelationID* of the reply message (*JMSReplyMessage*). If you do not specify a *uuid*, the replying Integration Server uses the *JMSMessageID* set by the JMS provider as the *JMSCorrelationID* of the reply message (*JMSReplyMessage*).

If you set the *uuid* in the JMS message request, you can link the value of the *uuid* field from the *JMSMessage* produced by the `pub.jms:sendAndWait` service to the *correlationID* input field of the `pub.jms:waitForReply` service. If you did not specify a *uuid*, you can link the *JMSMessageID* field from the *JMSMessage* produced by the `pub.jms:sendAndWait` to the *correlationID* input field.

The *timeout* value of the sending service specifies how long Integration Server will keep the request open while waiting for a reply. If a reply is not available at the time Integration Server executes the `pub.jms:waitForReply` service, Integration Server continues to wait for the document until the time specified in the *timeout* parameter elapses. If the service does not receive a reply by the time the *timeout* interval elapses, the service returns a null document.

The `pub.jms:waitForReply` service cannot be used to retrieve response to requests that were routed through the client side queue. To retrieve the response, create a JMS trigger that subscribes to the reply queue.

If the `pub.jms:sendAndWait` service executes and the message is sent directly to the JMS provider (i.e., it is not sent to the client side queue), the `JMSMessage\header\JMSMessageID` contains a unique identifier assigned by the JMS provider. If the `JMSMessageID` field is null after the service executes, the JMS provider was not available at the time the service executed. Integration Server wrote the message to the client side queue.

### See Also

[pub.jms:sendAndWait](#)

---

## pub.jms.wmjms:receiveStream

WmPublic. Receives a large message stream from a queue or topic on the webMethods Broker.

### Input Parameters

---

<i>consumer</i>	<b>Object</b> A message consumer object that the service uses to receive the large message stream. Create the message consumer object using the <a href="#">pub.jms:createConsumer</a> service.
<i>timeout</i>	<b>java.lang.Long</b> Optional. Time to wait (in milliseconds) for the first part of the message stream. If you do not specify a <i>timeout</i> value, the consumer does not wait.

### Output Parameters

---

<i>stream</i>	<b>Object</b> A <code>com.webmethods.jms.impl.WmJMSInputStream</code> received by the consumer.  If the <i>timeout</i> value elapses before a message is received, <i>stream</i> will be null.
---------------	--

### Usage Notes

When using webMethods Broker as the JMS provider, the webMethods message streaming feature allows you to stream large amounts of data or a large file from a message producer to a message consumer.

**Important:** You can only send and receive large messages from Integration Server when working with the webMethods Broker. For more information about how the webMethods message streaming feature works on the webMethods Broker, see the *webMethods Broker Messaging Programmer's Guide*.

Large message streams cannot be sent or received as part of a transaction. If the JMS connection alias used by the consumer has a transaction type of `LOCAL_TRANSACTION` or `XA_TRANSACTION`, Integration Server throws an exception, specifically `com.wm.app.b2b.server.jms.JMSSubsystemException`, when it executes the [pub.jms.wmjms:receiveStream](#) service.

The *consumer* that you use to receive the message determines the destination from which this services receives messages and the JMS connection alias used to receive the messages. You can create a message consumer object using the [pub.jms:createConsumer](#) service.

The *timeout* value specifies how long the message consumer waits for the initial part of the message stream. If a message is not returned when the time out period elapses, the [pub.jms.wmjms:receiveStream](#) returns a null value.

The *read timeout* is the maximum length of time the consumer waits between receiving subsequent pieces of the message stream. After the read timeout elapses, the consumer calls `InputStream.read()` to read the next byte of the stream. If the next byte of the stream is not available, Integration Server throws a `WmReadTimeoutException`. The read timeout only applies after the consumer receives the first part of the message stream. The `watt.server.jms.wmjms.lms.readTimeout` property determines the read timeout value. The default is 30000 milliseconds.

Make sure to code your service to handle a `WmReadTimeoutException`. When an `WmReadTimeoutException` occurs, it suggests that Integration Server did not receive the entire message stream. When this occurs, you need to close the stream, which will acknowledge it to the webMethods Broker. You can close the stream from a Java service by calling `Input.Stream.close`. You can also close the stream using the [pub.io:close](#) service.

If the connection between the Integration Server and webMethods Broker fails during execution of the [pub.jms.wmjms:receiveStream](#) service, Integration Server throws a `WmConnectionException`. When this occurs, Integration Server rolls the message back to the webMethods Broker automatically. The message can be received when the connection to the webMethods Broker is re-established.

You can code your service to implement recoverability logic. This means that the next time the message stream is received, the service re-processes the message stream from the point at which processing stopped. To resume processing from the correct point, the service needs to keep track of the message ID and byte position. For more details, `com.webmethods.jms.impl.WmJMSInputStream`.

After the [pub.jms.wmjms:receiveStream](#) receives and processes the last part of the message stream, you need to close the stream. `InputStream.read()` returns "-1" when the end of the stream is reached. You can close the stream from a Java service by calling `Input.Stream.close`. You can also close the stream using the [pub.io:close](#) service. Closing the stream explicitly acknowledges the message to the provider.

The *consumer* used to receive large message streams from the webMethods Broker can specify an *acknowledgmentMode* of `AUTO_ACKNOWLEDGE` or `CLIENT_ACKNOWLEDGE`. webMethods Broker does not permit the use of the *acknowledgmentMode* is `DUPS_OK_ACKNOWLEDGE` for the webMethods message streaming feature.

You might want to use the scheduler capabilities within Integration Server to schedule a service that receives and then process large messages from webMethods Broker. For more information about scheduling services, see *webMethods Integration Server Administrator's Guide*.

### See Also

[pub.io:close](#)  
[pub.jms:createConsumer](#)  
[pub.jms.wmjms:sendStream](#)  
[pub.jms.wmjms:sendStream](#)

---

## pub.jms.wmjms:sendStream

WmPublic. Sends a large message stream to the webMethods Broker.

### Input Parameters

---

*connectionAliasName*      **String** Name of the JMS connection alias that you want to use to send the message.

*destinationName*      **String** Name or lookup name of the Destination to which you want to send the message. Specify the lookup name of the Destination object when the JMS connection alias uses JNDI to retrieve administered objects. Specify the provider-specific name of the Destination when the JMS connection alias uses the native webMethods API to connect directly to the webMethods Broker.

*destinationType*      **String** Optional. Type of destination to which you want to send the message. Specify one of the following:

- `QUEUE` to send the message to a particular receiver/queue. This is the default.
- `TOPIC` to send the message to a topic.

**Note:** You need to specify a *destinationType* only if you specified a *connectionAliasName* that uses the native webMethods API.

*stream*      **Object** A stream for the message you want to send to the webMethods Broker.

## Output Parameters

---

None.

## Usage Notes

When using the webMethods Broker as the JMS provider, the webMethods message streaming feature allows you to stream large amounts of data or a large file from a message producer to a message consumer. You can only send and receive large messages from Integration Server when working with the webMethods Broker. For more information about how the webMethods message streaming feature works on the webMethods Broker, see the *webMethods Broker Messaging Programmer's Guide*.

Large message streams cannot be sent or received as part of a transaction. If *connectionAliasName* specifies a JMS connection alias with a transaction type of LOCAL\_TRANSACTION or XA\_TRANSACTION, Integration Server throws the exception `com.wm.app.b2b.server.jms.JMSSubsystemException` when it executes the [pub.jms.wmjms:sendStream](#) service.

If the connection between Integration Server and the webMethods Broker fails before the [pub.jms.wmjms:sendStream](#) sends the entire message stream, you need to re-send the entire stream when the connection is re-established.

## See Also

[pub.jms.wmjms:receiveStream](#)



---

# 16 JSON Folder

---

■ Data Type Mapping .....	514
■ Summary of Elements in This Folder .....	515

You use the elements in the JSON folder to convert JSON content into a document (IData object), and a document (IData object) into JSON content.

## Data Type Mapping

The following table shows how JSON data types map to Integration Server data types during data conversion.

JSON	Integration Server
object	Document
string	String
number (integer)	Integer or Long Java wrapper. For more information about converting JSON integers, see the <i>decodeIntegerAsLong</i> input parameters in " <a href="#">pub.json:jsonStreamToDocument</a> " on page 517 and " <a href="#">pub.json:jsonStringToDocument</a> " on page 518.
number (real)	Float or Double Java wrapper. For more information about converting real numbers, see the <i>decodeRealAsDouble</i> input parameter in " <a href="#">pub.json:jsonStreamToDocument</a> " on page 517 and " <a href="#">pub.json:jsonStringToDocument</a> " on page 518.
True/False	Boolean Java wrapper
Array of JSON type	Array of Integration Server type
null	null
All others	String

**Note:** If an object has a `toString()` implementation, Integration Server uses that implementation.

If the object does not provide a `toString()` implementation, Integration Server uses `Object.toString()`. `Object.toString()` returns the class name and hexadecimal representation of the hash code of the object, such as `"javax.naming.InitialContext@3ae6f00b"`.

**Note:** If JSON text begins with an array at the root and the array is unnamed, when parsing the JSON text, Integration Server uses a fixed name of *\$rootArray* for the array value. The *\$rootArray* field appears in the pipeline. When creating a JSON response, if the pipeline contains *\$rootArray* with an array value at its root, Integration Server discards the *\$rootArray* name and transforms the array value into a JSON array.

## Summary of Elements in This Folder

The following elements are available in this folder:

Element	Package and Description
<a href="#">pub.json:documentToJSONString</a>	WmPublic. Converts a document (IData object) to a JSON string.
<a href="#">pub.json:jsonStreamToDocument</a>	WmPublic. Converts content from the JSON content stream to a document (an IData object).
<a href="#">pub.json:jsonStringToDocument</a>	WmPublic. Converts a JSON string to a document (an IData object).

### pub.json:documentToJSONString

WmPublic. Converts a document (IData object) to a JSON string.

#### Input Parameters

<i>document</i>	<b>Document</b> The document (IData object) to be converted to a JSON string.
<i>prettyPrint</i>	<p><b>String</b> Optional. Formats the <i>jsonString</i> output parameter for human readability by adding carriage returns and indentation to the JSON content. Set to:</p> <ul style="list-style-type: none"> <li>■ <code>true</code> to format <i>jsonString</i> output variable for human readability.</li> <li>■ <code>false</code> to leave the <i>jsonString</i> output variable in its unformatted state. The service will not add any additional carriage returns or indentation to the JSON content.</li> </ul>

- `null` to use the `prettyPrint` setting already in effect for the HTTP client making the request, as follows:
  - If the HTTP client request includes `jsonPrettyPrint=true` in the URI, JSON pretty printing is in effect.
  - If the HTTP client request includes `jsonPrettyPrint=false` in the URI, JSON pretty printing is not in effect.
  - If the HTTP client request does not include the `jsonPrettyPrint` parameter, the service uses the value of the `watt.server.json.prettyPrint` configuration parameter. For more information about `watt.server.json.prettyPrint`, see *webMethods Integration Server Administrator's Guide*.

## Output Parameters

---

*jsonString*      **String** JSON string resulting from the conversion of *document*.

### Usage Notes

To turn a document in a pipeline into a JSON response to send over HTTP, the application's service can:

1. Use `pub.json:documentToJSONString` to turn a document (IData object) in the pipeline into a string of JSON content.
2. Call `pub.client:http` to send the JSON string as an HTTP request.
3. Set the Content-Type header field to `application/json`.

The JSON standard requires that field names be enclosed in double quotes. However, you may need the service to produce unquoted field names if the generated JSON text will be processed by an older JavaScript interpreter. The `watt.server.json.quoteFieldNames` server configuration parameter determines whether the `pub.json:documentToJSONString` service encloses all generated JSON field names in double quotes. Set this parameter to `true` to instruct the `pub.json:documentToJSONString` service to enclose field names in quotes in the output JSON text. Set this parameter to `false` to instruct the service to omit the double quotes around field names in the generated JSON text. The default is `true`.

**Note:** Use caution when setting `watt.server.json.quoteFieldNames` to `false` as this causes the `pub.json:documentToJSONString` service to generate non-standard JSON text. This can cause interoperability issues if the JSON text is sent to other organizations.

---

## pub.json:jsonStreamToDocument

WmPublic. Converts content from the JSON content stream to a document (an IData object).

### Input Parameters

---

*jsonStream*

**java.io.InputStream** JSON content in an input stream to convert to a document (an IData object).

*decodeRealAsDouble*

**String** Optional. Converts real numbers from *jsonStream* to either a Float or Double Java wrapper type. Set to:

- `true` to convert real numbers to Double Java wrapper types. This is the default.
- `false` to convert real numbers to Float Java wrapper types.

**Note:** The *decodeRealAsDouble* parameter overrides the value specified by the `watt.server.json.decodeRealAsDouble` server configuration parameter. If no value is supplied for *decodeRealAsDouble*, Integration Server uses the value set in `watt.server.json.decodeRealAsDouble`. For more information about `watt.server.json.decodeRealAsDouble`, see *webMethods Integration Server Administrator's Guide*.

*decodeIntegerAsLong*

**String** Optional. Converts integers from *jsonStream* to either a Long or Integer Java wrapper type. Set to:

- `true` to convert integers to Long Java wrapper types. This is the default.
- `false` to convert integers to Integer Java wrapper types.

**Note:** The *decodeRealAsDouble* parameter overrides the value specified by the `watt.server.json.decodeIntegerAsLong` server configuration parameter. If no value is supplied for *decodeIntegerAsLong*, Integration Server uses the value specified in the `watt.server.json.decodeIntegerAsLong` property. For more information about

watt.server.json.decodeIntegerAsLong, see *webMethods Integration Server Administrator's Guide*.

## Output Parameters

---

*document*                      **Document** Document (IData object) resulting from the conversion of *jsonStream*.

## Usage Notes

If `watt.server.http.jsonFormat` is set to `stream`, Integration Server places a *jsonStream* variable in the pipeline when it receives an HTTP request containing JSON content. You can then use `pub.json.jsonStreamToDocument` to parse *jsonStream* into pipeline variables. If `watt.server.http.jsonFormat` is set to `parsed`, which is the default, the JSON content handler parses the JSON content in the HTTP request automatically. In this case, the `pub.json.jsonStreamToDocument` does not need to be invoked. For more information about `watt.server.http.jsonFormat`, see *webMethods Integration Server Administrator's Guide*.

The JSON standard requires that field names be enclosed in double quotes. However, when parsing legacy JavaScript as JSON text it may be helpful to allow unquoted field names as JavaScript does not require field names to be enclosed in double quotes. The server configuration parameter `watt.server.json.allowUnquotedFieldNames` specifies whether or not unquoted field names are allowed in JSON text passed to the `pub.json.jsonStringToDocument` and `pub.json.jsonStreamToDocument` services. If this parameter is set to `true`, the `pub.json.jsonStringToDocument` and `pub.json.jsonStreamToDocument` services allow unquoted field names in any supplied JSON text. If this parameter is set to `false`, the services throw a `ServiceException` when encountering unquoted field names. The default is `false`.

## pub.json:jsonStringToDocument

---

WmPublic. Converts a JSON string to a document (an IData object).

### Input Parameters

---

*jsonString*                      **String** JSON content in a string to convert to a document (IData object).

*decodeRealAsDouble*            **String** Optional. Converts real numbers from *jsonString* to either a Float or Double Java wrapper type. Set to:

- `true` to convert real numbers to Double Java wrapper types. This is the default.
- `false` to convert real numbers to Float Java wrapper types.

**Note:** The *decodeRealAsDouble* parameter overrides the value specified by the `watt.server.json.decodeRealAsDouble` server configuration parameter. If no value is supplied for *decodeRealAsDouble*, Integration Server uses the value set in `watt.server.json.decodeRealAsDouble`. For more information about `watt.server.json.decodeRealAsDouble`, see *webMethods Integration Server Administrator's Guide*.

*decodeIntegerAsLong*

**String** Optional. Converts integers from *jsonString* to either a Long or Integer Java wrapper type. Set to:

- `true` to convert integers to Long Java wrapper types. This is the default.
- `false` to convert integers to Integer Java wrapper types.

**Note:** The *decodeRealAsDouble* parameter overrides the value specified by the `watt.server.json.decodeIntegerAsLong` server configuration parameter. If no value is supplied for *decodeIntegerAsLong*, Integration Server uses the value specified in the `watt.server.json.decodeIntegerAsLong` property. For more information about `watt.server.json.decodeIntegerAsLong`, see *webMethods Integration Server Administrator's Guide*.

## Output Parameters

*document*

**Document** Document (IData object) resulting from the conversion of *jsonString*.

## Usage Notes

The JSON standard requires that field names be enclosed in double quotes. However, when parsing legacy JavaScript as JSON text it may be helpful to allow unquoted field names as JavaScript does not require field names to be enclosed in double quotes. The server configuration parameter `watt.server.json.allowUnquotedFieldNames` specifies whether or not unquoted field names are allowed in JSON text passed to the `pub.json:jsonStringToDocument` and `pub.json:jsonStreamToDocument` services. If this parameter is set to `true`, the `pub.json:jsonStringToDocument` and `pub.json:jsonStreamToDocument` services allow unquoted field names in any supplied JSON text. If this parameter is set to `false`, the services throw a `ServiceException` when encountering unquoted field names. The default is `false`.



# 17 List Folder

---

■ Summary of Elements in this Folder .....	522
--	-----

You use the elements in the list folder to retrieve, replace, or add elements in an Object List, Document List, String List, or Vector. You also use list services to convert String Lists to Document Lists or a Vector to an Array.

## Summary of Elements in this Folder

The following elements are available in this folder:

Element	Package and Description
<a href="#">pub.list:addItemToVector</a>	WmPublic. Adds an item or a list of items to a java.util.Vector object.
<a href="#">pub.list:appendToDocumentList</a>	WmPublic. Adds documents to a document list.
<a href="#">pub.list:appendToStringList</a>	WmPublic. Adds Strings to a String list.
<a href="#">pub.list:sizeOfList</a>	WmPublic. Returns the number of elements in a list.
<a href="#">pub.list:stringListToDocumentList</a>	WmPublic. Converts a String list to a document list.
<a href="#">pub.list:vectorToArray</a>	WmPublic. Converts a java.util.Vector object to an array.

## pub.list:addItemToVector

WmPublic. Adds an item or a list of items to a java.util.Vector object.

### Input Parameters

*vector* **java.util.Vector** Optional. The vector object to which you want to add an item or list of items. If no value is specified, the service creates a new java.util.Vector object to which the item(s) will be added.

*item* **Object** Optional. Item to be added to the vector object.

**Note:** You can use either *item* or *itemList* to specify the input object. If both *item* and *itemList* input parameters are

specified, the item as well as the list of items will be added to the vector object.

<i>itemList</i>	<b>Object[]</b> Optional. List of items to be added to the vector object.
<i>addNulls</i>	<p><b>String</b> Optional. Specifies whether a null item can be added to the vector object. Set to:</p> <ul style="list-style-type: none"> <li>■ <code>false</code> to prevent null values from being added to the vector object. This is the default.</li> <li>■ <code>true</code> to allow null values to be added to the vector object.</li> </ul>

### Output Parameters

<i>vector</i>	<b>java.util.Vector</b> Updated vector object with the list of items added or an empty vector in case no items are added.
---------------	---

### Usage Notes

Either of the optional input parameters, *item* or *itemList*, is required.

## pub.list:appendToDocumentList

WmPublic. Adds documents to a document list.

### Input Parameters

<i>toList</i>	<b>Document List</b> Optional. List to which you want to append documents. If you do not specify <i>toList</i> , the service creates a new list.
<i>fromList</i>	<b>Document List</b> Optional. Documents you want to append to the end of <i>toList</i> .
<i>fromItem</i>	<b>Document</b> Optional. Document you want to append to the end of <i>toList</i> . If you specify both <i>fromList</i> and <i>fromItem</i> , the service adds the document specified in <i>fromItem</i> after the documents in <i>fromList</i> .

---

## Output Parameters

*toList*                      **Document List** The *toList* document list with the documents in *fromList* and *fromItem* appended to it.

### Usage Notes

The documents contained in *fromList* and *fromItem* are not actually appended as entries to *toList*. Instead, references to the documents in *fromList* and *fromItem* are appended as entries to *toList*. Consequently, any changes made to the documents in *fromList* and *fromItem* also affect the resulting *toList*.

---

## pub.list:appendToStringList

WmPublic. Adds Strings to a String list.

### Input Parameters

*toList*                      **String List** Optional. List to which you want to append Strings. If the value of *toList* is null, a null pointer exception error is thrown. If you do not specify *toList*, the service creates a new list.

*fromList*                    **String List** Optional. List of Strings to add to *toList*. Strings are added after the entries of *toList*.

*fromItem*                   **String** Optional. String you want to append to the end of *toList*. If you specify both *fromList* and *fromItem*, the service adds the String specified in *fromItem* after the Strings specified in *fromList*.

---

### Output Parameters

*toList*                      **String List** The *toList* String list with the Strings from *fromList* and *fromItem* appended to it.

### Usage Notes

The Strings contained in *fromList* and *fromItem* are not actually appended as entries to *toList*. Instead, references to the Strings in *fromList* and *fromItem* are appended as entries to *toList*. Consequently, any changes made to the Strings in *fromList* and *fromItem* also affect the resulting *toList*.

---

## pub.list:sizeOfList

WmPublic. Returns the number of elements in a list.

### Input Parameters

---

*fromList*                    **Document List, String List, or Object List** Optional. List whose size you want to discover. If *fromList* is not specified, the service returns a *size* of 0.

### Output Parameters

---

*size*                         **String** Number of entries in *fromList* .

*fromList*                    **Document List, String List, or Object List** Original list.

### Usage Notes

For example, if *fromList* consists of:

```
fromList [0] = "a"
```

```
fromList [1] = "b"
```

```
fromList [2] = "c"
```

The result would be:

```
size ="3"
```

---

## pub.list:stringListToDocumentList

WmPublic. Converts a String list to a document list.

### Input Parameters

---

*fromList*                    **String List** Optional. List of Strings (a `String[]`) that you want to convert to a list of documents (an `IData[]`). If *fromList* is not specified, the service returns a zero length array for *toList* .

*key*                         **String** Optional. Key name to use in the generated document list.

---

## Output Parameters

---

*toList*                      **Document List** Resulting document list.

### Usage Notes

Creates a document list containing one document for each element in the *fromList*. Each document will contain a single String element named *key*.

For example, if *fromList* consists of:

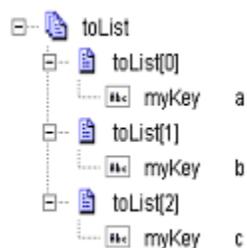
*fromList*[0] = "a"

*fromList*[1] = "b"

*fromList*[2] = "c"

*key* = "myKey"

The result would be:




---

## pub.list:vectorToArray

---

WmPublic. Converts a java.util.Vector object to an array.

### Input Parameters

---

*vector*                      **java.util.Vector** The object to be converted to an array.

*stronglyType*              **String** Optional. If this option is specified, the service expects all items in the vector to have the same Java type as the first non-null item in the vector. If the service detects an item of a different type, an error is thrown.

Set to:

- `false` to convert the vector to an object array. This is the default.

- `true` to convert the vector to a strongly typed array holding the same type of objects.

### Output Parameters

---

*array*

**Object[]** Converted object array.



# 18 Math Folder

---

■ Summary of Elements in this Folder .....	530
--	-----

You use the elements in the math folder to perform mathematical operations on string-based numeric values.

**Note:** Services that operate on integer values use Java's long data type (64-bit, two's complement). Services that operate on float values use Java's double data type (64-bit IEEE 754). If extremely precise calculations are critical to your application, you should write your own Java services to perform math functions.

## Summary of Elements in this Folder

The following elements are available in this folder:

Element	Package and Description
<a href="#">pub.math:absoluteValue</a>	WmPublic. Returns the absolute value of the input number.
<a href="#">pub.math:addFloatList</a>	WmPublic. Adds a list of floating point numbers (represented in a string list) and returns the sum.
<a href="#">pub.math:addFloats</a>	WmPublic. Adds one floating point number (represented as a String) to another and returns the sum.
<a href="#">pub.math:addIntList</a>	WmPublic. Adds a list of integers (represented in a String list) and returns the sum.
<a href="#">pub.math:addInts</a>	WmPublic. Adds one integer (represented as a String) to another and returns the sum.
<a href="#">pub.math:addObjects</a>	WmPublic. Adds one java.lang.Number object to another and returns the sum.
<a href="#">pub.math:divideFloats</a>	WmPublic. Divides one floating point number (represented as a String) by another ( <i>num1/num2</i> ) and returns the quotient.
<a href="#">pub.math:divideInts</a>	WmPublic. Divides one integer (represented as a String) by another ( <i>num1/num2</i> ) and returns the quotient.

Element	Package and Description
<a href="#">pub.math:divideObjects</a>	WmPublic. Divides one java.lang.Number object by another ( $num1/num2$ ) and returns the quotient.
<a href="#">pub.math:max</a>	WmPublic. Returns the largest number from a list of numbers.
<a href="#">pub.math:min</a>	WmPublic. Returns smallest number from a list of numbers.
<a href="#">pub.math:multiplyFloatList</a>	WmPublic. Multiplies a list of floating point numbers (represented in a String list) and returns the product.
<a href="#">pub.math:multiplyFloats</a>	WmPublic. Multiplies one floating point number (represented as String) by another and returns the product.
<a href="#">pub.math:multiplyIntList</a>	WmPublic. Multiplies a list of integers (represented in a String list) and returns the product.
<a href="#">pub.math:multiplyInts</a>	WmPublic. Multiplies one integer (represented as a String) by another and returns the product.
<a href="#">pub.math:multiplyObjects</a>	WmPublic. Multiplies one java.lang.Number object by another and returns the product.
<a href="#">pub.math:randomDouble</a>	WmPublic. Returns the next pseudorandom, uniformly distributed double between 0.0 and 1.0.
<a href="#">pub.math:roundNumber</a>	WmPublic. Returns a rounded number.
<a href="#">pub.math:subtractFloats</a>	WmPublic. Subtracts one floating point number (represented as a String) from another and returns the difference.
<a href="#">pub.math:subtractInts</a>	WmPublic. Subtracts one integer (represented as a String) from another and returns the difference.
<a href="#">pub.math:subtractObjects</a>	WmPublic. Subtracts one java.lang.Number object from another and returns the difference.
<a href="#">pub.math:toNumber</a>	WmPublic. Converts a string to numeric data type.

---

## pub.math:absoluteValue

WmPublic. Returns the absolute value of the input number.

### Input Parameters

---

*num*                    **String** Number whose absolute value is to be returned.

### Output Parameters

---

*positiveNumber*    **String** Absolute value of the input number.

---

## pub.math:addFloatList

WmPublic. Adds a list of floating point numbers (represented in a string list) and returns the sum.

### Input Parameters

---

*numList*    **String List** Numbers (floating point numbers represented in a string list) to add.

### Output Parameters

---

*value*            **String** Sum of the numbers in *numList* . If a sum cannot be produced, *value* contains one of the following:

<b>Value</b>	<b>Description</b>
<i>Infinity</i>	The computation produces a positive value that overflows the representable range of a float type.
<i>-Infinity</i>	The computation produces a negative value that overflows the representable range of a float type.
<i>0.0</i>	The computation produces a value that underflows the representable range of a float type (for example, adding a number to infinity).

NaN      The computation produces a value that cannot be represented as a number (for example, any operation that uses NaN as input, such as  $10.0 + \text{NaN} = \text{NaN}$ ).

### Usage Notes

Make sure the strings that are passed to the service in *numList* are in a locale-neutral format (that is, using the pattern `-####.##`). Passing locally formatted strings may result in unexpected results. For example, calling `pub.math.addFloats` in a German locale with the arguments `1,23` and `2,34` will result in the value `357`, not `3.57` or `3,57`.

---

## pub.math:addFloats

WmPublic. Adds one floating point number (represented as a String) to another and returns the sum.

### Input Parameters

---

*num1*      **String** Number to add.

*num2*      **String** Number to add.

*precision*      **String** Optional. Number of decimal places to which the sum will be rounded. The default value is null.

The *precision* parameter, if specified, will override the behavior set by the `watt.server.math.floatOperation.mode` property. For information about the `watt.server.math.floatOperation.mode` property, see the *webMethods Integration Server Administrator's Guide*.

### Output Parameters

---

*value*      **String** Sum of the numbers in *num1* and *num2*. If a sum cannot be produced, *value* contains one of the following:

Value	Description
Infinity	The computation produces a positive value that overflows the representable range of a float type.
-Infinity	The computation produces a negative value that overflows the representable range of a float type.

0.0	The computation produces a value that underflows the representable range of a float type (for example, adding a number to infinity).
NaN	The computation produces a value that cannot be represented as a number (for example, any operation that uses NaN as input, such as $10.0 + \text{NaN} = \text{NaN}$ ).

### Usage Notes

Make sure the strings that are passed to the service in *num1* and *num2* are in a locale-neutral format (that is, using the pattern `-####.##`). Passing locally formatted strings may result in unexpected results. For example, calling `pub.math:addFloats` in a German locale with the arguments `1,23` and `2,34` will result in the value `357`, not `3.57` or `3,57`.

Use the `watt.server.math.floatOperation.mode` property to specify whether the `pub.math:addFloats` service return the exact result of an operation involving two floating point numbers, the result as calculated by the JVM, or the result based on a fixed number of decimal places. For information about the `watt.server.math.floatOperation.mode` property, see the *webMethods Integration Server Administrator's Guide*.

---

## pub.math:addIntList

WmPublic. Adds a list of integers (represented in a String list) and returns the sum.

### Input Parameters

*numList*      **String List** Numbers (integers represented as Strings) to add.

### Output Parameters

*value*      **String** Sum of the numbers in *numList*.

### Usage Notes

Make sure the strings that are passed to the service in *numList* are in a locale-neutral format (that is, using the pattern `-####.##`). Passing locally formatted strings may result in unexpected results. For example, calling `pub.math:addFloats` in a German locale with the arguments `1,23` and `2,34` will result in the value `357`, not `3.57` or `3,57`.

---

## pub.math:addInts

WmPublic. Adds one integer (represented as a String) to another and returns the sum.

### Input Parameters

---

*num1*      **String** Number (integer represented as a String) to add.

*num2*      **String** Number (integer represented as a String) to add.

### Output Parameters

---

*value*      **String** Sum of *num1* and *num2*.

### Usage Notes

Make sure the result of your calculation is less than 64 bits in width (the maximum width for the long data type). If the result exceeds this limit, it will generate a data overflow.

Make sure the strings that are passed to the service in *num1* and *num2* are in a locale-neutral format (that is, using the pattern -####.##). Passing locally formatted strings may result in unexpected results. For example, calling [pub.math:addFloats](#) in a German locale with the arguments 1,23 and 2,34 will result in the value 357, not 3.57 or 3,57.

---

## pub.math:addObjects

WmPublic. Adds one java.lang.Number object to another and returns the sum.

### Input Parameters

---

*num1*      **java.lang.Number** Number to add. See the Usage Notes for supported sub-classes.

*num2*      **java.lang.Number** Number to add. See the Usage Notes for supported sub-classes.

### Output Parameters

---

*value*      **java.lang.Number** Sum of the numeric values of *num1* and *num2*.

### Usage Notes

This service accepts the following sub-classes of `java.lang.Number`: `java.lang.Byte`, `java.lang.Double`, `java.lang.Float`, `java.lang.Integer`, `java.lang.Long`, `java.lang.Short`.

This service applies the following rules for binary numeric promotion to the operands in order:

- If either operand is of type `Double`, the other is converted to `Double`.
- Otherwise, if either operand is of type `Float`, the other is converted to `Float`.
- Otherwise, if either operand is of type `Long`, the other is converted to `Long`.
- Otherwise, both operands are converted to type `Integer`.

These promotion rules mirror the Java rules for numeric promotion of numeric types.

---

## pub.math:divideFloats

`WmPublic`. Divides one floating point number (represented as a `String`) by another ( $num1/num2$ ) and returns the quotient.

### Input Parameters

---

*num1*     **String** Number (floating point number represented as a `String`) that is the dividend.

*num2*     **String** Number (floating point number represented as a `String`) that is the divisor.

*precision*     **String** Optional. Number of decimal places to which the quotient will be rounded. The default value is null.

The *precision* parameter, if specified, will override the behavior set by the `watt.server.math.floatOperation.mode` property. For information about the `watt.server.math.floatOperation.mode` property, see the *webMethods Integration Server Administrator's Guide*.

### Output Parameters

---

*value*     **String** The quotient of  $num1 / num2$ . If a quotient cannot be produced, *value* contains one of the following:

Value	Description
-------	-------------

Infinity	The computation produces a positive value that overflows the representable range of a float type.
- Infinity	The computation produces a negative value that overflows the representable range of a float type.
0.0	The computation produces a value that underflows the representable range of a float type (for example, dividing a number by infinity).
NaN	The computation produces a value that cannot be represented as a number (for example, the result of an illegal operation such as dividing zero by zero or any operation that uses NaN as input, such as $10.0 + \text{NaN} = \text{NaN}$ ).

### Usage Notes

Make sure the strings that are passed to the service in *num1* and *num2* are in a locale-neutral format (that is, using the pattern `-####.##`). Passing locally formatted strings may result in unexpected results. For example, calling `pub.math:addFloats` in a German locale with the arguments `1,23` and `2,34` will result in the value `357`, not `3.57` or `3,57`.

Use the `watt.server.math.floatOperation.mode` property to specify whether the `pub.math:divideFloats` service return the exact result of an operation involving two floating point numbers, the result as calculated by the JVM, or the result based on a fixed number of decimal places. For information about the `watt.server.math.floatOperation.mode` property, see the *webMethods Integration Server Administrator's Guide*.

---

## pub.math:divideInts

WmPublic. Divides one integer (represented as a String) by another (*num1/num2*) and returns the quotient.

### Input Parameters

---

*num1*      **String** Number (integer represented as a String) that is the dividend.

*num2*      **String** Number (integer represented as a String) that is the divisor.

### Output Parameters

---

*value*      **String** The quotient of *num1* / *num2*.

### Usage Notes

Make sure the strings that are passed to the service in *num1* and *num2* are in a locale-neutral format (that is, using the pattern -####.##). Passing locally formatted strings may result in unexpected results. For example, calling `pub.math:addFloats` in a German locale with the arguments `1,23` and `2,34` will result in the value `357`, not `3.57` or `3,57`.

---

## pub.math:divideObjects

WmPublic. Divides one `java.lang.Number` object by another (*num1/num2*) and returns the quotient.

### Input Parameters

---

*num1*     **java.lang.Number** Number that is the dividend. See the Usage Notes for supported sub-classes.

*num2*     **java.lang.Number** Number that is the divisor. See the Usage Notes for supported sub-classes.

### Output Parameters

---

*value*     **java.lang.Number** Quotient of *num1* / *num2*.

### Usage Notes

This service accepts the following sub-classes of `java.lang.Number`: `java.lang.Byte`, `java.lang.Double`, `java.lang.Float`, `java.lang.Integer`, `java.lang.Long`, `java.lang.Short`.

This service applies the following rules for binary numeric promotion to the operands in order:

- If either operand is of type `Double`, the other is converted to `Double`.
- Otherwise, if either operand is of type `Float`, the other is converted to `Float`.
- Otherwise, if either operand is of type `Long`, the other is converted to `Long`.
- Otherwise, both operands are converted to type `Integer`.

These promotion rules mirror the Java rules for numeric promotion of numeric types.

---

## pub.math:max

WmPublic. Returns the largest number from a list of numbers.

### Input Parameters

---

*numList*      **String List** List of numbers from which the largest number is to be returned.

### Output Parameters

---

*maxValue*      **String** Largest number from the list of numbers.

---

## pub.math:min

WmPublic. Returns smallest number from a list of numbers.

### Input Parameters

---

*numList*      **String List** List of numbers from which the smallest number is to be returned.

### Output Parameters

---

*minValue*      **String** Smallest number from the list of numbers.

---

## pub.math:multiplyFloatList

WmPublic. Multiplies a list of floating point numbers (represented in a String list) and returns the product.

### Input Parameters

---

*numList*      **String List** Numbers (floating point numbers represented as Strings) to multiply.

### Output Parameters

---

*value*      **String** Product of the numbers in *numlist* . If a product cannot be produced, *value* contains one of the following:

Value	Description
Infinity	The computation produces a positive value that overflows the representable range of a float type.
-Infinity	The computation produces a negative value that overflows the representable range of a float type.
0.0	The computation produces a value that underflows the representable range of a float type (for example, multiplying a number by infinity).
NaN	The computation produces a value that cannot be represented as a number (for example, the result of an illegal operation such as multiplying zero by zero or any operation that uses NaN as input, such as $10.0 + \text{NaN} = \text{NaN}$ ).

### Usage Notes

Make sure the strings that are passed to the service in *numList* are in a locale-neutral format (that is, using the pattern `-####.##`). Passing locally formatted strings may result in unexpected results. For example, calling `pub.math:addFloats` in a German locale with the arguments `1,23` and `2,34` will result in the value `357`, not `3.57` or `3,57`.

## pub.math:multiplyFloats

WmPublic. Multiplies one floating point number (represented as String) by another and returns the product.

### Input Parameters

<i>num1</i>	<b>String</b> Number (floating point number represented as a String) to multiply.
<i>num2</i>	<b>String</b> Number (floating point number represented as a String) to multiply.
<i>precision</i>	<b>String</b> Optional. Number of decimal places to which the product will be rounded. The default value is null.  The <i>precision</i> parameter, if specified, will override the behavior set by the <code>watt.server.math.floatOperation.mode</code> property. For information about the <code>watt.server.math.floatOperation.mode</code> property, see the <i>webMethods Integration Server Administrator's Guide</i> .

---

## Output Parameters

---

*value* **String** Product of the numeric values of *num1* and *num2*. If a product cannot be produced, *value* contains one of the following:

<b>Value</b>	<b>Description</b>
Infinity	The computation produces a positive value that overflows the representable range of a float type.
-Infinity	The computation produces a negative value that overflows the representable range of a float type.
0.0	The computation produces a value that underflows the representable range of a float type (for example, multiplying a number by infinity).
NaN	The computation produces a value that cannot be represented as a number (for example, the result of an illegal operation such as multiplying zero by zero or any operation that uses NaN as input, such as $10.0 + \text{NaN} = \text{NaN}$ ).

### Usage Notes

Make sure the strings that are passed to the service in *num1* and *num2* are in a locale-neutral format (that is, using the pattern -####.##). Passing locally formatted strings may result in unexpected results. For example, calling `pub.math:addFloats` in a German locale with the arguments `1,23` and `2,34` will result in the value `357`, not `3.57` or `3,57`.

Use the `watt.server.math.floatOperation.mode` property to specify whether the `pub.math:multiplyFloats` service return the exact result of an operation involving two floating point numbers, the result as calculated by the JVM, or the result based on a fixed number of decimal places. See *webMethods Integration Server Administrator's Guide* for more information about the `watt.server.math.floatOperation.mode` property.

---

## pub.math:multiplyIntList

WmPublic. Multiplies a list of integers (represented in a String list) and returns the product.

---

### Input Parameters

---

*numList*      **String List** Numbers (floating point numbers represented as Strings) to multiply.

### Output Parameters

---

*value*          **String** Product of the numbers in *numList* .

### Usage Notes

Make sure the result of your calculation is less than 64 bits in width (the maximum width for the long data type). If the result exceeds this limit, it will generate a data overflow.

Make sure the strings that are passed to the service in *numList* are in a locale-neutral format (that is, using the pattern -####.##). Passing locally formatted strings may result in unexpected results. For example, calling [pub.math:addFloats](#) in a German locale with the arguments 1,23 and 2,34 will result in the value 357, not 3.57 or 3,57.

---

## pub.math:multiplyInts

WmPublic. Multiplies one integer (represented as a String) by another and returns the product.

### Input Parameters

---

*num1*      **String** Number (integer represented as a String) to multiply.

*num2*      **String** Number (integer represented as a String) to multiply.

### Output Parameters

---

*value*          **String** Product of *num1* and *num2* .

### Usage Notes

Make sure the result of your calculation is less than 64 bits in width (the maximum width for the long data type). If the result exceeds this limit, it will generate a data overflow.

Make sure the strings that are passed to the service in *num1* and *num2* are in a locale-neutral format (that is, using the pattern -####.##). Passing locally formatted strings

may result in unexpected results. For example, calling `pub.math:addFloats` in a German locale with the arguments `1, 23` and `2, 34` will result in the value `357`, not `3.57` or `3,57`.

---

## pub.math:multiplyObjects

WmPublic. Multiplies one `java.lang.Number` object by another and returns the product.

### Input Parameters

---

*num1*      **java.lang.Number** Number to multiply. See the Usage Notes for supported sub-classes.

*num2*      **java.lang.Number** Number to multiply. See the Usage Notes for supported sub-classes.

### Output Parameters

---

*value*      **java.lang.Number** Product of *num1* and *num2* .

### Usage Notes

This service accepts the following sub-classes of `java.lang.Number`: `java.lang.Byte`, `java.lang.Double`, `java.lang.Float`, `java.lang.Integer`, `java.lang.Long`, `java.lang.Short`.

This service applies the following rules for binary numeric promotion to the operands in order:

- If either operand is of type `Double`, the other is converted to `Double`.
- Otherwise, if either operand is of type `Float`, the other is converted to `Float`.
- Otherwise, if either operand is of type `Long`, the other is converted to `Long`.
- Otherwise, both operands are converted to type `Integer`.

These promotion rules mirror the Java rules for numeric promotion of numeric types.

---

## pub.math:randomDouble

WmPublic. Returns the next pseudorandom, uniformly distributed double between 0.0 and 1.0.

Random number generators are often referred to as pseudorandom number generators because the numbers produced tend to repeat themselves over time.

**Input Parameters**

---

None.

**Output Parameters**

---

*number*     **String** Generated random number.

---

## pub.math:roundNumber

WmPublic. Returns a rounded number.

**Input Parameters**

---

*num*             **String** Number to be rounded.

*numberOfDigits*     **String** Specifies the number of digits to which you want to round the number.

*roundingMode*     **String** Optional. Specifies the rounding method.  
  
Valid values for the *roundingMode* parameter are RoundHalfUp, RoundUp, RoundDown, RoundCeiling, RoundFloor, RoundHalfDown, and RoundHalfEven. The default value is RoundHalfUp.

**Output Parameters**

---

*roundedNumber*     **String** The rounded number.

---

## pub.math:subtractFloats

WmPublic. Subtracts one floating point number (represented as a String) from another and returns the difference.

**Input Parameters**

---

*num1*             **String** Number (floating point number represented as a String).

- num2* **String** Number (floating point number represented as a String) to subtract from *num1*.
- precision* **String** Optional. Number of decimal places to which the difference will be rounded. The default value is null.
- The *precision* parameter, if specified, will override the behavior set by the `watt.server.math.floatOperation.mode` property. For information about the `watt.server.math.floatOperation.mode` property, see the *webMethods Integration Server Administrator's Guide*.

### Output Parameters

- value* **String** Difference of *num1* - *num2*. If a difference cannot be produced, *value* contains one of the following:

Value	Description
Infinity	The computation produces a positive value that overflows the representable range of a float type.
-Infinity	The computation produces a negative value that overflows the representable range of a float type.
0.0	The computation produces a value that underflows the representable range of a float type (for example, subtracting a number from infinity).
NaN	The computation produces a value that cannot be represented as a number (for example, the result of an illegal operation such as multiplying zero by zero or any operation that uses NaN as input, such as $10.0 - \text{NaN} = \text{NaN}$ ).

### Usage Notes

Make sure the strings that are passed to the service in *num1* and *num2* are in a locale-neutral format (that is, using the pattern `-####.##`). Passing locally formatted strings may result in unexpected results. For example, calling `pub.math:addFloats` in a German locale with the arguments `1,23` and `2,34` will result in the value `357`, not `3.57` or `3,57`.

Use the `watt.server.math.floatOperation.mode` property to specify whether the `pub.math:subtractFloats` service return the exact result of an operation involving two floating point numbers, the result as calculated by the JVM, or the result based on a fixed number of decimal places. For more information about the

---

watt.server.math.floatOperation.mode property, see *webMethods Integration Server Administrator's Guide*

---

## pub.math:subtractInts

WmPublic. Subtracts one integer (represented as a String) from another and returns the difference.

### Input Parameters

---

*num1*     **String** Number (integer represented as a String).

*num2*     **String** Number (integer represented as a String) to subtract from *num1* .

### Output Parameters

---

*value*     **String** Difference of *num1* - *num2* .

### Usage Notes

Make sure the result of your calculation is less than 64 bits in width (the maximum width for the long data type). If the result exceeds this limit, it will generate a data overflow.

Make sure the strings that are passed to the service in *num1* and *num2* are in a locale-neutral format (that is, using the pattern -####.##). Passing locally formatted strings may result in unexpected results. For example, calling [pub.math:addFloats](#) in a German locale with the arguments 1,23 and 2,34 will result in the value 357, not 3.57 or 3,57.

---

## pub.math:subtractObjects

WmPublic. Subtracts one java.lang.Number object from another and returns the difference.

### Input Parameters

---

*num1*     **java.lang.Number** Number. See the Usage Notes for supported sub-classes.

*num2*     **java.lang.Number** Number to subtract from *num1* . See Usage Notes for supported sub-classes.

---

## Output Parameters

---

*value*     **java.lang.Number** Difference of *num1* - *num2* .

### Usage Notes

This service accepts the following sub-classes of `java.lang.Number`: `java.lang.Byte`, `java.lang.Double`, `java.lang.Float`, `java.lang.Integer`, `java.lang.Long`, `java.lang.Short`.

This service applies the following rules for binary numeric promotion to the operands. The following rules are applied in order:

- If either operand is of type `Double`, the other is converted to `Double`.
- Otherwise, if either operand is of type `Float`, the other is converted to `Float`.
- Otherwise, if either operand is of type `Long`, the other is converted to `Long`.
- Otherwise, both operands are converted to type `Integer`.

These promotion rules mirror the Java rules for numeric promotion of numeric types.

---

## pub.math:toNumber

WmPublic. Converts a string to numeric data type.

### Input Parameters

---

*num*            **String** Number (represented as a string) to be converted to numeric format.

*convertAs*    **String** Optional. Specifies the Java numeric data type to which the *num* parameter is to be converted.

Valid values for the *convertAs* parameter are `java.lang.Double`, `java.lang.Float`, `java.lang.Integer`, `java.math.BigDecimal`, `java.math.BigInteger`, `java.lang.Long`. The default value is `java.lang.Double`.

### Output Parameters

---

*num*            **java.lang.Number** Converted numeric object.



---

# 19 Metadata Folder

---

- Summary of Elements in this Folder ..... 550

You use the elements in the metadata folder to publish metadata about Integration Server packages and administrative assets to the CentraSite shared registry.

## Summary of Elements in this Folder

The following elements are available in this folder:

Element	Package and Description
<a href="#">pub.metadata.assets:publishPackages</a>	WmAssetPublisher. Publishes metadata about Integration Server packages and administrative assets to the CentraSite shared registry.

### pub.metadata.assets:publishPackages

WmAssetPublisher. Publishes metadata about Integration Server packages, the supported assets in the package, and administrative assets to the CentraSite shared registry.

#### Input Parameters

<i>packages</i>	<b>String</b> A comma-separated list of Integration Server packages about which you want to publish metadata.
<i>includeServerAssets</i>	<p><b>String</b> Whether to publish information about Integration Server administrative assets to the CentraSite shared registry. Specify one of the following values:</p> <ul style="list-style-type: none"> <li>■ <code>True</code> to indicate that information about administrative assets will be published to the CentraSite shared registry. This is the default.</li> <li>■ <code>False</code> to indicate that information about administrative assets will not be published to the CentraSite shared registry.</li> </ul>

#### Output Parameters

None.

#### Usage Notes

- Use the [pub.metadata.assets:publishPackages](#) service to publish metadata about Integration Server packages and administrative assets to CentraSite on a scheduled basis. To schedule this service, create a scheduled task that executes this service.

For instructions about creating a scheduled task, see *webMethods Integration Server Administrator's Guide*

- For the `pub.metadata.assets:publishPackages` service to execute successfully, Integration Server must be configured to connect to CentraSite and the CentraSite connection must be available. If Integration Server is not configured to connect to CentraSite or the connection is not available, Integration Server returns a `ServiceException`. For information about configuring the connection to CentraSite, see *webMethods Integration Server Administrator's Guide*.
- When this service executes, it publishes metadata using Integration Server credentials.



# 20 MIME Folder

---

■ Summary of Elements in this Folder .....	554
--	-----

You use the elements in the mime folder to create MIME messages and extract information from MIME messages.

## Summary of Elements in this Folder

The following elements are available in this folder:

Service	Function
<a href="#">pub.mime:addBodyPart</a>	WmPublic. Adds a body part (header fields and content) to a specified MIME object.
<a href="#">pub.mime:addMimeHeader</a>	WmPublic. Adds one or more header fields to a specified MIME object.
<a href="#">pub.mime:createMimeData</a>	WmPublic. Creates a MIME object.
<a href="#">pub.mime:getBodyPartContent</a>	WmPublic. Retrieves the content (payload) from the specified MIME object.
<a href="#">pub.mime:getBodyPartHeader</a>	WmPublic. Returns the list of header fields for the specified body part.
<a href="#">pub.mime:getContentType</a>	WmPublic. Returns the value of the Content-Type message header from the specified MIME object.
<a href="#">pub.mime:getEnvelopeStream</a>	WmPublic. Generates an InputStream representation of a MIME message from a specified MIME object.
<a href="#">pub.mime:getMimeHeader</a>	WmPublic. Returns the list of message headers from a specified MIME object.
<a href="#">pub.mime:getNumParts</a>	WmPublic. Returns the number of body parts in the specified MIME object.
<a href="#">pub.mime:getPrimaryContentType</a>	WmPublic. Returns the top-level portion of a MIME object's Content-Type value.
<a href="#">pub.mime:getSubContentType</a>	WmPublic. Returns the sub-type portion of a MIME object's Content-Type value.

Service	Function
<a href="#">pub.mime:mergeHeaderAndBody</a>	WmPublic. Concatenates the contents of the header and body returned by the <a href="#">pub.client:http</a> service.

## pub.mime:addBodyPart

WmPublic. Adds a body part (header fields and content) to a specified MIME object.

### Input Parameters

<i>mimeData</i>	<b>Document</b> MIME object to which you want to add a body part. (This IData object is produced by <a href="#">pub.mime:createMimeData</a> .)
<i>content</i>	<b>java.io.InputStream or Object</b> Content that you want to add to the MIME object. <i>content</i> can be an InputStream or another MIME object. Use an InputStream to add an ordinary payload. Use a MIME object to add a payload that is itself a MIME message.
<i>isEntityStream</i>	<b>String</b> Flag that specifies whether <i>content</i> is to be treated as a MIME entity.
	<p><b>Important:</b> This parameter is only used if <i>content</i> is an InputStream.</p> <p>Set this parameter to one of the following values:</p> <ul style="list-style-type: none"> <li>■ <i>yes</i> to treat <i>content</i> as a MIME entity. <code>addBodyPart</code> will strip out the header fields from the top of <i>content</i> and add them to <i>mimeData</i> as part headers. The remaining data will be treated as the payload.</li> </ul> <p><b>Note:</b> <code>addBodyPart</code> assumes that all data up to the first blank line represents the entity's header fields.</p> <ul style="list-style-type: none"> <li>■ <i>no</i> to treat <i>content</i> as an ordinary payload.</li> </ul>
<i>mimeHeader</i>	<b>Document</b> Specifies the part headers that you want to add with this body part. Key names represent the names of the header fields. The values of the keys represent the values of the header fields.

For example, if you wanted to add the following header fields:

```
X-Doctype: RFQ
X-Severity: 10
```

You would set *mimeHeader* as follows:

Key	Value
<i>X-Doctype</i>	RFQ
<i>X-Severity</i>	10

Be aware that the following MIME headers are automatically inserted by [pub.mime:getEnvelopeStream](#) when it generates the MIME message:

```
Message-ID
MIME-Version
```

Additionally, you use the *content*, *encoding*, and *description* parameters to set the following fields:

```
Content-Type
Content-Transfer-Encoding
Content-Description
```

If you set these header fields in *mimeHeader* and you create a single-part message, the values in *contenttype*, *encoding*, and *description*, if specified, will override those in *mimeHeader*. See usage notes.

*contenttype*

**String** Optional. The value of the Content-Type header for this body part. For single-part messages, this value overrides the Content-Type value in *mimeHeader*, if one is present. Defaults to text/plain.

See usage notes.

*encoding*

**String** Optional. Specifies how the body part is to be encoded for transport and sets the value of the Content-Transfer-Encoding header. For single-part messages, this value overrides the Content-Transfer-Encoding value in *mimeHeader*, if one is present. Defaults to 7bit.

See usage notes.

**Note:** This parameter determines how the payload is to be encoded for transport. When you add a payload to *mimeData*, it should be in its original format. The [pub.mime:getEnvelopeStream](#) service will perform the encoding (as specified by *encoding*) when it generates the final MIME message.

Set to:

- `7bit` to specify that *content* is 7-bit, line-oriented text that needs no encoding. This is the default.
- `8bit` to specify that *content* is 8-bit, line-oriented text that needs no encoding.

**Note:** This encoding value is not recommended for messages that will be transported via SMTP over the Internet, because the data can be altered by intervening mail servers that can't accommodate 8-bit text. To safely transport 8-bit text, use quoted-printable encoding instead.

- `binary` to specify that *content* contains binary information that needs no encoding.

**Note:** This encoding value is not recommended for messages that will be transported via SMTP over the Internet, because the data can be altered by intervening mail servers that can't accommodate binary data. To safely transport binary data, use base64 encoding instead.

- `quoted-printable` to specify that *content* contains 7 or 8-bit, line-oriented text that you want to encode using the quoted-printable encoding scheme.
- `base64` to specify that *content* contains an arbitrary sequence of octets that you want to encode using the base64 encoding scheme.
- `uuencode` to specify that *content* contains an arbitrary sequence of octets that you want to encode using the uuencode encoding scheme.

*description*

**String** Optional. Specifies the value of the `Content-Description` header for this body part.

*multipart*

**String** Optional. Flag that determines how `addBodyPart` behaves if *mimeData* already contains one or more body parts.

By default, `addBodyPart` simply appends a new body part to *mimeData* if it already contains a payload. (This allows you to construct multi-part messages.) However, you can override this behavior if you want to either replace the existing payload with the new body part or throw an exception under these circumstances (see *replace* parameter, below).

Set to:

- `yes` to append a new body part to `mimeData` . This is the default.
- `no` to replace the existing payload with the new body part. (Depending on the value of `replace` , this setting may cause `addBodyPart` to throw an exception.)

*replace*

**String** Optional. Flag that specifies whether `addBodyPart` replaces the existing payload or throws an exception when it receives a `mimeData` that already contains a payload. This parameter is only used when `multipart` is set to `no`.

Set to:

- `yes` to replace the existing payload with the new body part. This is the default.
- `no` to throw an exception.

## Output Parameters

---

*mimeData*

**Document** MIME object to which the body part was added.

## Usage Notes

This service operates on the MIME object (*mimeData* ) produced by [pub.mime:createMimeData](#).

The way in which the *contenttype* and *encoding* parameters are applied depends on whether the finished message is single-part or multipart.

For single-part messages:

- *contenttype* specifies the Content-Type for the entire MIME message. It overrides any value assigned to the Content-Type header in *mimeHeader* . If Content-Type is not specified in *contenttype* or *mimeHeader* , the value of the Content-Type header defaults to text/plain.
- *encoding* specifies the Content-Transfer-Encoding for the entire MIME message. It overrides any value assigned to the Content-Transfer-Encoding header in *mimeHeader* . If Content-Transfer-Encoding is not specified in *encoding* or *mimeHeader* , the value of the Content-Transfer-Encoding header defaults to 7bit.

For multipart messages:

- *contenttype* specifies the Content-Type for an individual body part. The Content-Type for the entire MIME message is automatically set to multipart/mixed, or to multipart/*subType* if a subtype was specified when the MIME object was created. See [pub.mime:createMimeData](#).

- *encoding* specifies the Content-Transfer-Encoding for an individual body part. The Content-Transfer-Encoding header in *mimeHeader*, if present, specifies the encoding for the entire MIME message. If Content-Transfer-Encoding is not specified in *mimeHeader*, or if the specified value is not valid for a multipart message, the value of the Content-Transfer-Encoding header defaults to 7bit. (7bit, 8bit, and binary are the only encoding values valid for multipart messages.)

For general information about MIME messages and using the MIME services, see the *MIME-S/MIME Developer's Guide*.

### See Also

[pub.mime:createMimeData](#)  
[pub.mime:getBodyPartContent](#)  
[pub.mime:addMimeHeader](#)

### Examples

For examples of how to use this service, see the following services in the certified samples area of the Knowledge Center on the Empower Product Support Website <https://empower.softwareag.com>.

`samples.mime:build_SimpleMIME`

`samples.mime:build_MultipartMIME`

---

## pub.mime:addMimeHeader

WmPublic. Adds one or more header fields to a specified MIME object.

### Input Parameters

---

<i>mimeData</i>	<b>Document</b> MIME object to which you want the header fields added. (This IData object is produced by <a href="#">pub.mime:createMimeData</a> .)
<i>mimeHeader</i>	<b>Document</b> Header fields that you want to add to the MIME object. Key names represent the names of the header fields. The values of the keys represent the values of the header fields. For example, to add the following header fields:

```
X-Doctype: RFQ
X-Severity: 10
```

You would set *mimeHeader* as follows:

<u>Key</u>	<u>Description</u>
------------	--------------------

*X-Doctype*      RFQ

*X-Severity*      10

Be aware that the following MIME headers are automatically inserted by [pub.mime:getEnvelopeStream](#) when it generates the MIME message:

```
Message-ID
MIME-Version
```

If you set these values in *mimeHeader*, [pub.mime:getEnvelopeStream](#) will overwrite them at run time.

## Output Parameters

*mimeData*      **Document** MIME object to which the header fields were added.

## Usage Notes

This service operates on the MIME object (*mimeData*) produced by [pub.mime:createMimeData](#).

If you add MIME headers before you add multiple body parts, the header fields will be added to each of the body parts. If you do not want this behavior, either drop *mimeHeader* from the pipeline immediately after you execute `addMimeHeader`, or invoke `addMimeHeader` after you've added all body parts to the MIME object.

Be aware that the *contentType* and *encoding* parameters used by the [pub.mime:addBodyPart](#) service will override any Content-Type or Content-Transfer-Encoding settings in *mimeData*. Moreover, in certain cases, the [pub.mime:getEnvelopeStream](#) will override these settings when it generates a multipart message. For information about how the Content-Type or Content-Transfer-Encoding headers are derived at run time, see the Usage Notes under [pub.mime:addBodyPart](#).

For general information about MIME messages and using the MIME services, see the *MIME-S/MIME Developer's Guide*.

## See Also

- [pub.mime:createMimeData](#)
- [pub.mime:getMimeHeader](#)
- [pub.mime:addBodyPart](#)

## Examples

For examples of how to use this service, see the following services in the certified samples area of the Knowledge Center on the Empower Product Support Website <https://empower.softwareag.com>.

samples.mime:build\_SimpleMIME

## pub.mime:createMimeData

WmPublic. Creates a MIME object.

If no *input* parameter is passed to this service, the service creates an empty MIME object. Otherwise, the service creates a MIME object containing the elements (header fields and content) from the MIME message in *input*.

- If you are building a MIME message, you use this service to create an empty MIME object. You populate the empty MIME object with header fields and content, and then pass it to [pub.mime:getEnvelopeStream](#), which produces the finished MIME message.
- If you are extracting data from a MIME message, you use this service to parse the original MIME message into a MIME object so that you can extract its header fields and content using other webMethods services.

### Input Parameters

*input* **java.io.InputStream** Optional. MIME entity you want to parse. If *input* is not provided, `createMimeData` creates an empty MIME object.

*mimeHeader* **Document** Optional. Specifies header fields that you want to add to the MIME object. Key names represent the names of the header fields. The values of the keys represent the values of the header fields.

**Note:** This parameter is ignored when *input* is passed to this service.

For example, if you wanted to add the following header fields:

```
X-Doctype: RFQ
X-Severity: 10
```

You would set *mimeHeader* as follows:

Key	Value
<i>X-Doctype</i>	RFQ
<i>X-Severity</i>	10

Be aware that the following MIME headers are automatically inserted by [pub.mime:getEnvelopeStream](#) when it generates the MIME message:

```
Message-ID
```

## MIME-Version

If you set these values in *mimeHeader*, [pub.mime:getEnvelopeStream](#) will overwrite them at run time.

- subType* **String** Optional. String that specifies the subtype portion of the Content Type header, when the message is a multipart message and you want something other than the default value of `mixed`. For example, if you want the Content Type header to be `multipart/related` in the resulting message, set *subType* to `related`.
- subType* is ignored if the resulting message is not a multipart message.
- decodeHeaders* **String** Optional. Specifies how the MIME header is to be decoded.
- Set to:
- " " (empty String) to decode headers based on the value of the global `watt.server.mime.decodeHeaders` property. This is the default.
  - `NONE` to specify that the MIME header or body part headers do not need decoding.
  - `ONLY_MIME_HEADER` to decode the MIME header only.
  - `ONLY_BODY_PART_HEADERS` to decode the body part headers only.
  - `BOTH` to decode the MIME header and the body part headers.

## Output Parameters

---

- mimeData* **Document** MIME object. If *input* was passed to `createMimeData`, *mimeData* will contain the parsed MIME message. If *input* was not passed to `createMimeData`, *mimeData* will be empty.
- encrypted* **String** Conditional. Indicates whether input was an encrypted message. This parameter is not present when the service creates a new, empty MIME object. A value of:
- `true` indicates that the message is encrypted (the original message stream is in *stream*).
  - `false` indicates that the message is not encrypted.
- signed* **String** Conditional. Flag whose value indicates whether *input* was a signed message. This parameter is not present when the service creates a new, empty MIME object. A value of:

- `true` indicates that the message is signed (the original message stream is in *stream*).
- `false` indicates that the message is not signed.

***certsOnly***

**String** Conditional. Flag whose value indicates whether *input* contained only digital certificates. (This type of message can be produced by the [pub.smime:createCertsOnlyData](#) service and allows digital certificates to be transported via the network as a MIME message.) This parameter is not present when the service creates a new, empty MIME object. A value of:

- `true` indicates that the message contains only certificates.
- `false` indicates that the message contains a regular payload.

***stream***

**java.io.InputStream** Conditional. `InputStream` containing the original MIME message from *input*. This parameter is present only when *input* is an S/MIME message.

**Usage Notes**

All of the other MIME services operate on the *mimeData* `IData` object produced by this service. They do not operate directly on MIME message streams.

**Important:** You can examine the contents of *mimeData* during testing and debugging. However, because the internal structure of *mimeData* is subject to change without notice, **do not** explicitly set or map data to/from these elements in your service. To manipulate or access the contents of *mimeData*, use **only** the MIME services that Integration Server provides.

For general information about MIME messages and using the MIME services, see the *MIME-S/MIME Developer's Guide*.

**See Also**

[pub.mime:addMimeHeader](#)  
[pub.mime:addBodyPart](#)  
[pub.mime:getMimeHeader](#)  
[pub.mime:getBodyPartContent](#)  
[pub.mime:getEnvelopeStream](#)

**Examples**

For examples of how to use this service, see the following services in the certified samples area of the Knowledge Center on the Empower Product Support Website <https://empower.softwareag.com>.

`samples.mime:build_SimpleMIME`

samples.mime:build\_MultipartMIME

samples.mime:extract\_SimpleMIME

samples.mime:extract\_MultipartMIME

---

## pub.mime:getBodyPartContent

WmPublic. Retrieves the content (payload) from the specified MIME object.

You use this service for both single-part and multi-part messages.

To retrieve content from a multi-part message, you set the *index* (to select the part by index number) or *contentID* (to select the part by *contentID* value) parameter to specify the body part whose content you want to retrieve. To get the content from a single-part message, you omit the *index* and *contentID* parameters or set *index* to 0.

### Input Parameters

---

<i>mimeData</i>	<b>Document</b> MIME object whose content you want to retrieve. (This IData object is produced by <a href="#">pub.mime:createMimeData</a> .)
<i>index</i>	<b>String</b> Optional. Index number of the body part whose content you want to retrieve (if you want to retrieve the content from a specific body part). The first body part is index number zero.  <b>Note:</b> If <i>contentID</i> is specified, <i>index</i> is ignored.
<i>contentID</i>	<b>String</b> Optional. Value of the Content-ID header field of the body part whose content you want to retrieve (if you want to retrieve the payload from a specific body part).

### Output Parameters

---

<i>content</i>	<b>IData</b> The payload of the specified body part.
<i>encrypted</i>	<b>String</b> Flag whose value indicates whether <i>content</i> is an encrypted MIME message. A value of: <ul style="list-style-type: none"> <li>■ <code>true</code> indicates that <i>content</i> is an encrypted message.</li> <li>■ <code>false</code> indicates that <i>content</i> is not an encrypted message.</li> </ul>
<i>signed</i>	<b>String</b> Flag indicating whether <i>content</i> is a signed MIME message. A value of: <ul style="list-style-type: none"> <li>■ <code>true</code> indicates that <i>content</i> is a signed MIME message.</li> </ul>

- `false` indicates that *content* is not a signed MIME message.

*certsOnly*

**String** Flag whose value indicates whether *content* is a certs-only MIME message. A value of:

- `true` indicates that *content* is a certs-only message.
- `false` indicates that *content* is not a certs-only message.

### Usage Notes

This service operates on the MIME object (*mimeData*) produced by [pub.mime:createMimeData](#).

If you omit *index* or *contentID* when retrieving content from a multi-part message, `getBodyPartContent` returns the payload from the first body part. If you use *index* or *contentID* to select a body part that does not exist in *mimeData*, *content* will be null.

For general information about MIME messages and using the MIME services, see the *MIME-S/MIME Developer's Guide*.

### See Also

[pub.mime:createMimeData](#)  
[pub.mime:addBodyPart](#)  
[pub.mime:getBodyPartHeader](#)

### Examples

For examples of how to use this service, see the following services in the certified samples area of the Knowledge Center on the Empower Product Support Website <https://empower.softwareag.com>.

`samples.mime:extract_SimpleMIME`

`samples.mime:extract_MultipartMIME`

---

## pub.mime:getBodyPartHeader

WmPublic. Returns the list of header fields for the specified body part.

### Input Parameters

---

*mimeData*

**Document** MIME object whose message headers you want to retrieve. (This IData object is produced by [pub.mime:createMimeData](#))

*index*

**String** Optional. Index number of the body part whose header fields you want to retrieve. The first body part is index zero.

**Note:** If *contentID* is specified, *index* is ignored.

- contentID* **String** Optional. Value of the `Content-ID` header field of the body part whose header fields you want to retrieve.
- decodeHeaders* **String** Conditional. Flag whose value indicates whether to decode encoded headers in the MIME object. Set to:
- `true` to indicate that the headers should be decoded.
  - `false` to indicate that the headers should not be decoded. This is the default.

## Output Parameters

*mimeHeader* **Document** IData object containing the message headers. Key names represent the names of the header fields. The value of a key represents the value of that header field.

For example, if the original message contained the following message header fields:

```
Content-Type: text/xml
X-Doctype: RFQ
X-Severity: 0
```

`get Body Part Header` would return the following IData object:

Key	Value
<i>Content-Type</i>	text/xml
<i>X-Doctype</i>	RFQ
<i>X-Severity</i>	0

## Usage Notes

This service operates on the MIME object (*mimeData*) produced by [pub.mime.createMimeData](#).

If you omit *index* or *contentID*, `getBodyPartHeader` returns the message headers from the first body part. If you use *index* or *contentID* to select a body part that does not exist in *mimeData*, *content* will be null.

For general information about MIME messages and using the MIME services, see the *MIME-S/MIME Developer's Guide*.

---

**See Also**

[pub.mime:createMimeData](#)  
[pub.mime:addBodyPart](#)  
[pub.mime:getMimeHeader](#)

---

## pub.mime:getContentType

WmPublic. Returns the value of the Content-Type message header from the specified MIME object.

---

**Input Parameters**

*mimeData*                      **Document** MIME object whose Content-Type you want to discover. (This IData object is produced by [pub.mime:createMimeData](#).)

---

**Output Parameters**

*contentType*                      **String** Value of the MIME object's Content-Type header field. Note that this service returns only the media type and subtype portion of this header field's value. It does not return any parameters the value may include. For example, if the message's Content-Type header were:

```
Content-Type: text/plain;charset=UTF8
```

*contentType* would contain:

```
text/plain
```

---

**Usage Notes**

This service operates on the MIME object (*mimeData*) produced by [pub.mime:createMimeData](#).

For general information about MIME messages and using the MIME services, see the *MIME-S/MIME Developer's Guide*.

**See Also**

[pub.mime:createMimeData](#)  
[pub.mime:getSubContentType](#)  
[pub.mime:getPrimaryContentType](#)  
[pub.mime:getMimeHeader](#)  
[pub.mime:getBodyPartHeader](#)

---

## pub.mime:getEnvelopeStream

WmPublic. Generates an InputStream representation of a MIME message from a specified MIME object.

### Input Parameters

---

<i>mimeData</i>	<b>Document</b> MIME object from which you want to generate the MIME message. (This IData object is produced by <a href="#">pub.mime:createMimeData</a> .)
<i>index</i>	<b>String</b> Optional. Index number of the body part for which you want to generate the MIME message (if you want to generate the message from a specific body part). The first body part is index number zero.
<i>contentID</i>	<b>String</b> Optional. Value of the <code>Content-ID</code> header field of the body part from which you want to generate the MIME message (if you want to generate the message from a specific body part).  <b>Note:</b> If <i>index</i> is specified, <i>contentID</i> is ignored.
<i>suppressHeaders</i>	<b>String List</b> Optional. Names of header fields that are to be omitted from message. You can use this option to exclude header fields that <code>getEnvelopeStream</code> generates by default, such as <code>Content-Type</code> and <code>content-encoding</code> .
<i>createMultipart</i>	<b>String</b> Optional. Specifies whether a multipart message is to be created, even if <i>mimeData</i> contains only one body part. Set to: <ul style="list-style-type: none"><li>■ <code>yes</code> to create a multipart message (Content-Type message header is set to "multipart/mixed").</li><li>■ <code>no</code> to create a message based on the number of body parts in <i>mimeData</i>. This is the default.<ul style="list-style-type: none"><li>■ If the message contains only one body part, Content-Type is set according to the <i>contenttype</i> setting specified when that body part was added to <i>mimeData</i>.</li><li>■ If the message contains multiple body parts, Content-Type is automatically set to "multipart/mixed."</li></ul></li></ul>

---

## Output Parameters

---

*envStream*                    **java.io.InputStream** The MIME message as an InputStream.

### Usage Notes

This service operates on the MIME object (*mimeData*) produced by [pub.mime:createMimeData](#).

If you omit *index* or *contentID*, `getEnvelopeStream` generates the MIME message from the entire contents of the *mimeData*. If you use *index* or *contentID* to select a body part that does not exist in *mimeData*, *content* will be null.

`getEnvelopeStream` automatically inserts the `MIME-Version` and `Message-ID` message headers into the MIME message it puts into *envStream*.

For general information about MIME messages and using the MIME services, see the *MIME-S/MIME Developer's Guide*.

### See Also

[pub.mime:createMimeData](#)  
[pub.mime:addBodyPart](#)  
[pub.mime:addMimeHeader](#)

### Examples

For examples of how to use this service, see the following services in the certified samples area of the Knowledge Center on the Empower Product Support Website <https://empower.softwareag.com>.

`samples.mime:build_SimpleMIME`

`samples.mime:build_MultipartMIME`

---

## pub.mime:getMimeHeader

WmPublic. Returns the list of message headers from a specified MIME object.

### Input Parameters

---

*mimeData*                    **Document** MIME object whose message headers you want to retrieve. (This IData object is produced by [pub.mime:createMimeData](#).)

---

## Output Parameters

---

*mimeHeader* **Document** Conditional. An IData object containing the message headers. Key names represent the names of the header fields. The value of a key represents the value of the header fields.

For example, if the original message contained the following message header fields:

```
Message-ID: <002e01c0f150$6f33010a@sgx.com>
From: "Purch01@GSX.com" <Purch01@GSX.com>To:
<EXPEst@exprint.com>
MIME-Version: 1.0
Content-Type: text/xml
X-Doctype: RFQ
X-Severity: 0
```

`getMimeHeader` would return the following:

Key	Value
Message-ID	<002e01c0f150\$6f33010a@sgx.com>
From	"Purch01@GSX.com" <Purch01@GSX.com>
To	<EXPEst@exprint.com>
MIME-Version	1.0
Content-Type	text/xml
X-Doctype	RFQ
X-Severity	0

### Usage Notes

This service operates on the MIME object (*mimeData*) produced by [pub.mime:createMimeData](#).

For general information about MIME messages and using the MIME services, see the *MIME-S/MIME Developer's Guide*.

### See Also

[pub.mime:createMimeData](#)  
[pub.mime:addMimeHeader](#)  
[pub.mime:getBodyPartHeader](#)

---

## pub.mime:getNumParts

WmPublic. Returns the number of body parts in the specified MIME object.

### Input Parameters

---

*mimeData*                    **Document** MIME object whose parts you want to count. (This IData object is produced by [pub.mime:createMimeData](#).)

### Output Parameters

---

*numParts*                    **String** The number of body parts in the MIME object.

### Usage Notes

This service operates on the MIME object (*mimeData*) produced by `createMimeData`.

For general information about MIME messages and using the MIME services, see the *MIME-S/MIME Developer's Guide*.

### See Also

[pub.mime:createMimeData](#)  
[pub.mime:getBodyPartContent](#)  
[pub.mime:addBodyPart](#)

### Examples

For examples of how to use this service, see the following service in the certified samples area of the Knowledge Center on the Empower Product Support Website <https://empower.softwareag.com>.

`samples.mime:extract_MultipartMIME`

---

## pub.mime:getPrimaryContentType

WmPublic. Returns the top-level portion of a MIME object's Content-Type value.

### Input Parameters

---

*mimeData*                    **Document** MIME object whose Content-Type you want to discover. (This IData object is produced by [pub.mime:createMimeData](#).)

---

## Output Parameters

---

*primContentType* **String** Message's top-level Content-Type. For example, if the message's Content-Type header were:

```
Content-Type: multipart/mixed
```

*primContentType* would contain:

```
multipart
```

## Usage Notes

This service operates on the MIME object (*mimeData*) produced by [pub.mime:createMimeData](#).

For general information about MIME messages and using the MIME services, see the *MIME-S/MIME Developer's Guide*.

## See Also

- [pub.mime:createMimeData](#)
- [pub.mime:getContentType](#)
- [pub.mime:addMimeHeader](#)
- [pub.mime:getBodyPartHeader](#)

---

## pub.mime:getSubContentType

WmPublic. Returns the sub-type portion of a MIME object's Content-Type value.

## Input Parameters

---

*mimeData* **Document** MIME object whose sub-type you want to discover. (This IData object is produced by [pub.mime:createMimeData](#).)

## Output Parameters

---

*subContentType* **String** Message's sub-type. For example, if the message's Content-Type header were:

```
Content-Type: multipart/mixed
```

*subContentType* would contain:

```
mixed
```

## Usage Notes

This service operates on the MIME object (*mimeData*) produced by [pub.mime:createMimeData](#).

For general information about MIME messages and using the MIME services, see the *MIME-S/MIME Developer's Guide*.

## See Also

[pub.mime:createMimeData](#)  
[pub.mime:getContentType](#)  
[pub.mime:addMimeHeader](#)  
[pub.mime:getBodyPartHeader](#)

---

## pub.mime:mergeHeaderAndBody

WmPublic. Concatenates the contents of the header and body returned by the [pub.client:http](#) service.

You can use this service to reassemble the message into its original form so that it can be used as input to the [pub.mime:createMimeData](#) service (or any other service that requires the entire http response as an `InputStream`).

## Input Parameters

---

*headerLines*      **Document** IData object containing the message headers returned by [pub.client:http](#). (The message headers are returned in the *lines* document inside the *header* output parameter that is produced by [pub.client:http](#).)

*body*              **Document** IData object containing the body of the message returned by [pub.client:http](#). This document must contain the body of the message in one of the following keys:

<u>Key</u>	<u>Description</u>
<i>bytes</i>	<b>byte[ ]</b> Optional. Body of the message (if <a href="#">pub.client:http</a> returned the body as a <code>byte[ ]</code> ).
<i>stream</i>	<b>java.io.InputStream</b> Optional. The body of the message (if <a href="#">pub.client:http</a> returned the body as an <code>InputStream</code> ).

## Output Parameters

---

*stream*                    **java.io.InputStream** InputStream containing the reassembled tap message.

## Usage Notes

Use this service to merge the results produced by [pub.client:http](#) to get the original MIME message.

## See Also

[pub.client:http](#)

[pub.mime:createMimeData](#)

# 21 OAuth Folder

---

Use the elements in the `oauth` folder to authorize a client application to access data on Integration Server using the OAuth 2.0 Authorization Framework.

Client applications use these services to interact with Integration Server when Integration Server is configured to act as the OAuth authorization server. In this chapter, *authorization server* refers to the Integration Server that acts as the authorization server. For information about configuring Integration Server as the OAuth authorization server, see *webMethods Integration Server Administrator's Guide*.

**Note:** Before using these services, you must have already registered the client with the authorization server and received a client identifier. You will use this information to configure the services in this folder. For information about registering a client, see *webMethods Integration Server Administrator's Guide*.

## Summary of Elements in this Folder

The following elements are available in this folder:

Element	Package and Description
<code>pub.oauth:authorize</code>	Initiates an authorization request from a client application to the authorization server.
<code>pub.oauth:getAccessToken</code>	Requests an access token from the authorization server.
<code>pub.oauth:refreshAccessToken</code>	Requests a fresh token from the authorization server.

### `pub.oauth:authorize`

Initiates an authorization request from a client application to the authorization server.

#### Input Parameters

<code>response_type</code>	<p><b>String</b> The grant type preferred by the client. This parameter informs the authorization server how to respond to the client. Set to:</p> <ul style="list-style-type: none"> <li>code for the authorization code grant. When set to <code>code</code>, the response from authorization server must include an OAuth authentication code the client can exchange for an access token.</li> </ul>
----------------------------	--

- `token` for an implicit grant type. When set to `token`, the response from authorization server includes an OAuth access token for the client.

For more information about grant types, see ["Usage Notes" on page 577](#).

*client\_id*

**String** The client identifier generated by the authorization server when the client application is registered. The *client\_id* is used to authenticate the client to the authorization server.

*redirect\_uri*

**String** Optional. The URI that the authorization server will use to redirect the client when the client is authorized.

This parameter is required if the client is registered with more than one redirect URI. The value for *redirect\_uri* must match one of the client's registered redirect URIs.

*scope*

**String** Optional. The name of the scope associated with the client. The scope defines the level of access requested by the client.

Specify the name of one or more scopes. Use a space to separate the name of the scopes. For example:

```
scope1 scope2 scope3
```

The scopes you specify must already exist on the authorization server. For information about creating a scope, see *webMethods Integration Server Administrator's Guide*.

*state*

**String** Optional. A unique string used to maintain the state between the request and callback. When the authorization server redirects the user to the *redirect\_uri*, the value for *state* will be included in the response. Software AG recommends using this parameter to protect against cross-site request forgery (CSRF) attacks.

## Output Parameters

---

None.

## Usage Notes

This service must be invoked using HTTPS unless the **Require HTTPS** setting on the **Security > OAuth > Edit OAuth Global Settings** page is disabled.

When you register a client, you must consider the grant type the client should use to obtain an access token. Integration Server supports the following grant types:

- **Authorization code.** Requires the client to authenticate to the authorization server before obtaining an access token. The authentication code supplied by the authorization server is included in the redirection URI. The client can refresh an expired token. To implement an authorization code grant, set the *response\_type* to `code`.
- **Implicit.** Less secure than the authorization code grant. It does not require the client to authenticate to the authorization server. The authentication server includes the access token in the redirection URI. The client cannot refresh an expired token. To implement an implicit grant, set the *response\_type* to `token`.

Authentication code is not persisted in the cache. If Integration Server is restarted after the authorization code is issued but before the access token is requested, Integration Server will reject the request for the access token.

---

## pub.oauth:getAccessToken

Requests an access token from the authorization server.

The authorization server validates the request and generates an access token and a refresh token (if one is requested). The tokens, along with the client identifier, token expiration interval, and scope are stored in the authorization server's cache.

### Input Parameters

---

<i>grant_type</i>	<p><b>String</b> Specifies the type of grant flow required by the client. Since this service is for authorization code grant flows, you must specify <code>authorization_code</code>.</p>
<i>client_id</i>	<p><b>String</b> The client identifier generated by the authorization server when the client application is registered. The <i>client_id</i> is used to authenticate the client to the authorization server.</p> <p>Public clients must provide a value for <i>client_id</i>. Confidential clients do not need to provide a value for this parameter because they are required to use HTTP authentication to identify themselves.</p>
<i>code</i>	<p><b>String</b> The OAuth authorization code received from the authorization server.</p>

---

*redirect\_uri*                      **String** The URI the authorization server will use to redirect the client when the client is authorized.

### Output Parameters

---

*access\_token*                      **String** The access token issued by the authorization server.

*token\_type*                      **String** The type of access token issued by the authorization server. The value is `Bearer`.

*expires\_in*                      **String** The number of seconds for which the access token is valid.

*refresh\_token*                      **String** Optional. The refresh token issued by the authorization server. You can use this token to obtain new access tokens using the same authorization grant.

If the client is registered with a refresh limit of 0, no refresh token is issued.

### Usage Notes

This service is used with authentication code grant flows only.

This service must be invoked using HTTPS unless the **Require HTTPS** setting on the **Security > OAuth > Edit OAuth Global Settings** page is disabled.

Clients must invoke this service via an HTTP POST request.

Confidential clients must authenticate requests by supplying their credentials in the HTTP Authorization header.

Authentication code is not persisted in the cache. If Integration Server is restarted after the authorization code is issued but before the access token is requested, Integration Server will reject the request for the access token.

When Integration Server acts as the authorization server, the *token\_type* output parameter is always `Bearer`. The authorization server retains the information about the bearer tokens it issues, including the user information. When the client presents a bearer token to the resource server, the resource server checks with the authorization server to see whether the user is allowed to access the requested folders and services.

The tokens, authorization codes, and client information are stored in the authorization server's caches. By default, these caches maintain up to 1000 elements in memory and 10000 elements on disk. If the cache size is exceeded, OAuth performance can be affected on Integration Server and can lead to errors if the disk runs out of space. If you anticipate that your authorization server's cache will exceed the default size, you should increase the **Maximum Elements In Memory**, **Maximum Elements On Disk**, or **Maximum Off-Heap** settings for Integration Server. For information about changing these settings,

---

see *webMethods Integration Server Administrator's Guide*. If the cache is distributed, see the BigMemory Max product documentation for 4.1 at [www.terracotta.org/documentation](http://www.terracotta.org/documentation) for additional considerations when sizing distributed caches.

---

## pub.oauth:refreshAccessToken

Requests a refresh token from the authorization server. If the authorization server issued a refresh token to the client with the initial request, the client can use this service to submit a token refresh request when that initial refresh token expires.

### Input Parameters

---

<i>grant_type</i>	<b>String</b> Specify the type of grant flow required by the client. For refresh tokens, you must specify <code>refresh_token</code> .
<i>refresh_token</i>	<b>String</b> Refresh token issued to the client by the authentication server.
<i>scope</i>	<b>String</b> Optional. Specify the name of one or more scopes required by the client. Use a space to separate multiple scopes.  The value for <i>scope</i> must match or be a subset of the value you provided for the <code>pub.oauth:authorize</code> and <code>pub.oauth:getAccessToken</code> services.  The scope of the refresh token can be smaller than the original request. It cannot contain any scope tokens that were not in the original request.

### Output Parameters

---

<i>access_token</i>	<b>String</b> The access token issued by the authorization server.
<i>token_type</i>	<b>String</b> The type of access token issued by the authorization server. The value is <code>Bearer</code> .
<i>expires_in</i>	<b>String</b> The number of seconds for which the access token is valid.
<i>refresh_token</i>	<b>String</b> The refresh token issued by the authorization server. You can use this token to obtain new access tokens using the same authorization grant.

*scope* **String** Conditional. The name of the scopes requested by the client.

### Usage Notes

This service is used with authorization grant flows only.

This service must be invoked using HTTPS unless the **Require HTTPS** setting on the **Security > OAuth > Edit OAuth Global Settings** page is disabled.

Clients must invoke this service via an HTTP POST request.

Confidential clients must authenticate requests by supplying their credentials in the HTTP Authorization header.

When Integration Server acts as the authorization server, the *token\_type* output parameter is always `Bearer`. The authorization server retains the information about the bearer tokens it issues, including the user information. When the client presents a bearer token to the resource server, the resource server checks with the authorization server to see whether the user is allowed to access the requested folders and services.



# 22 Packages Folder

---

■ Summary of Elements in this Folder .....	584
--	-----

You use the elements in the packages folder to install, load, and/or alter the status of a package on the Integration Server.

## Summary of Elements in this Folder

The following elements are available in this folder:

Element	Package and Description
<a href="#">pub.packages:activatePackage</a>	WmPublic. Activates (makes available to clients) an inactive package.
<a href="#">pub.packages:backupPackage</a>	WmPublic. Creates a backup copy of a specified package.
<a href="#">pub.packages:disablePackage</a>	WmPublic. Disables a package, thus prohibiting access to the services in the package.
<a href="#">pub.packages:enablePackage</a>	WmPublic. Enables a package that has been disabled.
<a href="#">pub.packages:installPackage</a>	WmPublic. Installs a package that has been published to this server.
<a href="#">pub.packages:recoverPackage</a>	WmPublic. Recovers a package that exists in the server's salvage directory.
<a href="#">pub.packages:reloadPackage</a>	WmPublic. Loads a new copy of the package into memory from disk.
<a href="#">pub.packages.hotdeployment:cancel</a>	WmPublic. Cancels a hot deployment operation.

### **pub.packages:activatePackage**

WmPublic. Activates (makes available to clients) an inactive package.

You use this service to activate a package that was not activated when it was initially installed or recovered.

**Note:** This service activates packages from an *inactive* state (that is, packages that are installed on the server but are not registered in the active-package list). To enable a package that is in a *disabled* state, you use [pub.packages:enablePackage](#).

### Input Parameters

---

*package*                      **String** Name of the package that you want to activate. Package names are case sensitive.

### Output Parameters

---

*message*                      **String** Message from server. (This is the same message that you receive when you activate a package with the Integration Server Administrator.)

### Usage Notes

This service will throw an exception if the package specified in *package* does not exist or cannot otherwise be activated.

When a package is activated, it is loaded into memory in an enabled state (that is, `activatePackage` automatically activates *and* enables the package.) You do not need to explicitly enable it with [pub.packages:enablePackage](#).

### See Also

[pub.packages:enablePackage](#)  
[pub.packages:installPackage](#)  
[pub.packages:recoverPackage](#)

---

## pub.packages:backupPackage

WmPublic. Creates a backup copy of a specified package.

### Input Parameters

---

*packageName*                      **String** Name of the package to back up.

### Output Parameters

---

None.

## Usage Notes

The service creates the backup in a file named *packageName.zip*, where *packageName* is the name of the original package installed in Integration Server. The *packageName.zip* is placed in the following directory:

*Integration Server\_directory*\instances\*instance\_name* \replicate\inbound

If a package with the same name as the file produced by this service already exists in the *Integration Server\_directory*\instances\*instance\_name* \replicate\inbound directory, the service overwrites the existing .zip file with the backup copy that the service creates.

The backed up package is an exact copy of the specified package. Package metadata, such as creation timestamp, will be the same in the backup as in the original package. This is unlike package replication or package archiving in which the creation timestamp reflects the time the package was replicated or archived.

---

## pub.packages:disablePackage

WmPublic. Disables a package, thus prohibiting access to the services in the package.

### Input Parameters

<i>package</i>	<b>String</b> Name of the package that you want to disable. Package names are case sensitive.
----------------	---

### Output Parameters

<i>message</i>	<b>String</b> Message from server. (This is the same message that you receive when you disable a package with the Integration Server Administrator.)
----------------	--

## Usage Notes

When a package is disabled, the services in the package are no longer available to the clients. To re-enable a package that has been disabled, use [pub.packages:enablePackage](#).

**Important:** Never disable the WmRoot package. Doing so would disable the server.

Be aware that if you disable a package while services in the package are being executed, those services will most likely fail. `disablePackage` does not wait for in-progress services to finish before disabling a package.

This service will throw an exception if the package specified in *package* does not exist or cannot otherwise be disabled.

**See Also**

[pub.packages:enablePackage](#)

---

## pub.packages:enablePackage

WmPublic. Enables a package that has been disabled.

**Note:** This service enables a package that is in a *disabled* state (that is, a package that has been disabled through the Integration Server Administrator or the [pub.packages:disablePackage](#) service). To activate a package that is in an *inactive* state, you use `enablePackage`.

---

### Input Parameters

*package*                      **String** Name of the package that you want to enable. Package names are case sensitive.

---

### Output Parameters

*message*                      **String** Message from server. (This is the same message that you receive when you enable a package with the Integration Server Administrator.)

### Usage Notes

When you enable a package, the package is reloaded into memory from disk.

This service will throw an exception if the package specified in *package* does not exist, has not been activated, or cannot otherwise be enabled.

**See Also**

[pub.packages:disablePackage](#)  
[pub.packages:activatePackage](#)  
[pub.packages:reloadPackage](#)

---

## pub.packages:installPackage

WmPublic. Installs a package that has been published to this server.

---

## Input Parameters

---

*packageFile*

**String** Name of the distribution file that contains the package that you want to install. This file must reside in the server's inbound directory (*Integration Server\_directory*\instances\*instance\_name* \replicate\inbound).

When specifying *packageFile* ,

- **Do** include the .zip extension in the file name.
- **Do not** include the directory path.

For example: myPackageFileAug2001.zip

*activateOnInstall*

**String** Flag that specifies whether you want Integration Server to automatically activate the package after it is installed. Set to:

- *yes* to activate the package after installation and make it immediately available to clients. This is the default.
- *no* to install the package without activating it afterwards. If you install a package in this mode, it will not be accessible until it is explicitly activated through the Integration Server Administrator or the [pub.packages:activatePackage](#) service.

*archiveOnInstall*

**String** Optional. Flag that specifies whether you want Integration Server to archive the package automatically after it is installed. Set to:

- *yes* to archive the package after installation. If you choose to archive the package automatically, Integration Server moves the package from the *Integration Server\_directory*\instances\*instance\_name* \replicate\inbound directory to the *Integration Server\_directory*\instances\*instance\_name* \replicate\archive directory. This is the default.
- *no* to install the package without archiving it.

---

## Output Parameters

---

*message*

**String** Message from server. (This is the same message that is displayed when you install a package with the Integration Server Administrator.)

## Usage Notes

If the installed package replaces an existing package on the server, `pub.packages:installPackage` will automatically put a backup copy of the existing package in `Integration Server_directory\instances\instance_name\replicate\salvage` before it installs the new package.

This service will throw an exception if the file named in `packageFile` does not exist or cannot otherwise be installed correctly.

## See Also

[pub.packages:activatePackage](#)

[pub.packages:recoverPackage](#)

---

## pub.packages:recoverPackage

WmPublic. Recovers a package that exists in the server's salvage directory.

The salvage directory (`Integration Server_directory\instances\instance_name\replicate\salvage`) is where the server keeps packages that are deleted with the "safe delete" option or replaced with newer installed versions.

## Input Parameters

---

<i>package</i>	<b>String</b> Name of the package that you want to recover. Package names are case sensitive.
<i>activateOnRecover</i>	<b>String</b> Flag that specifies whether you want the server to automatically activate the package after it is recovered. Set to: <ul style="list-style-type: none"><li>■ <code>yes</code> to activate the package after it is recovered and make it immediately available to clients.</li><li>■ <code>no</code> to recover the package without activating it afterwards. If you recover a package in this mode, it will not be accessible until it is explicitly activated through the Integration Server Administrator or the <a href="#">pub.packages:activatePackage</a> service.</li></ul>

## Output Parameters

---

<i>message</i>	<b>String</b> Message from server. (This is the same message that is displayed when you recover a package with the Integration Server Administrator.)
----------------	---

### Usage Notes

You can only recover packages that exist in the server's salvage directory.

If you recover a package that is currently installed on the server, the package from the salvage directory *replaces* the version that is currently installed. (Be aware that the server *does not* retain a copy of the version that it replaces.)

This service will throw an exception if the file named in *package* does not exist in the server's salvage directory or cannot otherwise be recovered.

### See Also

[pub.packages:activatePackage](#)

---

## pub.packages:reloadPackage

WmPublic. Loads a new copy of the package into memory from disk.

If you make changes to the service in a package while the server is running, you must use `reloadPackage` to put those changes into effect.

### Input Parameters

---

*package*                      **String** Name of the package that you want to reload. Package names are case sensitive.

### Output Parameters

---

*message*                      **String** Message from server. (This is the same message that is displayed when you reload a package with the Integration Server Administrator.)

### Usage Notes

Be aware that if you reload a package while services in the package are being executed, those services will most likely fail. `reloadPackage` does not wait for in-progress services to finish before reloading a package.

This service will throw an exception if the file named in *package* does not exist or cannot otherwise be reloaded.

---

## pub.packages.hotdeployment:cancel

WmPublic. Cancels a hot deployment operation.

You use the hot deployment feature in Integration Server to upgrade custom packages in Integration Server while ensuring that the Integration Server assets are available for processing without any noticeable downtime. You can use the `pub.packages.hotdeployment:cancel` service to cancel a hot deployment operation if Integration Server is unable to start installing the package because of delay in the completion of in-flight tasks. For more information about hot deployment of packages, see *webMethods Integration Server Administrator's Guide*.

### Input Parameters

---

*package*                      **String** Name of the package that is being upgraded using hot deployment operation that you want to cancel. Package names are case sensitive.

### Output Parameters

---

*message*                      **String** Message from server indicating the status of the cancellation of the hot deployment operation.



# 23 Publish Folder

---

■ Summary of Elements in this Folder .....	594
--	-----

You use the elements in the publish folder to publish messages to webMethods Universal Messaging or webMethods Broker. Other Integration Servers subscribe to and receive these messages using webMethods messaging triggers.

## Summary of Elements in this Folder

The following elements are available in this folder:

Element	Package and Description
<a href="#">pub.publish:deliver</a>	WmPublic. Delivers a document to a specific destination.
<a href="#">pub.publish:deliverAndWait</a>	WmPublic. Requests a reply document from a specific client. The service waits for the reply or indicates that the <a href="#">pub.publish:waitForReply</a> service should retrieve the reply later.
<a href="#">pub.publish:documentResolverSpec</a>	WmPublic. Specification for the signature of a service that determines whether a document's status is New, Duplicate, or In Doubt.
<a href="#">pub.publish:envelope</a>	WmPublic. Document type that defines the content and structure of the envelope that accompanies a published document.
<a href="#">pub.publish:getRedeliveryCount</a>	WmPublic. Retrieves the redelivery count for a document.
<a href="#">pub.publish:publish</a>	WmPublic. Publishes a document locally or to the messaging provider.
<a href="#">pub.publish:publishAndWait</a>	WmPublic. Broadcasts a request for a document from any client subscribed to a specific document type. The service waits for the reply or indicates that the <a href="#">pub.publish:waitForReply</a> service should retrieve the reply later.
<a href="#">pub.publish:reply</a>	WmPublic. Delivers a reply document to the requesting client.
<a href="#">pub.publish:syncToBroker</a>	WmPublic. <i>Deprecated</i> - Replaced by <a href="#">pub.publish:syncToProvider</a> .

Element	Package and Description
<a href="#">pub.publish:syncToProvider</a>	Synchronizes one or more publishable document types with their associated definition on the messaging provider.
<a href="#">pub.publish:waitForReply</a>	WmPublic. Retrieves the reply for an asynchronous request. If a reply is not available, the Integration Server continues to wait for the document until the time specified in the <i>waitTime</i> parameter of the <a href="#">pub.publish:deliverAndWait</a> or <a href="#">pub.publish:publishAndWait</a> service elapses.
<a href="#">pub.publish.notification:error</a>	WmPublic. Publishable document type that defines the document that Integration Server generates and delivers when a trigger encounters an error or exception condition during processing.

## pub.publish:deliver

WmPublic. Delivers a document to a specific destination.

### Input Parameters

<i>documentTypeName</i>	<p><b>String</b> Fully qualified name of the publishable document type being delivered.</p> <p>The publishable document type must be in sync with the associated provider definition. The provider definition is a representation of the publishable document type and its properties on the provider. In the case of Universal Messaging, this is a channel. For Broker, it is a Broker document type. If the document type and provider definition are not synchronized, publication might fail.</p> <p>A publishable document type that uses the IS_LOCAL_CONNECTION alias cannot be delivered because delivery can only occur with a messaging provider (Broker or Universal Messaging).</p>
<i>document</i>	<p><b>Document</b> Document (IData object) conforming to the publishable document type in <i>documentTypeName</i>.</p>
<i>destId</i>	<p><b>String</b> The destination to which the document will be delivered. You can specify the ID for an individual</p>

trigger or the default client of an Integration Server as the destination. When you deliver a document to the default client of Integration Server, Integration Server routes the delivered document to any subscribers on that Integration Server.

When working with Universal Messaging, the destination ID is a subject. When working with Broker, the destination ID is the client ID.

The destination ID for a trigger uses the following naming convention:

*clientPrefix\_folderName\_subfolderName\_triggerName*

Where *clientPrefix* is the client prefix for the messaging connection alias used by the trigger, *folderName* is the folder in which the trigger is located, *subfolderName* is the subfolder in which the trigger is located, and *triggerName* is the local name of the trigger. For example, for a trigger with the fully qualified name `myFolder.mySubFolder.myTrigger` that uses a messaging connection alias with the client prefix “myAlias”, the destination ID of the trigger is: `myAlias_myFolder_mySubfolder_myTrigger`.

The destination ID for the default client of an Integration Server uses the following naming convention:

*clientPrefix\_DefaultClient*

Where *clientPrefix* is the client prefix for the messaging connection alias used by subscribers to the publishable document type on the Integration Server.

**Note:** The naming conventions for destination IDs are the same on Universal Messaging and Broker. If you change the messaging provider for a publishable document type and therefore the triggers that subscribe to it, you do not need to change the *destID* for any services that deliver a document to the trigger or default client.

If you specify an invalid client ID, the Integration Server delivers the document to the messaging provider, but the messaging provider never delivers the document to the intended recipient and no error is produced.

*delayUntilServiceSuccess*

**String** Optional. Flag indicating whether the Integration Server should publish the document when the `pub.publish:deliver` service executes or after the top-level service successfully completes. If the top-level

service fails, the Integration Server will not publish the document.

Set to:

- `true` to delay publishing until after the top-level service executes successfully.
- `false` to publish the document when the `pub.publish:deliver` service executes. This is the default.

## Output Parameters

---

None.

## Usage Notes

You can use the `pub.publish:deliver` service to deliver a document to a destination where the destination can be a `webMethods` messaging trigger or the default client of an Integration Server. For a trigger that receives messages from Universal Messaging, you specify the subject that corresponds to the trigger or the Integration Server. For a trigger that receives messages from the Broker, you specify the client ID that corresponds to the trigger or the Integration Server.

To view the subjects for a trigger used with Universal Messaging, use Integration Server Administrator to open the **Settings > Messaging > webMethods Messaging Trigger Management** screen. In the **Individual webMethods Messaging Trigger** list, click the name of the trigger for which you want to view the client ID. On the **Settings > Messaging > webMethods Messaging Trigger Management > triggerName** screen, under **Destination ID for Deliver**, the **Trigger Client ID** field displays the client ID for the trigger. The **Default Client ID** field displays the client of the Integration Server.

To view a list of client IDs on the Broker, use the Broker user interface within My `webMethods` or use Designer to test the publishable document type that you want to deliver.

For a Universal Messaging connection alias the **Publish Wait Time While Reconnecting** value specifies the number of milliseconds that a publishing service using alias will wait for a connection to the Universal Messaging server to be re-established after the connection fails. If Integration Server re-establishes the connection before the **Publish Wait Time While Reconnecting** elapses, the publishing service continues executing. If the specified time elapses before a connection is re-established, the publishing service ends with an `ISRuntimeException`. Make sure to code your service to handle this situation.

If outbound client-side queuing is disabled (the `watt.server.publish.useCSQ` property is set to "never"), Integration Server throws a `ServiceException` if the Broker is not available when this service executes. Make sure to code your service to handle this situation.

If the `pub.publish:deliver` services publishes a document to Universal Messaging and use of the client side queue is disabled for the Universal Messaging connection alias, Integration Server throws an `ISRuntimeException` if the Universal Messaging server is

not available when the service executes. Make sure to code your service to handle this situation.

For more information about how the Integration Server, Universal Messaging, and Broker deliver documents and for information about building a service that delivers a document, see the *Publish-Subscribe Developer's Guide*.

### See Also

[pub.publish:publish](#)  
[pub.publish:deliverAndWait](#)  
[pub.publish:envelope](#)

---

## pub.publish:deliverAndWait

WmPublic. Requests a reply document from a specific client. The service waits for the reply or indicates that the [pub.publish:waitForReply](#) service should retrieve the reply later.

### Input Parameters

---

<i>documentTypeName</i>	<p><b>String</b> Fully qualified name of the publishable document type being delivered.</p> <p>The publishable document type must be in sync with the associated provider definition. The provider definition is a representation of the publishable document type and its properties on the provider. In the case of Universal Messaging, this is a channel. For Broker, it is a Broker document type. If the document type and provider definition are not synchronized, publication might fail.</p> <p>A publishable document type that uses the IS_LOCAL_CONNECTION alias cannot be delivered because delivery can only occur with a messaging provider (Broker or Universal Messaging).</p>
<i>document</i>	<p><b>Document</b> Document (IData object) conforming to the publishable document type in <i>documentTypeName</i>.</p>
<i>receiveDocumentTypeName</i>	<p><b>String</b> Optional. Fully qualified name of the publishable document type expected as a reply. If no value is specified, the service uses the first reply document of any type it receives, as long as the value of <i>tag</i> in the envelope of the reply document matches the <i>tag</i> in the envelope of the published document. All other reply documents are discarded.</p>

*destId*

**String** The destination to which the document will be delivered. You can specify the ID for an individual trigger or the default client of an Integration Server as the destination. When you deliver a document to the default client of Integration Server, Integration Server routes the delivered document to any subscribers on that Integration Server.

When working with Universal Messaging, the destination ID is a subject. When working with Broker, the destination ID is the client ID.

The destination ID for a trigger uses the following naming convention:

*clientPrefix\_folderName\_subfolderName\_triggerName*

Where *clientPrefix* is the client prefix for the messaging connection alias used by the trigger, *folderName* is the folder in which the trigger is located, *subfolderName* is the subfolder in which the trigger is located, and *triggerName* is the local name of the trigger. For example, for a trigger with the fully qualified name `myFolder.mySubFolder.myTrigger` that uses a messaging connection alias with the client prefix "myAlias", the destination ID of the trigger is: `myAlias_myFolder_mySubfolder_myTrigger`.

The destination ID for the default client of an Integration Server uses the following naming convention:

*clientPrefix\_DefaultClient*

Where *clientPrefix* is the client prefix for the messaging connection alias used by subscribers to the publishable document type on the Integration Server.

**Note:** The naming conventions for destination IDs are the same on Universal Messaging and Broker. If you change the messaging provider for a publishable document type and therefore the triggers that subscribe to it, you do not need to change the *destID* for any services that deliver a document to the trigger or default client.

If you specify an invalid client ID, the Integration Server delivers the document to the messaging provider, but the messaging provider never delivers the document to the intended recipient and no error is produced.

*waitTime*

**String** Optional. Specifies the time to wait (in milliseconds) for the response to arrive. If no value is specified, the service waits indefinitely until it receives a reply.

*async*

**String** Optional. Flag specifying whether this is an asynchronous or synchronous request/reply. Set to:

- `true` to indicate that this is an asynchronous request/reply. After publishing the document, the Integration Server executes the next step in the flow service immediately. Integration Server does not wait for a reply before continuing service execution.

**Note:** To retrieve the reply to an asynchronous request, invoke the [pub.publish:waitForReply](#) service.

- `false` to indicate that this is a synchronous request/reply. After publishing the document, Integration Server waits for a reply before executing the next step in the flow service. This is the default.

## Output Parameters

*receivedDocument*

**Document** A Document (IData object) received as reply.

**Important:** Integration Server treats all reply documents as volatile documents. If the Integration Server shuts down before processing the reply document, the reply document is lost.

*tag*

**String** Conditional. A unique identifier for a deliver request. The Integration Server uses the *tag* value to match the requesting document with its corresponding reply document.

The service produces a *tag* output value only when the *async* field is set to `true`. The *tag* value is required input when using the [pub.publish:waitForReply](#) service to retrieve the reply.

**Note:** The *tag* output value is the same value that the Integration Server places in the *tag* field of the request document's envelope.

## Usage Notes

You can use the [pub.publish:deliverAndWait](#) service to deliver a document to a destination where the destination can be a webMethods messaging trigger or the default client of an Integration Server. For a trigger that receives messages from Universal Messaging, you specify the subject that corresponds to the trigger or the Integration Server. For a trigger that receives messages from the Broker, you specify the client ID that corresponds to the trigger or the Integration Server.

To view the subjects for a trigger used with Universal Messaging, use Integration Server Administrator to open the **Settings > Messaging > webMethods Messaging TriggerManagement** screen. In the **Individual webMethods Messaging Trigger** list, click the name of the trigger for which you want to view the client ID. On the **Settings > Messaging > webMethods Messaging TriggerManagement > triggerName** screen, under **Destination ID for Deliver**, the **Trigger Client ID** field displays the client ID for the trigger. The **Default Client ID** field displays the client of the Integration Server.

To view a list of client IDs on the Broker, use the Broker user interface within My webMethods or use Designer to test the publishable document type that you want to deliver.

For a Universal Messaging connection alias the **Publish Wait Time While Reconnecting** value specifies the number of milliseconds that a publishing service using alias will wait for a connection to the Universal Messaging server to be re-established after the connection fails. If Integration Server re-establishes the connection before the **Publish Wait Time While Reconnecting** elapses, the publishing service continues executing. If the specified time elapses before a connection is re-established, the publishing service ends with an `ISRuntimeException`. Make sure to code your service to handle this situation.

You can use the `pub.publish:deliverAndWait` service to initiate and continue a private conversation between two clients. This is a variation of the request/reply model. One client executes a service that delivers a document to a specific client. This document requests information from the receiving client.

- In a synchronous request/reply, the delivering service stops executing while it waits for a response. When the service receives a reply document from the specified client, the servers resumes executing. If the *waitTime* elapses before the service receives a reply, Integration Server ends the request, and the service returns a null document indicating that the request timed out. Integration Server then executes the next step in the flow service. If a reply document arrives after the flow service resumes execution, Integration Server rejects the document and creates a journal log message stating that the document was rejected because there is no service thread waiting for the document.
- In an asynchronous request/reply, the delivering service continues executing the steps in the service after publishing the document. To retrieve the reply, the delivering service must invoke the `pub.publish:waitForReply` service. If the wait time elapses before the `pub.publish:waitForReply` service receives a document, the `pub.publish:waitForReply` service returns a null document indicating that the request timed out.

A service that contains multiple asynchronous deliver requests allows the service to deliver all the requests before collecting the replies. This approach can be more efficient than delivering a request, waiting for a reply, and then delivering the next request.

If you create a service that contains multiple asynchronous requests, make sure to link the *tag* output to another field in the pipeline. Each asynchronous delivery produces a *tag* field in the pipeline. If the *tag* field is not linked to another field, the next asynchronous delivery request (that is, the next execution of the `pub.publish:deliverAndWait` service) will overwrite the first *tag* value.

Use `pub.publish:deliverAndWait` if you need to know that a specific client successfully received and processed the request document.

If the publishable document type in `documentTypeName` is associated with a Universal Messaging connection alias, the **Enable Request/Reply Channel and Listener** check box must be selected for the alias. When this check box is selected, Integration Server ensures that a request/reply channel exists for the Universal Messaging connection alias on the Universal Messaging server and that Integration Server has a listener that subscribes to the alias-specific request/reply channel. If the check box is cleared, there will be no channel in which Universal Messaging can collect replies and no listener with which Integration Server can retrieve replies. The `pub.publish:deliverAndWait` service will end with a `ServiceException` if the **Enable Request/Reply Channel and Listener** check box is not selected for the Universal Messaging connection alias.

If outbound client-side queuing is disabled (the `watt.server.publish.useCSQ` property is set to "never"), Integration Server throws a `ServiceException` if the webMethods Broker is not available when this service executes. Make sure to code your service to handle this situation.

If the `pub.publish:deliverAndWait` services publishes a document to Universal Messaging and use of the client side queue is disabled for the Universal Messaging connection alias, Integration Server throws an `ISRuntimeException` if the Universal Messaging server is not available when the service executes. Make sure to code your service to handle this situation.

For more information about how to build a services that initiate synchronous or asynchronous request/reply scenarios, see the *Publish-Subscribe Developer's Guide*.

#### See Also

[pub.publish:waitForReply](#)  
[pub.publish:publishAndWait](#)  
[pub.publish:reply](#)  
[pub.publish:envelope](#)

---

## pub.publish:documentResolverSpec

`WmPublic`. Specification for the signature of a service that determines whether a document's status is `New`, `Duplicate`, or `In Doubt`.

#### Input Parameters

---

<code>documentTypeName</code>	<b>String</b> Fully qualified name of the document whose status is <code>In Doubt</code> .
<code>redeliveryCount</code>	<b>String</b> Number of times the document has been redelivered to the trigger queue on the Integration Server.

---

<i>uuid</i>	<b>String</b> Universally unique identifier for the document. The publishing application assigns the <i>uuid</i> to a document.
<i>document</i>	<b>Document</b> The document (IData object) whose status needs to be resolved. This document must conform to the publishable document type specified in <i>documentTypeName</i> .
<i>transport</i>	<b>String</b> The transport (such as LOCAL, BROKER, or UM) used to send the document to the Integration Server.
<i>triggerName</i>	<b>String</b> The name of the webMethods messaging trigger that received the document whose status needs to be resolved.

### Output Parameters

---

<i>status</i>	<p><b>String</b> Indicates the status of the document. The value of this field determines whether the Integration Server processes the document, discards the document, or sends the document to the audit log. The <i>status</i> field must have one of the following values.</p> <ul style="list-style-type: none"><li>■ <b>NEW</b> indicates that the document is new and has not been processed by the trigger. Integration Server instructs the trigger to process the document.</li><li>■ <b>DUPLICATE</b> indicates that the document is a duplicate of one already processed by the trigger. Integration Server discards the document and generates a journal log message.</li><li>■ <b>IN_DOUBT</b> indicates that the status of the document is still in doubt. The document resolver service could not conclusively determine whether the trigger already processed the document. If the audit log is a database, the audit subsystem logs the document and the Integration Server generates a journal log message.</li></ul>
<i>message</i>	<p><b>String</b> Conditional. A user-specified string that indicates why the document status is <b>DUPLICATE</b> or <b>IN_DOUBT</b>. Integration Server writes the message to the journal log when the server discards the document or routes it to the audit log.</p>

### Usage Notes

The `pub.publish:documentResolverSpec` must be used as the signature for any service used to resolve the processing status of a document. For information about building a document resolver service and enabling exactly once processing for a webMethods messaging trigger, see the *Publish-Subscribe Developer's Guide*.

Use the [pub.jms:documentResolverSpec](#) as the signature for a document resolver service used to determine the status of a JMS message received by a JMS trigger.

### See Also

[pub.jms:documentResolverSpec](#)

---

## pub.publish:envelope

WmPublic. Document type that defines the content and structure of the envelope that accompanies a published document.

The envelope records information such as the sender's address, the time the document was sent, password and certificate information, and other useful information for routing and control. Every publishable document type contains a document reference to this document type.

### Read/Write Parameters

---

You can set the following parameters within your service. Broker supports all of the read/write parameters. However, webMethods Universal Messaging supports only *activation*, *businessContext*, *priority*, *replyTo*, and *tag*. Universal Messaging ignores the values of any other read/write parameters set in the document envelope and passes them through to subscribers.

*activation*                      **String** Optional. A unique identifier that any messaging client (including the Integration Server) assigns to all documents published as a result of the one-time execution of the integration solution. If a document does not have an activation ID, the Integration Server assigns one when the document is published.

If you are using a trigger to join documents published by different services, you must explicitly set the activation ID of the documents. The services that publish the documents must assign the same activation ID to the documents.

*appLastSeqn*                    **java.lang.Integer** Optional. This field is provided for backward compatibility.

*appPassword*                    **String** Optional. The password of the user specified in *appUserName*. If the resource that processes the document requires authentication before it begins processing, specify the password in this field.

*appSeqn*                            **java.lang.Integer** Optional. This field is provided for backward compatibility.

---

<i>appUserName</i>	<b>String</b> Optional. The user name for logging into the application that processes the document. Use the <i>appPassword</i> field to specify the password for this user name.
<i>businessContext</i>	<b>String</b> Optional. Used by the Integration Server to track business process context and audit context across multiple Integration Servers.  <b>Important</b> The <i>businessContext</i> field is reserved for internal use by the Integration Server. Do not set or overwrite the value of the <i>businessContext</i> field.
<i>controlLabel</i>	<b>java.lang.Short[ ]</b> Optional. This field is provided for backward compatibility.
<i>errorsTo</i>	<b>String</b> Optional. The client ID to which the Integration Server sends an error notification document if errors occur during document processing by subscribers. If this parameter is not set, error notifications will be sent to the document publisher. The errors document is an instance of <a href="#">pub.publish.notification:error</a> .
<i>errorRequestsTo</i>	<b>String</b> Optional. This field is provided for backward compatibility.
<i>locale</i>	<b>String</b> Optional. Locale of the publishing client expressed as a URN (Uniform Resource Name). Trigger services examine the <i>locale</i> value to determine the locale to use when processing the document. If the <i>locale</i> field is empty, the locale of the current Integration Server is used instead.
<i>maxResults</i>	<b>java.lang.Integer</b> Optional. This field is provided for backward compatibility.
<i>priority</i>	<b>java.lang.Integer</b> Optional. Indicates how quickly the document should be published and processed. A value of 0 is the lowest processing priority; a value of 9 indicates expedited processing. Set a message priority in the document envelope when publishing the document. The default priority is 4.
<i>replyTo</i>	<b>String</b> Optional. The client ID to which the replies to the published document should be sent. If this parameter is not set, replies will be sent to the document publisher as specified in <i>pubId</i> .

<i>runLevel</i>	<b>java.lang.Integer</b> Optional. This field is provided for backward compatibility.
<i>signature</i>	<b>byte[ ]</b> Optional. A byte sequence that holds a digital signature. Specify a digital signature if clients receiving this document requires one.
<i>signatureType</i>	<b>String</b> Optional. The type of digital signature being used.
<i>startResult</i>	<b>java.lang.Integer</b> Optional. This field is provided for backward compatibility.
<i>tag</i>	<b>java.lang.Integer</b> Optional. Used with <a href="#">pub.publish:publishAndWait</a> and <a href="#">pub.publish:deliverAndWait</a> to match a request document with its corresponding reply document.
	<b>Important</b> The <i>tag</i> field is reserved for internal use by the Integration Server. Do not set or overwrite the value of the <i>tag</i> field in the envelope.
<i>trackId</i>	<b>String</b> Optional. A unique identifier assigned to a published document by the publishing client application. If no value is specified, Integration Server populates this field with the value of the <i>uuid</i> field.
<i>transactionId</i>	<b>String</b> Optional. This field is provided for backwards compatibility.
<i>transformState</i>	<b>String</b> Optional. An indication of a document's current state, set by a publishing client application that transforms data. For example, a client could publish a document with a <i>transformState</i> value of "USEnglish" and a receiving client could translate the document into French and publish it with a <i>transformState</i> value of "French."

### Read-only Parameters

webMethods Universal Messaging, webMethods Broker, or Integration Server set the following parameters. You cannot set these parameters within your service, but you can retrieve their values from published documents.

**Note:** Of the read-only parameters listed below, webMethods Universal Messaging supports the *destID*, *pubID*, and *uuid* parameters only.

---

<i>age</i>	<b>java.lang.Integer</b> Optional. The cumulative time, in seconds, that the document spends on all webMethods Brokers. The webMethods Broker starts tracking the document age when it receives the document from the publishing client. The webMethods Broker stops tracking the document age when the subscribing client removes the document from the client queue. If the document is routed to successive webMethods Brokers, <i>age</i> also includes the length of time the document spends on the other webMethods Brokers.
<i>connectionIntegrity</i>	<b>String</b> Optional. An indication of whether the received document passed over a link that is not secure. This field can have one of the following values: <ul style="list-style-type: none"><li>■ <i>&lt;empty&gt;</i> indicates that at some point, the document passed through a connection that was not encrypted.</li><li>■ <code>U.S.Export</code> indicates that all the connections used to transport the event had an encryption strength of <code>ENCRYPT_LEVEL_US_EXPORT</code> or greater.</li><li>■ <code>U.S.Domestic</code> indicates that the event traveled exclusively over connections with an encryption strength of <code>ENCRYPT_LEVEL_US_DOMESTIC</code>.</li></ul>
<i>destId</i>	<b>String</b> Optional. The ID of the destination to which the document is being delivered. The publisher sets the <i>destID</i> when it publishes the document. For example, the Integration Server uses the <i>destID</i> value specified in the <a href="#">pub.publish:deliver</a> service or the <a href="#">pub.publish:deliverAndWait</a> service to populate the <i>destID</i> value in the document envelope.
<i>enqueueTime</i>	<b>java.util.Date</b> Optional. The date and time that the webMethods Broker placed the document into the client queue.
<i>logBroker</i>	<b>String</b> Optional. The name of the webMethods Broker that contains the document in its document log. The webMethods Broker sets this parameter when webMethods Broker-based document logging and the logging utility are enabled.
<i>logHost</i>	<b>String</b> Optional. The host name and port number of the webMethods Broker that contains the document in its document log. The webMethods Broker sets this parameter when webMethods Broker-based document logging and the logging utility are enabled.

<i>pubDistinguishedName</i>	<b>String</b> Optional. The distinguished name of the publisher's SSL certificate. The webMethods Broker sets this parameter when the publisher has an SSL connection to the webMethods Broker and clears this parameter when the publisher has a non-SSL connection.
<i>pubId</i>	<b>String</b> Optional. An identifier for the document publisher assigned by the messaging provider.
<i>pubNetAddr</i>	<b>byte[ ]</b> Optional. The IP address and port number of the document's publisher.
<i>pubSeqn</i>	<b>java.lang.Long</b> Optional. This field is provided for backwards compatibility.
<i>pubLabel</i>	<b>java.lang.Short[ ]</b> Optional. This field is provided for backwards compatibility.
<i>recvTime</i>	<b>java.util.Date</b> Optional. The date and time the document was received by the messaging provider.
<i>route</i>	<b>Document List</b> Optional. Information about the webMethods Brokers through which a document passed. When a webMethods Broker receives a document, the webMethods Broker sets the <i>broker</i> and <i>recvTime</i> keys. When the webMethods Broker places the document in the queue for the next webMethods Broker, the first webMethods Broker sets <i>enqueueTime</i> . The webMethods Broker only sets these fields when document is forwarded from one webMethods Broker to another. The webMethods Broker does not set these fields when the publishing and receiving clients are connected to the same webMethods Broker.

<b>Key</b>	<b>Description</b>
<i>broker</i>	<b>String</b> Optional. The name of the webMethods Broker.
<i>recvTime</i>	<b>java.util.Date</b> Optional. The time the webMethods Broker received the document from the publishing client or another webMethods Broker.
<i>enqueueTime</i>	<b>java.util.Date</b> Optional. The time the webMethods Broker placed the

---

document in the queue for the next webMethods Broker.

*uuid* **String** Optional. Universally unique identifier for the document. Integration Server assigns the UUID when it publishes the document. The receiving Integration Server uses the UUID to detect duplicate documents.

### Output Parameters

---

None.

### Usage Notes

For more information about setting and using a document's envelope parameters, see *webMethods Broker Java Client API Reference*, *webMethods Broker Client C API Programmer's Guide*, and *Publish-Subscribe Developer's Guide*.

### See Also

[pub.publish:deliver](#)  
[pub.publish:deliverAndWait](#)  
[pub.publish:publish](#)  
[pub.publish:publishAndWait](#)  
[pub.publish:publishAndWait](#)  
[pub.publish.notification:error](#)

---

## pub.publish:getRedeliveryCount

WmPublic. Retrieves the redelivery count for a document.

The redelivery count indicates the number of times the document has been redelivered to the trigger queue on the Integration Server. A document is redelivered to a trigger queue if the Integration Server shuts down before processing and acknowledging the document.

### Input Parameters

---

None.

### Output Parameters

---

*redeliveryCount* **String** Specifies the number of times the trigger on Integration Server has received the document. The redelivery count can be one of the following:

<u>A value of...</u>	<u>Indicates...</u>
-1	The transport used to send the document does not maintain a document redelivery count. For example, a document received from a webMethods Broker version 6.0.1 has a redelivery count of -1. (webMethods Brokers that are version 6.0.1 or earlier do not maintain document redelivery counts.)  Integration Server may or may not have received the document before.
0	The document has been received only once.
> 0	The number of times document has been redelivered.

### Usage Notes

If you do not want to use the exactly once processing capabilities, you can invoke the `pub.publish:getRedeliveryCount` service within your trigger service. The redelivery count for a document can provide an initial indication of whether the trigger has already processed the document.

Integration Server retrieves the redelivery count for the document currently maintained in the invoke state. That is, Integration Server retrieves the redelivery count for the document that caused the trigger service to execute.

When a trigger service satisfied by an All (AND) join condition invokes `pub.publish:getRedeliveryCount`, the `pub.publish:getRedeliveryCount` service returns the redelivery count for the last document received by the join. For example, suppose that documents A and B satisfied an All (AND) join condition. If the Integration Server receives document A first and document B second, when `pub.publish:getRedeliveryCount` executes, it retrieves the redelivery count for document B.

---

## **pub.publish:publish**

WmPublic. Publishes a document locally or to the messaging provider.

This service broadcasts the document (that is, distributes the document to all clients that subscribe to it).

---

## Input Parameters

---

<i>documentTypeName</i>	<p><b>String</b> Fully qualified name of the publishable document type of which you are publishing an instance.</p> <p>If you intend to publish the document to a messaging provider (Broker or Universal Messaging), the publishable document type must be in sync with the associated provider definition. The provider definition is a representation of the publishable document type and its properties on the provider. In the case of Universal Messaging, this is a channel. For Broker, it is a Broker document type. If the document type and provider definition are not synchronized, publication might fail.</p>
<i>document</i>	<p><b>Document</b> Document (IData object) conforming to the document type in <i>documentTypeName</i> .</p>
<i>local</i>	<p><b>String</b> Optional. Flag specifying whether the document is to be published locally or to the messaging provider. Set to:</p> <ul style="list-style-type: none"><li>■ <code>true</code> to publish locally (to this Integration Server only).</li><li>■ <code>false</code> to publish to the messaging provider using the messaging connection alias specified in the document type of which you are publishing an instance. This is the default.</li></ul> <p>The <i>local</i> parameter is applicable only when the publishable document type uses Broker as the messaging provider.</p> <p>If the publishable document type specifies the <code>IS_LOCAL_CONNECTION</code> alias, the document can be published locally only. Integration Server ignores the value of <i>local</i> .</p> <p>A publishable document type that specifies Universal Messaging as the messaging provider cannot be published locally. If the document type in <i>documentTypeName</i> is associated with a Universal Messaging connection alias, Integration Server ignores the value of <i>local</i> .</p>
<i>delayUntilServiceSuccess</i>	<p><b>String</b> Optional. Flag indicating whether the publish should happen when the <code>pub.publish:publish</code> service executes or after the top-level service successfully</p>

completes. If the top-level service fails, Integration Server will not publish the document. Set to:

- `true` to delay publishing until after the top-level service executes successfully.

**Note:** Integration Server does not return the `status` output parameter when `delayUntilServiceSuccess` is set to `true`.

- `false` to publish the document when the publish service executes. This is the default.

## Output Parameters

*status*

**String** Status indicating whether the service was successful. Integration Server reports status only for locally published documents. A value of:

- `success` indicates that the service executed successfully.

**Note:** If at least one subscribing trigger has room in its queue, the status is set to `success`.

- `noSubscriber` indicates that Integration Server does not contain any triggers that subscribe to the document.
- `capacityExceeded` indicates that the document could not be placed in the queue of the subscribing trigger because the trigger queue is currently at its maximum capacity.

**Note:** Integration Server reports this status only when the `watt.server.publish.local.rejectOOS` property is set to `true`.

## Usage Notes

You can use `pub.publish:publish` to publish documents to Universal Messaging, Broker, or locally within Integration Server.

The messaging connection alias assigned to a publishable document type determines the message provider to which the Integration Server publishes the document. Only subscribers to the document type on that messaging provider will receive the published document. The **Connection alias name** property for the document type specifies the message provider. If there is no specified message provider alias, Integration Server publishes the document to the provider in the default messaging connection alias.

---

If the messaging connection alias specified for the document type does in *documentTypeName* does not exist when the `pub.publish:publish` service executes, Integration Server issues a `ServiceException`. Make sure to code your service to handle this situation.

Integration Server writes a message to the journal log whenever it rejects or discards a document.

For a Universal Messaging connection alias the **Publish Wait Time While Reconnecting** value specifies the number of milliseconds that a publishing service using alias will wait for a connection to the Universal Messaging server to be re-established after the connection fails. If Integration Server re-establishes the connection before the **Publish Wait Time While Reconnecting** elapses, the publishing service continues executing. If the specified time elapses before a connection is re-established, the publishing service ends with an `ISRuntimeException`. Make sure to code your service to handle this situation.

If the `pub.publish:publish` services publishes a document to Broker and outbound client-side queuing is disabled (the `watt.server.publish.useCSQ` property is set to "never"), Integration Server issues a `ServiceException` if the Broker is not available when the service executes. Make sure to code your service to handle this situation.

If the `pub.publish:publish` services publishes a document to Universal Messaging and use of the client side queue is disabled for the Universal Messaging connection alias, Integration Server throws an `ISRuntimeException` if the Universal Messaging server is not available when the service executes. Make sure to code your service to handle this situation.

Integration Server issues a `ServiceException` when the dispatcher is shut down during the execution of this service. In this situation, Integration Server does not save the data in the outbound document store (client side queue), and the document will not appear in the outbound document store. Make sure to code your service to handle this situation.

For more information about building a service that publishes a document locally or to a messaging provider, see the *Publish-Subscribe Developer's Guide*.

### See Also

- [pub.publish:deliver](#)
- [pub.publish:publishAndWait](#)
- [pub.publish:envelope](#)

---

## pub.publish:publishAndWait

`WmPublic`. Broadcasts a request for a document from any client subscribed to a specific document type. The service waits for the reply or indicates that the `pub.publish:waitForReply` service should retrieve the reply later.

---

## Input Parameters

---

<i>documentTypeName</i>	<p><b>String</b> Fully qualified name of the publishable document type being published.</p> <p>If you intend to publish the document to a messaging provider (Broker or Universal Messaging), the publishable document type must be in sync with the associated provider definition. The provider definition is a representation of the publishable document type and its properties on the provider. In the case of Universal Messaging, this is a channel. For Broker, it is a Broker document type. If the document type and provider definition are not synchronized, publication might fail.</p>
<i>document</i>	<p><b>Document</b> Document (IData object) conforming to the document type in <i>documentTypeName</i>.</p>
<i>receiveDocumentTypeName</i>	<p><b>String</b> Optional. Fully qualified name of the document type expected as a reply. If no value is specified, the service uses the first reply document of any type it receives, as long as the value of <i>tag</i> in the reply document envelope matches the <i>tag</i> in the envelope of the published document. All other reply documents are discarded.</p>
<i>local</i>	<p><b>String</b> Optional. Flag specifying whether the document is to be published locally or to the webMethods Broker. Set to:</p> <ul style="list-style-type: none"><li>■ <code>true</code> to publish locally (to this Integration Server only).</li><li>■ <code>false</code> to publish to the webMethods Broker. This is the default.</li></ul> <p>The <i>local</i> parameter is applicable only when the publishable document type uses Broker as the messaging provider.</p> <p>The <i>local</i> parameter is applicable only when the publishable document type uses Broker as the messaging provider.</p> <p>If the publishable document type specifies the <code>IS_LOCAL_CONNECTION</code> alias, the document can be published locally only. Integration Server ignores the value of <i>local</i>.</p> <p>A publishable document type that specifies Universal Messaging as the messaging provider cannot be published locally. If the document type in <i>documentTypeName</i> is associated with a Universal Messaging connection alias, Integration Server ignores the value of <i>local</i>.</p>

<i>waitTime</i>	<b>String</b> Optional. Time to wait (in milliseconds) for the response to arrive. If no value is specified, the service waits indefinitely for a reply.
<i>async</i>	<p><b>String</b> Optional. Flag specifying whether this is an asynchronous or synchronous publish. Set to:</p> <ul style="list-style-type: none"> <li>■ <code>true</code> to indicate that this is an asynchronous request/reply. After publishing the document, the Integration Server executes the next step in the flow service immediately. The Integration Server does not wait for a reply before continuing service execution.</li> </ul> <p><b>Note:</b> To retrieve the reply to an asynchronous publish, invoke the <code>pub.publish:waitForReply</code> service.</p> <ul style="list-style-type: none"> <li>■ <code>false</code> to indicate that this is a synchronous request/reply. After publishing the document, the Integration Server waits for a reply before executing the next step in the flow service. This is the default.</li> </ul>

## Output Parameters

<i>receivedDocument</i>	<p><b>Document</b> Document (IData object) received as response. If no matching document is received within the wait time, this will be null.</p> <p><b>Important:</b> Integration Server treats all reply documents as volatile documents. If Integration Server shuts down before processing the reply document, the reply document is lost.</p>
<i>tag</i>	<p><b>String</b> Conditional. A unique identifier for a publish request. Integration Server uses the <i>tag</i> value to match the request document with its corresponding reply document.</p> <p>The service produces a <i>tag</i> output value only when the <i>async</i> field is set to <code>true</code>. The <i>tag</i> value is required input when using the <code>pub.publish:waitForReply</code> service to retrieve the reply.</p> <p><b>Note:</b> The <i>tag</i> output value is the same value that Integration Server places in the <i>tag</i> field of the request document's envelope.</p>
<i>status</i>	<p><b>String</b> Conditional. Status indicating whether the service was successful. Integration Server reports status only for locally published documents. A value of:</p>

- `success` indicates that the service executed successfully.

**Note:** If at least one subscribing trigger has room in its queue, the status is set to `success`.

- `requestTimedOut` indicates that the service timed out (that is, the `waitTime` specified in the service elapsed before Integration Server received a reply).
- `noSubscriber` indicates that Integration Server does not contain any triggers that subscribe to the document.
- `capacityExceeded` indicates that the document could not be placed in the queue of the subscribing trigger because the trigger queue is currently at its maximum capacity.

**Note:** Integration Server reports the `capacityExceeded` status only when the `watt.server.publish.local.rejectOOS` property is set to `true`.

## Usage Notes

You can use the `pub.publish:publishAndWait` service to publish a document to Universal Messaging, Broker, or locally within Integration Server.

Integration Server writes a message to the journal log whenever it rejects or discards a document.

You can use the `pub.publish:publishAndWait` service to initiate a request/reply. The publishing client broadcasts a request for information. Subscribers to the broadcast document compose and send a reply document that contains the information the publisher requested.

A single publish and wait request might receive many response documents. The Integration Server that made the publish and wait request uses only the first reply document it receives from the webMethods Broker. The Integration Server discards all other replies. *First* is arbitrarily defined. There is no guarantee provided for the order in which the messaging provider processes incoming replies. If you need a reply document from a specific client, use the `pub.publish:deliverAndWait` service instead.

The `pub.publish:publishAndWait` service can be useful in situations where multiple sources contain the response data. For example, suppose that an enterprise uses one application for managing customer data, another for storing master customer records, and a mainframe system for saving customer lists. Each of these applications could answer a published request for customer data. The publishing service will use the first reply document it receives.

A service can issue a publish and wait request in a synchronous or asynchronous manner.

- In a synchronous request/reply, the publishing flow service stops executing while it waits for a response. One of the following occurs:
  - If the service receives a reply before the *waitTime* elapses, the service resumes execution.
  - If the *waitTime* elapses before the service receives a reply, Integration Server ends the request, and the service returns a null document that indicates that the request timed out. Integration Server then executes the next step in the flow service.

If a reply document arrives after the flow service resumes execution, Integration Server rejects the document and creates a journal log message stating that the document was rejected because there was no thread waiting for the document.
- In an asynchronous request/reply, the publishing flow service continues executing the steps in the service after publishing the document. To retrieve the reply, the publishing flow service must invoke the `pub.publish:waitForReply` service. If the wait time elapses before the `pub.publish:waitForReply` service receives a document, the `pub.publish:waitForReply` service returns a null document indicating that the request timed out.

If a service contains multiple asynchronous publish and wait invocations, you can build the service such that the service publishes all the requests before collecting the replies. This approach can be more efficient than publishing a request, waiting for a reply, and then publishing the next request.

If you create a service that contains multiple asynchronous requests, make sure to link the *tag* output to another field in the pipeline. Each asynchronously published request produces a *tag* field in the pipeline. If the *tag* field is not linked to another field, the next asynchronously published request (that is, the next execution of the `pub.publish:publishAndWait` service) will overwrite the first *tag* value.

If the client side queue is disabled for the messaging connection alias and the document type has a storage type of guaranteed, the `pub.publish:waitForReply` service ends with a `ServiceException` if the messaging provider is not available. Make sure to code your publishing service to handle this situation.

The messaging connection alias assigned to a publishable document type determines the message provider to which the Integration Server publishes the document. Only subscribers to the document type on that messaging provider will receive the published document. The **Connection alias name** property for the document type specifies the message provider. If there is no specified message provider alias, Integration Server publishes the document to the provider in the default messaging connection alias.

If the publishable document type in *documentTypeName* is associated with a Universal Messaging connection alias, the **Enable Request/Reply Channel and Listener** check box must be selected for the alias. When this check box is selected, Integration Server ensures that a request/reply channel exists for the Universal Messaging connection alias on the Universal Messaging server and that Integration Server has a listener that subscribes to the alias-specific request/reply channel. If the check box is cleared, there will be no channel in which Universal Messaging can collect replies and no listener with which

Integration Server can retrieve replies. The `pub.publish:publishAndWait` service will end with a `ServiceException` if the **Enable Request/Reply Channel and Listener** check box is not selected for the Universal Messaging connection alias.

For more information about building a service that follows the request/reply model, see the *Publish-Subscribe Developer's Guide*.

### See Also

[pub.publish:waitForReply](#)  
[pub.publish:reply](#)  
[pub.publish:envelope](#)

---

## pub.publish:reply

WmPublic. Delivers a reply document to the requesting client.

### Input Parameters

---

<i>receivedDocumentEnvelope</i>	<b>Document</b> Optional. The envelope of the document to which you are replying. By default <i>receivedDocumentEnvelope</i> specifies the envelope of the document that triggered this service. In case of a join, <i>receivedDocumentEnvelope</i> will specify the last document that satisfied the join condition. However, you may specify the envelope of any published document to which you want to reply.
<i>documentTypeName</i>	<b>String</b> Fully qualified name of the publishable document type for the document that you are sending as a reply. Keep in mind that the publisher of the requesting document might be expecting a reply document that conforms to specific publishable document type.
<i>document</i>	<b>Document</b> The reply IData object. This document must conform to the publishable document type specified in <i>documentTypeName</i> .
<i>delayUntilServiceSuccess</i>	<b>String</b> Optional. Flag indicating whether Integration Server should publish the document when the <code>pub.publish:reply</code> service executes or after the top-level service successfully completes. If the top-level service fails, Integration Server will not publish the document.  Set to:

- `true` to delay publishing until after the top-level service executes successfully.
- `false` to publish the document when the publish service executes. This is the default.

## Output Parameters

---

None.

## Usage Notes

You can use the `pub.publish:reply` service to publish a document to Universal Messaging, Broker, or locally within Integration Server.

All reply documents are volatile documents. If the requesting Integration Server shuts down before processing the reply document, the reply document is lost.

Integration Server always encodes reply messages as `IData`.

If the `replyTo` envelope parameter is set in the request document, the reply document is delivered to that destination; otherwise, the reply document is sent to the publisher specified in the `pubId` parameter in the envelope of the request document.

This service correctly maps the required fields from the request document to the reply document.

A reply document can be a simple acknowledgment, or it can contain information asked for by the publisher of the request document.

Integration Server sends the reply document using the same connection that the trigger used to retrieve the request document. Integration Server ignores the messaging connection alias assigned to the publishable document type in `documentTypeName`.

Because Integration Server sends the reply document using the same connection that the trigger used to retrieve the request document, when Universal Messaging is the messaging provider, the publishable document types to which the trigger subscribes must be associated with a Universal Messaging connection alias for which the **Enable Request/Reply Channel and Listener** check box is selected.

If you are building a service to reply to documents that meet join conditions, keep the following in mind:

- **All (AND) join conditions.** If the replying service executes because two or more documents satisfied an All (AND) join condition, Integration Server uses the envelope of the last document that satisfied the join condition to determine where to send the reply document. If you want Integration Server to use the envelope of a different document, link the envelope of that document to `receivedDocumentEnvelope`. If you want to reply to all documents received as part of an All (AND) join, invoke `pub.publish:reply` once for each document received and map the envelope from the received document to `receivedDocumentEnvelope` for each call.

- **Any (OR) or Only one (XOR) join conditions.** If the replying service executes because a document satisfied an Any (OR) or Only one (XOR) join condition, do not map or assign a value to `receivedDocumentEnvelope`. It is impossible to know which document in the Any (OR) or Only one (XOR) join will be received first. For example, suppose that an Only one (XOR) join condition specified document types A and B. Integration Server uses the envelope of the document it received first as the `receivedDocumentEnvelope` value. If you map the envelope of document A to `receivedDocumentEnvelope`, but Integration Server receives document B first, your replying service will fail.

Services that publish or deliver a document and wait for a reply can specify a publishable document type to which reply documents must conform. If the reply document is not of the type specified in the `receiveDocumentTypeName` parameter of the [pub.publish:publishAndWait](#) or [pub.publish:deliverAndWait](#) service, the publishing service will wait forever for a reply. Work closely with the developer of the publishing service to make sure that your reply document is an instance of the correct publishable document type.

For more information about building a reply service, see the *Publish-Subscribe Developer's Guide*.

#### See Also

[pub.publish:deliverAndWait](#)  
[pub.publish:publishAndWait](#)  
[pub.publish:envelope](#)

---

## pub.publish:syncToBroker

WmPublic. *Deprecated* - Replaced by [pub.publish:syncToProvider](#).

Synchronizes one or more publishable document types with the corresponding Broker document types by pushing the publishable document types to webMethods Broker.

**Note:** This service can be used to synchronize publishable document types for which the **Connection alias name** property is a Broker connection alias only.

---

### Input Parameters

*documentTypes*      **String List** Fully qualified names of the publishable document types to synchronize with their associated provider definition.

---

### Output Parameters

*successfulPDTs*      **String List** Conditional. Fully qualified names of the publishable document types that Integration Server updated successfully on the provider. The service only produces this

output parameter if one or more publishable document types updated successfully.

*unsuccessfulPDTs*

**String List** Conditional. Fully qualified names of the publishable document types that Integration Server did not update successfully on the provider. The service only produces this output parameter if one or more publishable document types did not update successfully.

*errors*

**Document List** Conditional. Lists any errors that occurred during document type synchronization.

*warnings*

**Document List** Conditional. Lists any warnings that occurred during document type synchronization.

### Usage Notes

You can use `pub.publish:syncToBroker` as a start up service for a package to avoid using Software AG Designer to synchronize document types.

You can use the `pub.publish:syncToBroker` service with Broker only. The use of `pub.publish:syncToBroker` with Universal Messaging is not supported. Use [pub.publish:syncToProvider](#) or Designer to synchronize publishable document types with corresponding provider definitions (channels) on Universal Messaging.

The **Connection alias name** property for a document type specifies the messaging connection alias assigned to the document type which indicates the provider to which a document type will be pushed.

If the *documentTypes* parameter contains document types that are not publishable or do not exist, Integration Server lists the errors in the *errors* parameter.

---

## pub.publish:syncToProvider

WmPublic. Synchronizes one or more publishable document types with their associated provider definition by pushing the publishable document types to the associated message provider.

### Input Parameters

---

*documentTypes*

**String List** Fully qualified names of the publishable document types to synchronize with their associated provider definition.

---

## Output Parameters

---

<i>successfulPDTs</i>	<b>String List</b> Conditional. Fully qualified names of the publishable document types that Integration Server updated successfully on the provider. The service only produces this output parameter if one or more publishable document types updated successfully.
<i>unsuccessfulPDTs</i>	<b>String List</b> Conditional. Fully qualified names of the publishable document types that Integration Server did not update successfully on the provider. The service only produces this output parameter if one or more publishable document types did not update successfully.
<i>errors</i>	<b>Document List</b> Conditional. Lists any errors that occurred during document type synchronization.
<i>warnings</i>	<b>Document List</b> Conditional. Lists any warnings that occurred during document type synchronization.

## Usage Notes

You can use `pub.publish:syncToProvider` as a start up service for a package to avoid using Software AG Designer to synchronize document types.

The **Connection alias name** property for a document type specifies the messaging connection alias assigned to the document type which indicates the provider to which a document type will be pushed.

If the *documentTypes* parameter contains document types that are not publishable or do not exist, Integration Server lists the errors in the *errors* parameter.

The `pub.publish:syncToProvider` service replaces `pub.publish:syncToBroker`, which has been deprecated.

---

## pub.publish:waitForReply

WmPublic. Retrieves the reply for an asynchronous request. If a reply is not available, Integration Server continues to wait for the document until the time specified in the *waitTime* parameter of the `pub.publish:deliverAndWait` service or `pub.publish:publishAndWait` service elapses.

---

## Input Parameters

---

<i>tag</i>	<b>String</b> A unique identifier for the publish request for which you are retrieving a reply. Integration Server uses the <i>tag</i> value
------------	--

to match the request document with its corresponding reply document.

## Output Parameters

---

*receivedDocument* **Document** Document (IData object) received as the reply to the request. If the request expires (that is, the *waitTime* elapses) before Integration Server receives the reply document, the *receivedDocument* field contains a null document.

**Important** Integration Server treats all reply documents as volatile documents. If Integration Server shuts down before processing the reply document, the reply document is lost.

## Usage Notes

The *waitTime* value of the publishing service specifies how long Integration Server will keep the request open while waiting for a reply. When building an asynchronous request/reply service, keep the following information about the *waitTime* in mind:

- The waiting interval for the reply document starts when Integration Server executes the request service ([pub.publish:deliverAndWait](#) or [pub.publish:publishAndWait](#)). The execution of the [pub.publish:waitForReply](#) service does not affect the *waitTime* interval.
- If the *waitTime* interval elapses before the [pub.publish:waitForReply](#) service executes, the service immediately returns a null document which indicates that the wait time has expired.
- If Integration Server has not received the reply when the [pub.publish:waitForReply](#) service executes, the service waits the remainder of the *waitTime* interval. If Integration Server does not receive a reply by the time the *waitTime* interval elapses, the request completes. The service returns a null document which indicates that the wait time has expired.
- If the reply document arrives after the *waitTime* interval elapses, Integration Server rejects the document because the request is closed.

A single publish and wait request might receive many response documents. The Integration Server that made the publish and wait request uses only the first reply document it receives. Integration Server discards all other replies. *First* is arbitrarily defined. There is no guarantee provided for the order in which the messaging provider processes incoming replies. If you need a reply document from a specific recipient, use the [pub.publish:deliverAndWait](#) service instead.

For more information about building an asynchronous request/reply service, see the *Publish-Subscribe Developer's Guide*.

## See Also

[pub.publish:deliverAndWait](#)  
[pub.publish:publishAndWait](#)

---

## pub.publish.notification:error

WmPublic. Publishable document type that defines the document that Integration Server generates and delivers when a trigger encounters an error or exception condition during processing.

**Note:** Integration Server does not generate and return an error message when the trigger received a document from Universal Messaging.

Integration Server generates an error document if the trigger service cannot successfully process a document for one of the following reasons:

- The trigger service encounters an exception condition (that is not an `ISRuntimeException`) during execution.
- Integration Server makes the maximum number of attempts to re-execute the trigger service and the service still fails because of a transient error condition.
- Some other system exception occurred.

**Note:** Integration Server does not generate an error document if the subscribing trigger is part of a disabled process model version because the trigger service associated with a disabled process model version never executes.

Integration Server delivers the error document to the client ID specified in the `errorsTo` field contained in the received document's envelope. If the `errorsTo` field is empty, the Integration Server delivers the error document to the original document's publisher (as specified in the `pubId` envelope field). The error document notifies the publisher or other designated recipient that the subscriber cannot process the document successfully.

**Note:** If a trigger service cannot process a locally published document successfully, Integration Server produces and delivers an error document only if the Integration Server is connected to a webMethods Broker.

---

## Parameters

<i>adapterType</i>	<b>String</b> Optional. The resource producing the error. Integration Server sets the value of this field to <code>Integration Server</code> .
<i>errorCategory</i>	<b>String</b> Optional. Type of exception. Integration Server sets the value of this field to <code>Application</code> .

<i>errorText</i>	<b>String</b> Optional. Exception text message. At Dispatcher debug level 9, a stack trace of the exception will also be returned.
<i>eventID</i>	<b>java.lang.Long</b> Optional. The event ID of the document that caused this exception. If the trigger service executed because a document satisfied a join condition, then the <i>eventID</i> is the event ID of the last document that satisfied the condition.
<i>_env</i>	<b>Document</b> Optional. A document reference to the <a href="#">pub.publish:envelope</a> document type.

### Usage Notes

The client to which Integration Server delivers the error document needs to subscribe to the `pub.publish.notification:error` document type. If the client does not have a trigger that subscribes to this document type, the client will never receive or process the error document. If the client receiving the error document is an Integration Server, it generates the message `[ISS.0098.0024V2] No trigger available for incoming Document pub.publish.notification:error`.

### See Also

[pub.publish:envelope](#)



# 24 Remote Folder

---

■ Summary of Elements in this Folder .....	628
--	-----

You use the elements in the remote folder to invoke services on other webMethods Integration Server.

You can also use remote services for guaranteed delivery transactions. For more information about guaranteed delivery transactions, see the *Guaranteed Delivery Developer's Guide* and *webMethods Integration Server Administrator's Guide*.

## Summary of Elements in this Folder

The following elements are available in this folder:

Element	Package and Description
<code>pub.remote:invoke</code>	WmPublic. Invokes a service on a remote webMethods Integration Server.
<code>pub.remote.gd:end</code>	WmPublic. Ends a guaranteed delivery transaction.
<code>pub.remote.gd:getStatus</code>	WmPublic. Returns the status of the guaranteed delivery transaction.
<code>pub.remote.gd:invoke</code>	WmPublic. Invokes the service for a guaranteed delivery transaction by making a synchronous call to a remote webMethods Integration Server.
<code>pub.remote.gd:restart</code>	WmPublic. Restarts an expired guaranteed delivery transaction.
<code>pub.remote.gd:retrieve</code>	WmPublic. Retrieves the results of a guaranteed delivery transaction submitted asynchronously or synchronously to a remote webMethods Integration Server.
<code>pub.remote.gd:send</code>	WmPublic. Makes a guaranteed one-way call (fire-and-forget) to the webMethods Integration Server to invoke a service for which no output is needed or expected.
<code>pub.remote.gd:start</code>	WmPublic. Starts a guaranteed delivery transaction.
<code>pub.remote.gd:submit</code>	WmPublic. Invokes a service for a guaranteed delivery transaction by making an asynchronous call to a remote webMethods Integration Server.

---

## pub.remote:invoke

WmPublic. Invokes a service on a remote webMethods Integration Server.

The remote server is identified by an alias, which is configured on the **Remote Servers** tab in the Integration Server Administrator. Connection and authentication to the remote server is managed transparently to the caller of this service.

All current pipeline inputs are passed to the remote service. To improve performance and minimize the amount of data sent over the wire, scope the pipeline to a separate document or drop unneeded fields before invoking this service. The same advice applies to the output values of the remote service because all values returned from the service are sent over the wire in response to the caller.

---

### Input Parameters

*\$alias*

**String** Name of the target server on which to invoke the specified service. This name and its associated connection attributes are defined on the **Create Remote Server Alias** screen in the Integration Server Administrator.

**Note:** If you protect the alias using an Access Control List, the user invoking `invoke` must be a member of this list or the invocation will fail.

*\$service*

**String** Fully qualified name of the service to invoke on the remote server, in the format *folderName.folderName:serviceName* (for example: `wm.server:ping`).

*\$scope*

**String** Flag that specifies how the session to the remote server should be managed. Set to:

- `SESSION` to store the remote session in the current user session. This is the default.

Further calls by the same user to `pub.remote:invoke` for the same server alias reuse the existing remote session with the server.

Stateful interactions with the remote server are maintained and protected inside the current user's session.

When the current user disconnects, the remote session expires, or the local server is shut down, the remote session is automatically disconnected.

- `GLOBAL` to store the remote session in a shared pool of sessions. If another user invokes a service on the same remote server with `GLOBAL` scope, the session will be reused.

Stateful interactions with the remote server could be destroyed by other users' invocations.

When the remote session expires due to inactivity or the local server is shut down, the remote session is automatically disconnected.

*\$close*

**String** Optional. Flag to indicate whether Integration Server closes the connection to the remote server after the service invocation or keeps the connection open until it times out. Set to:

- `true` to close the connection to the remote server immediately after the service invocation.
- `false` to keep the connection open until it times out. This is the default.

**Note:** If the `watt.server.trigger.reuseSession` server configuration parameter is set to `true`, Integration Server expects trigger sessions to be reused and the connection to the remote server to remain open. Hence, if `watt.server.trigger.reuseSession` parameter is set to `true`, Integration Server will ignore the value of the *\$close* parameter when a trigger service for a webMethods messaging trigger executes the `pub.remote.invoke` service.

*\$clusterRetry*

**String** Optional. Flag to indicate whether Integration Servers should retry a failed connection request on other Integration Servers in the cluster. Set to:

- `true` to retry a request automatically on other Integration Servers in the cluster if the initial attempt to connect to a remote Integration Server fails. Integration Server will attempt to connect to each Integration Servers in the cluster until the connection is made or all Integration Servers have been tried with no success. If the service cannot connect to another Integration Server in the cluster, the service tries to connect to the retry server specified in the alias definition for the remote server.
- `false` to issue an error if the attempt to connect to the remote Integration Server fails. If the alias definition for the remote server specifies a retry server, the service tries to connect to that server.

**Note:** Once a connection to a remote server has been established, that connection is cached and reused. The *\$clusterRetry* setting is established when the connection is first created and used. Subsequent

invokes to the same remote server will not change the `$clusterRetry` setting, even if a different value is passed in the pipeline. Client applications must determine whether or not they want cluster retries before establishing the connection.

---

### Output Parameters

Returns the output of the invoked service. The output signature matches the output signature of the invoked service.

### Usage Notes

If `pub.remote:invoke` does not receive a response within the timeout period specified in the server's `watt.net.timeout` parameter, it will throw an exception. For information about the `watt.net.timeout` parameter, see *webMethods Integration Server Administrator's Guide*.

---

## pub.remote.gd:end

WmPublic. Ends a guaranteed delivery transaction.

---

### Input Parameters

*tid*                      **String** Transaction ID of the transaction you want to end.

---

### Output Parameters

None.

### Usage Notes

This service is used to eliminate a guaranteed delivery transaction from the jobstore.

---

## pub.remote.gd:getStatus

WmPublic. Returns the status of the guaranteed delivery transaction.

---

### Input Parameters

*tid*                      **String** Transaction identification number.

---

## Output Parameters

---

<i>status</i>	<p><b>String</b> Current status of the transaction. <i>status</i> can have one of the following values:</p> <ul style="list-style-type: none"><li>■ <b>NEW</b> indicates that the transaction is new.</li><li>■ <b>PENDING</b> indicates that the transaction is pending.</li><li>■ <b>DONE</b> indicates that the transaction is completed.</li><li>■ <b>FAILED</b> indicates that the transaction expired because the time-to-live or the retry limit has been exceeded.</li><li>■ <b>UNKNOWN</b> indicates that the transaction identification number in <i>tid</i> is not recognized.</li></ul>
---------------	---

### Usage Notes

Use the [pub.remote.gd:restart](#) service to restart a FAILED (expired) guaranteed delivery transaction.

---

## pub.remote.gd:invoke

WmPublic. Invokes the service for a guaranteed delivery transaction by making a synchronous call to a remote webMethods Integration Server.

---

### Input Parameters

---

<i>service</i>	<p><b>String</b> Name of the service to be run on the remote webMethods Integration Server.</p>
<i>tid</i>	<p><b>String</b> Transaction identification number for the service.</p>
<i>inputs</i>	<p><b>Document</b> Optional. Document (IData object) containing the inputs for the service.</p>

---

### Output Parameters

---

<i>results</i>	<p><b>Document</b> Conditional. Document (IData object) containing the pipeline as it exists after the service is invoked.</p>
----------------	--

### Usage Notes

To use an asynchronous call to the server to invoke a service for a guaranteed delivery transaction, use the [pub.remote.gd:submit](#) service.

If the remote server does not respond within the timeout limit specified in this server's `watt.net.timeout` setting, the Integration Server treats it as a failed attempt and retries the request.

---

## pub.remote.gd:restart

WmPublic. Restarts an expired guaranteed delivery transaction.

### Input Parameters

---

*tid*                      **String** Transaction identification number for the guaranteed delivery transaction you want to restart.

### Output Parameters

---

None.

### Usage Notes

If a guaranteed delivery transaction failed because of server or network failure, use this service to restart the transaction without resubmitting it.

---

## pub.remote.gd:retrieve

WmPublic. Retrieves the results of a guaranteed delivery transaction submitted asynchronously or synchronously to a remote webMethods Integration Server.

### Input Parameters

---

*tid*                      **String** Transaction identification number.

*block*                    **String** Optional. Flag that specifies whether to block or poll for the results of the transaction. Set to:

- `true` to wait until the invoked service completes before retrieving results. This is also known as blocking mode. This is the default.

- `false` to retrieve the results immediately, whether or not the invoked service is completed. This is also known as polling mode.

---

### Output Parameters

*results*                      **Document** Conditional. Document (IData object) containing the results of the service in the guaranteed delivery transaction.

### Usage Notes

If *block* is `false`, and the results of the transaction are still pending when this service executes, the results are returned as null.

---

## pub.remote.gd:send

WmPublic. Makes a guaranteed one-way call (fire-and-forget) to the webMethods Integration Server to invoke a service for which no output is needed or expected.

---

### Input Parameters

*service*                      **String** Service to be run on the remote Integration Server.

*tid*                              **String** Transaction identification number for the service.

*inputs*                        **Document** Optional. Document (IData object) containing the inputs for the service.

---

### Output Parameters

None.

### Usage Notes

The results of the service specified in *service* cannot be retrieved. However, errors that occur will be logged when the guaranteed delivery transaction ends.

Use the `pub.remote.gd:send` service to invoke a service remotely only if you want to run a guaranteed delivery transaction and are not concerned about the results of the invoked service. After `pub.remote.gd:send` completes the call, the service ends the transaction; therefore, you do not need to use the [pub.remote.gd:end](#) service to end the transaction.

---

## pub.remote.gd:start

WmPublic. Starts a guaranteed delivery transaction.

### Input Parameters

---

<i>alias</i>	<b>String</b> Name of the webMethods Integration Server on which you want to invoke a guaranteed delivery transaction. This name and its associated connection attributes are defined on the <b>Remote Servers</b> tab of the Integration Server Administrator.
<i>tll</i>	<b>String</b> Optional. Transaction time-to-live measured in minutes. The transaction expires when <i>tll</i> is exceeded.  Default is the value set in the <code>watt.tx.defaultTTLmins</code> property or, if the property is not set, 30 minutes.
<i>retries</i>	<b>String</b> Optional. Maximum number of times to retry the transaction. Default is 0 (no retry limit).
<i>followtid</i>	<b>String</b> Optional. Identification number of the transaction you want this guaranteed delivery transaction to follow. The current transaction executes only after the transaction indicated by <i>followtid</i> completes.

### Output Parameters

---

<i>tid</i>	<b>String</b> Transaction identification number.
------------	--

---

## pub.remote.gd:submit

WmPublic. Invokes a service for a guaranteed delivery transaction by making an asynchronous call to a remote webMethods Integration Server.

### Input Parameters

---

<i>service</i>	<b>String</b> Service to be run on the remote webMethods Integration Server.
<i>tid</i>	<b>String</b> Transaction identification number for the service.

*inputs*                      **Document** Optional. Document (IData object) containing the inputs for the service.

### **Output Parameters**

---

None.

### **Usage Notes**

To remove the transaction from the remote webMethods Integration Server, use the [pub.remote.gd:end](#) service.

To use a synchronous call to invoke the service, use the [pub.remote.gd:invoke](#) service.

# 25 Replicator Folder

---

■ Summary of Elements in this Folder .....	638
--	-----

You use the elements in the replicator folder to replicate packages across webMethods Integration Servers. This folder contains services that you can use to push packages from your webMethods Integration Servers to a subscriber's server. It also contains services that you can use to pull packages from a publisher's server to your webMethods Integration Server.

## Summary of Elements in this Folder

The following elements are available in this folder:

Element	Package and Description
<a href="#">pub.replicator:addReleaseRegistryEntry</a>	WmPublic. Adds an entry to the webMethods Integration Server's Package Release Registry.
<a href="#">pub.replicator:deleteReleaseRegistryEntry</a>	WmPublic. Deletes an entry from the webMethods Integration Server's Package Release Registry.
<a href="#">pub.replicator:distributeViaFTP</a>	WmPublic. Allows a publisher to send a package to a subscriber via FTP or allows a subscriber to retrieve a package from a publisher via FTP.
<a href="#">pub.replicator:distributeViaSvcPull</a>	WmPublic. Pulls a specified package release from a publisher's server.
<a href="#">pub.replicator:distributeViaSvcPush</a>	WmPublic. Pushes a package from your server to a list of subscribers (other webMethods Integration Servers).
<a href="#">pub.replicator:generateReplicationEvent</a>	WmPublic. Generates a replication event.
<a href="#">pub.replicator:getLocalReleasedList</a>	WmPublic. Returns all entries in your webMethods Integration Server's Package Release Registry.
<a href="#">pub.replicator:getRemoteReleasedList</a>	WmPublic. Queries the publisher for released packages.
<a href="#">pub.replicator:notifyPackageRelease</a>	WmPublic. Sends an e-mail message to subscribers who have said that they want

Element	Package and Description
	to be notified when a new release becomes available.
<a href="#">pub.replicator:packageCreation</a>	WmPublic. Creates a distribution file (a zip file) for the package.

## pub.replicator:addReleaseRegistryEntry

WmPublic. Adds an entry to the webMethods Integration Server's Package Release Registry.

### Input Parameters

<i>package</i>	<b>String</b> Name of the package. The service confirms that this package exists on the server before adding an entry to the Package Release Registry.
<i>name</i>	<b>String</b> Name of the release. This name could be different from the name of the package.
<i>version</i>	<b>String</b> Version number of the release, in the format <i>##</i> or <i>##.##</i> (for example, <i>1.2</i> or <i>1.2.1</i> ).
<i>build</i>	<b>String</b> Build number of the release (for example, <i>12, 530</i> ).
<i>patchNums</i>	<b>String</b> One or more comma-separated patch numbers included in this release.
<i>JVMVersion</i>	<b>String</b> Minimum JVM version number that this release requires.
<i>description</i>	<b>String</b> Brief description of this release. You may want to use this parameter to summarize the nature and purpose of the release.

### Output Parameters

*packages* **Document List** Entries in the server's Package Release Registry.

<u>Key</u>	<u>Description</u>
------------	--------------------

---

<i>name</i>	<b>String</b> Name of the release.
<i>version</i>	<b>String</b> Version number of the release, in the format <i>##</i> or <i>##.##</i> (for example, 1.2 or 1.2.1).
<i>build</i>	<b>String</b> Conditional. Build number of the release (for example, 12, 530).
<i>patch_nums</i>	<b>String</b> Conditional. Comma-separated list of patch numbers included in this release.
<i>time</i>	<b>String</b> Time when the package was released.
<i>jvm_version</i>	<b>String</b> Minimum JVM version number that the release requires.
<i>description</i>	<b>String</b> Conditional. Brief description of this release.
<i>source_server_version</i>	<b>String</b> Version number of Integration Server that released the package.

### Usage Notes

Before using this service, use [pub.replicator:packageCreation](#) to create a package zip file in the server's outbound directory. When you use `addReleaseRegistryEntry` to add an entry to the Package Release Registry, the package name you specify in *package* should match the package name you specified in [pub.replicator:packageCreation](#).

---

## pub.replicator:deleteReleaseRegistryEntry

WmPublic. Deletes an entry from the webMethods Integration Server's Package Release Registry.

### Input Parameters

---

<i>packageName</i>	<b>String</b> Name of the release that you want to delete.
--------------------	--

---

## Output Parameters

---

*packages*

**Document List** Entries that remain in the server's Package Release Registry.

<u>Key</u>	<u>Description</u>
<i>name</i>	<b>String</b> Name of the release.
<i>version</i>	<b>String</b> Version number of the release, in the format <i>##</i> or <i>###</i> (for example, 1.2 or 1.2.1).
<i>build</i>	<b>String</b> Conditional. Build number of the release (for example, 12, 530).
<i>patch_nums</i>	<b>String</b> Conditional. Comma-separated list of patch numbers included in this release.
<i>time</i>	<b>String</b> Time when the package was released.
<i>jvm_version</i>	<b>String</b> Minimum JVM version number that the release requires.
<i>description</i>	<b>String</b> Conditional. Brief description of the release.
<i>source_server_version</i>	<b>String</b> Version number of Integration Server that released the package.

---

## pub.replicator:distributeViaFTP

WmPublic. Allows a publisher to send a package to a subscriber via FTP or allows a subscriber to retrieve a package from a publisher via FTP.

### Input Parameters

---

*packageName*

**String** Name of the released package.

<i>action</i>	<b>String</b> Flag that specifies whether you want to send (put) a package to another Integration Server or whether you want to retrieve (get) a package from another Integration Server.  Set to: <ul style="list-style-type: none"><li>■ <code>get</code> to retrieve a package from the publisher's server. This is the default.</li><li>■ <code>put</code> to send a package to a subscriber's server.</li></ul>
<i>serverhost</i>	<b>String</b> Host name or IP address of the remote Integration Server.
<i>serverport</i>	<b>String</b> Number of the FTP port on the remote Integration Server.
<i>username</i>	<b>String</b> User name that your server will use to log on to the remote Integration Server.
<i>password</i>	<b>String</b> Password that your server will use to log on to the remote Integration Server.

#### Output Parameters

---

None.

---

## pub.replicator:distributeViaSvcPull

WmPublic. Pulls a specified package release from a publisher's server.

#### Input Parameters

---

<i>packageName</i>	<b>String</b> Name of the release.
<i>publisher</i>	<b>String</b> Alias of the publisher's server.

#### Output Parameters

---

None.

---

## pub.replicator:distributeViaSvcPush

WmPublic. Pushes a package from your server to a list of subscribers (other webMethods Integration Servers).

### Input Parameters

---

*packageName*            **String** The name of the release.

*subscriber*            **String List** List of the subscriber's host names or IP addresses.

### Output Parameters

---

None.

---

## pub.replicator:generateReplicationEvent

WmPublic. Generates a replication event.

You might invoke this service in conjunction with other services to make the package replication process generate an event. The replication event handler would listen for this event and perform some prescribed action that you have specified.

### Input Parameters

---

*packageName*            **String** Name of the package.

*action*                **String** User-defined string that describes the replication event, such as "pulled" or "pushed."

### Output Parameters

---

None.

---

## pub.replicator:getLocalReleasedList

WmPublic. Returns all entries in your webMethods Integration Server's Package Release Registry.

### Input Parameters

---

None.

---

## Output Parameters

---

*packages*

**Document List** Entries in the server's Package Release Registry.

<b>Key</b>	<b>Description</b>
<i>name</i>	<b>String</b> Name of the release.
<i>version</i>	<b>String</b> Version number of the release, in the format <i>##</i> or <i>##.##</i> (for example, <i>1.2</i> or <i>1.2.1</i> ).
<i>build</i>	<b>String</b> Conditional. Build number of the release (for example, <i>12, 530</i> ).
<i>patch_nums</i>	<b>String</b> Conditional. Comma-separated list of the patch numbers included in this release.
<i>time</i>	<b>String</b> Time when the package was released.
<i>jvm_version</i>	<b>String</b> Minimum JVM version number that the release requires.
<i>description</i>	<b>String</b> Conditional. Brief description of the release.
<i>source_server_version</i>	<b>String</b> Version number of webMethods Integration Server that released the package.

---

## pub.replicator:getRemoteReleasedList

WmPublic. Queries the publisher for released packages.

This service gets a list of released packages to which your server subscribes. You can use the list to find out if any new packages, or newer versions of existing packages, have been released.

---

## Input Parameters

---

*publisher*      **String** Alias of the publishing server.

## Output Parameters

---

*packages*      **Document List** List of released packages on the publishing server to which you subscribe.

<b>Key</b>	<b>Description</b>
<i>name</i>	<b>String</b> Name of the release.
<i>version</i>	<b>String</b> Version number of the release, in the format <code>##</code> or <code>###</code> (for example, <code>1.2</code> or <code>1.2.1</code> ).
<i>build</i>	<b>String</b> Conditional. Build number of the release (for example, <code>12, 530</code> ).
<i>patch_nums</i>	<b>String</b> Conditional. Comma-separated list of the patch numbers included in this release.
<i>time</i>	<b>String</b> Time when the package was released.
<i>jvm_version</i>	<b>String</b> Minimum JVM version number that the release requires.
<i>description</i>	<b>String</b> Conditional. Brief description of the release.
<i>source_server_version</i>	<b>String</b> Version number of webMethods Integration Server that released the package.

---

## pub.replicator:notifyPackageRelease

WmPublic. Sends an e-mail message to subscribers who have said that they want to be notified when a new release becomes available.

---

### Input Parameters

---

*packageName*            **String** Name of the release.

### Output Parameters

---

None.

---

## pub.replicator:packageCreation

WmPublic. Creates a distribution file (a zip file) for the package.

---

### Input Parameters

---

*package*                **String** Name of the package.

*name*                    **String** Name of the release.

*version*                **String** Version number of the release, in the format `##` or `###`  
(for example, `1.2` or `1.2.1`).

*build*                   **String** Build number of the release (for example, `12, 530`).

*patchNums*            **String** Comma-separated list of patch numbers included in the  
release.

*targetPkgVersion*    **String** Version number of the target package. To prevent  
the installation program from overwriting an existing  
(higher) version of the package, this field is checked when the  
subscriber installs this package over an existing package.

*targetServerVersion* **String** Version number of the webMethods Integration Server  
that this release requires.

*JVMVersion*           **String** Minimum JVM version number that this release  
requires.

*description*          **String** Brief description of this release. You might use this  
parameter to summarize the nature and purpose of the release.

*type*                   **String** Flag indicating the type of release. Set to:  
■ `full` to indicate a full package. This is the default.

- `partial` to indicate a patch or an update for the package.

*filter*

**String** Flag that specifies whether all files are to be included in the distribution file or only selected files.

If only selected files are to be included, use this parameter in conjunction with *fileList* to specify which files to include.

Set to:

- `includeall` to include all the files in the distribution file. This is the default.
- `include` to include selected files in the distribution file.
- `exclude` to include all except selected files in the distribution file.

*fileList*

**String List** Names of files to include or exclude from the distribution file, depending on the value of *filter*.

*fileNamePattern*

**String** Pattern string that specifies the names of files to be included in the distribution file. The asterisk (\*) is the only wildcard character allowed in a pattern string. All other characters are treated literally (for example, `*.java`, `*.dsp`).

*filesToDeleteList*

**String List** Optional. The names of files that will be deleted from the target package when the subscribing server installs the package created by this service.

## Output Parameters

---

*\$result*

**String** Conditional. If the distribution file is created successfully, this parameter contains the value `OK`. If the distribution file was not created successfully, this parameter is not present in the output signature and the service throws an exception.

## Usage Notes

After you use `packageCreation` to create the package, use [pub.replicator:addReleaseRegistryEntry](#) to add an entry to the Package Release Registry. The package name you specify in `packageCreation` should match the package name you specify in [pub.replicator:addReleaseRegistryEntry](#).



# 26 Report Folder

---

■ Summary of Elements in this Folder .....	650
--	-----

You use the elements in the report folder to apply an output template to an IData object. The services can be used in order to generate any type of dynamic XML, EDI, or HTML document.

## Summary of Elements in this Folder

The following elements are available in this folder:

Element	Package and Description
<a href="#">pub.report:runFileTemplate</a>	WmPublic. Applies a template file to a document (IData object).
<a href="#">pub.report:runFileTemplateOnPipe</a>	WmPublic. Applies a template to the pipeline.
<a href="#">pub.report:runStringTemplate</a>	WmPublic. Applies an output template to a specified document (IData object).
<a href="#">pub.report:runStringTemplateOnPipe</a>	WmPublic. Applies a template to the pipeline.
<a href="#">pub.report:runTemplate</a>	WmPublic. Applies a template in a file to a specified document (IData object).
<a href="#">pub.report:runTemplateOnPipe</a>	WmPublic. Applies a template in a file to the pipeline.

## pub.report:runFileTemplate

WmPublic. Applies a template file to a document (IData object).

### Input Parameters

<i>\$template</i>	<b>java.io.File</b> Template file.
<i>\$values</i>	<b>Document</b> Document (IData object) to bind against <i>\$template</i> .
<i>fileEncoding</i>	<b>String</b> Optional. The encoding of the template file. If <i>fileEncoding</i> is not specified, the default file encoding specified in the <code>watt.server.netEncoding</code> server parameter or the system file encoding will be used. Examples: SJIS, ASCII, ISO8859_1.

---

## Output Parameters

*\$txt*                      **String** Results from applying *\$template* to *\$values*.

### Usage Notes

If a template is not available in a templates directory of any of the packages on the server, you can use this service by passing in a File object representing the template.

---

## pub.report:runFileTemplateOnPipe

WmPublic. Applies a template to the pipeline.

### Input Parameters

*\$template*                      **java.io.File** Template file.

*fileEncoding*                      **String** Optional. The encoding of the template file. If *fileEncoding* is not specified, the default file encoding specified in the `watt.server.netEncoding` server parameter or the system file encoding will be used. Examples: SJIS, ASCII, ISO8859\_1.

### Output Parameters

*\$txt*                              **String** Results from applying *\$template* to the pipeline.

### Usage Notes

If a template is not available in a templates directory of any of the packages on the server, you can use this service to pass a File object representing the template file.

---

## pub.report:runStringTemplate

WmPublic. Applies an output template to a specified document (IData object).

### Input Parameters

*\$template*                      **String** Template to apply.

*\$values*                              **Document** Document (IData object) to bind against *\$template*.

---

## Output Parameters

---

*\$txt*                      **String** Results from applying *\$template* to *\$values* .

### Usage Notes

This service is typically invoked from other services that already have a template in a String object and an IData object that will be used to bind against the template.

---

## pub.report:runStringTemplateOnPipe

WmPublic. Applies a template to the pipeline.

---

### Input Parameters

---

*\$template*                      **String** Template to apply to pipeline.

---

### Output Parameters

---

*\$txt*                              **String** Result from applying *\$template* to the pipeline.

### Usage Notes

This service is typically invoked from other services that already have a template in a String object and need the template to bind against the pipeline.

---

## pub.report:runTemplate

WmPublic. Applies a template in a file to a specified document (IData object).

---

### Input Parameters

---

*\$template*                      **String** Name of the template file (for example, *mytemp.html* or *mytemp.xml*).

*\$package*                      **String** Name of the package where the template resides (for example, *Default*).

*\$values*                        **Document** Document (IData object) to bind against *\$template* .

*fileEncoding*      **String** Optional. The encoding of the template file. If *fileEncoding* is not specified, the default file encoding specified in the `watt.server.netEncoding` server parameter or the system file encoding will be used. Examples: SJIS, ASCII, ISO8859\_1.

---

### Output Parameters

*\$txt*      **String** Result from applying the template to *\$values* .

### Usage Notes

The service locates the output template by its file name and the name of the package in which it resides. To apply a template that resides in `IntegrationServer_directory \packages \Default \templates \mytemp.xml`, invoke the service with the following values.

*\$template* : mytemp.xml

*\$package* : Default

---

## pub.report:runTemplateOnPipe

WmPublic. Applies a template in a file to the pipeline.

---

### Input Parameters

*\$template*      **String** Name of template file (for example, `mytemp.html` or `mytemp.xml`).

*\$package*      **String** Name of the package in which the template resides (for example, `Default`).

*fileEncoding*      **String** Optional. The encoding of the template file. If *fileEncoding* is not specified, the default file encoding specified in the `watt.sever.netEncoding` server parameter or the system file encoding will be used. Examples: SJIS, ASCII, ISO8859\_1.

---

### Output Parameters

*\$txt*      **String** Results from applying the template file to the pipeline.

### Usage Notes

The service locates the output template by its file name and the name of the package in which it resides. For example, to apply a template that resides in

*IntegrationServer\_directory* \packages\Default\templates\mytemp.xml, invoke the service with the following values.

*\$template*: mytemp.xml

*\$package*: Default

# 27 Scheduler Folder

---

■ Summary of Elements in this Folder .....	656
--	-----

You use the elements in the scheduler folder to execute services at the times you specify. Services that you schedule are referred to as *user tasks* or just *tasks*. The Scheduler feature on the webMethods Integration Server handles execution of the tasks.

## Summary of Elements in this Folder

The following elements are available in this folder:

Element	Package and Description
<a href="#">pub.scheduler:addComplexTask</a>	WmPublic. Adds a complex task to the Scheduler.
<a href="#">pub.scheduler:addOneTimeTask</a>	WmPublic. Adds a task that runs only once to the Scheduler.
<a href="#">pub.scheduler:addRepeatingTask</a>	WmPublic. Adds a recurring task to the Scheduler.
<a href="#">pub.scheduler:cancelTask</a>	WmPublic. Removes a task from the Scheduler.
<a href="#">pub.scheduler:getTaskIDs</a>	WmPublic. Retrieves a list of identification numbers for tasks currently in the Scheduler.
<a href="#">pub.scheduler:getTaskInfo</a>	WmPublic. Retrieves information about a task on the Scheduler.
<a href="#">pub.scheduler:getUserTaskList</a>	WmPublic. Returns a list of scheduled user tasks.
<a href="#">pub.scheduler:migrateTasksToJDBC</a>	WmPublic. Migrates scheduled user tasks from the Integration Server embedded database to an external database.
<a href="#">pub.scheduler:resumeTask</a>	WmPublic. Resumes a suspended task.
<a href="#">pub.scheduler:suspendTask</a>	WmPublic. Suspends a task on the Scheduler.
<a href="#">pub.scheduler:updateComplexTask</a>	WmPublic. Updates a complex task on the Scheduler.

Element	Package and Description
<code>pub.scheduler:updateOneTimeTask</code>	WmPublic. Updates a one-time task on the Scheduler.
<code>pub.scheduler:updateRepeatingTask</code>	WmPublic. Updates a repeating task to the Scheduler.

## pub.scheduler:addComplexTask

WmPublic. Adds a complex task to the Scheduler.

The webMethods Integration Server runs the service for a complex task on the day(s) and time(s) that you specify either during a specified date range or indefinitely.

### Input Parameters

<i>service</i>	<b>String</b> Name of the service you want to schedule for execution on the server.
<i>description</i>	<b>String</b> Text string describing this task.
<i>target</i>	<p><b>String</b> Server or servers on which the task is to run. (Clustered environments only). Set to:</p> <ul style="list-style-type: none"> <li>any to run the task on any server in the cluster. The task will run on only <i>one</i> of the servers.</li> </ul> <p>For example, suppose that all the servers in your cluster share a single database for a parts inventory application, and that a particular function needs to run against that database once a day. Any of the servers can perform this task, therefore you can specify the <code>all</code> option to schedule a task to run on any of the servers.</p> <div style="background-color: #f0f0f0; padding: 10px; border: 1px solid #ccc;"> <p><b>Note:</b> There is no predetermined order in which servers in the cluster are selected to run tasks. Rather, the first server to detect that a task is ready to be executed runs it.</p> </div> <p>For more information about how Integration Server handles the scheduling of tasks in a clustered environment, see the chapter about managing services <i>webMethods Integration Server Administrator's Guide</i>.</p> <ul style="list-style-type: none"> <li><code>all</code> to run the task on all servers in the cluster.</li> </ul>

For example, suppose you run an application on each server in the cluster, and each server maintains its own database for that application. If you need to run a cleanup task against all the databases every day, you can schedule a task to run every day on all the servers in the cluster.

For more information about how Integration Server handles the scheduling of tasks in a clustered environment, see the chapter about managing services *webMethods Integration Server Administrator's Guide*.

- `hostname` to run the task on a specific server in the cluster.

*lateness*

**String** The number of minutes (after the scheduled execution time) after which the server is to take a special action for a late task. You specify the action to be taken in the *latenessAction* parameter, described below. The server checks scheduled tasks at startup, and again periodically. If the server finds a task that is overdue and has exceeded the lateness period, the server performs the requested lateness action. If the server finds a task that is overdue but has *not* yet exceeded the lateness period, the server starts the task immediately.

*latenessAction*

**String** Action to take if a task has missed its scheduled start time by a number of minutes you specified with the *lateness* parameter. Possible actions are:

- **run immediately or 0** - Runs the task immediately
- **skip and run at next scheduled interval or 1** - Skips this execution of the task and runs it again at the next scheduled run time.
- **suspend or 2** - Places the task in a suspended state until an administrator resumes or cancels the task.

*runAsUser*

**String** Optional. User ID under which the service is to be executed. If you do not specify a user name, the "Default" access rights are used.

*inputs*

**Document** Optional. Document (IData object) containing the input to the scheduled service.

**Note:** You can also assign values to input parameters of services using the **Assign Inputs** option while scheduling a task in Integration Server Administrator.

*startTime*

**String** Optional. Time at which the task is scheduled to start, in the format *HH:mm:ss*. If you do not specify a *startTime*, the current time is used.

---

<i>startDate</i>	<b>String</b> Optional. Date on which the task is scheduled to start, in the format <i>yyyy/MM/dd</i> . If you do not specify a <i>startDate</i> , the current date is used.
<i>endTime</i>	<b>String</b> Optional. Time at which the task expires, in the format <i>HH:mm:ss</i> . If you do not specify an <i>endTime</i> , the server uses the current time.
<i>endDate</i>	<b>String</b> Optional. Date on which the task expires, in the format <i>yyyy/MM/dd</i> . If you do not specify an <i>endDate</i> , the server executes this service for an indefinite period of time.
<i>months</i>	<b>String List</b> Optional. Months during which the task is scheduled to run. Months are represented by integers between 1 and 12, where "1" indicates January and "12" indicates December. If you do not specify <i>months</i> , the task will run every month.
<i>hours</i>	<b>String List</b> Optional. Hours at which the task is scheduled to run. Hours are represented by integers between 0 and 23. If you do not specify <i>hours</i> , the task runs every hour.
<i>minutes</i>	<b>String List</b> Optional. Minutes at which the task is scheduled to run. Minutes are represented by integers between 0 and 59. If you do not specify <i>minutes</i> , the task runs every minute.
<i>daysOfMonth</i>	<b>String List</b> Optional. Days of the month on which the task is scheduled to run. Days are represented by integers between 1 and 31. If you do not specify <i>daysOfMonth</i> , the task runs every day of the month.
<i>daysOfWeek</i>	<b>String List</b> Optional. Days of the week on which the task is scheduled to run. Days are represented by integers between 1 and 7, where "1" indicates Sunday and "7" indicates Saturday. If you do not specify <i>daysOfWeek</i> , the task runs every day of the week.

### Output Parameters

---

<i>taskId</i>	<b>String</b> Identification number of the task added to the scheduler.
<i>type</i>	<b>String</b> Code indicating the type of task added. For this type of task, the value of <i>type</i> will be <code>complex</code> .

*taskAdded* **String** Indicates whether the task was successfully added to the Scheduler. If the task was successfully added to the Scheduler, *taskAdded* contains `true`. If the task was not successfully added, the server throws an exception and terminates the service.

---

## pub.scheduler:addOneTimeTask

WmPublic. Adds a task that runs only once to the Scheduler.

The Integration Server executes the service a single time on the date and time you specify.

### Input Parameters

---

*service* **String** Name of the service you want to schedule for execution.

*description* **String** Text string describing this task.

*target* **String** Server or servers on which the task is to run.(Clustered environments only). Set to:

- `any` to run the task on any server in the cluster. The task will run on only *one* of the servers.

For example, suppose that all the servers in your cluster share a single database for a parts inventory application, and that a particular function needs to run against that database once a day. Any of the servers can perform this task, therefore you can specify the `all` option to schedule a task to run on any of the servers.

**Note:** There is no predetermined order in which servers in the cluster are selected to run tasks. Rather, the first server to detect that a task is ready to be executed runs it.

For more information about how Integration Server handles the scheduling of tasks in a clustered environment, see the chapter about managing services *webMethods Integration Server Administrator's Guide*.

- `all` to run the task on all servers in the cluster.

For example, suppose you run an application on each server in the cluster, and each server maintains its own database for that application. If you need to run a cleanup task against all

the databases every day, you can schedule a task to run every day on all the servers in the cluster.

For more information about how Integration Server handles the scheduling of tasks in a clustered environment, see the chapter about managing services *webMethods Integration Server Administrator's Guide*.

- `hostname` to run the task on a specific server in the cluster.

*lateness*

**String** The number of minutes (after the scheduled execution time) after which the server is to take a special action for a late task. You specify the action to be taken in the *latenessAction* parameter, described below. The server checks scheduled tasks at startup, and again periodically. If the server finds a task that is overdue and has exceeded the lateness period, the server performs the requested lateness action. If the server finds a task that is overdue but has *not* yet exceeded the lateness period, the server starts the task immediately.

*latenessAction*

**String** Action to take if a task has missed its scheduled start time by a number of minutes you specified with the *lateness* parameter. Possible actions are:

- **run immediately or 0** - Runs the task immediately
- **skip and run at next scheduled interval or 1** - Skips this execution of the task and runs it again at the next scheduled run time.
- **suspend or 2** - Places the task in a suspended state until an administrator resumes or cancels the task.

*runAsUser*

**String** Optional. User ID under which the service is to be executed. If you do not specify a user name, the "Default" access rights are used.

*inputs*

**Document** Optional. Document (IData object) containing input to the scheduled service.

**Note:** You can also assign values to input parameters of services using the **Assign Inputs** option while scheduling a task in Integration Server Administrator

*date*

**String** Date on which to run the service, in the format *yyyy/MM/dd*.

*time*

**String** Time at which to run the service, in the format *HH:mm:ss*.

---

## Output Parameters

---

<i>taskID</i>	<b>String</b> Identification number of the task added to the scheduler.
<i>type</i>	<b>String</b> Code indicating the type of task added. For this type of task, the value of <i>type</i> will be <code>once</code> .
<i>taskAdded</i>	<b>String</b> Indicates whether the task was successfully added to the Scheduler. If the task was successfully added to the Scheduler, <i>taskAdded</i> contains <code>true</code> . If the task was not successfully added, the server throws an exception and terminates the service.

---

## pub.scheduler:addRepeatingTask

WmPublic. Adds a recurring task to the Scheduler.

The webMethods Integration Server continually executes a repeating task at the interval you specify.

---

### Input Parameters

---

<i>service</i>	<b>String</b> Name of the service you want to schedule for execution on the server.
<i>description</i>	<b>String</b> Text string describing this task.
<i>target</i>	<p><b>String</b> Server or servers on which the task is to run. (Clustered environments only). Set to:</p> <ul style="list-style-type: none"> <li>■ <code>any</code> to run the task on any server in the cluster. The task will run on only <i>one</i> of the servers.</li> </ul> <p>For example, suppose that all the servers in your cluster share a single database for a parts inventory application, and that a particular function needs to run against that database once a day. Any of the servers can perform this task, therefore you can specify the <code>all</code> option to schedule a task to run on any of the servers.</p>

**Note:** There is no predetermined order in which servers in the cluster are selected to run tasks. Rather, the first server to detect that a task is ready to be executed runs it.

For more information about how Integration Server handles the scheduling of tasks in a clustered environment, see the chapter about managing services in *webMethods Integration Server Administrator's Guide*.

- `all` to run the task on all servers in the cluster.

For example, suppose you run an application on each server in the cluster, and each server maintains its own database for that application. If you need to run a cleanup task against all the databases every day, you can schedule a task to run every day on all the servers in the cluster.

For more information about how Integration Server handles the scheduling of tasks in a clustered environment, see the chapter about managing services in *webMethods Integration Server Administrator's Guide*.

- `hostname` to run the task on a specific server in the cluster.

*lateness*

**String** The number of minutes (after the scheduled execution time) after which the server is to take a special action for a late task. You specify the action to be taken in the *latenessAction* parameter, described below. The server checks scheduled tasks at startup, and again periodically. If the server finds a task that is overdue and has exceeded the lateness period, the server performs the requested lateness action. If the server finds a task that is overdue but has *not* yet exceeded the lateness period, the server starts the task immediately.

*latenessAction*

**String** Action to take if a task has missed its scheduled start time by a number of minutes you specified with the *lateness* parameter. Possible actions are:

- **run immediately or 0** - Runs the task immediately
- **skip and run at next scheduled interval or 1** - Skips this execution of the task and runs it again at the next scheduled run time.
- **suspend or 2** - Places the task in a suspended state until an administrator resumes or cancels the task.

*runAsUser*

**String** Optional. User ID under which the service is to be executed. If you do not specify a user name, the "Default" access rights are used.

*inputs*

**Document** Optional. Document (IData object) containing input to the scheduled service.

**Note:** You can also assign values to input parameters of services using the **Assign Inputs** option while scheduling a task in Integration Server Administrator.

<i>startTime</i>	<b>String</b> Optional. Time at which the task is scheduled to start, in <i>HH:mm:ss</i> format. If you do not specify a <i>startTime</i> , the current time is used.
<i>startDate</i>	<b>String</b> Optional. Date on which the task is scheduled to start, in <i>yyyy/MM/dd</i> format. If you do not specify <i>date</i> , the current date is used.
<i>endTime</i>	<b>String</b> Optional. Time at which the task expires, in <i>HH:mm:ss</i> format. If you do not specify an <i>endTime</i> , the server uses the current time.
<i>endDate</i>	<b>String</b> Optional. Date on which the task expires, in <i>yyyy/MM/dd</i> format. If you do not specify an <i>endDate</i> , the server executes this service for an indefinite period of time.
<i>interval</i>	<b>String</b> Time interval (measured in seconds) between executions of the task.
<i>doNotOverlap</i>	<p><b>String</b> Optional. Flag that indicates whether you want executions of this task to overlap. Set to:</p> <ul style="list-style-type: none"> <li>■ <code>true</code> to prevent executions of the scheduled task from overlapping. After a scheduled task finishes executing, the Scheduler waits the number of seconds specified in <i>interval</i> before running the task again.</li> <li>■ <code>false</code> to allow executions of the scheduled task to overlap. The Scheduler runs the task every time the value of <i>interval</i> elapses. This is the default.</li> </ul>

## Output Parameters

<i>taskID</i>	<b>String</b> Identification number of the task added to the Scheduler.
<i>type</i>	<b>String</b> Code indicating the type of task added. For this type of task, the value of <i>type</i> will be <code>repeat</code> .
<i>taskAdded</i>	<b>String</b> Indicates whether the task was successfully added to the Scheduler. If the task was successfully added to the Scheduler, <i>taskAdded</i> contains <code>true</code> . If the task was not

---

successfully added, the server throws an exception and terminates the service.

---

## pub.scheduler:cancelTask

WmPublic. Removes a task from the Scheduler.

### Input Parameters

---

*taskID*

**String** Identification number of the task to remove from the Scheduler.

If your server runs as part of a cluster of servers, and you have scheduled a task to run on all servers in the cluster, note the following before canceling a task:

- If you cancel a parent task, the task will be canceled on all servers in the cluster.
- If you cancel a child task, the task will be canceled only on the server on which the child task was scheduled to run.

For more information about parent and child tasks, see [pub.scheduler:getTaskInfo](#) or the chapter about managing services in *webMethods Integration Server Administrator's Guide*.

### Output Parameters

---

*taskCancelled*

**String** Indicates whether the task was successfully removed from the Scheduler. If the task was successfully removed from the Scheduler, *taskCancelled* contains `true`. If the task was not successfully removed, the server throws an exception and terminates the service.

### Usage Notes

For information about the tasks on the Scheduler, run the [pub.scheduler:getTaskIDs](#) and [pub.scheduler:getTaskInfo](#) services.

---

## pub.scheduler:getTaskIDs

WmPublic. Retrieves a list of identification numbers for tasks currently in the Scheduler.

---

## Input Parameters

---

*running* **String** Specifies whether the service returns task IDs for all tasks or just tasks that are running. If you specify “false” (the default), the service returns task IDs for all tasks. If you specify “true,” the service returns task IDs for just those tasks with the status “running.”

## Output Parameters

---

*taskIDs* **String List** Identification numbers for the tasks on the Scheduler.

---

## pub.scheduler:getTaskInfo

WmPublic. Retrieves information about a task on the Scheduler.

---

## Input Parameters

---

*taskID* **String** Task identification number.

## Output Parameters

---

*type* **String** Code indicating the task's type. Will be one of the following:

```
complex
once
repeat
```

*runAsUser* **String** The user ID whose access rights are used to execute the service.

*target* **String** Server or servers on which the task is to run. (Clustered environments only). A value of:

- `$any` indicates that the task will run on any, but only one, server in the cluster.

For more information about scheduled tasks in a clustered environment, see the chapter about managing services in *webMethods Integration Server Administrator's Guide*.

- `$all` indicates that the task will run on all servers in a cluster.

When you schedule a task to run on all servers in the cluster, the server divides the task into a main or *parent* task, and a *child* task for each server in the cluster. You can perform some actions (activate, suspend, delete) individually on the child tasks, but if you want to change the characteristics of a task, you must do so through the parent task.

For a parent task, this service returns \$all in the Target parameter.

For each child task, this service returns the hostname:port on which the task is to run.

For more information about scheduled tasks in a clustered environment, see the chapter about managing services in *webMethods Integration Server Administrator's Guide*.

- `hostname` indicates that the task will run on this particular server. This service returns `hostname:port` if:
  - Your server is running in a cluster, a task was scheduled to run on all servers in the cluster, and this is one of the child tasks. (See the description of \$all above.)
  - Your server is running in a cluster and you requested a specific server.
  - Your server is not running in a cluster.

For more information about scheduled tasks in a clustered environment, see the chapter about managing services in *webMethods Integration Server Administrator's Guide*.

*description* **String** Text string describing this task.

*lateness* **String** The number of minutes (after the scheduled execution time) after which the server is to take a special action for a late task. You specify the action to be taken in the *latenessAction* parameter, described below. The server checks scheduled tasks at startup, and again periodically. If the server finds a task that is overdue and has exceeded the lateness period, the server performs the requested lateness action. If the server finds a task that is overdue but has *not* yet exceeded the lateness period, the server starts the task immediately.

*latenessAction* **String** Action to take if a task has missed its scheduled start time by a number of minutes you specify in the *lateness* parameter. Possible actions are:

- **run immediately or 0** - Runs the task immediately

- **skip and run at next scheduled interval or 1** - Skips this execution of the task and runs it again at the next scheduled run time.
- **suspend or 2** - Places the task in a suspended state until an administrator resumes or cancels the task.

*service* **String** Name of the service associated with the task.

*nextRun* **String** Next date and time that the task is scheduled to run. The date and time is expressed as the number of milliseconds from January 1, 1970, 00:00:00 GMT.

*execState* **String** Current state of the task.

Tasks can be in one of the following states:

<u>A value of...</u>	<u>Indicates that...</u>
0	The task is currently active.
1	The task is currently running.
2	The task has been suspended or has expired.

For tasks that are scheduled to run on all servers in the cluster, you might see different statuses among the parent and child tasks. For example, the parent's status might be Active, while one child's status is Active, and another child's status is Suspended.

In general, the status of the parent task will be Active if at least one child task is active or running, Suspended if all child tasks are suspended, or Expired, if all child tasks are expired.

*inputs* **Document** Conditional. Document (IData object) containing the inputs, if any, to the scheduled service.

*oneTimeTaskInfo* **Document** Conditional. Information about the complex task represented by *taskID*. This parameter is present only if *type* is *once*.

<u>Key</u>	<u>Description</u>
<i>date</i>	<b>String</b> Conditional. Date on which to run the task, in <i>yyyy/MM/dd</i> format.

---

	<i>time</i>	<b>String</b> Conditional. Time at which to run the task, in <i>HH:mm:ss</i> format.
<i>repeatingTaskInfo</i>		<b>Document</b> Conditional. Information about the task represented by <i>taskID</i> . This parameter is present only if <i>type</i> is <i>repeat</i> .
	<b>Key</b>	<b>Description</b>
	<i>interval</i>	<b>String</b> Conditional. Time interval (measured in seconds) between repetitions of the task.
	<i>doNotOverlap</i>	<b>String</b> Conditional. Indicates whether recurrences of this task will overlap.
<i>complexTaskInfo</i>		<b>Document</b> Conditional. Information about the task. This parameter is present only if <i>type</i> is <i>complex</i> .
	<b>Key</b>	<b>Description</b>
	<i>startDate</i>	<b>String</b> Conditional. Date on which the task is scheduled to start, in <i>yyyy/MM/dd</i> format.
	<i>startTime</i>	<b>String</b> Conditional. Time at which the task is scheduled to start, in <i>HH:mm:ss</i> format.
	<i>endDate</i>	<b>String</b> Conditional. Date on which the task expires, in <i>yyyy/MM/dd</i> format.
	<i>endTime</i>	<b>String</b> Conditional. Time at which the task expires, in <i>HH:mm:ss</i> format.
	<i>minutes</i>	<b>String List</b> Conditional. Minutes at which the task is scheduled to run. Minutes are represented by integers between 0 and 59.
	<i>hours</i>	<b>String List</b> Conditional. Hours when the task is scheduled to run. Hours are represented by integers between 0 and 23.

<i>months</i>	<b>String List Conditional.</b> Months during which the task is scheduled to run. Months are represented by integers between 1 and 12, where “1” indicates January and “12” indicates December.
<i>daysOfWeek</i>	<b>String List Conditional.</b> Days of the week on which the task is scheduled to run. Days are represented by integers between 1 and 7, where “1” indicates Sunday and “7” indicates Saturday.
<i>daysOfMonth</i>	<b>String List Conditional.</b> Days of the month on which the task is scheduled to run. Days are represented by integers between 1 and 31.

---

## pub.scheduler:getUserTaskList

WmPublic. Returns a list of scheduled user tasks.

### Input Parameters

---

None.

### Output Parameters

---

<i>tasks</i>	<b>Document List</b> List of one-time and simple repeating tasks.
<i>extTasks</i>	<b>Document List</b> List of complex repeating tasks.

---

## pub.scheduler:migrateTasksToJDBC

WmPublic. Migrates scheduled user tasks from the Integration Server embedded database to an external database.

Integration Server stores information about certificate maps and scheduled jobs in a database that is associated with the **ISInternal** functional alias. When you install Integration Server, you can select whether this database will exist as an embedded database that is shipped with Integration Server, or an external RDBMS that you set up. If you chose to use the embedded database at install time, but later want to use an external RDBMS instead, you can use the `pub.scheduler:migrateTasksToJDBC` service to copy

or move information about user scheduled tasks from the embedded database to the external RDBMS.

### Input Parameters

---

*move* **Boolean** Specifies whether the tasks are to be deleted from the embedded database after the migration successfully completes. If set to false, the default, the tasks remain in the embedded database. If set to true, the tasks are removed from the embedded database.

### Output Parameters

---

*numberOfTaskMigrated* **String** The number of user scheduled tasks that were migrated.

*successful* **String** Indicates whether or not the migration was successful. The service returns "true" if all tasks were successfully migrated, otherwise "false."

### Usage Notes

This service copies scheduled user tasks only; it does not copy or move information about certificate maps.

Before running this service you must install the external IS Internal database component and define a database connection for it. For instructions, refer to *Installing Software AG Products*.

When you run the service, it looks in the embedded database for scheduled user tasks and writes any tasks it finds to the database identified by the **ISInternal** functional alias, which is defined on the **Settings > JDBC Pools** screen of the Integration Server Administrator.

---

## pub.scheduler:resumeTask

WmPublic. Resumes a suspended task.

### Input Parameters

---

*taskID* **String** Identification number of the task to resume.

If your server runs as part of a cluster of servers, and you have scheduled a task to run on all servers in the cluster, note the following before resuming a task:

- If you resume a parent task, the task will be resumed on all servers in the cluster.
- If you resume a child task, the task will be resumed only on the server on which the child task was scheduled to run.

For more information about parent and child tasks, see [pub.scheduler:getTaskInfo](#) or the chapter about managing services in *webMethods Integration Server Administrator's Guide*.

## Output Parameters

*taskResumed* **String** Indicates whether the task was successfully resumed. If the task was successfully resumed, *taskResumed* contains `true`. If the task was not successfully resumed, the server throws an exception and terminates the service.

## pub.scheduler:suspendTask

WmPublic. Suspends a task on the Scheduler.

## Input Parameters

*taskID* **String** Identification number of the task to suspend.

If your server runs as part of a cluster of servers, and you have scheduled a task to run on all servers in the cluster, note the following before canceling a task:

- If you suspend a parent task, the task will be suspended on all servers in the cluster.
- If you suspend a child task, the task will be suspended only on the server on which the child task was scheduled to run.

For more information about parent and child tasks, see [pub.scheduler:getTaskInfo](#) or the chapter about managing services in *webMethods Integration Server Administrator's Guide*.

## Output Parameters

*taskSuspended* **String** Indicates whether the task was successfully suspended. If the task was successfully suspended, *taskSuspended* contains `true`. If the task was not successfully suspended, the server throws an exception and terminates the service.

## Usage Notes

If you want to cancel a task or remove a task from the scheduler, use the [pub.scheduler:cancelTask](#) service.

---

## pub.scheduler:updateComplexTask

WmPublic. Updates a complex task on the Scheduler.

The webMethods Integration Server runs the service for a complex task on the day(s) and time(s) that you specify either during a specified date range or indefinitely.

### Input Parameters

---

<i>taskID</i>	<b>String</b> Identification number of the task to be updated.
<i>service</i>	<b>String</b> Optional. Name of the service you want to schedule for execution on the server.
<i>description</i>	<b>String</b> Optional. Text string describing this task.
<i>target</i>	<b>String</b> Optional. Server or servers on which the task is to run. (Clustered environments only). Set to: <ul style="list-style-type: none"><li>■ <code>any</code> to run the task on any server in the cluster. The task will run on only <i>one</i> of the servers.  For example, suppose that all the servers in your cluster share a single database for a parts inventory application, and that a particular function needs to run against that database once a day. Any of the servers can perform this task, therefore you can specify the <code>all</code> option to schedule a task to run on any of the servers.  <b>Note:</b> There is no predetermined order in which servers in the cluster are selected to run tasks. Rather, the first server to detect that a task is ready to be executed runs it.  For more information about how Integration Server handles the scheduling of tasks in a clustered environment, see the chapter about managing services in <i>webMethods Integration Server Administrator's Guide</i>.</li><li>■ <code>all</code> to run the task on all servers in the cluster. For clustered environments only.</li></ul>

For example, suppose you run an application on each server in the cluster, and each server maintains its own database for that application. If you need to run a cleanup task against all the databases every day, you can schedule a task to run every day on all the servers in the cluster.

For more information about how Integration Server handles the scheduling of tasks in a clustered environment, see the chapter about managing services in *webMethods Integration Server Administrator's Guide*.

- `hostname` to run the task on a specific server in the cluster.

*lateness*

**String** Optional. The number of minutes (after the scheduled execution time) after which the server is to take a special action for a late task. You specify the action to be taken in the *latenessAction* parameter, described below. The server checks scheduled tasks at startup, and again periodically. If the server finds a task that is overdue and has exceeded the lateness period, the server performs the requested lateness action. If the server finds a task that is overdue but has *not* yet exceeded the lateness period, the server starts the task immediately.

*latenessAction*

**String** Optional. Action to take if a task has missed its scheduled start time by a number of minutes you specified with the *lateness* parameter. Possible actions are:

- **run immediately or 0** - Runs the task immediately
- **skip and run at next scheduled interval or 1** - Skips this execution of the task and runs it again at the next scheduled run time.
- **suspend or 2** - Places the task in a suspended state until an administrator resumes or cancels the task.

*doNotOverlap*

**String** Optional. Flag that indicates whether you want executions of this task to overlap. Set to:

- `true` to prevent executions of the scheduled task from overlapping. After a scheduled task finishes executing, the Scheduler waits the number of seconds specified in *interval* before running the task again.
- `false` to allow executions of the scheduled task to overlap. The Scheduler runs the task every time the value of *interval* elapses. This is the default.

*runAsUser*

**String** Optional. User ID under which the service is to be executed. If you do not specify a user name, the "Default" access rights are used.

---

<i>inputs</i>	<b>Document</b> Optional. Document (IData object) containing input to the scheduled service.
<i>startTime</i>	<b>String</b> Optional. Time at which the task is scheduled to start, in <i>HH:mm:ss</i> format. If you do not specify a <i>startTime</i> , the current time is used.
<i>startDate</i>	<b>String</b> Optional. Date on which the task is scheduled to start, in <i>yyyy/MM/dd</i> format. If you do not specify <i>date</i> , the current date is used.
<i>endTime</i>	<b>String</b> Optional. Time at which the task expires, in <i>HH:mm:ss</i> format. If you do not specify an <i>endTime</i> , the server uses the current time.
<i>endDate</i>	<b>String</b> Optional. Date on which the task expires, in <i>yyyy/MM/dd</i> format. If you do not specify an <i>endDate</i> , the server executes this service for an indefinite period of time.
<i>months</i>	<b>String List</b> Optional. Months during which the task is scheduled to run. Months are represented by integers between 1 and 12, where "1" indicates January and "12" indicates December. If you do not specify <i>months</i> , the task will run every month.
<i>hours</i>	<b>String List</b> Optional. Hours at which the task is scheduled to run. Hours are represented by integers between 0 and 23. If you do not specify <i>hours</i> , the task runs every hour.
<i>minutes</i>	<b>String List</b> Optional. Minutes at which the task is scheduled to run. Minutes are represented by integers between 0 and 59. If you do not specify <i>minutes</i> , the task runs every minute.
<i>daysOfMonth</i>	<b>String List</b> Optional. Days of the month on which the task is scheduled to run. Days are represented by integers between 1 and 31. If you do not specify <i>daysOfMonth</i> , the task runs every day of the month.
<i>daysOfWeek</i>	<b>String List</b> Optional. Days of the week on which the task is scheduled to run. Days are represented by integers between 1 and 7, where "1" indicates Sunday and "7" indicates Saturday. If you do not specify <i>daysOfWeek</i> , the task runs every day of the week.

---

## Output Parameters

---

<i>type</i>	<b>String</b> Code indicating the type of task that was updated. For this type of task, the value of <i>type</i> will be <code>complex</code> .
<i>taskUpdated</i>	<b>String</b> Indicates whether the task was successfully updated. If the task was successfully updated, <i>taskUpdated</i> contains <code>true</code> . If the task was not successfully updated, the server throws an exception and terminates the service.

## Usage Notes

You can use [pub.scheduler:getTaskIDs](#) and [pub.scheduler:getTaskInfo](#) services to get information about the task you want to update.

This service updates only the fields for which you provide input parameters. If you want to clear the information in an optional field, specify blanks in the parameter for that field.

You can also assign values to input parameters of services using the **Assign Inputs** option while scheduling a task in Integration Server Administrator.

---

## pub.scheduler:updateOneTimeTask

WmPublic. Updates a one-time task on the Scheduler.

### Input Parameters

---

<i>taskID</i>	<b>String</b> Identification number of the task to be updated.
<i>service</i>	<b>String</b> Optional. Name of the service to be scheduled.
<i>description</i>	<b>String</b> Optional. Text string describing this task.
<i>target</i>	<p><b>String</b> Optional. <b>Server or servers on which the task is to run.</b> (Clustered environments only). Set to:</p> <ul style="list-style-type: none"> <li>■ <code>any</code> to run the task on any server in the cluster. The task will run on only <i>one</i> of the servers.</li> </ul> <p>For example, suppose that all the servers in your cluster share a single database for a parts inventory application, and that a particular function needs to run against that database once a day. Any of the servers can perform this task,</p>

therefore you can specify the `all` option to schedule a task to run on any of the servers.

**Note:** There is no predetermined order in which servers in the cluster are selected to run tasks. Rather, the first server to detect that a task is ready to be executed runs it.

For more information about how Integration Server handles the scheduling of tasks in a clustered environment, see the chapter about managing services in *webMethods Integration Server Administrator's Guide*.

- `all` to run the task on all servers in the cluster.

For example, suppose you run an application on each server in the cluster, and each server maintains its own database for that application. If you need to run a cleanup task against all the databases every day, you can schedule a task to run every day on all the servers in the cluster.

For more information about how Integration Server handles the scheduling of tasks in a clustered environment, see the chapter about managing services in *webMethods Integration Server Administrator's Guide*.

- `hostname` to run the task on a specific server in the cluster.

#### *lateness*

**String** Optional. The number of minutes (after the scheduled execution time) after which the server is to take a special action for a late task. You specify the action to be taken in the *latenessAction* parameter, described below. The server checks scheduled tasks at startup, and again periodically. If the server finds a task that is overdue and has exceeded the *lateness* period, the server performs the requested *lateness* action. If the server finds a task that is overdue but has *not* yet exceeded the *lateness* period, the server starts the task immediately.

#### *latenessAction*

**String** Optional. Action to take if a task has missed its scheduled start time by a number of minutes you specified with the *lateness* parameter. Possible actions are:

- **run immediately or 0** - Runs the task immediately
- **skip and run at next scheduled interval or 1** - Skips this execution of the task and runs it again at the next scheduled run time.
- **suspend or 2** - Places the task in a suspended state until an administrator resumes or cancels the task.

---

<i>runAsUser</i>	<b>String</b> Optional. User ID under which the service is to be executed.
<i>inputs</i>	<b>Document</b> Optional. Document (IData object) containing inputs to the scheduled service.
<i>date</i>	<b>String</b> Optional. Date on which to run the task, in <i>yyyy/MM/dd</i> format.
<i>time</i>	<b>String</b> Optional. Time at which to run the service, in <i>HH:mm:ss</i> format.

### Output Parameters

---

<i>type</i>	<b>String</b> Code indicating the type of task that was updated. For this type of task, the value of <i>type</i> will be <code>once</code> .
<i>taskUpdated</i>	<b>String</b> Indicates whether the task was successfully updated. If the task was successfully updated, <i>taskUpdated</i> contains <code>true</code> . If the task was not successfully updated, the server throws an exception and terminates the service.

### Usage Notes

This service updates only the fields for which you provide input parameters. If you want to clear the information in an optional field, specify blanks in the parameter for that field.

You can also assign values to input parameters of services using the **Assign Inputs** option while scheduling a task in Integration Server Administrator.

---

## pub.scheduler:updateRepeatingTask

WmPublic. Updates a repeating task to the Scheduler.

### Input Parameters

---

<i>taskId</i>	<b>String</b> Identification number of the task to be updated.
<i>service</i>	<b>String</b> Optional. Name of the service run by the task.
<i>description</i>	<b>String</b> Optional. Text string describing this task.

*target*

**String** Optional. Server or servers in the cluster on which the task is to run. (Clustered environments only). Set to:

- `any` to run the task on any server in the cluster. The task will run on only *one* of the servers.

For example, suppose that all the servers in your cluster share a single database for a parts inventory application, and that a particular function needs to run against that database once a day. Any of the servers can perform this task, therefore you can specify the `all` option to schedule a task to run on any of the servers.

**Note:** There is no predetermined order in which servers in the cluster are selected to run tasks. Rather, the first server to detect that a task is ready to be executed runs it.

For more information about how Integration Server handles the scheduling of tasks in a clustered environment, see the chapter about managing services in *webMethods Integration Server Administrator's Guide*.

- `all` to run the task on all servers in the cluster.

For example, suppose you run an application on each server in the cluster, and each server maintains its own database for that application. If you need to run a cleanup task against all the databases every day, you can schedule a task to run every day on all the servers in the cluster.

For more information about how Integration Server handles the scheduling of tasks in a clustered environment, see the chapter about managing services in *webMethods Integration Server Administrator's Guide*.

- `hostname` to run the task on a specific server in the cluster.

*lateness*

**String** Optional. The number of minutes (after the scheduled execution time) after which the server is to take a special action for a late task. You specify the action to be taken in the *latenessAction* parameter, described below. The server checks scheduled tasks at startup, and again periodically. If the server finds a task that is overdue and has exceeded the lateness period, the server performs the requested lateness action. If the server finds a task that is overdue but has *not* yet exceeded the lateness period, the server starts the task immediately.

---

<i>latenessAction</i>	<p><b>String</b> Optional. Action to take if a task has missed its scheduled start time by a number of minutes you specified with the <i>lateness</i> parameter. Possible actions are:</p> <ul style="list-style-type: none"><li>■ <b>run immediately or 0</b> - Runs the task immediately</li><li>■ <b>skip and run at next scheduled interval or 1</b> - Skips this execution of the task and runs it again at the next scheduled run time.</li><li>■ <b>suspend or 2</b> - Places the task in a suspended state until an administrator resumes or cancels the task.</li></ul>
<i>runAsUser</i>	<p><b>String</b> Optional. User ID under which the service is to be executed. If you do not specify a user name, the “Default” access rights are used.</p>
<i>startTime</i>	<p><b>String</b> Optional. Time at which the task is scheduled to start, in <i>HH:mm:ss</i> format. If you do not specify a <i>startTime</i>, the current time is used.</p>
<i>startDate</i>	<p><b>String</b> Optional. Date on which the task is scheduled to start, in <i>yyyy/MM/dd</i> format. If you do not specify <i>date</i>, the current date is used.</p>
<i>endTime</i>	<p><b>String</b> Optional. Time at which the task expires, in <i>HH:mm:ss</i> format. If you do not specify an <i>endTime</i>, the server uses the current time.</p>
<i>endDate</i>	<p><b>String</b> Optional. Date on which the task expires, in <i>yyyy/MM/dd</i> format. If you do not specify an <i>endDate</i>, the server executes this service for an indefinite period of time.</p>
<i>inputs</i>	<p><b>Document</b> Optional. Document (IData object) containing inputs to the scheduled service.</p>
<i>interval</i>	<p><b>String</b> Optional. Time interval (measured in seconds) between repetitions of the task.</p>
<i>doNotOverlap</i>	<p><b>String</b> Optional. Flag indicating whether or not you want the executions of this task to overlap. Set to:</p> <ul style="list-style-type: none"><li>■ <code>true</code> to prevent executions of the scheduled task from overlapping. After a scheduled task finishes executing, the Scheduler waits the number of seconds specified in <i>interval</i> before running the task again.</li></ul>

- `false` to allow executions of the scheduled task to overlap. The Scheduler runs the task every time the value of *interval* elapses. This is the default.

## Output Parameters

---

<i>type</i>	<b>String</b> Code indicating the type of task updated. For this type of task, the value of <i>type</i> will be <code>repeat</code> .
<i>taskUpdated</i>	<b>String</b> Indicates whether the task was successfully updated. If the task was successfully updated, <i>taskUpdated</i> contains <code>true</code> . If the task was not successfully updated, the server throws an exception and terminates the service.

## Usage Notes

This service updates only the fields for which you provide input parameters. If you want to clear the information in an optional field, specify blanks in the parameter for that field.

You can also assign values to input parameters of services using the **Assign Inputs** option while scheduling a task in Integration Server Administrator.



# 28 Schema Folder

---

■ Summary of Elements in this Folder .....	684
--	-----

You use the elements in the schema folder to validate objects and to validate the pipeline.

## Summary of Elements in this Folder

The following elements are available in this folder:

Element	Package and Description
<a href="#">pub.schema:createXSD</a>	WmPublic. Creates an XML Schema definition from a document type, from the input and output parameters of a service, or from a specification.
<a href="#">pub.schema:validate</a>	WmPublic. Validates an object using an IS document type or a schema.
<a href="#">pub.schema:validatePipeline</a>	WmPublic. Validates the pipeline against a document type.
<a href="#">pub.schema.w3c</a>	WmPublic. This folder contains definitions for XML Schemas as defined in the W3C specification <i>XML Schema Part 2: Datatypes</i> .
<a href="#">pub.schema.w3c:datatypes</a>	WmPublic. A schema containing the simple type names for built-in schemas.
<a href="#">pub.schema.w3c:structures</a>	WmPublic. A schema containing the structural components for XML schema definitions.
<a href="#">pub.schema.w3c:xml</a>	WmPublic. A schema containing the XML namespace components, such as <code>xml:lang</code> and <code>xml:space</code> , as defined in the W3C specifications <i>Namespaces in XML and Extensible Markup Language (XML) 1.0</i> .
<a href="#">pub.schema.w3c:xsi</a>	WmPublic. A schema containing the XML Schema instance components, such as <code>xsi:nil</code> , <code>xsi:noNamespaceSchemaLocation</code> , <code>xsi:schemaLocation</code> , and <code>xsi:type</code> , as defined in the W3C XML Schema recommendation <i>Part 1: Structures</i> .

---

## pub.schema:createXSD

WmPublic. Creates an XML Schema definition from a document type, from the input and output parameters of a service, or from a specification.

### Input Parameters

---

*name* **String** Fully qualified name of a document type, service, or specification on the Integration Server.

### Output Parameters

---

*isSuccessful* **String** Flag indicating whether the schema definition was created successfully. A value of:

- `true` indicates that the schema definition was created successfully.
- `false` indicates that the schema definition was not created successfully. See *errors* for detailed information.

*xsd* **Document** Conditional. The schema definition *xsd* has the following keys:

Key	Description
<i>url</i>	<b>String</b> Conditional. Relative url of the generated schema.
<i>source</i>	<b>String</b> Conditional. Schema definition.

*errors* **Document List** Conditional. List of fatal errors, if any, that occurred when generating the XSD. Each document in the list has the following structure:

Key	Description
<i>error Message</i>	<b>String</b> Text of the error message.

When fatal errors occur, the service does not generate an XSD file.

*warnings*

**Document List** Conditional. List of non-fatal errors, if any, that were encountered while generating the XSD. Each document in the list has the following structure:

<b>Key</b>	<b>Description</b>
<i>warningMessage</i>	<b>String</b> Text of the warning message.

When non-fatal errors occur, the service generates the XSD file but also returns *warnings* to indicate that it encountered unusual or unexpected conditions during the process.

### Usage Notes

If the document type, service signature, or specification you are providing as input to `createXSD` contains fields that belong to multiple XML namespaces, `createXSD` generates multiple XML Schema definitions (one for each XML namespace) and imports them into the XML Schema contained in the *source* field. These imported XML Schema definitions appear as children of *xsd* in the pipeline.

When using `createXSD` to create an XML Schema definition, keep the following points in mind:

- Top-level strings are not allowed.
- String tables beneath the top level are not allowed.
- Field names must conform to QName lexical rules (that is, the prefix and local name must conform to NCName rules specified in <http://www.w3.org/TR/REC-xml-names/#NT-NCName>).
- Field names cannot contain a prefix without an associated XML namespace.
- Fields of type other than scalar string cannot have names that begin with the character @ or be named *\*body*.
- Fields at the same level (that is, beneath the same parent field in the input or output of the same signature) can have the same name but different types or properties. However, only one field's type and properties is used for all fields with that name at that level. Because the method used to select the field is not defined, Software AG recommends avoiding this case.
- Only one field named *\*body* can occur at the same level.
- Duplicate field names that begin with the character @ cannot repeat at the same level.
- Fields at different levels can have the same name with duplicate XML namespace values, even if the fields have different types or properties. However, only one field's type and properties are used for all fields with that name at that level. Because the method used to select the field is not defined, Software AG recommends avoiding this case.

- Object constraints are allowed. However, the Integration Server does not represent them in the XSD.
- Strings constrained by older schema types (types defined before the W3C XML 2001 Schema recommendations) are allowed. However, the Integration Server translates them into 2001 XML Schema types.
- If a document variable is considered to be open (the Allow unspecified fields property is set to true), Integration Server adds an `xsd:any` element to the complex type definition that corresponds to the document variable. However, when the `watt.core.schema.createSchema.omitXSDAny` sever configuration parameter is set to true, Integration Server omits the `xsd:any` element even when the document is considered to be open. For more information about this server configuration parameter, see *webMethods Integration Server Administrator's Guide*.

If you use `createXSD` to create multiple XML Schema definitions that refer to each other, place the XSD files in the same folder or base path. To ensure that the references resolve correctly, make sure the relative URLs specified in the XSD files reflect the names of the XSD files within this folder or base path.

---

## pub.schema:validate

WmPublic. Validates an object using an IS document type, XML document type, or an IS schema.

### Input Parameters

---

*object*

**Document** Where Document is an IData or an XMLData-formatted document, `com.wm.lang.xml.Document` or `com.wm.lang.xml.Node` Document or object to be validated.

*conformsTo*

**String** Document type or schema to validate *object* against.

- If *object* is a Document (IData object), *conformsTo* must specify the fully qualified name of an IS document type on the Integration Server.
- If *object* is a XMLData-formatted document, *conformsTo* must specify the fully qualified name of an XML document type on Integration Server.

**Note:** One or more XML document types and XML fields are created from a single XML schema definition. To validate an XMLData-formatted Document using the XML document types created from an XML Schema definition, you need specify only one of the XML document types or XML fields created from the schema. Integration Server locates the related XML

document types, using the complete collection of XML document types and XML fields created from the XML schema definition during validation.

- If *object* is a `com.wm.lang.xml.Document` or `com.wm.lang.xml.Node` object, *conformsTo* must specify the fully qualified name of an IS schema on the Integration Server.

The specified IS schema is needed only for validating nodes with "Names" that are not from XML namespaces (that is, qualified nodes whose XML Namespace Name properties are absent). Integration Server can only locate the IS schema if its fully qualified name is provided.

If the XML document (`com.wm.lang.xml.Document` or `com.wm.lang.xml.Node`) contains namespace-qualified tags, *conformsTo* is ignored. Instead, Integration Server uses the XML namespaces declared in the instance document to locate the IS schemas that contain definitions and declarations for that XML namespace.

**Note:** When validating a document type created from an XML schema definition, if you want to use the OR operator for pattern-matching, use the `|` operator after the pattern string. If the `|` operator is used at the start of the regular expression, Integration Server treats it as an empty string.

<i>maxErrors</i>	<p><b>String</b> Optional. Number of errors to be collected. Default value is 1. When the number of errors found is equal to <i>maxErrors</i>, the validation processor stops validation and returns the result. If <i>maxErrors</i> is set to -1, the validation processor returns all errors.</p>
<i>ignoreContent</i>	<p><b>String</b> Optional. Flag that specifies whether the validation processor will validate content keys of the type String, String List, or String Table.</p> <p>Set to:</p> <ul style="list-style-type: none"> <li>■ <code>true</code> to ignore content (that is, do not validate keys of these types).</li> <li>■ <code>false</code> to validate content. This is the default.</li> </ul>
<i>failIfInvalid</i>	<p><b>String</b> Optional. Flag that indicates whether the service should fail and throw an exception if the object is invalid. Set to:</p> <ul style="list-style-type: none"> <li>■ <code>true</code> to indicate that the service should fail if the object is invalid.</li> </ul>

- `false` to indicate that service should signal success and return errors to the pipeline if object is invalid. This is the default.

*schemaDomain*

**String** Optional. Schema domain in which the schema specified by the XML namespaces resides. If *schemaDomain* is not specified, Integration Server uses the default schema domain.

**Note:** This parameter only applies if *object* is a `com.wm.lang.xml.Document` or `com.wm.lang.xml.Node`

## Output Parameters

*isValid*

**String** Flag that indicates whether or not the validation was successful. A value of:

- `true` indicates that the validation was successful.
- `false` indicates that the validation was unsuccessful.

*errors*

**Document List** Errors encountered during validation. Each document will contain the following information:

Key	Description
<i>pathName</i>	<b>String</b> Location of the error in XQL.
<i>errorCode</i>	<b>String</b> Error code (for example, VV-001).
<i>errorMessage</i>	<b>String</b> Error message (for example, Missing Object).

## Usage Notes

When validating supplied XML against an IS document type, XML document type, or IS schema, Integration Server uses the Perl regular expression compiler by default. If you need to change this behavior so that Integration Server uses the Java regular expression compiler during validation, set the server configuration parameter `watt.core.datatype.usejavaregex` to `true`. For information about setting this configuration parameter, see *webMethods Integration Server Administrator's Guide*.

By default, when validating XML, Integration Server uses the Perl5 regular expression compiler. As a result, Integration Server uses Perl5 pattern matching to evaluate element content. The Perl5 pattern matching is not identical to the pattern matching defined by the World Wide Web Consortium (W3C) for the XML Schema standard. When validating XML against an IS schema or IS document type created from an XML Schema definition, an element defined to be of simple type with a pattern constraining facet

might be invalid according to the Perl5 pattern matching rules but valid according to the pattern matching rules associated with the XML Schema standard.

You can change how Integration Server performs pattern matching during validation by switching from the Perl5 regular expression compiler to the Java regular expression compiler. To do this, set the server configuration parameter `watt.core.datatype.usejavaregex` to `true`. When this parameter is set to `true`, Integration Server performs pattern matching as described by `java.util.regex.pattern`.

When validating against an IS document type, if the **Allow null** property is set to false for a field in the document type and the corresponding element in the instance document carries the attribute `xsi:nil`, Integration Server throws the following error

```
[ISC.0082.9026] Undefined Object found.
```

When validating against an IS document type, if the **Allow null** property is set to false for a field in the document type and the corresponding element in the instance document contains content or contains child elements, Integration Server throws the following error:

```
[ISC.0082.9024] FieldName cannot have content or child elements since xsi:nil is true.
```

When validating a `com.wm.lang.xml.Document` or `com.wm.lang.xml.Nodeobject`, Integration Server searches the named schema domain for the specified schema. If the schema cannot be found in the specified domain, Integration Server searches the default schema domain. Note that Integration Server searches the schema domain for a schema, not an individual component (element, attribute, complex type, etc) within the schema.

When validating XML, Integration Server uses the W3C recommendation *XML Schema Part 2: Datatypes*. If you want to validate XML for illegal values, set `ignoreContent` to `false` and the `watt.core.validation.w3cConformant` configuration parameter to `true`. For information about setting this configuration parameter, see *webMethods Integration Server Administrator's Guide*.

The `pub.schema.validate` service cannot be used to validate a node produced by the enhanced XML parser.

---

## pub.schema:validatePipeline

WmPublic. Validates the pipeline against a document type.

### Input Parameters

---

<i>conformsTo</i>	<b>String</b> Fully qualified name of the document type that you want to validate the pipeline against.
<i>maxErrors</i>	<b>String</b> Optional. Number of errors to be collected. Default value is 1. When the number of errors found is equal to <i>maxErrors</i> , the validation processor stops validation and returns the

result. If *maxErrors* is set to -1, the validation processor returns all errors.

*ignoreContent*

**String** Optional. Flag that specifies whether the validation processor will validate content keys of the type String, String List, or String Table. Set to:

- `true` to ignore content (that is, do not validate keys of these types).
- `false` to validate content. This is the default.

*failIfInvalid*

**String** Optional. Flag that indicates whether the service should fail and throw an exception if the object is invalid. Set to:

- `true` to indicate that service should fail if object is invalid.
- `false` to indicate that service should simply signal success and return errors to the pipeline if object is invalid. This is the default.

## Output Parameters

*isValid*

**String** Flag that indicates whether or not the validation was successful. A value of:

- `true` indicates that the validation was successful.
- `false` indicates that the validation was unsuccessful.

*errors*

**Document List** Errors encountered during validation. Each document will contain the following information:

Key	Description
<i>pathName</i>	<b>String</b> Location of the error in XQL.
<i>errorCode</i>	<b>String</b> Error code (for example, vv-001).
<i>errorMessage</i>	<b>String</b> Error message (for example, Missing Object).

## pub.schema.w3c

WmPublic. This folder contains definitions for XML Schemas as defined in the W3C specification *XML Schema Part 2: Datatypes*.

For more information about schemas and datatypes, see *webMethods Service Development Help*.

---

## **pub.schema.w3c:datatypes**

WmPublic. A schema containing the simple type names for built-in schemas.

---

## **pub.schema.w3c:structures**

WmPublic. A schema containing the structural components for XML schema definitions.

---

## **pub.schema.w3c:xml**

WmPublic. A schema containing the XML namespace components, such as `xml:lang` and `xml:space`, as defined in the W3C specifications *Namespaces in XML and Extensible Markup Language (XML) 1.0*.

---

## **pub.schema.w3c:xsi**

WmPublic. A schema containing the XML Schema instance components, such as `xsi:nil`, `xsi:noNamespaceSchemaLocation`, `xsi:schemaLocation`, and `xsi:type`, as defined in the W3C XML Schema recommendation *Part 1: Structures*.

# 29 Security Folder

---

■ About the Security Elements .....	694
■ Summary of Elements in this Folder .....	695

You use the elements in the security folder to control which client certificates are sent to other services and digitally sign data and process digital signatures. You can also use the elements to store and retrieve outbound passwords to access secure resources.

## About the Security Elements

---

Use the elements in the security folder to:

- Control which client certificates are sent to other services.
- Digitally sign data.
- Process digital signatures.
- Store and retrieve outbound passwords to access secure resources.
- Manage Integration Server keystores and truststores.
- Secure XML documents.

The services `pub.security.keystore:setKeyAndChain`, `pub.security:setKeyAndChainFromBytes`, and `pub.security:clearKeyAndChain` are used to control which client certificate the webMethods Integration Server presents to remote servers. You need to use these services to switch between certificates and certificate chains if you are not using aliases for remote servers. For more information about aliases for remote servers, see *webMethods Integration Server Administrator's Guide*.

The `pub.security.outboundPasswords` services support the use of encrypted outbound passwords to access secure resources. You may wish to have a flow service access a secure resource such as a remote Integration Server, proxy server, or database. The service would need to provide a valid password to access the resource. The `pub.security.outboundPasswords` services allow a flow service to store passwords in and retrieve passwords from the Integration Server's outbound password store. The outbound password store is an encrypted store of passwords managed by the Integration Server. For more information about the outbound password store, see *webMethods Integration Server Administrator's Guide*.

The `pub.security.keystore` services allow you to configure Integration Server SSL through access to its keys and associated certificates. These keys and certificates are now stored securely in industry-standard keystore and truststore files. For more information about Integration Server keystores and truststores, see *webMethods Integration Server Administrator's Guide*.

The `pub.security.xml` services are based on the Apache Security APIs. These services support encryption and digital signing of outbound XML documents from Integration Server, and decryption and signature verification of inbound XML from partner applications. The services provide the most commonly-used XML security options, including:

- Signing/encrypting the entire XML document or the content of specific nodes
- Selection of the signing and encryption algorithms

- Use of enveloping and enveloped signatures

## Summary of Elements in this Folder

The following elements are available in this folder:

Element	Package and Description
<a href="#">pub.security.clearKeyAndChain</a>	WmPublic. Clears the set key and certificate chain and reverts back to the default key and certificate chain for the subsequent set of invoked services.
<a href="#">pub.security.setKeyAndChain</a>	WmPublic. <i>Deprecated</i> - Replaced by <a href="#">pub.security.keystore.setKeyAndChain</a> .
<a href="#">pub.security.setKeyAndChainFromBytes</a>	WmPublic. Associates a key and certificate chain with the subsequent set of invoked services. Use this service to associate a key and certificate chain that is different from the default settings, and if your key and certificate information is located in byte arrays (rather than files).
<a href="#">pub.security.enterpriseGateway.alertSpec</a>	WmPublic. Specification for flow services used to send alerts about violations of webMethods Enterprise Gateway rules.
<a href="#">pub.security.enterpriseGateway.customFilterSpec</a>	WmPublic. Specification for services that can be invoked by the custom filter in Enterprise Gateway rules.
<a href="#">pub.security.keystore.getCertificate</a>	WmPublic. Returns the trusted certificate, stored in a truststore, that corresponds to the certificate's alias.
<a href="#">pub.security.keystore.getKeyAndChain</a>	WmPublic. Returns a private key and its associated certificate chain from a designated keystore.

Element	Package and Description
<a href="#">pub.security.keystore:getTrustedCertificates</a>	WmPublic. Returns the trusted certificates located in a specified truststore.
<a href="#">pub.security.keystore:setKeyAndChain</a>	WmPublic. Associates a key and certificate chain with the subsequent set of invoked services. Use this service to associate a key and certificate chain that is different from the default settings, and if your key and certificate information is stored in a keystore file.
<a href="#">pub.security.keystore.pkcs7:sign</a>	WmPublic. Creates a PKCS7 signed Data object.
<a href="#">pub.security.outboundPasswords:setPassword</a>	WmPublic. Stores a key and password in the password store.
<a href="#">pub.security.outboundPasswords:getPassword</a>	WmPublic. Retrieves a password from the password store for a given key.
<a href="#">pub.security.outboundPasswords:listKeys</a>	WmPublic. Lists the keys in the password store.
<a href="#">pub.security.outboundPasswords:removePassword</a>	WmPublic. Removes a password from the password store for a given key.
<a href="#">pub.security.outboundPasswords:updatePassword</a>	WmPublic. Changes the password value for a key already in the password store.
<a href="#">pub.security.pkcs7:sign</a>	WmPublic. Creates a PKCS7 SignedData object.
<a href="#">pub.security.pkcs7:verify</a>	WmPublic. Processes a digital signature to guarantee that the data associated with the signature has not been modified.
<a href="#">pub.security:userInfoSpec</a>	WmPublic. Specification for the signature of a UserInfo service that performs custom processing based on

Element	Package and Description
	the personally identifiable information in the OpenID Connect UserInfo token returned from the OpenID Provider
<a href="#">pub.security.util:createMessageDigest</a>	WmPublic. Generates a message digest for a given message.
<a href="#">pub.security.util:getCertificateInfo</a>	WmPublic. Retrieves information such as serial number, issuer, and expiration date from a digital certificate.
<a href="#">pub.security.util:loadPKCS7CertChain</a>	WmPublic. Converts a certificate chain that is in PKCS #7 format to a list of byte arrays.
<a href="#">pub.security.util:createSecureString</a>	WmPublic. Creates a WmSecureString object from either a Java String, byte array, or character array.
<a href="#">pub.security.util:convertSecureString</a>	WmPublic. Returns a WmSecureString in Java String, byte array, or character array format.
<a href="#">pub.security.util:destroySecureString</a>	WmPublic. Destroys a WmSecureString such that it no longer resides in memory and is removed from the pipeline.
<a href="#">pub.security.xml:decryptXML</a>	WmPublic. Decrypts the encrypted XML, and returns the XML as either a string or stream object.
<a href="#">pub.security.xml:encryptXML</a>	WmPublic. Encrypt an XML document or node in an XML document.
<a href="#">pub.security.xml:signXML</a>	WmPublic. Digitally sign an outgoing XML node or document.
<a href="#">pub.security.xml:verifyXML</a>	WmPublic. Verifies a signed XML document, or node in an XML document, and returns information

---

Element	Package and Description
	about the success or failure of the verification.

---

## pub.security:clearKeyAndChain

WmPublic. Clears the set key and certificate chain and reverts back to the default key and certificate chain for the subsequent set of invoked services.

### Input Parameters

---

None.

### Output Parameters

---

None.

### Usage Notes

The following scenario describes a situation in which you would use the [pub.security.keystore:setKeyAndChain](#) and [pub.security:clearKeyAndChain](#) services.

Company A has a webMethods Integration Server with one certificate chain. Company A wants to start trading with two new companies: Company B and Company C. Due to explicit business decisions, both Company B and Company C require that secure requests to their servers use certificates issued by their company's certificate authority. Company A now has three certificate sets that it must manage: one for connections to B, one for connections to C, and one for all other requests. Below is a high-level process flow of what Company A would do if documents needed to be forwarded to companies B, C, and D (some arbitrary partner without the stringent security).

Assume all network communication is done using HTTPS. Documents are sent to the companies in the following order: Company D, Company B, Company C, Company D. All data transfers make use of the [pub.client:http](#) service.

1. Invoke [pub.client:http](#) to send data to Company D.
2. Invoke [pub.security.keystore:setKeyAndChain](#) using the key and certificate chain for Company B.
3. Invoke [pub.client:http](#) to send data to Company B.
4. Invoke [pub.security.keystore:setKeyAndChain](#) using the key and certificate chain for Company C.
5. Invoke [pub.client:http](#) to send data to Company C.
6. Invoke [pub.security:clearKeyAndChain](#) to revert back to the default key and certificate chain for Company A's server.

7. Invoke [pub.client:http](#) to send data to Company D.

### See Also

[pub.security.keystore:setKeyAndChain](#)

---

## pub.security:setKeyAndChain

WmPublic. *Deprecated* - Replaced by [pub.security.keystore:setKeyAndChain](#).

Associates a key and certificate chain with the subsequent set of invoked services. Use this service to associate a key and certificate chain that is different from the default settings, and if your key and certificate information is located in files (rather than byte arrays).

### Input Parameters

---

<i>privKeyFile</i>	<b>String</b> Absolute (for example, D:\certs\cert1.der) or relative path of the file containing the private key. A relative path is the path relative to the directory from which the Integration Server has been started (for example, <i>Integration Server_directory</i> \instances\ <i>instance_name</i> \config \certs\cert1.der).
<i>certFiles</i>	<b>String List</b> of file names containing the certificates that comprise the certificate chain. The list should start with the user's certificate followed by (in order) intermediate certificates and the root CA certificate.  Absolute or relative paths of the files can be specified.

### Output Parameters

---

None.

---

## pub.security:setKeyAndChainFromBytes

WmPublic. Associates a key and certificate chain with the subsequent set of invoked services. Use this service to associate a key and certificate chain that is different from the default settings, and if your key and certificate information is located in byte arrays (rather than files).

---

## Input Parameters

<i>provoke</i>	<b>Object</b> A byte array containing the client's private key.
<i>certs</i>	<b>Object List</b> List of byte arrays containing the client's certificate chain. The list should start with the user's certificate followed by (in sequence) intermediate certificates and the root CA certificate.

---

## Output Parameters

None.

## Usage Notes

To enable this service to work properly if you use the FTPS protocol, you must set the *secure* parameter to `True` in the [pub.client:http](#) and [pub.client:ftp:login](#) services.

You can use `pub.security:clearKeyAndChain` with `pub.security:setKeyAndChainFromBytes`. See the Usage Notes for [pub.security:clearKeyAndChain](#) for more information about using the `pub.security:setKeyAndChainFromBytes` service.

---

## pub.security.enterpriseGateway:alertSpec

WmPublic. Specification for flow services used to send alerts about violations of webMethods Enterprise Gateway rules.

Use this specification as the signature of the flow service that Integration Server invokes when a request violates an Enterprise Gateway rule. For more information about webMethods Enterprise Gateway, see *webMethods Integration Server Administrator's Guide*.

---

## Input Parameters

<i>ruleName</i>	<b>String</b> The Enterprise Gateway rule that the request violates.				
<i>alertInfo</i>	<b>Document List</b> Information for Enterprise Gateway alert notification.				
	<table> <thead> <tr> <th><u>Key</u></th> <th><u>Description</u></th> </tr> </thead> <tbody> <tr> <td><i>alertAction</i></td> <td><b>String</b> The action that the Integration Server functioning as the Enterprise Gateway Server takes when a request violates an Enterprise Gateway rule.</td> </tr> </tbody> </table>	<u>Key</u>	<u>Description</u>	<i>alertAction</i>	<b>String</b> The action that the Integration Server functioning as the Enterprise Gateway Server takes when a request violates an Enterprise Gateway rule.
<u>Key</u>	<u>Description</u>				
<i>alertAction</i>	<b>String</b> The action that the Integration Server functioning as the Enterprise Gateway Server takes when a request violates an Enterprise Gateway rule.				

---

	<p>The action will have one of these values:</p> <ul style="list-style-type: none"> <li>■ DENY if the server denies the request and issues alerts as configured.</li> <li>■ ALERT if the server allows the request and issues alerts as configured.</li> </ul>
<i>requestType</i>	<p><b>String</b> The specific request type to which the server applies the Enterprise Gateway rule. The values are:</p> <ul style="list-style-type: none"> <li>■ ALL if the rule applies to all requests.</li> <li>■ SOAP if the rule applies to SOAP requests only.</li> <li>■ REST if the rule applies to REST requests only.</li> <li>■ INVOKE if the rule applies to INVOKE requests only.</li> </ul>
<i>filterName</i>	<p><b>String</b> The filter that the request satisfies. The values are:</p> <ul style="list-style-type: none"> <li>■ DoSFilter if the request satisfies the Denial of Service settings specified for Enterprise Gateway rules.</li> <li>■ MsgSizeLimitFilter if the request satisfies the Message Size Limit filter in the rule.</li> <li>■ OAuthFilter if the request satisfies the OAuth filter in the rule.</li> <li>■ mobileAppProtectionFilter if the request satisfies the Mobile Application Protection filter in the rule.</li> <li>■ None if the request violates a rule that has no filters.</li> </ul>
<i>message</i>	<p><b>String</b> The details about the filter condition that the request satisfies.</p>
<i>requestUser</i>	<p><b>String</b> The user ID that sent the request that violates the Enterprise Gateway rule. This field is empty if authentication is disabled on</p>

Enterprise Gateway Server. For more information about authentication on Enterprise Gateway Server, see *webMethods Integration Server Administrator's Guide*.

<i>requestHost</i>	<b>String</b> The host name or IP address of the client that sent the request.
<i>requestTime</i>	<b>String</b> Date and time at which the request reached Enterprise Gateway Server. This parameter is in the format <code>yyyy-MM-dd HH:mm:ss z</code> , where <code>z</code> indicates the time zone.
<i>resourcePath</i>	<b>String</b> The path of the resource to be accessed by the request.
<i>serverHost</i>	<b>String</b> The host name or IP address of the Enterprise Gateway Server that received the client request.
<i>serverPort</i>	<b>String</b> The number of the Enterprise Gateway Server port that received the client request.

### Output Parameters

---

None.

## pub.security.enterpriseGateway:customFilterSpec

---

WmPublic. Specification for services that can be invoked by the custom filter in Enterprise Gateway rules.

Using the service that has the `pub.security.enterpriseGateway:customFilterSpec` specification as its signature, you can extract the HTTP headers and payload from a request and act on it as per your business requirements. Upon processing the headers, you can choose to forward the request to the Internal Server or deny the request and return an error message to the user. For more information about webMethods Enterprise Gateway, see *webMethods Integration Server Administrator's Guide*.

### Input Parameters

---

<i>requestHeaders</i>	<b>Document</b> IData object containing the standard HTTP headers in the request along with any custom headers that are present
-----------------------	---

in the request. The following are the standard headers that will be extracted from the request: `User-Agent`, `Accept`, `Host`, `Authorization`, `Cookie`, `Content-Type`, `Accept-Language`, and `Content-Length`.

*payload*                    **Object** Object containing the payload in the request.

---

### Output Parameters

---

*forwardRequest*            **java.lang.Boolean** Flag indicating whether Enterprise Gateway Server forwards the request to the Internal Server. A value of:

- `True` indicates that the request will be forwarded to the Internal Server.
- `False` indicates that the request will be denied and an error message is returned to the user.

---

## pub.security.keystore:getCertificate

`WmPublic`. Returns the trusted certificate, stored in a truststore, that corresponds to the certificate's alias.

---

### Input Parameters

---

*trustStoreAlias*            **String** Alias for the truststore containing the certificate.

*certAlias*                    **String** Alias identifying a particular trusted certificate within a truststore.

---

### Output Parameters

---

*certificate*                    **byte[ ]** A byte array containing the trusted certificate.

### Usage Notes

For information about using aliases for keystores, truststores, and private keys, see *webMethods Integration Server Administrator's Guide*

---

## pub.security.keystore:getKeyAndChain

WmPublic. Returns a private key and its associated certificate chain from a designated keystore.

### Input Parameters

---

<i>keyStoreAlias</i>	<b>String</b> Alias for the keystore that contains the private key of interest and its certificate.
<i>keyAlias</i>	<b>String</b> Alias for the private key stored in the specified keystore.

### Output Parameters

---

<i>privateKey</i>	<b>java.security.PrivateKey</b> Object representing the private key.
<i>certChain</i>	<b>byte[ ][ ]</b> List of byte arrays representing the certificate chain associated with the private key.

### Usage Notes

For information about using aliases for keystores, truststores, and private keys, see *webMethods Integration Server Administrator's Guide*.

---

## pub.security.keystore:getTrustedCertificates

WmPublic. Returns the trusted certificates located in a specified truststore.

### Input Parameters

---

<i>trustStoreAlias</i>	<b>String</b> Name of the truststore alias.
------------------------	---

### Output Parameters

---

<i>certificates</i>	<b>byte[ ][ ]</b> Trusted certificates, as a list of byte arrays.
---------------------	---

### Usage Notes

For information about using aliases for keystores, truststores, and private keys, see *webMethods Integration Server Administrator's Guide*.

---

## pub.security.keystore:setKeyAndChain

WmPublic. Associates a key and certificate chain with the subsequent set of invoked services. Use this service to associate a key and certificate chain that is different from the default settings, and if your key and certificate information is stored in a keystore file.

### Input Parameters

---

<i>keyStoreAlias</i>	<b>String</b> Name of the keystore alias.
<i>keyAlias</i>	<b>String</b> Alias of the private key located in the keystore.

### Output Parameters

---

None.

### Usage Notes

This service replaces [pub.security:setKeyAndChain](#), which is deprecated.

For information about using aliases for keystores, truststores, and private keys, see *webMethods Integration Server Administrator's Guide*

### See Also

[pub.security:clearKeyAndChain](#)

---

## pub.security.keystore.pkcs7:sign

WmPublic. Creates a PKCS7 signed Data object.

### Input Parameters

---

*signerInfo* **Document List** Information about a single signer of the signed data object. Each *signerInfo* requires either a certificate chain and a private key or a key alias that references them.

<b>Key</b>	<b>Description</b>
<i>keyStoreAlias</i>	<b>String</b> Name of the keystore alias.
<i>keyAlias</i>	<b>String</b> Alias of the private key.

<i>hashAlgorithm</i>	<p><b>String</b> The algorithm to use when computing the digest of the provided data. Specify:</p> <ul style="list-style-type: none"> <li>■ MD5 (the default)</li> <li>■ SHA-1</li> <li>■ SHA-256</li> <li>■ SHA-384</li> <li>■ SHA-512</li> </ul>
<i>data</i>	<p><b>byte[]</b> Optional. Data in the form of a byte array to be digitally signed.</p> <p><b>Note:</b> If both <i>data</i> and <i>dataAsStream</i> are provided, <i>dataAsStream</i> takes precedence.</p>
<i>dataAsStream</i>	<p>java.io.InputStream Optional. Data in the form of an input stream to be digitally signed.</p> <p><b>Note:</b> If both <i>data</i> and <i>dataAsStream</i> are provided, <i>dataAsStream</i> takes precedence.</p>
<i>detachedSignature</i>	<p><b>String</b> Flag specifying whether to generate a detached signature. A detached signature does not include the data that was signed. Set to:</p> <ul style="list-style-type: none"> <li>■ <code>true</code> to generate a detached signature.</li> <li>■ <code>false</code> to generate an implicit signature (one that includes the signed data). This is the default.</li> </ul>

## Output Parameters

---

<i>signature</i>	<p><b>byte[]</b> Conditional. Signature generated in the form of a byte array from the supplied data. This is a DER-encoded representation of the SignedData object as specified in PKCS#7. <i>signature</i> is returned when the input parameter <i>data</i> is provided.</p>
<i>signatureAsStream</i>	<p>java.io.OutputStream Conditional. Signature generated in the form of an output stream from the supplied data. <i>signatureAsStream</i> is returned when the input parameter <i>dataAsStream</i> is provided.</p>

### Usage Notes

This service supersedes [pub.security.pkcs7:sign](#), which is deprecated.

For information about using aliases for keystores, truststores, and private keys, see *webMethods Integration Server Administrator's Guide*.

---

## pub.security.outboundPasswords:setPassword

WmPublic. Stores a key and password in the password store.

### Input Parameters

---

<i>key</i>	<b>String</b> Key to be associated with the password entry.
<i>value</i>	<b>WmSecureString</b> Password to be stored.
<i>isInternal</i>	<b>String</b> "true" if this should be saved as an internal password; "false" if it should be saved as a public password. Default is "false". (See " <a href="#">Internal and Public Passwords</a> " on page 708 for more information.)

### Output Parameters

---

<i>result</i>	<b>String</b> "true" if password was successfully stored; "false" otherwise.
<i>message</i>	<b>String</b> "successful" or reason for failure.

### Usage Notes

This is the basic process a flow service should follow to store an outbound password:

1. Call [pub.security.util:createSecureString](#) to create a WmSecureString object containing the password to be stored.

For security reasons, the flow service should be run manually requiring an authorized person to type the password to be stored. This will eliminate the need to save the password on disk in an unencrypted format.

2. Call [pub.security.outboundPasswords:setPassword](#) to save the password in encrypted form in the outbound password store.

The [pub.security.outboundPasswords:setPassword](#) service requires a key to be supplied which is basically a key to the password. This key must be saved in some way; any

flow service wishing to use the password to access a secure resource will need to supply the key to retrieve the password from the outbound password store.

3. Once the password is successfully stored, call `pub.security.util:destroySecureString` to remove the password from memory.

### Internal and Public Passwords

Internal passwords are passwords for use by the Integration Server itself to access secure resources (e.g., remote Integration Servers, JDBC connection pools, LDAP servers, etc.). Internal passwords are managed using the Integration Server Administrator and are stored in the outbound password store. Flow services are also allowed to store passwords in the outbound password store. However, by default, passwords stored by a flow service are considered "public," as opposed to internal. This distinction allows flow services to use the outbound password store as a secure mechanism for storing and retrieving passwords, but protects the Integration Server's internal passwords.

When calling any of the `pub.security.outboundPasswords` services (i.e. `setPassword`, `getPassword`, `listKeys`, `removePassword`, and `updatePassword`) the `isInternal` input parameter indicates whether the service is working with internal or public passwords. Note that even if this parameter is set to "true", you cannot access internal passwords if the Integration Server is configured to deny access to internal passwords. Access to internal passwords is controlled by the `watt.security.ope.AllowInternalPasswordAccess` configuration parameter on the Integration Server; for more information see *webMethods Integration Server Administrator's Guide*.

---

## pub.security.outboundPasswords:getPassword

WmPublic. Retrieves a password from the password store for a given key.

### Input Parameters

---

<i>key</i>	<b>String</b> Key of the password entry to be retrieved.
<i>isInternal</i>	<b>String</b> "true" if this is an internal password; "false" if it is public. By default, this is "false". If you specify incorrectly whether the password is internal or public, the retrieve operation will fail. (For more information about internal and public passwords, see " <a href="#">Internal and Public Passwords</a> " on page 708.)

### Output Parameters

---

<i>value</i>	<b>WmSecureString</b> Value of the retrieved password.
--------------	--

---

<i>result</i>	<b>String</b> "true" if the password value was successfully retrieved; "false" otherwise.
<i>message</i>	<b>String</b> "successful" or reason for failure.

### Usage Notes

This is the basic process a flow service should follow to retrieve an outbound password:

1. Call [pub.security.outboundPasswords:getPassword](#) with the key to the password to be retrieved.

If the key is unknown, you can call [pub.security.outboundPasswords:listKeys](#) to retrieve a list of keys currently in the outbound password store.

The [pub.security.outboundPasswords:getPassword](#) service returns a `WmSecureString` object containing the retrieved password.

2. Call [pub.security.util:convertSecureString](#) to convert the password to a usable format.

The password can then be passed to the authenticating mechanism of the secure resource.

3. When done accessing the secure resource, call [pub.security.util:destroySecureString](#) to remove the password from memory.

---

## pub.security.outboundPasswords:listKeys

WmPublic. Lists the keys in the password store.

### Input Parameters

---

<i>isInternal</i>	<b>String</b> "true" if you want keys for internal passwords; "false" if you want keys for public passwords. By default this is "false". (For more information about internal and public passwords, see <a href="#">"Internal and Public Passwords" on page 708.</a> )
-------------------	--

### Output Parameters

---

<i>key</i>	<b>IData</b> List of keys in the password store.
<i>result</i>	<b>String</b> "true" if the list of keys was successfully retrieved; "false" otherwise.

---

## pub.security.outboundPasswords:removePassword

WmPublic. Removes a password from the password store for a given key.

### Input Parameters

---

<i>key</i>	<b>String</b> Key of the password to be removed.
<i>isInternal</i>	<b>String</b> "true" if this is an internal password; "false" if it is public. By default, this is "false". If you specify incorrectly whether the password is internal or public, the remove operation will fail. (For more information about internal and public passwords, see <a href="#">"Internal and Public Passwords" on page 708.</a> )

### Output Parameters

---

<i>result</i>	<b>String</b> "true" if the password was successfully removed; "false" otherwise.
<i>message</i>	<b>String</b> "successful" or reason for failure.

---

## pub.security.outboundPasswords:updatePassword

WmPublic. Changes the password value for a key already in the password store.

### Input Parameters

---

<i>key</i>	<b>String</b> Key of the password to be updated.
<i>newPassword</i>	<b>WmSecureString</b> New password value for the key.
<i>isInternal</i>	<b>String</b> "true" if this is an internal password; "false" if it is public. By default, this is "false". If you specify incorrectly whether the password is internal or public, the update operation will fail. (For more information about internal and public passwords, see <a href="#">"Internal and Public Passwords" on page 708.</a> )

---

## Output Parameters

---

<i>result</i>	<b>String</b> "true" if the password value was successfully changed; "false" otherwise.
<i>message</i>	<b>String</b> "successful" or reason for failure.

---

## pub.security.pkcs7:sign

WmPublic. *Deprecated* - Replaced by [pub.security.keystore.pkcs7:sign](#).

Creates a PKCS7 SignedData object.

This service enables multiple entities to sign the specified data. Each *signerInfo* block contained in the resulting signature contains two authenticated attributes: the content type and a timestamp.

## Input Parameters

---

*signerInfo* **Document List** Information about a single signer of the signed data object. Each *signerInfo* requires either a certificate chain and a private key or a key alias that references them.

<b>Key</b>	<b>Description</b>
<i>certChain</i>	<b>java.security.cert.X509Certificate[ ] or byte[ ][ ]</b> Certificate chain of the signer. The subject that is performing the signature should be the first certificate in this chain, while the root Certifying Authority should be the last. The key provided should correspond to the public key contained in the first certificate of the chain.
<i>key</i>	<b>java.security.PrivateKey or byte[ ]</b> Private key that will be used to digitally sign the data. The private key can be any asymmetric encryption key that is supported by the webMethods Integration Server (for example, DSA or RSA).

---

<i>keyAlias</i>	<b>String</b> Alias of the certificate chain and private key in the key store. This key is not currently used.
<i>hashAlgorithm</i>	<b>String</b> The algorithm to use when computing the digest of the provided data (SHA-1 or MD5). The default value is MD5.
<i>data</i>	<b>byte[ ]</b> Data to be digitally signed.
<i>detachedSignature</i>	<b>String</b> Flag specifying whether to generate a detached signature. A detached signature does not include the data that was signed. Set to: <ul style="list-style-type: none"><li>■ <code>true</code> to generate a detached signature.</li><li>■ <code>false</code> to generate an implicit signature (one that includes the signed data). This is the default.</li></ul>

### Output Parameters

---

<i>signature</i>	<b>byte[ ]</b> Signature generated from the supplied data. This is a DER-encoded representation of the SignedData object as specified in PKCS#7.
------------------	--

### Usage Notes

This service is superseded by [pub.security.keystore.pkcs7:sign](#).

---

## pub.security.pkcs7:verify

WmPublic. Processes a digital signature to guarantee that the data associated with the signature has not been modified.

### Input Parameters

---

<i>signature</i>	<b>byte[ ]</b> Optional. Signature to use to determine whether the signed data is intact (a DER-encoded representation of the SignedData object as specified in PKCS#7) where the signature is in the form of a byte array. If you are processing a detached signature, pass the signature in <i>signature</i> . If you are processing an implicit signature, pass the entire signed message in <i>signature</i> .
------------------	--

	<p><b>Note:</b> If both <i>signature</i> and <i>signatureAsStream</i> are provided, <i>signatureAsStream</i> takes precedence.</p>
<i>signatureAsStream</i>	<p>java.io.InputStream Optional. Signature to use to determine whether the signed data is intact where the signature is in the form of an input stream.</p>
	<p><b>Note:</b> If both <i>signature</i> and <i>signatureAsStream</i> are provided, <i>signatureAsStream</i> takes precedence.</p>
<i>data</i>	<p><b>byte[ ]</b> Optional. Data in the form of a byte array that was signed. If you are processing a detached signature, you must supply <i>data</i> . If you are only processing an implicitly signed data as a byte array, you do not need to supply <i>data</i> because both the data and the signature reside in <i>signature</i> . If you are processing a detached signature and an implicitly signed data as a byte array, you must supply <i>data</i> .</p>
	<p><b>Note:</b> If both <i>data</i> and <i>dataAsStream</i> are provided, <i>dataAsStream</i> takes precedence.</p>
<i>dataAsStream</i>	<p>java.io.InputStream Optional. Data in the form of an input stream that was signed. If you are processing a detached signature, you must supply <i>dataAsStream</i> . If you are only processing an implicitly signed data as an input stream, you do not need to supply <i>dataAsStream</i> because both the data and the signature reside in <i>signatureAsStream</i> . If you are processing a detached signature and an implicitly signed data as an input stream, you must supply <i>dataAsStream</i> .</p>
	<p><b>Note:</b> If both <i>data</i> and <i>dataAsStream</i> are provided, <i>dataAsStream</i> takes precedence.</p>
<i>detachedSignature</i>	<p><b>String</b> Optional. Flag indicating whether the message has a detached signature. Set to:</p> <ul style="list-style-type: none"> <li>■ <code>true</code> when the message has a detached signature.</li> <li>■ <code>false</code> when the message has an implicit signature. This is the default.</li> </ul>
	<p><b>Note:</b></p> <ul style="list-style-type: none"> <li>■ If <i>detachedSignature</i> is <code>true</code> and <i>signature</i> is provided, then <i>data</i> should be provided.</li> <li>■ If <i>detachedSignature</i> is <code>true</code> and <i>signatureAsStream</i> is provided, then <i>dataAsStream</i> should be provided.</li> </ul>

*signerCertChain* **byte[ ][ ]** Optional. Certificate chains of the parties that signed the message.

**Note:** If the signers included the certificate chain with the digital signature, you do not need to supply *signerCertChain*.

## Output Parameters

*content* **byte[ ]** Conditional. The data (for example, the document that was originally signed) extracted in the form of a byte array from an implicit signature. If you are verifying a detached signature, *content* is not returned.

**Note:** The extracted data is returned in *content* even if signature verification fails.

*content* is returned when the input parameter *data* is provided.

*contentAsStream* **java.io.OutputStream** Conditional. The data extracted in the form of an output stream from an implicit signature. *contentAsStream* is returned when the input parameter *dataAsStream* is provided.

*signerInfo* **Document List** Information about the signers. Each document in the list provides the following information about a single signer:

Key	Description
<i>certChain</i>	<b>java.security.cert.X509Certificate[ ]</b> Certificate chain of the signer. The chain will appear in hierarchical order, starting with the signer's X.509 certificate in element 0.
<i>timeStamp</i>	<b>java.util.Date</b> Time at which the signer signed the data.
<i>trusted</i>	<b>String</b> Flag indicating whether the certificate chain presented by the signer is trusted. A value of: <ul style="list-style-type: none"> <li>■ <code>true</code> indicates that the chain is trusted.</li> </ul>

- false indicates that the chain is not trusted.

*status*

**String** Flag indicating whether the signatures were successfully verified. If successful, *status* contains `verified`. If the signatures were not successfully verified, *status* contains an error message.

---

## pub.security:userInfoSpec

WmPublic. Specification for the signature of a UserInfo service that performs custom processing based on the personally identifiable information in the OpenID Connect UserInfo token returned from the OpenID Provider.

### Input Parameters

---

*userInfoClaims*

**Document** Optional. A document (IData) containing the claims from the UserInfo token returned by the OpenID Provider's UserInfo Endpoint.

*userInfoError*

**Document** Optional. A document (IData) containing the error when the OpenID Provider returns an unsuccessful response.

Integration Server redirects the User Agent to the OpenID Provider's Authorization Endpoint to authenticate the End User. If the OpenID Provider responds to that redirection with an error, Integration Server will populate *error* with the error information.

<u>Key</u>	<u>Description</u>
<i>statusCode</i>	<b>String</b> Optional. The HTTP status code returned by the OpenID Provider's UserInfo Endpoint.
<i>errorType</i>	<b>String</b> The type of error returned by the OpenID Provider's UserInfo Endpoint. The possible values are defined in section 4.1.2.1 of the OAuth 2.0 specification.
<i>errorDescription</i>	<b>String</b> Optional. A description of the error returned by the OpenID Provider's UserInfo Endpoint.

---

## Output Parameters

---

*userInfoResponse*

**Document** A document (IData) for the result of the UserInfo service.

### Key

### Description

*rejectRequest*

**java.lang.Boolean** Whether or not to reject the request.

- True instructs Integration Server to reject the request.

Integration Server does not access the requested resource in the original request from the User Agent. Processing of the request stops and a response is sent to the User Agent. Integration Server uses the contents of *rejectRequest*, *statusCode*, *statusCode* and *responseEntity* in the response.

- False instructs Integration Server to accept the request.

Integration Server continues processing of the request by selecting an OpenID Provider User and setting the user for the current session. Integration Server sends the User Agent an HTTP 302 response with a session cookie, redirecting the User Agent back to the originally requested resource.

*statusCode*

**java.lang.Integer** Conditional. HTTP status code to be returned in the response sent to the User Agent.

*statusCode* is required when *rejectRequest* is set to false.

If *rejectRequest* is true, *statusCode* is not returned to the User Agent.

*reasonPhrase*

**String** Conditional. HTTP reason phrase to be included in the response sent to the User Agent.

*reasonPhrase* is optional when *rejectRequest* is set to false.

If *rejectRequest* is true, *reasonPhrase* is not returned to the User Agent.

*responseHeaders*

**Document List** Conditional. A document list (IData []) containing the header fields to set in the response sent to the User Agent.

*responseHeaders* is optional when *rejectRequest* is set to false.

If *rejectRequest* is true, *responseHeaders* is not returned to the User Agent.

<u>Key</u>	<u>Description</u>
<i>fieldName</i>	<b>String</b> Name of the header field to set.
<i>fieldValue</i>	<b>String</b> Value of the header field to set.

*responseEntity*

**String** Conditional. Body of the response to be sent to the User Agent.

*responseEntity* is optional when *rejectRequest* is set to false.

If *rejectRequest* is true, *responseEntity* is not returned to the User Agent.

## Usage Notes

The `pub.security.userInfoSpec` must be used as the signature for any service used as a UserInfo service. A UserInfo service performs custom processing based on the personally identifiable information in the OpenID Connect UserInfo token returned from the OpenID Provider. The logic of the UserInfo service must determine whether to accept or reject the original resource request made by the User Agent.

Integration Server can only reject a request using a UserInfo service if the UserInfo service is invoked synchronously. When registering the UserInfo service with the OpenID Provider, make sure to register the service as synchronous if you want to be able to reject a User Agent request based on the output of the UserInfo service.

Integration Server populates *userInfoClaims* or *error* depending on the response from the OpenID Provider's UserInfo Endpoint.

---

## pub.security.util:createMessageDigest

WmPublic. Generates a message digest for a given message.

### Input Parameters

---

*algorithm* **String** Name of the algorithm that you want to use to compute the message digest. Must be one of the following: MD5, SHA-1, SHA-256, SHA-384, or SHA-512.

*input* **byte[ ]** Optional. Message for which you want the digest generated where the message is in the form of a byte array.

**Note:** If both *input* and *inputAsStream* are provided, *inputAsStream* takes precedence.

*inputAsStream* **java.io.InputStream** Optional. Message for which you want to generate a message digest where the message is in the form of an input stream.

**Note:** If both *input* and *inputAsStream* are provided, *inputAsStream* takes precedence.

### Output Parameters

---

*output* **byte[ ]** Conditional. Computed digest in the form of a byte array. *output* is returned when the input parameter *input* is provided.

*outputAsStream* **OutputStream** Conditional. Computed digest in the form of an output stream. *outputAsStream* is returned when the input parameter *inputAsStream* is provided.

---

## pub.security.util:getCertificateInfo

WmPublic. Retrieves information such as serial number, issuer, and expiration date from a digital certificate.

---

## Input Parameters

---

*certificate*      **byte[] java.security.cert.X509Certificate** The certificate whose information you want to retrieve.

## Output Parameters

---

*info*      **Document** Information from the certificate.

<u>Key</u>	<u>Description</u>						
<i>version</i>	<b>java.lang.Number</b> X509 certificate version number.						
<i>serialNumber</i>	<b>String</b> Serial number of the certificate.						
<i>signature</i>	<b>String</b> Signature algorithm used by the issuer to sign this certificate.						
<i>issuer</i>	<b>Document</b> Detailed information about the CA that signed the certificate, such as name, location, and e-mail address.						
<i>validity</i>	<b>Document</b> The time period over which the certificate is valid.						
	<table> <thead> <tr> <th><u>Key</u></th> <th><u>Description</u></th> </tr> </thead> <tbody> <tr> <td><i>notBefore</i></td> <td><b>String</b> First date on which this certificate is valid (for example, 3/15/00 3:36PM).</td> </tr> <tr> <td><i>notAfter</i></td> <td><b>String</b> Last date on which this certificate is valid (for example, 3/15/00 3:36PM).</td> </tr> </tbody> </table>	<u>Key</u>	<u>Description</u>	<i>notBefore</i>	<b>String</b> First date on which this certificate is valid (for example, 3/15/00 3:36PM).	<i>notAfter</i>	<b>String</b> Last date on which this certificate is valid (for example, 3/15/00 3:36PM).
<u>Key</u>	<u>Description</u>						
<i>notBefore</i>	<b>String</b> First date on which this certificate is valid (for example, 3/15/00 3:36PM).						
<i>notAfter</i>	<b>String</b> Last date on which this certificate is valid (for example, 3/15/00 3:36PM).						
<i>subject</i>	<b>Document</b> Detailed information about the owner of the certificate, such as name, location, and mail address.						

<i>subjectPublicKey Algorithm</i>	<b>String</b> Encryption algorithm with which the certificate's key is designed to be used (for example, RSA or DSA).
---------------------------------------	---

---

## pub.security.util:loadPKCS7CertChain

WmPublic. Converts a certificate chain that is in PKCS #7 format to a list of byte arrays.

### Input Parameters

---

<i>certificateChain</i>	<b>byte[ ]</b> The certificate chain in PKCS #7 format.
-------------------------	---

### Output Parameters

---

<i>certificates</i>	<b>byte[ ][ ]</b> List of byte arrays in which each byte[ ] in the list contains a certificate from <i>certificateChain</i> .
---------------------	---

---

## pub.security.util:createSecureString

WmPublic. Creates a WmSecureString object from either a Java String, byte array, or character array.

WmSecureString is a mutable alternative to Java String. It allows the characters in the string to be explicitly removed from memory. Any password you wish to store in the Integration Server's outbound password store must be converted to a WmSecureString.

### Input Parameters

---

<i>string</i>	<b>String</b> Java String to made into a WmSecureString.
<i>bytes</i>	<b>byte[ ]</b> Byte array to be made into a WmSecureString.
<i>chars</i>	<b>char[ ]</b> Character array to be made into a WmSecureString.
<i>encoding</i>	<b>String</b> If a byte array is supplied as an input parameter, <i>encoding</i> specifies the Java encoding of the byte array. This may be any encoding supported by Java String. By default, if no encoding is specified, then the default JVM encoding is used.

---

### Output Parameters

---

*secureString*      **WmSecureString** WmSecureString created from the supplied input parameters.

### Usage Notes

Only one of the input parameters (i.e. *string*, *bytes*, or *chars*) may be specified. If more than one is specified, an exception will be thrown. An exception is also thrown if none of these is specified.

---

## pub.security.util:convertSecureString

WmPublic. Returns a WmSecureString in Java String, byte array, or character array format.

### Input Parameters

---

*secureString*      **WmSecureString** WmSecureString to be converted.

*returnAs*            **String** Format into which the WmSecureString is to be converted. Valid options are byte[], char[], and Java String. If a value for this parameter is not specified, the default is to convert the WmSecureString to a String.

### Output Parameters

---

*string*              **String** The WmSecureString converted to a Java String.

*bytes*                **byte[]** The WmSecureString converted to a native Java byte array.

*chars*                **char[]** The WmSecureString converted to a native Java character array.

---

## pub.security.util:destroySecureString

WmPublic. Destroys a WmSecureString such that it no longer resides in memory and is removed from the pipeline.

---

### Input Parameters

---

*secureString*      **WmSecureString** WmSecureString to be destroyed.

### Output Parameters

---

None.

---

## pub.security.xml:decryptXML

WmPublic. Decrypts the encrypted XML, and returns the XML as either a string or stream object.

---

### Input Parameters

---

*xmlData*      **String** Optional. Encrypted XML that needs to be decrypted as plain text.

*xmlStream*      **InputStream** Optional. Encrypted XML in the form of an input stream.

**Note:** If both *xmlData* and *xmlStream* are provided, *xmlStream* takes precedence; Integration Server uses the *xmlStream* value and returns only *decryptedXMLStream* .

*keyStoreAlias*      **String** Optional. Alias of the keystore that contains the private key used for decryption.

*keyAlias*      **String** Optional. Alias of the private key, contained in the keystore specified by the *keyStoreAlias* parameter, that is used for decryption.

*encoding*      **String** Optional. Specifies the encoding to use if the encoding cannot be extracted from the XML. If encoding is not specified in the XML document or in the *encoding* parameter, Integration Server uses UTF-8.

The *encoding* value must be a valid IANA encoding.

---

## Output Parameters

---

<i>decryptedXMLData</i>	<b>String</b> Conditional. Decrypted XML data. <i>decryptedXMLData</i> is returned when the input parameter <i>xmldata</i> is provided.
<i>decryptedXMLStream</i>	<b>Object</b> Conditional. A decrypted XML OutputStream object. <i>decryptedXMLStream</i> is returned when the input parameter <i>xmlStream</i> is provided.

## Usage Notes

There are several prerequisites to using `pub.security.xml:decryptXML`:

- Certificates must be configured for Integration Server and the client with which it is exchanging secure XML.
- The sending, encrypting client must have access to Integration Server's public key before the document exchange can occur.
- Integration Server stores its certificates in keystores and truststores. You must configure a keystore and truststore for Integration Server before using the XML encryption services.

You access the public and private keys for Integration Server through aliases. For information about Integration Server keystores and truststores, refer to *webMethods Integration Server Administrator's Guide*.

The `pub.security.xml:decryptXML` service works as follows:

1. The external system sends the XML document encrypted with the Integration Server's public key.
2. Integration Server receives the document and passes it to the XML service.
3. Integration Server uses the private key member of the key pair to decrypt the XML.
4. The decrypted XML is returned from the service.

If both *xmldata* and *xmlStream* are provided, *xmlStream* takes precedence; Integration Server uses the *xmlStream* value and returns only *decryptedXMLStream*.

*keyAlias* and *keyStoreAlias* should either both be provided or both be absent from the input. If no value is provided for these parameters, Integration Server uses the private key/certificate specified for the Decryption Key. If no value is specified for Decryption Key, Integration Server uses the SSL Key.

For information about configuring the Decryption Key and SSL Key keystore aliases, refer to *webMethods Integration Server Administrator's Guide*.

---

## pub.security.xml:encryptXML

WmPublic. Encrypt an XML document or node in an XML document.

### Input Parameters

---

<i>xmlData</i>	<b>String</b> Optional. The XML to be encrypted.
<i>xmlStream</i>	<b>InputStream</b> Optional. Input stream to the XML that needs to be encrypted.  <b>Note:</b> If both <i>xmlData</i> and <i>xmlStream</i> are provided, <i>xmlStream</i> takes precedence.
<i>nodeSelectors</i>	<b>String List</b> XPath to the node to be encrypted. If the value for this parameter is left empty, no XML will be encrypted.
<i>nsDecls</i>	<b>Document</b> Optional. Mapping of the namespace prefixes to the namespace URIs. The first column contains the prefixes and the second column contains the corresponding URIs.
<i>recipientID</i>	<b>String</b> Optional. Name of the client to which the XML will be sent. The user name and certificate must be configured with Integration Server certificate mapping. The client name entry is mapped to a valid X.509 certificate, and both are stored in Integration Server.  For information about Integration Server certificate mapping, see <i>webMethods Integration Server Administrator's Guide</i> .
<i>recipientCert</i>	<b>Byte[]</b> Optional. The certificate containing the public key that will be used to encrypt the XML. If the input parameters <i>recipientCert</i> and <i>recipientID</i> are both provided, <i>recipientCert</i> is used.
<i>contentOnly</i>	<b>Boolean</b> Optional. Indicates whether the XML tags surrounding the content will be encrypted along with the content. Set to: <ul style="list-style-type: none"><li>■ <code>true</code> to encrypt only the content.</li><li>■ <code>false</code> to encrypt both the tags and the content. This is the default.</li></ul>

<i>algorithm</i>	<p><b>String</b> Optional. The symmetric key algorithm to use for encryption. Set to:</p> <ul style="list-style-type: none"> <li>■ <code>tripleDES-cbc</code> for the algorithm at <a href="http://www.w3.org/2001/04/xmlenc#tripleDES-cbc">http://www.w3.org/2001/04/xmlenc#tripleDES-cbc</a></li> </ul> <p>This is the default.</p> <ul style="list-style-type: none"> <li>■ <code>aes256-cbc</code> for the algorithm at <a href="http://www.w3.org/2001/04/xmlenc#aes256-cbc">http://www.w3.org/2001/04/xmlenc#aes256-cbc</a></li> <li>■ <code>aes192-cbc</code> for the algorithm at <a href="http://www.w3.org/2001/04/xmlenc#aes192-cbc">http://www.w3.org/2001/04/xmlenc#aes192-cbc</a></li> <li>■ <code>aes128-cbc</code> for the algorithm at <a href="http://www.w3.org/2001/04/xmlenc#aes128-cbc">http://www.w3.org/2001/04/xmlenc#aes128-cbc</a></li> </ul> <div style="border: 1px solid #ccc; background-color: #f0f0f0; padding: 5px; margin-top: 10px;"> <p><b>Note:</b> If you are using <code>aes256-cbc</code> or <code>aes192-cbc</code> with JVM 1.6, make sure the unlimited policy jar files have been installed.</p> </div>
<i>encryptedKeyAlgorithm</i>	<p><b>String</b> Optional. The symmetric key that is randomly generated, and then encrypted with the receiver's public key. This encryption uses an asymmetric algorithm if public/private key pairs are being used. Set to:</p> <ul style="list-style-type: none"> <li>■ <code>rsa-1_5</code> for the algorithm at <a href="http://www.w3.org/2001/04/xmlenc#rsa-1_5">http://www.w3.org/2001/04/xmlenc#rsa-1_5</a></li> </ul> <p>This is the default.</p> <ul style="list-style-type: none"> <li>■ <code>rsa-oaep-mgf1p</code> for the algorithm at <a href="http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p">http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p</a></li> </ul>
<i>encoding</i>	<p><b>String</b> Optional. Specifies the encoding to use if the encoding cannot be extracted from the XML. If encoding is not specified in the XML document or in the <i>encoding</i> parameter, Integration Server uses UTF-8.</p> <p>The <i>encoding</i> value must be a valid IANA encoding.</p>

## Output Parameters

---

<i>encryptedXMLData</i>	<p><b>String</b> Conditional. Encrypted XML data. <i>encryptedXMLData</i> is returned when the input parameter <i>xmlData</i> is provided.</p>
<i>encryptedXMLStream</i>	<p><b>OutputStream</b> Conditional. Encrypted XML in the form of an OutputStream. <i>encryptedXMLStream</i> is returned when the input parameter <i>xmlStream</i> is provided.</p>

## Usage Notes

If both *xmlData* and *xmlStream* are provided, *xmlStream* takes precedence.

There are several prerequisites to using the `pub.security.xml:encryptXML` service:

- Certificates must be configured for Integration Server and the client with which it is exchanging encrypted XML.
- Before an encrypted XML document can be exchanged between Integration Server and an external system, the external system must share its public key.
- Prior to use of `pub.security.xml:encryptXML`, Integration Server must have access to the partner's public key. Such access is possible through:
  - An Integration Server certificate mapping (for information, refer to *webMethods Integration Server Administrator's Guide*).
  - A copy of the partner's X.509 certificate that is available to Integration Server.

In `pub.security.xml:encryptXML`, the certificate/public key is specified through one of the following input parameters: the client's name (through *recipientID*), or the public key of the partner application (through *recipientCert*).

Because encryption is a processing-intensive activity, it is recommended to only encrypt the XML nodes requiring protection.

## Signing and Encrypting the Same XML Document

You can use both encryption and signing in the same XML document.

- If you sign and encrypt *different* XML elements in a document, you can run either `pub.security.xml:signXML` or `pub.security.xml:encryptXML` first.
- Typically, if you sign and encrypt the *same* XML elements in a document, you should sign the elements before encrypting them. That is, invoke `pub.security.xml:signXML` before invoking `pub.security.xml:encryptXML`.

---

## pub.security.xml:signXML

WmPublic. Digitally sign an outgoing XML node or document.

### Input Parameters

---

<i>xmlData</i>	<b>String</b> Optional. XML that needs to be signed.
<i>xmlStream</i>	<b>InputStream</b> Optional. Input stream containing the XML that needs to be signed.

**Note:** If both *xmlData* and *xmlStream* are provided, *xmlStream* takes precedence.

<i>uri</i>	<p><b>String</b> Optional. URI to the element to be signed.</p> <p>In combination with the <i>nodeSelectors</i> parameter, the <i>uri</i> identifies the nodes to be signed.</p>
<i>noNamespaceSchemaLocation</i>	<p><b>String</b> Optional. A URI that identifies the location of the XML schema definition that contains the ID attribute specified in <i>uri</i>.</p> <p>Provide a <i>noNamespaceSchemaLocation</i> when specifying an ID attribute for <i>uri</i> and the ID attribute resides in an XML schema with no namespaces.</p>
<i>schemaLocations</i>	<p><b>Document</b> Optional. Document (IData) containing name-value pairs for the XML namespace and the location of the XML schema definition that contains element declarations, attribute declarations, and type definitions for that namespace.</p> <p>Provide a <i>schemaLocation</i> when specifying an ID attribute for <i>uri</i> and the ID attribute resides in an XML schema for a particular namespace.</p> <p>For example,</p> <ul style="list-style-type: none"> <li>■ XML namespace = http://www.w3schools.com</li> <li>■ XML schema definition location = file:C:/note.xsd</li> </ul>
<i>nodeSelectors</i>	<p><b>String List</b> XPath notation that identifies the nodes to be signed. The locations of the XPaths are not absolute, but relative, and work within the context of the node (an XPath Axes).</p> <p><b>Important</b> <i>not</i> use absolute location XPaths here.</p>
<i>nsDecls</i>	<p><b>Document</b> Optional. Mapping of the namespace prefixes to the namespace URIs. The first column contains the prefixes and the second column contains the corresponding URIs.</p>
<i>isEnveloped</i>	<p><b>String</b> Optional. Indicates whether the signature should be enveloped or enveloping. Set to:</p> <ul style="list-style-type: none"> <li>■ <code>True</code> to indicate the signature is enveloped. This is the default.</li> </ul>

---

	<ul style="list-style-type: none"><li>■ <code>False</code> to indicate the signature is enveloping.</li></ul>
<i>signatureNodeSelector</i>	<b>String</b> Optional. XPath to the node where the signature is entered. Applicable only for enveloped signatures. If no value is provided, the signature is placed as a first child of the root node.
<i>signatureAlgorithm</i>	<b>String</b> Optional. Signature algorithm to use when signing the XML node or document. Specify one of the following or use the default value (first algorithm): <ul style="list-style-type: none"><li>■ SHA1 (default)</li><li>■ SHA256</li><li>■ SHA384</li><li>■ SHA512</li></ul>
<i>digestAlgorithm</i>	<b>String</b> Optional. Digest algorithm to use when signing the XML node or document. Specify one of the following or use the default value (first algorithm): <ul style="list-style-type: none"><li>■ SHA1 (default)</li><li>■ SHA256</li><li>■ SHA384</li><li>■ SHA512</li></ul>
<i>canonicalizationAlgorithm</i>	<b>String</b> Optional. Canonical algorithm used with the XML. Specify one of the following or use the default value (first algorithm): <ul style="list-style-type: none"><li>■ <a href="http://www.w3.org/TR/2001/rec-xml-c14n-20010315">http://www.w3.org/TR/2001/rec-xml-c14n-20010315</a> (default)</li><li>■ <a href="http://www.w3.org/TR/2001/rec-xml-c14n-20010315#WithComments">http://www.w3.org/TR/2001/rec-xml-c14n-20010315#WithComments</a></li><li>■ <a href="http://www.w3.org/2001/10/xml-exc-c14n#">http://www.w3.org/2001/10/xml-exc-c14n#</a></li><li>■ <a href="http://www.w3.org/2001/10/xml-exc-c14n#WithComments">http://www.w3.org/2001/10/xml-exc-c14n#WithComments</a></li></ul>
<i>signatureId</i>	<b>String</b> Optional. ID attribute for the signature node.
<i>keyStoreAlias</i>	<b>String</b> Optional. Name (alias) of the keystore that contains the private key/certificate.

---

<i>keyAlias</i>	<b>String</b> Optional. Name (alias) of the private key, contained in the keystore specified by the <i>keyStoreAlias</i> parameter, that is used for signing.
<i>includeCertChain</i>	<b>String</b> Optional. Indicates whether the certificate chain should be included in the signature. Set to: <ul style="list-style-type: none"><li>■ <code>True</code> to include the certificate chain in the signature.</li><li>■ <code>False</code> to leave the certificate chain out of the signature. This is the default.</li></ul>
<i>certData</i>	<b>String</b> Optional. Select the X509 certificate data that should be entered into the signature's key information. Note that the initials "SKI" in the last option stand for "Subject Key Identifier." <ul style="list-style-type: none"><li>■ <code>X509_CERTIFICATE</code> (default)</li><li>■ <code>X509_SUBJECT_NAME</code></li><li>■ <code>X509_ISSUER_SERIAL</code></li><li>■ <code>X509_SKI</code></li></ul>
<i>idXmlObject</i>	<b>String</b> Optional. Specifies the ID for the node that holds the original XML that is signed. Applicable only for enveloping signatures.
<i>encoding</i>	<b>String</b> Optional. Specifies the encoding to use if the encoding cannot be extracted from the XML. If encoding is not specified in the XML document or in the <i>encoding</i> parameter, Integration Server uses UTF-8. The <i>encoding</i> value must be a valid IANA encoding.
<i>addSignatureAsLastElement</i>	<b>Boolean</b> Optional. When <i>isEnveloped</i> is set to <code>True</code> , this parameter indicates the position at which Integration Server should add the signature element child to the root. Set to: <ul style="list-style-type: none"><li>■ <code>True</code> to add the signature element as the last child of the root.</li><li>■ <code>False</code> to add the signature element as the first child of the root. This is the default.</li></ul>

---

## Output Parameters

---

<i>signedXMLData</i>	<b>String</b> Conditional. Signed XML data. <i>signedXMLData</i> is returned when <i>xmlData</i> is provided.
<i>signedXMLStream</i>	<b>OutputStream</b> Conditional. Signed XML in the form of an OutputStream. <i>signedXMLStream</i> is returned when <i>xmlStream</i> is provided.

## Usage Notes

Before the signing/signature verification of XML can occur between Integration Server and an external system, the Integration Server *must* share the public key that corresponds to the private key with which the document is signed. Integration Server must share the public key with the external system that will be performing verification.

*keyAlias* and *keyStoreAlias* should either both be provided or both be absent from the input. If no value is provided for these parameters, Integration Server uses the private key/certificate specified for the Signing Key. If the Signing Key is not specified, Integration Server uses the SSL Key.

For information about configuring the Signing Key and SSL Key keystore aliases using the **Security > Certificates** screen in Integration Server Administrator, refer to *webMethods Integration Server Administrator's Guide*.

If both *xmlData* and *xmlStream* are provided, *xmlStream* takes precedence.

The *uri* and *nodeSelectors* parameters identify the nodes to be signed.

If *uri* is specified and *nodeSelectors* is not specified, Integration Server signs the entire node identified by *uri*.

If *uri* and *nodeSelectors* are specified, Integration Server determines which nodes to sign by locating the node specified by the *uri* and then applying the filter from *nodeSelectors*.

If *uri* is not specified and *nodeSelectors* is specified, Integration Server determines which nodes to sign by applying the filter in *nodeSelectors* to the entire XML.

If neither *uri* nor *nodeSelectors* are specified, Integration Server signs the entire XML.

You can use the value of an ID attribute as the *uri*.

For example, #sampleID

Where sampleID is an ID attribute that functions as a unique identifier for an element in an XML schema definition. In this example, Integration Server will locate the node with the ID attribute "sampleID" and then apply the filter specified by *nodeSelectors* to determine which nodes to sign.

## Signature Types

As opposed to a detached signature, which is kept apart from the original document, enveloping and enveloped signatures are tightly coupled with the original document.

An *enveloping* signature must be a parent node of the data being signed:

```
<!-- Example of Enveloping Signature --> <Signature>
<my_document>. . . </my_document> </Signature>
```

The following input parameters and values are applicable only to enveloping signatures:

- *isEnveloped*. Specify a value of "false" for enveloping. If *isEnveloped* is set to false, then:
  - If both *uri* and *idXmlObject* are null, Integration Server creates a dynamic unique value for both *uri* and *idXmlObject* and signs the XML.
  - If *idXmlObject* is provided and *uri* is null, Integration Server creates a *uri* with a value of *#idXmlObject\_value* and signs the XML.
  - If both *uri* and *idXmlObject* are provided and match the XML contract (for example, *uri*='#*idXmlObject* '), Integration Server signs the XML. If the *uri* and *idXmlObject* parameters do not match the contract, Integration Server issues an exception.
- *idXmlObject*. Specifies the ID for the node that holds the original, signed XML.

An *enveloped* signature must be a child node of the data being signed:

```
<!-- Example of Enveloped Signature --> <my_document>
<Signature> . . . </Signature> </my_document>
```

The following parameters and values are applicable only to enveloped signatures:

- *isEnveloped*. The default value of "true" specifies that the signature is enveloped.
- *signatureNodeSelector*. XPath to the node where the signature is entered. If no value is provided, the signature is placed as a first child of the root node.

pub.security.xml:signXML does *not* support detached signatures.

## Signing and Encrypting the Same XML Document

You can use both encryption and signing in the same XML document.

- If you sign and encrypt *different* XML elements in a document, you can run either pub.security.xml:signXML or pub.security.xml:encryptXML first.
- Typically, if you sign and encrypt the *same* XML elements in a document, you should sign the elements before encrypting them. That is, invoke pub.security.xml:signXML before invoking pub.security.xml:encryptXML.

---

## pub.security.xml:verifyXML

WmPublic. Verifies a signed XML document, or node in an XML document, and returns information about the success or failure of the verification.

### Input Parameters

---

<i>xmldata</i>	<b>String</b> Optional. Signed XML that needs to be verified.
<i>xmlStream</i>	<b>InputStream</b> Optional. Signed XML as an input stream that needs to be verified.  <b>Note:</b> If both <i>xmldata</i> and <i>xmlStream</i> are provided, <i>xmlStream</i> takes precedence.
<i>signatureSelectors</i>	<b>String Array</b> XPath expressions that are used to identify the signature; can be any valid XPath. Following is an example: <pre>//*[@ID="Sign001"]</pre>
<i>nsDecls</i>	<b>Document</b> Optional. Mapping of the namespace prefixes to the namespace URIs. The first column contains the prefixes and the second column contains the corresponding URIs.
<i>noNamespace SchemaLocation</i>	<b>String</b> Optional. Schema location for elements with no namespace. This parameter is used to locate the schema that defines elements without a namespace prefix.
<i>schemaLocations</i>	<b>Document</b> Optional. Holds the schema locations against the namespaces.
<i>encoding</i>	<b>String</b> Optional. Specifies the encoding to use if the encoding cannot be extracted from the XML. If encoding is not specified in the XML document or in the <i>encoding</i> parameter, Integration Server uses UTF-8.  The <i>encoding</i> value must be a valid IANA encoding.

### Output Parameters

---

<i>verificationResult</i>	<b>Boolean</b> Indicates whether the signed XML is authentic (true) or cannot be verified or shows signs of tampering (false).
---------------------------	--

---

<i>failedSignatureSelector</i>	<b>String</b> Conditional. In case of a verification failure (the digests do not equate), indicates which signature selector failed.
<i>failureReason</i>	<b>String</b> Conditional. This output parameter is populated only in the case of a verification failure. Its value indicates whether (1) the signature caused the failure, or (2) the signature is from an untrusted certificate. Possible values are: <ul style="list-style-type: none"><li>■ SIGNATURE FAILED</li><li>■ CERTIFICATE NOT TRUSTED</li></ul>
<i>certMap</i>	<b>Document List</b> Conditional. For each XPath in <i>signatureSelector</i> , <i>certMap</i> contains a document that identifies the XPath and the corresponding signing certificate found at that XPath. <p><i>certMap</i> is only returned if certificates were resolved for at least one the XPaths specified in <i>signatureSelector</i>.</p> <p>If Integration Server encounters a signature failure, Integration Server does not resolve any subsequent XPaths. <i>certMap</i> contains all of the XPaths and corresponding certificates that Integration Server could resolve up to the point of failure.</p>

### Usage Notes

If both *xmlData* and *xmlStream* are provided, *xmlStream* takes precedence.

Before `pub.security.xml:verifyXML` can verify a signature, the partner application's public key must have been made available to Integration Server, either through:

- Integration Server certificate mapping.
- The partner application having sent a copy of its certificate to Integration Server.

For information on Integration Server certificate mapping, refer to *webMethods Integration Server Administrator's Guide*.

The `pub.security.xml:verifyXML` service works as follows:

1. Integration Server receives the signed XML document.
2. Integration Server extracts the public key from the partner application's certificate.
3. Integration Server uses the public key to verify the authenticity of the XML document.



# 30 SMIME Folder

---

■ Summary of Elements in this Folder .....	736
--	-----

You use the elements in the smime folder to create digitally signed and/or encrypted MIME messages. You also use the services in this folder to process signed and encrypted MIME messages that are passed into the pipeline.

## Summary of Elements in this Folder

The following elements are available in this folder:

Element	Package and Description
<a href="#">pub.smime:createCertsOnlyData</a>	WmPublic. Generates a PKCS #7 certs-only S/MIME entity from an array of certificates.
<a href="#">pub.smime:createEncryptedData</a>	WmPublic. Encrypts a MIME message.
<a href="#">pub.smime:createSignedAndEncryptedData</a>	WmPublic. <i>Deprecated</i> - Replaced by <a href="#">pub.smime.keystore:createSignedAndEncryptedData</a> . Digitally signs a MIME message and then encrypts it.
<a href="#">pub.smime:createSignedData</a>	WmPublic. <i>Deprecated</i> - Replaced by <a href="#">pub.smime.keystore:createSignedData</a> . Digitally signs a MIME message.
<a href="#">pub.smime:processCertsOnlyData</a>	WmPublic. Extracts the certificates from a PKCS #7 certs-only S/MIME entity.
<a href="#">pub.smime:processEncryptedData</a>	WmPublic. <i>Deprecated</i> - Replaced by <a href="#">pub.smime.keystore:processEncryptedData</a> .
<a href="#">pub.smime:processSignedData</a>	WmPublic. Verifies the signature from a signed S/MIME entity and extracts the message from it.
<a href="#">pub.smime.keystore:createSignedAndEncryptedData</a>	WmPublic. Digitally signs and encrypts a MIME message.
<a href="#">pub.smime.keystore:createSignedData</a>	WmPublic. Creates signed S/MIME data.

Element	Package and Description
<a href="#">pub.smime.keystore:processEncryptedData</a>	WmPublic. Decrypts an encrypted S/MIME message.

## pub.smime:createCertsOnlyData

WmPublic. Generates a PKCS #7 certs-only S/MIME entity from an array of certificates.

This service can be used to develop mechanisms for transmitting certificates and certificate chains to other parties.

### Input Parameters

*certificates*      **byte[ ][ ]** The certificates that are to be encapsulated within the S/MIME entity. Each `byte[ ]` represents a single certificate.

### Output Parameters

*SMimeEncStream*      **java.io.InputStream** S/MIME entity.

### Usage Notes

For general information about MIME messages and using the MIME services, see the *MIME-S/MIME Developer's Guide*.

## pub.smime:createEncryptedData

WmPublic. Encrypts a MIME message.

### Input Parameters

*encStream*      **java.io.InputStream** MIME message that you want to encrypt (for example, the output produced by [pub.mime:getEnvelopeStream](#)).

*recipientCerts*      **byte[ ][ ]** The X.509 certificates of the recipients for whom this message will be encrypted. Each element in the list represents a certificate for a single recipient in the form of a `byte[ ]`.

**Note:** When you have multiple recipients, `createEncryptedData` creates a single message that is encrypted for all

recipients. It does not create a separate message for each recipient.

<i>encryptionAlg</i>	<b>String</b> Optional. Code specifying the encryption algorithm to use. Must be <code>TripleDES</code> (default), <code>DES</code> , or <code>RC2</code> .
<i>keyLength</i>	<b>String</b> Optional. Length of the encryption key for RC2 encryption. Must be <code>40</code> , <code>64</code> , or <code>128</code> (default).  This parameter is ignored if <i>encryptionAlg</i> is not RC2.

## Output Parameters

*SMimeEncStream*     **java.io.InputStream** The encrypted MIME message.

## Usage Notes

For general information about MIME messages and using the MIME services, see the *MIME-S/MIME Developer's Guide*.

## See Also

[pub.smime:createSignedData](#)  
[pub.smime:processEncryptedData](#)  
[pub.mime:getEnvelopeStream](#)

## Examples

See the following in the `WmSamples` package in the certified samples area of the Knowledge Center on Empower Product Support website at <https://empower.softwareag.com>:

`sample.smime:build_EncryptedSMime`

## pub.smime:createSignedAndEncryptedData

`WmPublic`. *Deprecated* - Replaced by [pub.smime.keystore:createSignedAndEncryptedData](#). Digitally signs a MIME message and then encrypts it.

**Important:** You must use this service when you want to create a message that is both signed and encrypted. You cannot produce this type of message using the [pub.smime:createSignedData](#) and [pub.smime:createEncryptedData](#) services.

## Input Parameters

---

*envStream* **java.io.InputStream** The MIME message that you want to sign and encrypt (for example, the output produced by `pub.mime.getEnvelopeStream`).

*privKey* **byte[]** Private key of the party signing the message.

*certificates* **byte[][]** Optional. The certificate chain of the party signing the message, where each `byte[]` represents a single certificate in the chain. Certificates must appear in hierarchical order, starting with the signer's certificate in element 0. The following list shows how the elements of a complete chain would appear for a certificate that was issued through two intermediate CAs:

Element	Contents
0	Signer's certificate.
1	Intermediary CA Certificate.
2	Intermediary CA Certificate.
3	Root CA Certificate.

**Note:** Although this parameter is optional, it should only be omitted if the party receiving the message is able to process this signature without an accompanying certificate chain.

*signerCert* **byte[]** Digital certificate of the party signing the message.

*explicit* **String** Optional. Flag indicating whether an implicit or explicit signature is to be generated. Set to:

- `true` to generate an explicit (detached) signature. This is the default.
- `false` to generate an implicit signature.

*recipientCerts* **byte[][]** X.509 certificates of the recipients for whom this message will be encrypted. Each element in the list contains the certificate for a single recipient in the form of a byte array.

<i>encryptionAlg</i>	<b>String</b> Optional. Code specifying the encryption algorithm to use. Must be <code>TripleDES</code> (default), <code>DES</code> , or <code>RC2</code> .
<i>keyLength</i>	<b>String</b> Optional. Length of the encryption key for RC2 encryption. Must be 40, 64, or 128 (default).  This parameter is ignored if <i>encryptionAlg</i> is not RC2.

---

### Output Parameters

*SMimeEnvStream*      **java.io.InputStream** Signed and encrypted MIME message.

### Usage Notes

This service is superseded by [pub.smime.keystore:createSignedAndEncryptedData](#).

For general information about MIME messages and using the MIME services, see the *MIME-S/MIME Developer's Guide*.

### See Also

- [pub.smime:createEncryptedData](#)
- [pub.smime:processEncryptedData](#)
- [pub.smime:createSignedData](#)
- [pub.smime:processSignedData](#)
- [pub.smime.keystore:createSignedData](#)
- [pub.mime:getEnvelopeStream](#)

### Examples

See the following in the WmSamples package in the certified samples area of the Knowledge Center on the Empower Product Support website at <https://empower.softwareag.com>:

```
sample.smime:build_SignedAndEncryptedSMime
```

---

## pub.smime:createSignedData

WmPublic. *Deprecated* - Replaced by [pub.smime.keystore:createSignedData](#). Digitally signs a MIME message.

---

### Input Parameters

*envStream*      **java.io.InputStream** MIME message that you want to sign (for example, the output produced by [pub.mime:getEnvelopeStream](#)).

*privKey* **byte[]** Private key of the party signing the message.

*certificates* **byte[][]** Optional. Certificate chain of the party that signed the message, where each `byte[]` represents a single certificate in the chain. Certificates must appear in hierarchical order, starting with the signer's certificate in element 0. The following shows how the elements of a complete chain would appear for a certificate that was issued through two intermediate CAs:

Element	Contents
0	Signer's certificate.
1	Intermediary CA Certificate.
2	Intermediary CA Certificate.
3	Root CA Certificate.

Although this parameter is optional, it should only be omitted if the party receiving the message is able to process this signature without an accompanying certificate chain.

*signerCert* **byte[]** Digital certificate of the party signing the message.

*explicit* **String** Optional. Flag indicating whether an implicit or explicit signature is generated. Set to:

- `true` to generate an explicit (detached) signature. This is the default.
- `false` to generate an implicit signature.

## Output Parameters

*SMimeEncStream* **java.io.InputStream** The signed MIME message.

## Usage Notes

This service is superseded by [pub.smime.keystore:createSignedData](#).

For general information about MIME messages and using the MIME services, see the *MIME-S/MIME Developer's Guide*.

---

**See Also**

[pub.smime:createEncryptedData](#)  
[pub.mime:getEnvelopeStream](#)

---

## pub.smime:processCertsOnlyData

WmPublic. Extracts the certificates from a PKCS #7 certs-only S/MIME entity.

---

**Input Parameters**

*SMimeEnvStream*      **java.io.InputStream** The certs-only S/MIME entity.

---

**Output Parameters**

*certificates*      **byte[ ][ ]** The extracted certificates. Each element in the list contains one of the extracted certificates represented as a **byte[ ]**.

---

## pub.smime:processEncryptedData

WmPublic. *Deprecated* - Replaced by [pub.smime.keystore:processEncryptedData](#).

Decrypts an encrypted S/MIME message.

---

**Input Parameters**

*SMimeEnvStream*      **java.io.InputStream** The encrypted S/MIME entity (for example, the output produced by [pub.smime:createEncryptedData](#)).

*recipientCert*      **byte[ ]** Digital certificate of the party receiving the message.

*privKey*      **byte[ ]** Private key of the party receiving the message (that is, the party whose public key was used to encrypt the message).

---

**Output Parameters**

*mimeData*      **Document** MIME object containing the decrypted MIME message.

---

<i>contentDigest</i>	<b>String</b> Message digest of the encrypted content, base64-encoded. (Some sites return this digest to the sender to acknowledge their receipt of the message.)
<i>encrypted</i>	<b>String</b> Conditional. Flag indicating whether the decrypted MIME entity is encrypted. A value of: <ul style="list-style-type: none"><li>■ <code>true</code> indicates that the MIME entity is encrypted.</li><li>■ <code>false</code> indicates that the MIME entity is not encrypted.</li></ul>
<i>signed</i>	<b>String</b> Conditional. Flag indicating whether the decrypted MIME entity is signed. A value of: <ul style="list-style-type: none"><li>■ <code>true</code> indicates that the MIME entity is signed.</li><li>■ <code>false</code> indicates that the MIME entity is not signed.</li></ul>
<i>certsOnly</i>	<b>String</b> Conditional. Flag indicating whether the decrypted MIME entity is a certs-only entity. A value of: <ul style="list-style-type: none"><li>■ <code>true</code> indicates that the MIME entity is a certs-only entity.</li><li>■ <code>false</code> indicates that the MIME entity is not a certs-only entity.</li></ul>
<i>stream</i>	<b>java.io.InputStream</b> Conditional. The decrypted MIME entity.

### Usage Notes

This service is superseded by [pub.smime.keystore:processEncryptedData](#).

If the decrypted message is signed or encrypted, *mimeData* will be empty, and the decrypted message will reside in *stream*. You can check the state of the *signed* and *encrypted* output variables to determine whether the decrypted message requires additional processing, and pass *stream* to the [pub.smime:processSignedData](#) or [pub.smime:processEncryptedData](#) service as necessary.

**Important:** You can examine the contents of *mimeData* during testing and debugging. However, because the internal structure of *mimeData* is subject to change without notice, *do not* explicitly set or map data to/from these elements in your service. To manipulate or access the contents of *mimeData*, use *only* the MIME services that Integration Server provides.

### See Also

[pub.smime:processSignedData](#)  
[pub.smime:createEncryptedData](#)

---

## pub.smime:processSignedData

WmPublic. Verifies the signature from a signed S/MIME entity and extracts the message from it.

### Input Parameters

---

*SMimeEnvStream*      **java.io.InputStream** Signed MIME entity (for example, the output produced by [pub.smime:createSignedData](#)).

*signerCertChain*      **byte[ ][ ]** Optional. Certificate chain of the party that signed the message, where each `byte[ ]` represents a single certificate in the chain. Certificates must appear in hierarchical order, starting with the signer's certificate in element 0. The following shows how the elements of a complete chain would appear for a certificate that was issued through two intermediate CAs:

Element	Contents
0	Signer's certificate.
1	Intermediary CA Certificate.
2	Intermediary CA Certificate.
3	Root CA Certificate.

**Note:** If the signer included the certificate chain with the digital signature, you do not need to supply *signerCertChain*.

### Output Parameters

---

*mimeData*      **Document** MIME object containing the extracted MIME entity.

*contentDigest*      **String** Message digest (base64-encoded) that was recalculated by `processSignedData`.

*signerCert*      **java.security.cert.X509Certificate** Signer's X.509 certificate.

*encrypted*      **String** Conditional. Flag indicating whether the extracted MIME entity is encrypted. A value of:

---

	<ul style="list-style-type: none"><li>■ <code>true</code> indicates that the MIME entity is encrypted.</li><li>■ <code>false</code> indicates that the MIME entity is not encrypted.</li></ul>		
<i>signed</i>	<p><b>String</b> Conditional. Flag indicating whether the extracted MIME entity is signed. A value of:</p> <ul style="list-style-type: none"><li>■ <code>true</code> indicates that the MIME entity is signed.</li><li>■ <code>false</code> indicates that the MIME entity is not signed.</li></ul>		
<i>certsOnly</i>	<p><b>String</b> Conditional. Flag indicating whether the extracted MIME entity is a certs-only entity. A value of:</p> <ul style="list-style-type: none"><li>■ <code>true</code> indicates that the MIME entity is a certs-only entity.</li><li>■ <code>false</code> indicates that the MIME entity is not a certs-only entity.</li></ul>		
<i>stream</i>	<p><b>java.io.InputStream</b> Conditional. Extracted MIME entity.</p>		
<i>verify</i>	<p><b>String</b> Flag indicating whether the signature was successfully processed. Success indicates that the signature was successfully verified with the supplied public key. A value of:</p> <ul style="list-style-type: none"><li>■ <code>true</code> indicates that signature processing was successful.</li><li>■ <code>false</code> indicates that signature processing failed. The signature could not be verified because an <i>errorCode</i> 1, 2, 3, or 4 occurred.</li></ul>		
<i>trusted</i>	<p><b>String</b> Flag indicating whether the signer certificate is trusted or not. A value of:</p> <ul style="list-style-type: none"><li>■ <code>true</code> indicates that the signer certificate is trusted.</li><li>■ <code>false</code> indicates that the signer certificate is not trusted.</li></ul>		
<i>errorCode</i>	<p><b>String</b> Conditional. Number indicating the kind of error that occurred while processing the signature. See <i>errorMessage</i> for possible values.</p> <p>If no error occurred, <i>errorCode</i> will not be returned.</p>		
<i>errorMessage</i>	<p><b>String</b> Conditional. Textual error message indicating what kind of error occurred while processing the signature. Error codes and messages are as follows:</p> <table><tr><td><u><b>errorCode</b></u></td><td><u><b>errorMessage</b></u></td></tr></table>	<u><b>errorCode</b></u>	<u><b>errorMessage</b></u>
<u><b>errorCode</b></u>	<u><b>errorMessage</b></u>		

---

1	Invalid signer certificate file information.
2	Certificate at index 'i' is not in recognizable format.
3	Invalid certificate input at index 'i'.
4	Signature cannot be verified.
5	Expired certificate chain.
6	Error in certificate chain.
7	Untrusted certificate.

### Usage Notes

If *verify* is false, the *errorCode* and *errorMessage* values will indicate the error that caused the failure. Note that *errorCode* values 5 through 7 do not represent signature-validation failures and, therefore, do not cause the *verify* flag to be set to false.

If the extracted entity is signed or encrypted, *mimeData* will be empty, and the extracted entity will reside in *stream*. You can check the state of the *signed* and *encrypted* output variables to determine whether the extracted entity requires additional processing, and pass *stream* to the [pub.smime:processEncryptedData](#) service as necessary.

**Important:** You can examine the contents of *mimeData* during testing and debugging. However, because the internal structure of *mimeData* is subject to change without notice, *do not* explicitly set or map data to/from these elements in your service. To manipulate or access the contents of *mimeData*, use *only* the MIME services that Integration Server provides.

### See Also

[pub.smime:processEncryptedData](#)  
[pub.smime:createSignedData](#)

### Examples

**Important:** See the following in the WmSamples packages in the certified samples area of the Knowledge Center on the Empower Product Support website at <https://empower.softwareag.com>:

sample.smime:extract\_SignedSMime

sample.smime:extract\_SignedAndEncryptedSMime

---

## pub.smime.keystore:createSignedAndEncryptedData

WmPublic. Digitally signs and encrypts a MIME message.

### Input Parameters

---

<i>envStream</i>	<b>java.io.InputStream</b> The MIME message that you want to sign and encrypt (for example, the output produced by <a href="#">pub.mime:getEnvelopeStream</a> ).
<i>keyStoreAlias</i>	<b>String</b> Alias of the keystore containing the signing key.
<i>keyAlias</i>	<b>String</b> Alias of the private key to be used for signing.
<i>explicit</i>	<b>String</b> Optional. Flag indicating whether an implicit or explicit signature is to be generated. Set to: <ul style="list-style-type: none"><li>■ <b>True</b> to generate an explicit (detached) signature. This is the default.</li><li>■ <b>False</b> to generate an implicit signature.</li></ul>
<i>recipientCerts</i>	<b>Object List</b> A list of byte[ ] for the partner certificates for whom this message will be encrypted.
<i>encryptionAlg</i>	<b>String</b> Optional. Code specifying the encryption algorithm to use. Must be TripleDES (default), DES, or RC2.
<i>keyLength</i>	<b>String</b> Optional. Length of the encryption key for RC2 encryption. Must be 40, 64, or 128 (default).  This parameter is ignored if encryptionAlg is not RC2.

### Output Parameters

---

<i>SMimeEnvStream</i>	<b>java.io.InputStream</b> Signed and encrypted data as a stream.
-----------------------	---

### Usage Notes

This service supersedes [pub.smime:createSignedAndEncryptedData](#)

You must use this service when you want to create a message that is both signed and encrypted.

For information about using aliases for keystores, truststores, and private keys, see *webMethods Integration Server Administrator's Guide*

---

## pub.smime.keystore:createSignedData

WmPublic. Creates signed S/MIME data.

### Input Parameters

---

<i>envStream</i>	<b>java.io.InputStream</b> MIME message that you want to sign (for example, the output produced by <a href="#">pub.mime:getEnvelopeStream</a> ).
<i>keyStoreAlias</i>	<b>String</b> Alias of the keystore.
<i>keyAlias</i>	<b>String</b> Alias of the private key of interest in the keystore.
<i>explicit</i>	<p><b>String</b> Optional. Flag indicating whether an implicit or explicit signature is generated. Set to:</p> <ul style="list-style-type: none"> <li>■ <code>True</code> to generate an explicit (detached) signature. This is the default.</li> <li>■ <code>False</code> to generate an implicit signature.</li> </ul>

### Output Parameters

---

<i>SMimeEnvStream</i>	<b>java.io.InputStream</b> The signed MIME stream.
-----------------------	--

### Usage Notes

This service supersedes [pub.smime:createSignedData](#).

For information about using aliases for keystores, truststores, and private keys, see *webMethods Integration Server Administrator's Guide*.

---

## pub.smime.keystore:processEncryptedData

WmPublic. Decrypts an encrypted S/MIME message.

### Input Parameters

---

<i>SMimeEnvStream</i>	<b>java.io.InputStream</b> The encrypted S/MIME stream.
-----------------------	---

*keyStoreAlias*      **String** Alias of the keystore containing the decryption key.

*keyAlias*      **String** Alias of the key used for decryption.

### Output Parameters

---

*mimeData*      **Document** The decrypted MIME message.

*contentDigest*      **String** Message digest of the encrypted content, base64-encoded. (Some sites return this digest to the sender to acknowledge their receipt of the message.)

*encrypted*      **String** Flag indicating whether the MIME entity passed in to the service was encrypted. A value of:

- `True` indicates that the MIME entity was encrypted.
- `False` indicates that the MIME entity was not encrypted.

*signed*      **String** Flag indicating whether the MIME entity passed in to the service was signed. A value of:

- `True` indicates that the MIME entity is signed.
- `False` indicates that the MIME entity is not signed.

*certsOnly*      **String** Flag indicating whether the MIME entity passed in to the service contained only digital certificates. A value of:

- `True` indicates that the MIME entity is a certs-only entity.
- `False` indicates that the MIME entity is not a certs-only entity.

*stream*      **java.io.InputStream** The decrypted MIME entity.

### Usage Notes

This service supersedes [pub.smime:processEncryptedData](#).

For information about using aliases for keystores, truststores, and private keys, see *webMethods Integration Server Administrator's Guide*.



# 31 SOAP Folder

---

■ Summary of Elements in this Folder .....	752
--	-----

You use the elements in the soap folder to compose and send SOAP messages and to receive and retrieve data from within them.

## Summary of Elements in this Folder

The following elements are available in this folder:

Element	Package and Description
<a href="#">pub.soap.handler:addBodyBlock</a>	WmPublic. Inserts a document into a SOAP message as a new body block.
<a href="#">pub.soap.handler:addFaultBlock</a>	WmPublic. Inserts a document into a SOAP message as a new fault block.
<a href="#">pub.soap.handler:addHeaderBlock</a>	WmPublic. Inserts a document into a SOAP message as a new header block.
<a href="#">pub.soap.handler:addHeaderElement</a>	WmPublic. <i>Deprecated</i> - Replaced by <a href="#">pub.soap.handler:addHeaderBlock</a> .
<a href="#">pub.soap.handler:generateDocumentTypesFromWSDL</a>	WmPublic. Generates IS document types from a WSDL.
<a href="#">pub.soap.handler:getBodyBlock</a>	WmPublic. Retrieves a body block from a SOAP message.
<a href="#">pub.soap.handler:getBodyBlockQNames</a>	WmPublic. Returns the QName for each body block in a SOAP message.
<a href="#">pub.soap.handler:getFaultBlock</a>	WmPublic. Retrieves a fault block from a SOAP message.
<a href="#">pub.soap.handler:getHeaderBlock</a>	WmPublic. Retrieves a header block from a SOAP message.
<a href="#">pub.soap.handler:getHeaderBlockQNames</a>	WmPublic. Returns the QName for each header block in a SOAP message.

Element	Package and Description
<a href="#">pub.soap.handler:getHeaderElement</a>	WmPublic. <i>Deprecated</i> - Replaced by <a href="#">pub.soap.handler:getHeaderBlock</a> .
<a href="#">pub.soap.handler:getInitialSOAPRequest</a>	WmPublic. Gets the initial SOAP request message from a given message context.
<a href="#">pub.soap.handler:getMessageAddressingProperties</a>	WmPublic. Gets the value of the specified message addressing properties in a SOAP message.
<a href="#">pub.soap.handler:getProperty</a>	WmPublic. Gets the value of a specified property from a message context.
<a href="#">pub.soap.handler:getServicePipeline</a>	WmPublic. Gets the service pipeline from a given message context.
<a href="#">pub.soap.handler:getSOAPMessage</a>	WmPublic. Gets the message addressing properties of the SOAP message in the provided message context.
<a href="#">pub.soap.handler:getWebServiceInvocationProperties</a>	WmPublic. Fetches the web service invocation properties.
<a href="#">pub.soap.handler:handlerSpec</a>	WmPublic. Specification to use as the signature for a service that acts as a header handler.
<a href="#">pub.soap.handler:hasFaultMessage</a>	WmPublic. Determines whether the SOAP message in a given message context contains a SOAP fault message.
<a href="#">pub.soap.handler:listConsumer</a>	WmPublic. Returns a list of the consumer handlers currently registered on Integration Server.

Element	Package and Description
<a href="#">pub.soap.handler:listProvider</a>	WmPublic. Returns a list of the provider handlers currently registered on Integration Server.
<a href="#">pub.soap.handler:registerConsumer</a>	WmPublic. <i>Deprecated</i> - Replaced by <a href="#">pub.soap.handler:registerWmConsumer</a> .
<a href="#">pub.soap.handler:registerProvider</a>	WmPublic. <i>Deprecated</i> - Replaced by <a href="#">pub.soap.handler:registerWmProvider</a> .
<a href="#">pub.soap.handler:registerWmConsumer</a>	WmPublic. Registers a handler for use with consumer.
<a href="#">pub.soap.handler:registerWmProvider</a>	WmPublic. Registers a header handler for use with provider.
<a href="#">pub.soap.handler:removeBodyBlock</a>	WmPublic. Removes a body block from a SOAP message, leaving an empty SOAP body, <SOAP-ENV:Body></SOAP-ENV:Body>.
<a href="#">pub.soap.handler:removeHeaderBlock</a>	WmPublic. Removes a header block from a SOAP message.
<a href="#">pub.soap.handler:removeHeaderElement</a>	WmPublic. <i>Deprecated</i> - Replaced by <a href="#">pub.soap.handler:removeHeaderBlock</a> .
<a href="#">pub.soap.handler:removeProperty</a>	WmPublic. Removes a property from a message context.
<a href="#">pub.soap.handler:setProperty</a>	WmPublic. Sets the value of a specific property in a message context.
<a href="#">pub.soap.handler:setSOAPMessage</a>	WmPublic. Sets the SOAP message in a message context.
<a href="#">pub.soap.handler:unregisterConsumer</a>	WmPublic. Unregisters a consumer web service descriptor handler.

Element	Package and Description
<a href="#">pub.soap.handler:unregisterProvider</a>	WmPublic. Unregisters a provider web service descriptor handler.
<a href="#">pub.soap.handler:updateFaultBlock</a>	WmPublic. Updates the fault code ( <i>code</i> ) and the fault string ( <i>reasons</i> ) in the existing fault block.
<a href="#">pub.soap.processor:list</a>	WmPublic. <i>Deprecated</i> - There is no replacement service. Returns a list of the SOAP processors that are currently registered on the Integration Server.
<a href="#">pub.soap.processor:processMessage</a>	WmPublic. <i>Deprecated</i> - There is no replacement service. Executes the Integration Server's default SOAP processor.
<a href="#">pub.soap.processor:processRPCMessage</a>	WmPublic. <i>Deprecated</i> - There is no replacement service. Executes the Integration Server's SOAP RPC processor.
<a href="#">pub.soap.processor:registerProcessor</a>	WmPublic. <i>Deprecated</i> - There is no replacement service. Registers a service as a SOAP processor on the Integration Server.
<a href="#">pub.soap.processor:unregisterProcessor</a>	WmPublic. <i>Deprecated</i> - There is no replacement service. Unregisters a SOAP processor by removing it from the registry.
<a href="#">pub.soap.schema:encoding</a>	WmPublic. Schema that defines the data types SOAP supports.
<a href="#">pub.soap.schema:encoding_1_2</a>	WmPublic. Schema that defines the data types SOAP 1.2 supports.
<a href="#">pub.soap.schema:envelope</a>	WmPublic. Schema that defines the structure of a SOAP message.

Element	Package and Description
<a href="#">pub.soap.schema:envelope_1_2</a>	WmPublic. Schema that defines the structure of a SOAP 1.2 message.
<a href="#">pub.soap.utils:addBodyEntry</a>	WmPublic. Inserts an entry into the body element of a SOAP message.
<a href="#">pub.soap.utils:addHeaderEntry</a>	WmPublic. Inserts an entry into the header element of a SOAP message.
<a href="#">pub.soap.utils:addTrailer</a>	WmPublic. Inserts a trailer in a SOAP message.
<a href="#">pub.soap.utils:callbackServiceSpec</a>	WmPublic. Defines the input signature for an outbound callback service.
<a href="#">pub.soap.utils:convertToVersionSpecificSOAPFault</a>	WmPublic. Converts the generic SOAP fault structure to the SOAP version-specific fault structure that is used by web service descriptors created in versions of Integration Server prior to 8.2.
<a href="#">pub.soap.utils:createSoapData</a>	WmPublic. Creates a SOAP object consisting of SOAP envelope, body, and header entries.
<a href="#">pub.soap.utils:createXOPObject</a>	WmPublic. Generates a <code>com.wm.util.XOPObject</code> instance from a <code>base64Binary</code> string, a byte array, or an input stream.
<a href="#">pub.soap.utils:exitUnableToUnderstand</a>	WmPublic. Terminates processing and returns a <code>mustUnderstand</code> fault to the client.
<a href="#">pub.soap.utils:getActor</a>	WmPublic. Retrieves the value of the <code>actor</code> attribute (for SOAP 1.1) or the <code>role</code> attribute (for SOAP 1.2) from a given header entry.

Element	Package and Description
<code>pub.soap.utils:getBody</code>	WmPublic. Retrieves the body from a SOAP message as a single node object.
<code>pub.soap.utils:getBodyEntries</code>	WmPublic. Retrieves the body entries from a SOAP message as an array of node objects.
<code>pub.soap.utils:getDocument</code>	WmPublic. Retrieves an entire SOAP message as a node object.
<code>pub.soap.utils:getEncoding</code>	WmPublic. Retrieves the encoding from a SOAP message as a single string.
<code>pub.soap.utils:getHeader</code>	WmPublic. Retrieves the header from a SOAP message as a single node object.
<code>pub.soap.utils:getHeaderEntries</code>	WmPublic. Retrieves the header entries from a SOAP message as an array of node objects.
<code>pub.soap.utils:getMustUnderstand</code>	WmPublic. Returns the mustUnderstand status for a given header entry.
<code>pub.soap.utils:getQName</code>	WmPublic. Returns the qualified name for a given node.
<code>pub.soap.utils:getTrailers</code>	WmPublic. Retrieves the trailers from a SOAP message.
<code>pub.soap.utils:getXOPObjectContent</code>	WmPublic. Retrieves the contents of a <code>com.wm.util.XOPObject</code> instance as a <code>base64Binary</code> string, a byte array, or an <code>InputStream</code> .
<code>pub.soap.utils:QName</code>	WmPublic. Document type that defines the structure of a qualified name.

Element	Package and Description
<a href="#">pub.soap.utils:removeBodyEntry</a>	WmPublic. Deletes a body entry from a SOAP message.
<a href="#">pub.soap.utils:removeHeaderEntry</a>	WmPublic. Deletes a header entry from a SOAP message.
<a href="#">pub.soap.utils:removeTrailer</a>	WmPublic. Deletes a trailer from a SOAP message.
<a href="#">pub.soap.utils:requestResponseSpec</a>	WmPublic. Defines the input/output signature for a custom processor and a target service for the default processor.
<a href="#">pub.soap.utils:resetWSDEffectivePolicy</a>	WmPublic. Returns the effective policy for a handler in a web service descriptor to the policy set in the <b>Policy</b> name property in Software AG Designer.
<a href="#">pub.soap.utils.setWSDEffectivePolicy</a>	WmPublic. Sets the effective policy for a handler in a web service descriptor.
<a href="#">pub.soap.utils:soapDataToBytes</a>	WmPublic. Converts a SOAP object to a Byte Array.
<a href="#">pub.soap.utils:soapDataToString</a>	WmPublic. Converts a SOAP object to a String.
<a href="#">pub.soap.utils:soapFault</a>	WmPublic. Document type that defines the generic SOAP fault structure used by web service descriptors created in Integration Server 8.2 and later.
<a href="#">pub.soap.utils:streamToSoapData</a>	WmPublic. Converts an InputStream containing a SOAP message to a SOAP object.

Element	Package and Description
<a href="#">pub.soap.utils:stringToSoapData</a>	WmPublic. Converts a String containing a SOAP message to a SOAP object.
<a href="#">pub.soap.utils:validateSoapData</a>	WmPublic. Verifies that a SOAP object represents a valid SOAP message.
<a href="#">pub.soap.wsa:action</a>	WmPublic. Document type that defines the contents of the <code>wsa:Action</code> WS-Addressing header.
<a href="#">pub.soap.wsa:faultTo</a>	WmPublic. Document type that defines the contents of the <code>wsa:FaultTo</code> WS-Addressing header.
<a href="#">pub.soap.wsa:from</a>	WmPublic. Document type that contains the details of the source of the message.
<a href="#">pub.soap.wsa:messageID</a>	WmPublic. Document type that defines the contents of the <code>wsa:MessageID</code> WS-Addressing header.
<a href="#">pub.soap.wsa:problemAction</a>	WmPublic. Document type that captures additional information about faults.
<a href="#">pub.soap.wsa:problemHeaderQName</a>	WmPublic. Document type that captures additional information about faults.
<a href="#">pub.soap.wsa:problemIRI</a>	WmPublic. Document type that captures the IRI that caused the problem.
<a href="#">pub.soap.wsa:relatesTo</a>	WmPublic. Document type that defines the contents of the <code>wsa:RelatesTo</code> WS-Addressing header.

Element	Package and Description
<a href="#">pub.soap.wsa:replyTo</a>	WmPublic. Document type that defines the contents of the <code>wsa:ReplyTo</code> WS-Addressing header.
<a href="#">pub.soap.wsa:retryAfter</a>	WmPublic. Document type that specifies the minimum duration in milliseconds that Integration Server waits before retransmitting a message.
<a href="#">pub.soap.wsa:to</a>	WmPublic. Document type that defines the contents of the <code>wsa:To</code> WS-Addressing header.
<a href="#">pub.soap.wsa:schema_wsa</a>	WmPublic. A schema containing the elements from <code>http://www.w3.org/2005/08/addressing</code> namespace.
<a href="#">pub.soap.wsa.submission:action</a>	WmPublic. Document type that defines the contents of the <code>wsa:Action</code> WS-Addressing header.
<a href="#">pub.soap.wsa.submission:faultTo</a>	WmPublic. Document type that defines the contents of the <code>wsa:FaultTo</code> WS-Addressing header.
<a href="#">pub.soap.wsa.submission:from</a>	WmPublic. Document type that contains the details about the source of the message.
<a href="#">pub.soap.wsa.submission:messageID</a>	WmPublic. Document type that defines the contents of the <code>wsa:MessageID</code> WS-Addressing header.
<a href="#">pub.soap.wsa.submission:relatesTo</a>	WmPublic. Document type that defines the contents of the <code>wsa:RelatesTo</code> WS-Addressing header.

Element	Package and Description
<a href="#">pub.soap.wsa.submission:replyTo</a>	WmPublic. Document type that specifies the destination to which the response message is to be sent.
<a href="#">pub.soap.wsa.submission:retryAfter</a>	WmPublic. Document type that specifies the minimum duration in milliseconds that Integration Server waits before retransmitting a message.
<a href="#">pub.soap.wsa.submission:to</a>	WmPublic. Document type that defines the contents of the <code>wsa:To</code> WS-Addressing header.
<a href="#">pub.soap.wsa.submission:schema_wsa_submission</a>	WmPublic. A schema containing the elements from <code>http://schemas.xmlsoap.org/ws/2004/08/addressing</code> namespace.
<a href="#">pub.soap.wsrn:closeSequence</a>	WmPublic. Closes a reliable messaging sequence.
<a href="#">pub.soap.wsrn:createSequence</a>	WmPublic. Sends a request to a reliable messaging destination to create a new reliable messaging sequence.
<a href="#">pub.soap.wsrn:sendAcknowledgementRequest</a>	WmPublic. Requests an acknowledgment for a message sequence.
<a href="#">pub.soap.wsrn:terminateSequence</a>	WmPublic. Terminates a reliable messaging sequence.
<a href="#">pub.soap.wsrn:waitUntilSequenceCompleted</a>	WmPublic. Requests an acknowledgment for a message sequence.

---

## pub.soap.handler:addBodyBlock

WmPublic. Inserts a document into a SOAP message as a new body block.

### Input Parameters

---

*messageContext* **Object** Message context containing the SOAP message to which to add a body block.

A message context contains properties for the SOAP message and provides access to the SOAP message. Integration Server creates the message context and passes it to the handler. All handlers invoked by a given instance of a SOAP request or SOAP response use the same message context, which enables you to use the message context to pass information among handlers.

*documentType* **String** Optional. Fully qualified name of the IS document type that specifies the structure of the document that you want to add as a new body block; that is, that specifies the structure of *inputBodyBlock/bodyDocument*. If specified, *inputBodyBlock/bodyDocument* must be an instance of this document type.

Specify a document type that Integration Server created while creating the consumer or the WSDL first provider web service descriptor or one that was created using the [pub.soap.handler:generateDocumentTypesFromWSDL](#) service.

If you do not specify a document type, the service performs a literal conversion of *inputBodyBlock/bodyDocument* into the SOAP body; that is, the conversion will not consider style or use.

*inputBodyBlock* **Document** Contents of the body block.

Key	Description
<i>operationQName</i>	<p><b>Document</b> Optional. Qualified name for the web service wrapper that the <code>addBodyBlock</code> service will use to wrap the XML payload that it generates by converting <i>bodyDocument</i> into XML.</p> <p>The <i>operationQName</i> document references the <a href="#">pub.soap.utils:QName</a> document type.</p>

**Note:**The *operationQName* is only relevant for RPC/Encoded and RPC/Literal services;

this service ignores *operationQName* for Document/Literal services.

<u>Key</u>	<u>Description</u>
<i>namespaceName</i>	<b>String</b> Namespace name for the web service operation.
<i>localName</i>	<b>String</b> Name of the web service operation on which the handler is being written.
<i>bodyDocument</i>	<b>Document</b> SOAP body block that you want to add to the SOAP message. Integration Server converts the document to an XML node and inserts it as a body block.
<i>wrapPayloadWithOperationQNameFromContext</i>	<p><b>String</b> Optional. Flag that indicates whether the service should obtain the qualified name for the web service wrapper from the message context or from <i>operationQName</i> . Set to:</p> <ul style="list-style-type: none"> <li>■ <code>true</code> to have the service obtain the qualified name for the web service wrapper from <i>messageContext</i> . If the <i>operationQName</i> parameters are specified, the service ignores them.</li> <li>■ <code>false</code> to have the service use the values you specify in the <i>operationQName</i> parameters. This is the default.</li> </ul> <p><b>Note:</b> For RPC/Encoded and RPC/Literal services, be sure to set <i>wrapPayloadWithOperationQNameFromContext</i> to <code>true</code> if you do not specify <i>operationQName</i> . This service ignores <i>wrapPayloadWithOperationQNameFromContext</i> for Document/Literal services.</p>
<i>validate</i>	<p><b>String</b> Optional. Flag that indicates whether you want the service to validate <i>inputBodyBlock/bodyDocument</i> against the IS document type specified in the <i>documentType</i> input parameter. If you do not use <i>documentType</i> to specify an IS document type, the validation will fail.</p> <p>Set <i>validate</i> to:</p> <ul style="list-style-type: none"> <li>■ <code>true</code> to validate <i>inputBodyBlock/bodyDocument</i> . If the validation fails, an exception is thrown.</li> <li>■ <code>false</code> to skip validating <i>inputBodyBlock/bodyDocument</i> . This is the default.</li> </ul>

---

## Output Parameters

---

*status* **String** Flag indicating the outcome of the service. A value of:

- `true` indicates that adding the body block was successful.
- `false` indicates that adding the body block failed.

**Note:** If the SOAP message already contains a body block, the service sets *status* to `false`.

## Usage Notes

If the SOAP message already contains a body block, first use the [pub.soap.handler:removeBodyBlock](#) service to remove the existing body block before using this service to add a new one.

## See Also

[pub.soap.handler:getBodyBlock](#)  
[pub.soap.handler:removeBodyBlock](#)

---

# pub.soap.handler:addFaultBlock

WmPublic. Inserts a document into a SOAP message as a new fault block.

---

## Input Parameters

---

*messageContext* **Object** Message context containing the SOAP message to which to add a fault block.

A message context contains properties for the SOAP message and provides access to the SOAP message. Integration Server creates the message context and passes it to the handler. All handlers invoked by a given instance of a SOAP request or SOAP response use the same message context, which enables you to use the message context to pass information among handlers.

*documentType* **String** Optional. Fully qualified name of the IS document type that specifies the structure of the document in *soapFault/detail*. If specified, *soapFault/detail* must be an instance of this document type.

Specify a document type that Integration Server created while creating the consumer or the WSDL first provider

web service descriptor or one that was created using the [pub.soap.handler:generateDocumentTypesFromWSDL](#) service.

If you do not specify a document type, the service performs a literal conversion of *soapFault/detail*; that is, the conversion will not consider style or use.

### *soapFault*

**Document** SOAP fault block that you want to add to the SOAP message. Integration Server converts the document to an XML node and inserts it as a fault block.

<u>Key</u>	<u>Descriptions</u>								
<i>code</i>	<b>Document</b> Contains the fault code and possible subcodes.								
	<table border="1"> <thead> <tr> <th><u>Key</u></th> <th><u>Descriptions</u></th> </tr> </thead> <tbody> <tr> <td><i>namespaceName</i></td> <td><b>String</b> Namespace name for the SOAP fault code.</td> </tr> <tr> <td><i>localName</i></td> <td><b>String</b> Code that identifies the fault.</td> </tr> <tr> <td><i>subCodes</i></td> <td> <p><b>Document List</b> Optional. One or more subcodes that provide further detail. For each subcode, include a Document in the <i>subCodes</i> Document List; each Document should contain:</p> <ul style="list-style-type: none"> <li>■ <i>namespaceName</i> for the subcode</li> <li>■ <i>localName</i> that identifies the subcode</li> </ul> </td> </tr> </tbody> </table>	<u>Key</u>	<u>Descriptions</u>	<i>namespaceName</i>	<b>String</b> Namespace name for the SOAP fault code.	<i>localName</i>	<b>String</b> Code that identifies the fault.	<i>subCodes</i>	<p><b>Document List</b> Optional. One or more subcodes that provide further detail. For each subcode, include a Document in the <i>subCodes</i> Document List; each Document should contain:</p> <ul style="list-style-type: none"> <li>■ <i>namespaceName</i> for the subcode</li> <li>■ <i>localName</i> that identifies the subcode</li> </ul>
<u>Key</u>	<u>Descriptions</u>								
<i>namespaceName</i>	<b>String</b> Namespace name for the SOAP fault code.								
<i>localName</i>	<b>String</b> Code that identifies the fault.								
<i>subCodes</i>	<p><b>Document List</b> Optional. One or more subcodes that provide further detail. For each subcode, include a Document in the <i>subCodes</i> Document List; each Document should contain:</p> <ul style="list-style-type: none"> <li>■ <i>namespaceName</i> for the subcode</li> <li>■ <i>localName</i> that identifies the subcode</li> </ul>								
<i>reasons</i>	<p><b>Document List</b> Reasons for the SOAP fault. Each Document in the Document List contains a human readable explanation of the cause of the fault.</p> <table border="1"> <thead> <tr> <th><u>Key</u></th> <th><u>Descriptions</u></th> </tr> </thead> <tbody> <tr> <td><i>*body</i></td> <td><b>String</b> Text explaining the cause of the fault.</td> </tr> </tbody> </table>	<u>Key</u>	<u>Descriptions</u>	<i>*body</i>	<b>String</b> Text explaining the cause of the fault.				
<u>Key</u>	<u>Descriptions</u>								
<i>*body</i>	<b>String</b> Text explaining the cause of the fault.								

---

	<i>@lang</i>	<b>String</b> Optional. Language for the human readable description.
	<i>node</i>	<b>String</b> Optional. URI to the SOAP node where the fault occurred.
	<i>role</i>	<b>String</b> Optional. Role in which the node was operating at the point the fault occurred.
	<i>detail</i>	<b>Document</b> Optional. Application-specific details about the SOAP fault.
<i>validate</i>		<p><b>String</b> Optional. Flag that indicates whether you want the service to validate <i>soapFault/detail</i> against the IS document type specified in the <i>documentType</i> input parameter. If you do not use <i>documentType</i> to specify an IS document, the validation will fail.</p> <p>Set <i>validate</i> to:</p> <ul style="list-style-type: none"> <li>■ <code>true</code> to validate <i>soapFault/detail</i> . If the validation fails, an exception is thrown.</li> <li>■ <code>false</code> to skip validating <i>soapFault/detail</i> This is the default.</li> </ul>

## Output Parameters

---

<i>status</i>	<p><b>String</b> Flag indicating the outcome of the service. A value of:</p> <ul style="list-style-type: none"> <li>■ <code>true</code> indicates that adding the fault block succeeded.</li> <li>■ <code>false</code> indicates that adding the fault block failed.</li> </ul>
---------------	---

**Note:** If the SOAP message already contains a fault block, the service sets *status* to false.

## See Also

[pub.soap.handler:getFaultBlock](#)

---

## pub.soap.handler:addHeaderBlock

WmPublic. Inserts a document into a SOAP message as a new header block.

---

## Input Parameters

---

*messageContext*      **Object** Message context containing the SOAP message to which to add a header block.

A message context contains properties for the SOAP message and provides access to the SOAP message. Integration Server creates the message context and passes it to the header handler. All handlers invoked by a given instance of a SOAP request or SOAP response use the same message context, which enables you to use the message context to pass information among handlers.

*documentType*      **String** Name of the IS document type that specifies the structure and namespaces of the document to use as a new header block.

Specify a document type that Integration Server created while creating the consumer or the WSDL first provider web service descriptor or one that was created using the [pub.soap.handler:generateDocumentTypesFromWSDL](#) service.

The QName of the first element in the IS document type is used as the QName of the header block to add to the SOAP message.

The *inputHeaderBlock/headerDocument* must be a instance of this document type.

*inputHeaderBlock*      **Document** Contains information used to create the header block to add to the SOAP message.

Key	Description
<i>mustUnderstand</i>	<p><b>String</b> Optional. Flag that indicates whether a mustUnderstand attribute is added to the new header block. The mustUnderstand attribute specifies whether recipients (the actor or role at which the header is targeted) are required to process a header entry. Recipients that cannot process a mandatory header entry must reject the message and return a SOAP fault.</p> <p>Set to:</p> <ul style="list-style-type: none"> <li>■ <code>true</code> to indicate that the attribute <code>mustUnderstand="true"</code> will be added</li> </ul>

to the header block. This indicates that processing the header entry is required.

- `false` to indicate that the `mustUnderstand` attribute will not be added to the header block. This indicates that processing the header entry is optional.

There is no default value.

If you do not set `mustUnderstand`, Integration Server omits the `mustUnderstand` attribute from the header entry, which is equivalent to setting `mustUnderstand` to `false`.

**Note:** In SOAP 1.2, the values of the `mustUnderstand` attribute changed from 0 and 1 to True and False; however, Integration Server processes both sets of values the same and performs any necessary conversions.

*role*

**String** Optional. Target of the header entry. The value of `role` determines the value of the actor or role attribute for the header entry. The actor or role attribute specifies a URI for the recipient of a header entry.

There is no default value. If you do not set `role`, Integration Server omits the actor attribute from the header entry, which is equivalent to setting `role` to `Ultimate receiver`.

- **For SOAP 1.1, set to:**

`Ultimate receiver` to omit the actor attribute from the header block. This indicates that the recipient is the ultimate destination of the SOAP message.

Next to add an actor attribute with the value `"http://schemas.xmlsoap.org/soap/actor/next"` to the header block.

None to add an actor attribute with the value "http://www.w3.org/2003/05/soap-envelope/role/none" to the header block.

*User-specified value* to specify the target of the header block. Typically, this will be a URI.

■ **For SOAP 1.2, set to:**

Ultimate receiver to omit the role attribute from the header block. This indicates that the recipient is the ultimate destination of the SOAP message.

Next to add a role attribute with the value "http://schemas.xmlsoap.org/soap-envelope/role/next" to the header block.

None to add a role attribute with the value "http://www.w3.org/2003/05/soap-envelope/role/none" to the header block.

*User-specified value* to specify the target of the header block. Typically, this will be a URI.

*headerDocument*

**Document** Document to add as a header block. Integration Server converts the document to an XML node and inserts it as a header block.

*validate*

**String** Optional. Flag that indicates whether you want the service to validate the *inputHeaderBlock/headerDocument* document against the IS document type specified in the *documentType* input parameter. If you do not use *documentType* to specify an IS document type, the validation will fail.

Set *validate* to:

- `true` to validate *inputHeaderBlock/headerDocument* . If the validation fails, an exception is thrown.
- `false` to skip validating *inputHeaderBlock/headerDocument* . This is the default.

---

## Output Parameters

---

None.

## Usage Notes

This service replaces [pub.soap.handler:addHeaderElement](#), which is deprecated.

## See Also

[pub.soap.handler:getHeaderBlock](#)

[pub.soap.handler:removeHeaderBlock](#)

---

# pub.soap.handler:addHeaderElement

WmPublic. *Deprecated* - Replaced by [pub.soap.handler:addHeaderBlock](#).

Inserts a document into a SOAP message as a new header element (block).

---

## Input Parameters

---

*messageContext*      **Object** Message context containing the SOAP message to which to add a header element.

A message context contains properties for the SOAP message and provides access to the SOAP message. Integration Server creates the message context and passes it to the header handler. All handlers invoked by a given instance of a SOAP request or SOAP response use the same message context, which enables you to use the message context to pass information among handlers.

*QName*      **Document** Optional. Qualified name (namespace name and local name) of the header element to add.

The *QName* document references the [pub.soap.utils:QName](#) document type.

**Note:** If you do not specify *QName* value, you must specify *documentType* .

*headerDocument*      **Document** Document to add as a header element. Integration Server converts the document to an XML node and inserts it as a child element of the header element specified in *QName* .

*documentType*      **String** Optional. Name of the IS document type that specifies the structure and namespaces of the document to use as a new header element. Integration Server uses the universal

name assigned to the IS document type to determine the qualified name to use for the new header element. If you specify *documentType*, *headerDocument* must be an instance of this document type.

**Note:** If you do not specify *documentType*, you must specify *QName*.

### *mustUnderstand*

**String** Optional. Sets the value of the `mustUnderstand` attribute for the new header element (block). The `mustUnderstand` attribute specifies whether recipients (the actor or role at which the header is targeted) are required to process a header entry. Recipients that cannot process a mandatory header entry must reject the message and return a SOAP fault.

Set to:

- `true` to indicate that processing the header entry is optional.
- `false` to indicate that processing the header entry is mandatory.

There is no default value.

If you do not set *mustUnderstand*, Integration Server omits the `mustUnderstand` attribute from the header entry, which is equivalent to setting *mustUnderstand* to `false`.

**Note:** In SOAP 1.2, the values of the `mustUnderstand` attribute changed from 0 and 1 to `True` and `False`; however, Integration Server processes both sets of values the same and performs any necessary conversions.

### *actor*

**String** Optional. Target of the header entry. The value of *actor* determines the value of the actor or role attribute for the header entry. The actor or role attribute specifies a URI for the recipient of a header entry.

If you do not specify a value for *actor*, the actor or role attribute will be blank in the SOAP header. In SOAP 1.1, this indicates that the recipient is the ultimate destination of the SOAP message. In SOAP 1.2, this is equivalent to supplying that attribute with the value `"http://www.w3.org/2003/05/soap-envelope/role/ultimateReceiver"`.

**Note:** In SOAP 1.2, the actor attribute is named `role`; however, Integration Server processes both names the same and performs any necessary conversions.

## Output Parameters

None.

## Usage Notes

*QName* and *documentType* are mutually exclusive. Even though the parameters are optional, you must specify one or the other. If you do not specify either, Integration Server displays the following error:

```
[ISS.0088.9422] One of the mutually exclusive parameter QName or documentType is missing or invalid.
```

If you specify values for *QName* and *documentType*, Integration Server uses the *QName* value and ignores *documentType*.

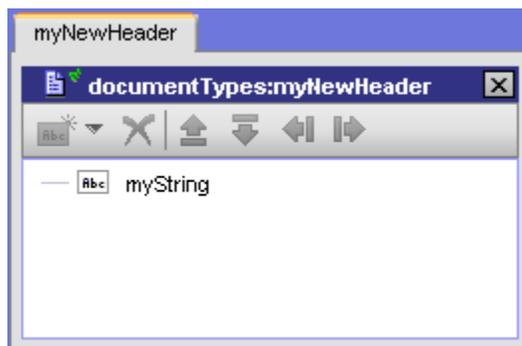
For more information about the `mustUnderstand` and `actor` attributes in SOAP 1.1, see the *Simple Object Access Protocol (SOAP) 1.1 - W3C Note 08 May 2000* at <http://www.w3.org/TR/SOAP/>.

For more information about the `mustUnderstand` and `role` attributes in SOAP 1.2, see the *Simple Object Access Protocol (SOAP) 1.2 specification* at <http://www.w3.org/TR/SOAP12/>.

## Example

Suppose that *messageContext* contains a SOAP message with an empty SOAP header and you want to add a header element by passing the `pub.soap.handler:addHeaderElement` service the following input parameters:

Input Parameter	Provided Value
<i>QName</i>	<i>namespaceName</i> userHandlerNamespaceName
	<i>localName</i> userHandlerLocalName
<i>headerDocument</i>	An instance of <code>documentTypes:myNewHeader</code> , which contains a single field of type <code>String</code> named <i>myString</i> . The value of <i>myString</i> is: Value of <code>myString</code> field.



Input Parameter	Provided Value
<i>documentType</i>	Not provided.
<i>mustUnderstand</i>	true
<i>actor</i>	soapActor

Execution of the [pub.soap.handler:addHeaderElement](#) service results in this SOAP header for SOAP 1.1:

```
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Header
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
<HDR:userHandlerLocalName
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:HDR="userHandlerNamespaceName"
SOAP-ENV:actor="soapActor" SOAP-ENV:mustUnderstand="1">
<myString>Value of myString field.</myString>
</HDR:userHandlerLocalName>
</SOAP-ENV:Header>
<SOAP-ENV:Body>
...
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Integration Server uses HDR as the namespace prefix for the namespace name of the header.

### See Also

[pub.soap.handler:getHeaderElement](#)  
[pub.soap.handler:removeHeaderElement](#)

## pub.soap.handler:generateDocumentTypesFromWSDL

WmPublic. Generates IS document types from a WSDL.

### Input Parameters

*wSDLURL* **String** Optional. Network accessible URL to a WSDL document or the path to and name of a WSDL on the same file system as the Integration Server.

**Note:** All the XML Schema definitions and WSDL documents imported by or included by the target WSDL document must be network accessible or on the local file system as well.

---

<i>wSDLString</i>	<b>String</b> Optional. WSDL document as a string.  <b>Note:</b> All the XML Schema definitions and WSDL documents imported by or included by the target WSDL document must be network accessible or on the local file system as well.
<i>targetPackageName</i>	<b>String</b> Name of the package in which to place the generated IS document types on Integration Server.
<i>targetFolderName</i>	<b>String</b> Name of the folder in which to place the generated IS document types. Use the format <i>folder.subfolder</i> to specify the folder name. The folder must be empty or must not yet exist.
<i>generateHeaderDocs</i>	<b>String</b> Optional. Flag indicating whether the service should generate the documents corresponding to headers in the SOAP messages described in the WSDL document. Set to: <ul style="list-style-type: none"><li>■ <code>true</code> to generate the documents. This is the default.</li><li>■ <code>false</code> to skip generating the documents.</li></ul>
<i>generateBodyDocs</i>	<b>String</b> Optional. Flag indicating whether the service should generate the documents corresponding to SOAP body in the SOAP messages described in the WSDL document. Set to: <ul style="list-style-type: none"><li>■ <code>true</code> to generate the documents.</li><li>■ <code>false</code> to skip generating the documents. This is the default.</li></ul>
<i>generateFaultDocs</i>	<b>String</b> Optional. Flag indicating whether the service should generate the documents corresponding to SOAP faults in the SOAP messages described in the WSDL document. Set to: <ul style="list-style-type: none"><li>■ <code>true</code> to generate the documents.</li><li>■ <code>false</code> to skip generating the documents. This is the default.</li></ul>
<i>generateXOPObjectForBase64Binary</i>	<b>String</b> Optional. Flag indicating whether the service should generate fields of type <code>com.wm.util.XOPObject</code> corresponding to the <code>xsd:base64Binary</code> elements. Set to: <ul style="list-style-type: none"><li>■ <code>true</code> to generate fields of type <code>com.wm.util.XOPObject</code>.</li><li>■ <code>false</code> to skip generating fields of type <code>com.wm.util.XOPObject</code>. This is the default.</li></ul>
<i>contentModelCompliance</i>	<b>String</b> Optional. Flag that specifies how strictly the service represents content models from the XML Schema definition in the generated IS document types. Set to:

- **Strict** to generate the IS document type only if Integration Server can represent the content models defined in the XML Schema definition correctly. Document type generation fails if Integration Server cannot accurately represent the content models in the source XML Schema definition.

Currently, Integration Server does not support repeating model groups, nested model groups, or the any attribute. If you select strict compliance, Integration Server does not generate an IS document type from any XML schema definition that contains those items.

- **Lax** to generate an IS document type that correctly represents the content models for the complex types defined in the XML schema definition, when possible. If Integration Server cannot correctly represent the content model in the XML Schema definition in the resulting IS document type, Integration Server generates the IS document type using a compliance mode of None.

When compliance is set to lax, Integration Server will generate the IS document type even if the content models in the XML schema definition cannot be represented correctly.

- **None** to generate an IS document type that does not necessarily represent or maintain the content models in the source XML Schema definition.

When compliance is set to none, Integration Server generates IS document types the same way they were generated in Integration Server releases prior to version 8.2.

## Output Parameters

*warnings*

**Document List** Conditional. Contains any warnings encountered while generating IS document types from the provided WSDL.

## Usage Notes

*wSDLURL* and *wSDLString* are mutually exclusive. Even though the parameters are optional, you must specify one or the other. If you do not specify either, Integration Server displays the following error:

```
ISS.0088.9422 Either parameter {0} or {1} must be provided.
```

If you specify values for *wSDLURL* and *wSDLString*, Integration Server uses *wSDLString* and ignores *wSDLURL*.

If the WSDL provided in *wSDLURL* or *wSDLString* is invalid, the service ends in error.

If the package specified in *targetPackageName* does not exist, the service ends in error.

If the folder specified in *targetFolderName* does not exist, Integration Server creates it when the service executes.

If the folder specified in *targetFolderName* exists but is not empty, the service ends in error.

If you want execute this service for the same WSDL more than once, make sure to specify a different *targetFolderName* or delete the contents of *targetFolderName* before re-executing the service for the WSDL again.

If the service ends in error, it throws any errors or warnings as an exception and does not create any IS document types. However, if the service encounters warnings, it places the warnings in the *warnings* field and generates any IS document types.

---

## pub.soap.handler:getBodyBlock

WmPublic. Retrieves a body block from a SOAP message.

### Input Parameters

---

<i>messageContext</i>	<p><b>Object</b> Message context containing the SOAP message from which to retrieve the body block.</p> <p>A message context contains properties for the SOAP message and provides access to the SOAP message. Integration Server creates the message context and passes it to the handler. All handlers invoked by a given instance of a SOAP request or SOAP response use the same message context, which enables you to use the message context to pass information among handlers.</p>
<i>documentType</i>	<p><b>String</b> Optional. Fully qualified name of the IS document type that specifies the structure of the SOAP body block to be retrieved from the SOAP message.</p> <p>Specify a document type that Integration Server created while creating the consumer or the WSDL first provider web service descriptor or one that was created using the <a href="#">pub.soap.handler:generateDocumentTypesFromWSDL</a> service.</p> <p>If you do not specify a document type, the service performs a literal conversion of body block; that is, the conversion will not consider style or use.</p>
<i>validate</i>	<p><b>String</b> Optional. Flag that indicates whether you want the service to validate the document returned in the <i>outputBodyBlock/bodyDocument</i> parameter against the IS document type specified in the <i>documentType</i> input parameter.</p>

If you do not use *documentType* to specify an IS document type, the validation will fail.

Set *validate* to:

- `true` to validate the body block. If the validation fails, an exception is thrown.
- `false` to skip validating the body block. This is the default.

## Output Parameters

---

*outputBodyBlock* **Document** Contents of the body block.

Key	Description						
<i>operationQName</i>	<p><b>Document</b> Qualified name for the web service wrapper that wraps the XML payload. The service only returns <i>operationQName</i> for RPC/Encoded and RPC/Literal services.</p> <p>The <i>operationQName</i> document references the <a href="#">pub.soap.utils:QName</a> document type.</p>						
	<table border="1"> <thead> <tr> <th style="text-decoration: underline;">Key</th> <th style="text-decoration: underline;">Description</th> </tr> </thead> <tbody> <tr> <td><i>namespaceName</i></td> <td><b>String</b> Namespace name for the web service operation.</td> </tr> <tr> <td><i>localName</i></td> <td><b>String</b> Name of the web service operation on which the handler is being written.</td> </tr> </tbody> </table>	Key	Description	<i>namespaceName</i>	<b>String</b> Namespace name for the web service operation.	<i>localName</i>	<b>String</b> Name of the web service operation on which the handler is being written.
Key	Description						
<i>namespaceName</i>	<b>String</b> Namespace name for the web service operation.						
<i>localName</i>	<b>String</b> Name of the web service operation on which the handler is being written.						
<i>bodyDocument</i>	<p>Contents of the body block. If <i>documentType</i> was specified, the service uses the IS document type to determine the structure of <i>bodyDocument</i>. Otherwise, the service performs a literal conversion of body block; that is, the conversion will not consider style or use.</p>						

### See Also

[pub.soap.handler:addBodyBlock](#)  
[pub.soap.handler:removeBodyBlock](#)

---

## pub.soap.handler:getBodyBlockQNames

WmPublic. Returns the QName for each body block in a SOAP message.

**Note:** RPC/Encoded and RPC/Literal services can have only one body block. However, Document/Literal services can have multiple body blocks.

---

### Input Parameters

*messageContext* **Object** Message context containing the SOAP message from which to retrieve the body block QNames.

A message context contains properties for the SOAP message and provides access to the SOAP message. Integration Server creates the message context and passes it to the handler. All handlers invoked by a given instance of a SOAP request or SOAP response use the same message context, which enables you to use the message context to pass information among handlers.

---

### Output Parameters

*bodyBlockQNames* **Document List** A list of documents containing the qualified names (namespace name and local name) for each body block.

The *bodyBlockQName* document references the [pub.soap.utils:QName](#) document type.

### See Also

[pub.soap.handler:removeBodyBlock](#)

---

## pub.soap.handler:getFaultBlock

WmPublic. Retrieves a fault block from a SOAP message.

---

### Input Parameters

*messageContext* **Object** Message context containing the SOAP message from which to retrieve the fault block.

A message context contains properties for the SOAP message and provides access to the SOAP message. Integration Server

creates the message context and passes it to the handler. All handlers invoked by a given instance of a SOAP request or SOAP response use the same message context, which enables you to use the message context to pass information among handlers.

*documentTypes*

**String List** Optional. Fully qualified name of the possible IS document types that could represent the structure of *soapFault/detail*.

Specify document types that Integration Server created while creating the consumer or the WSDL first provider web service descriptor or ones that were created using the [pub.soap.handler.generateDocumentTypesFromWSDL](#) service.

If you do not specify any document type, the service performs a literal conversion of the document; that is, the conversion will not consider style or use.

*validate*

**String** Optional. Flag that indicates whether you want the service to validate *soapFault/detail* against one of the IS document types specified in the *documentTypes* input parameter.

Set *validate* to:

- `true` to validate the contents of *soapFault/detail*.  
To validate the service uses an IS document type from *documentTypes* that matches the structure. If none of the IS document types specified in *documentTypes* match, validation fails. If you do not use *documentTypes* to specify IS document types, the validation fails. If the validation fails, an exception is thrown.
- `false` to skip validating the contents of *soapFault/detail*. This is the default.

## Output Parameters

---

**Document** Contents of the fault block.

<u>Key</u>	<u>Description</u>
<i>soapFault</i>	<b>Document</b> The retrieved fault block.
<u>Key</u>	<u>Description</u>

<i>code</i>	<b>Document</b> Contains the fault code and possible subcodes.								
	<table> <thead> <tr> <th><u>Key</u></th> <th><u>Descriptions</u></th> </tr> </thead> <tbody> <tr> <td><i>namespaceName</i></td> <td><b>String</b> Namespace name for the SOAP fault code.</td> </tr> <tr> <td><i>localName</i></td> <td><b>String</b> Code that identifies the fault.</td> </tr> <tr> <td><i>subCodes</i></td> <td> <b>Document List</b> Subcodes that provide further detail. Each Document in the <i>subCodes</i> Document List contains: <ul style="list-style-type: none"> <li>■ <i>namespaceName</i> for the subcode</li> <li>■ <i>localName</i> that identifies the subcode</li> </ul> </td> </tr> </tbody> </table>	<u>Key</u>	<u>Descriptions</u>	<i>namespaceName</i>	<b>String</b> Namespace name for the SOAP fault code.	<i>localName</i>	<b>String</b> Code that identifies the fault.	<i>subCodes</i>	<b>Document List</b> Subcodes that provide further detail. Each Document in the <i>subCodes</i> Document List contains: <ul style="list-style-type: none"> <li>■ <i>namespaceName</i> for the subcode</li> <li>■ <i>localName</i> that identifies the subcode</li> </ul>
<u>Key</u>	<u>Descriptions</u>								
<i>namespaceName</i>	<b>String</b> Namespace name for the SOAP fault code.								
<i>localName</i>	<b>String</b> Code that identifies the fault.								
<i>subCodes</i>	<b>Document List</b> Subcodes that provide further detail. Each Document in the <i>subCodes</i> Document List contains: <ul style="list-style-type: none"> <li>■ <i>namespaceName</i> for the subcode</li> <li>■ <i>localName</i> that identifies the subcode</li> </ul>								
<i>reasons</i>	<b>Document List</b> Reasons for the SOAP fault. Each Document in the Document List contains a human readable explanation of the cause of the fault.								
	<table> <thead> <tr> <th><u>Key</u></th> <th><u>Descriptions</u></th> </tr> </thead> <tbody> <tr> <td><i>*body</i></td> <td><b>String</b> Text explaining the cause of the fault.</td> </tr> <tr> <td><i>@lang</i></td> <td><b>String</b> Optional. Language for the human readable description.</td> </tr> </tbody> </table>	<u>Key</u>	<u>Descriptions</u>	<i>*body</i>	<b>String</b> Text explaining the cause of the fault.	<i>@lang</i>	<b>String</b> Optional. Language for the human readable description.		
<u>Key</u>	<u>Descriptions</u>								
<i>*body</i>	<b>String</b> Text explaining the cause of the fault.								
<i>@lang</i>	<b>String</b> Optional. Language for the human readable description.								
<i>node</i>	<b>String</b> URI to the SOAP node where the fault occurred.								
<i>role</i>	<b>String</b> Role in which the node was operating at the point the fault occurred.								
<i>detail</i>	<b>Document</b> Application-specific details about the SOAP fault.								

**See Also**

[pub.soap.handler:addFaultBlock](#)

---

## pub.soap.handler:getHeaderBlock

WmPublic. Retrieves a header block from a SOAP message.

## Input Parameters

---

<i>messageContext</i>	<p><b>Object</b> Message context containing the SOAP message from which to retrieve a header block.</p> <p>A message context contains properties for the SOAP message and provides access to the SOAP message. Integration Server creates the message context and passes it to the header handler. All handlers invoked by a given instance of a SOAP request or SOAP response use the same message context, which enables you to use the message context to pass information among handlers.</p>
<i>documentType</i>	<p><b>String</b> Fully qualified name of the IS document type that specifies the structure to apply to the content of the header block.</p> <p>Specify a document type that Integration Server created while creating the consumer or the WSDL first provider web service descriptor or one that was created using the <a href="#">pub.soap.handler:generateDocumentTypesFromWSDL</a> service.</p> <p>Integration Server uses the QName of the first field of the document type to determine which header block to retrieve from the SOAP message.</p>
<i>validate</i>	<p><b>String</b> Optional. Flag that indicates whether you want the service to validate the document returned in the <i>outputHeaderBlock/headerDocument</i> parameter against the IS document type specified in the <i>documentType</i> input parameter.</p> <p>Set <i>validate</i> to:</p> <ul style="list-style-type: none"> <li>■ <code>true</code> to validate <i>outputHeaderBlock/headerDocument</i> . If the validation fails, an exception is thrown.</li> <li>■ <code>false</code> to skip validating <i>outputHeaderBlock/headerDocument</i> . This is the default.</li> </ul>

## Output Parameters

---

<i>outputHeaderBlock</i>	<p><b>Document List</b> Content of the requested header block(s) as IData. This service returns multiple header blocks if the QName of the first field of the document type matches multiple header blocks in the SOAP message.</p>
--------------------------	---

Key	Description
-----	-------------

<i>mustUnderstand</i>	<p><b>String</b> Flag indicating whether the <code>mustUnderstand</code> attribute is present in the retrieved header block. A value of:</p> <ul style="list-style-type: none"> <li>■ <code>true</code> indicates that the header block contained <code>mustUnderstand="true"</code>.</li> <li>■ <code>false</code> indicates that the <code>mustUnderstand</code> attribute was absent from the header block or <code>mustUnderstand="false"</code>.</li> </ul>
<i>role</i>	<p><b>String</b> For a SOAP 1.1 message, specifies the value of the actor attribute. For a SOAP 1.2 message, specifies the value of the role attribute.</p> <p>The <i>role</i> parameter is not returned if the actor attribute is absent from a SOAP 1.1 message or if the role attribute is absent from a SOAP 1.2 message.</p>
<i>headerDocument</i>	<p><b>Document</b> Contents of the header block. The IS document type specified for the <i>documentType</i> input parameter determines structure of <i>headerDocument</i></p>

## Usage Notes

This service replaces [pub.soap.handler.getHeaderElement](#), which is deprecated.

If the very first field in the IS document type specified for *documentType* does not contain a QName, the service returns an empty *outputHeaderBlock* and does not throw an error.

If the QName of the very first field in the IS document type specified for *documentType* does not match a QName of a header block in the SOAP message, the service returns an empty *outputHeaderBlock* and does not throw an error.

If the QName of the very first field in the IS document type specified for *documentType* matches a QName of a header block in the SOAP message but the content of the header block does not match the fields in the IS document type, then the retrieved *headerDocument* fails validation with the error

[ISS.0088.9432] SOAP Header data does not conform to Header Record

## See Also

[pub.soap.handler.addHeaderBlock](#)  
[pub.soap.handler.removeHeaderBlock](#)

---

## pub.soap.handler:getHeaderBlockQNames

WmPublic. Returns the QName for each header block in a SOAP message.

### Input Parameters

---

*messageContext*      **Object** Message context containing the SOAP message from which to retrieve the header block QNames.

A message context contains properties for the SOAP message and provides access to the SOAP message. Integration Server creates the message context and passes it to the header handler. All handlers invoked by a given instance of a SOAP request or SOAP response use the same message context, which enables you to use the message context to pass information among handlers.

### Output Parameters

---

*headerBlockQNames*      **Document List** A list of documents containing the qualified names (namespace name and local name) in the header block.

The *headerBlockQName* document references the [pub.soap.utils:QName](#) document type.

### Usage Notes

You can use the [pub.soap.handler:getHeaderBlockQNames](#) to identify the header block QNames in a SOAP message and then use [pub.soap.handler:removeHeaderBlock](#) to remove specific header blocks.

### See Also

[pub.soap.handler:removeHeaderBlock](#)

---

## pub.soap.handler:getHeaderElement

WmPublic. *Deprecated* - Replaced by [pub.soap.handler:getHeaderBlock](#).

Retrieves a header element from a SOAP message.

---

## Input Parameters

---

<i>messageContext</i>	<p><b>Object</b> Message context containing the SOAP message from which to retrieve a header element.</p> <p>A message context contains properties for the SOAP message and provides access to the SOAP message. Integration Server creates the message context and passes it to the header handler. All handlers invoked by a given instance of a SOAP request or SOAP response use the same message context, which enables you to use the message context to pass information among handlers.</p>
<i>QName</i>	<p><b>Document</b> Optional. Qualified name (namespace name and local name) of the header element to retrieve.</p> <p>The <i>QName</i> document references the <a href="#">pub.soap.util:QName</a> document type. If you do not specify <i>QName</i>, you must specify <i>documentType</i>.</p>
<i>documentType</i>	<p><b>String</b> Optional. Fully qualified name of the IS document type that specifies the structure to impose on the resulting document. Integration Server uses the explicit universal name assigned to the document type to determine which header element to retrieve from the SOAP message.</p> <p>If you do not specify <i>documentType</i>, you must specify <i>QName</i>.</p>

---

## Output Parameters

---

*outputHeaderDocument* **Document** Header element from the SOAP message in the form of a document (IData).

### Usage Notes

*QName* and *documentType* are mutually exclusive. Even though the parameters are optional, you must specify one or the other. If you do not specify either, Integration Server displays the following error:

```
[ISS.0088.9422] One of the mutually exclusive parameter QName or documentType is missing or invalid.
```

If you specify values for *QName* and *documentType*, Integration Server uses *QName* and ignores *documentType*.

### Example

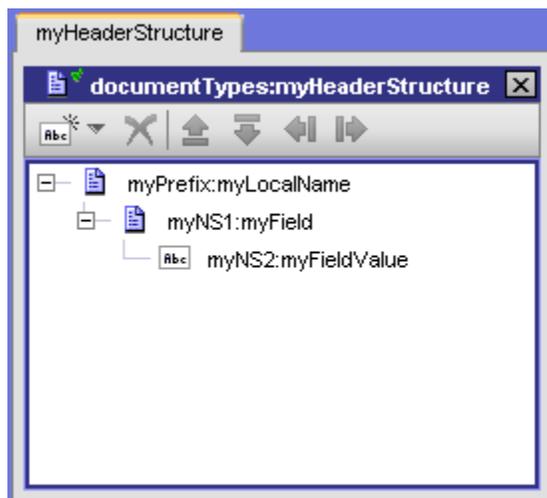
Suppose that *messageContext* contains a SOAP message with the following header:

```
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:user="userHandlerNamespaceName" xmlns:pfx="pfx1namespace"
xmlns:ns1="ns1namespace" xmlns:ns2="ns2namespace">
<SOAP-ENV:Header>
<user:userHandlerLocalName>
<pfx:myLocalName>
<ns1:myField>
<ns2:myFieldValue>someValue</ns2:myFieldValue>
</ns1:myField>
</pfx:myLocalName>
</user:userHandlerLocalName>
</SOAP-ENV:Header>
<SOAP-ENV:Body>
...
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Additionally, suppose that `pub.soap.handler.getHeaderElement` uses the following input values, where `messageContext` has already been obtained:

Input Parameter	Provided Value
<i>QName</i>	<i>namespaceName</i> userHandlerNamespaceName
	<i>localName</i> userHandlerLocalName
<i>documentType</i>	documentTypes:myHeaderStructure

The structure of `documentTypes:myHeaderStructure` looks like this:

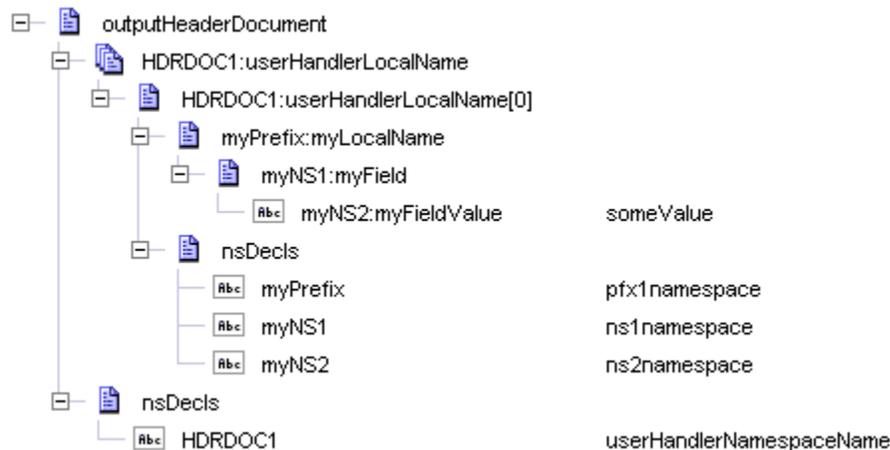


The prefixes in `documentTypes:myHeaderStructure` refer to the following namespaces.

Prefix	Namespace
myPrefix	pfx1namespace

Input Parameter	Provided Value
myNS1	ns1namespace
myNS2	ns2namespace

Execution of the [pub.soap.handler:getHeaderElement](#) service results in the following value for *outputHeaderDocument* :



Integration Server uses the following conventions in *outputHeaderDocument* :

- *outputHeaderDocument* always contains a document list named *HDRDOC1:localName* . The document list contains the header retrieved by the [pub.soap.handler:getHeaderElement](#) service.
- Integration Server uses HDRDOC1 as the prefix for the header element (block). The value of HDRDOC1 is the namespace name portion of the QName. The *outputHeaderDocument/nsDecls* document identifies the namespace associated with the namespace prefix of the requested header element (block).
- The *HDRDOC1:localName* [0] document contains an *nsDecls* document that identifies the namespace prefixes used within the retrieved header element. Integration Server replaces the prefixes used in the SOAP envelope with the prefixes that the document type specifies for the same namespaces.

Integration Server uses the same general structure when placing SOAP headers in the pipeline for IS services acting as web services. For more information, see "Server Configuration Parameters" in *webMethods Integration Server Administrator's Guide*.

## See Also

- [pub.soap.handler:addHeaderElement](#)
- [pub.soap.handler:removeHeaderElement](#)

---

## pub.soap.handler:getInitialSOAPRequest

WmPublic. Gets the initial SOAP request message from a given message context.

### Input Parameters

---

*messageContext*      **Object** Message context from which to retrieve the initial SOAP request message.

A message context contains properties for the SOAP message and provides access to the SOAP message. Integration Server creates the message context and passes it to the header handler. All handlers invoked by a given instance of a SOAP request or SOAP response use the same message context, which enables you to use the message context to pass information among handlers.

### Output Parameters

---

*initialSOAPMessage*      **Object** Object of type `javax.xml.soap.SOAPMessage` that represents the initial SOAP request message.

### Usage Notes

You can use the initial SOAP request message retrieved by this service in the outbound callback service. For more information about outbound callback services, see *Web Services Developer's Guide*.

You can use the SOAP message retrieved by this service with any existing public service that takes an object of type `javax.xml.soap.SOAPMessage` as input.

In case of consumer web service descriptors, the [pub.soap.handler:getInitialSOAPRequest](#) service returns the initial outbound SOAP request that Integration Server builds just before sending the SOAP request to the web service provider.

---

## pub.soap.handler:getMessageAddressingProperties

WmPublic. Gets the message addressing properties of the SOAP message in the provided message context.

---

## Input Parameters

---

*messageContext*      **Object** Message context from which to retrieve the message addressing property value.

A message context contains properties for the SOAP message and provides access to the SOAP message. Integration Server creates the message context and passes it to the header handler. All handlers invoked by a given instance of a SOAP request or SOAP response use the same message context, which enables you to use the message context to pass information among handlers.

## Output Parameters

---

*messageAddressingProperties*      **Document** Value of the specified message addressing property.

<u>Key</u>	<u>Description</u>						
<i>messageID</i>	<b>String</b> Conditional. Unique identifier of the SOAP message.						
<i>relatesTo</i>	<b>Document List</b> Conditional. Contains the relationship information to another SOAP message.						
	<table border="1"> <thead> <tr> <th><u>Key</u></th> <th><u>Description</u></th> </tr> </thead> <tbody> <tr> <td><i>value</i></td> <td><b>String</b> Message ID of the related message.</td> </tr> <tr> <td><i>relationshipType</i></td> <td><b>String</b> Conditional. The relationship type.</td> </tr> </tbody> </table>	<u>Key</u>	<u>Description</u>	<i>value</i>	<b>String</b> Message ID of the related message.	<i>relationshipType</i>	<b>String</b> Conditional. The relationship type.
<u>Key</u>	<u>Description</u>						
<i>value</i>	<b>String</b> Message ID of the related message.						
<i>relationshipType</i>	<b>String</b> Conditional. The relationship type.						
<i>action</i>	<b>String</b> Conditional. WS-Addressing action specified in the message addressing property of the SOAP message.						
<i>to</i>	<b>Document</b> Conditional. The endpoint reference that specifies the address of the intended receiver of the SOAP message. The <i>to</i> endpoint reference includes:						

- *attributes* , which includes namespaceName, localname, and their values.
- *address* and its attributes and values.
- *referenceParameters*
- *metadata* and its attributes and elements
- *extensibleElements* , which are any other elements usually provided for future extensions.

*from*

**Document** Conditional. The endpoint reference that specifies the source of the SOAP message. The *from* endpoint reference includes:

- *attributes* , which includes namespaceName, localname, and their values.
- *address* and its attributes and values.
- *referenceParameters*
- *metadata* and its attributes and elements.
- *extensibleElements* , which are any other elements usually provided for future extensions.

*replyTo*

**Document** Conditional. The endpoint reference that specifies the destination address of the response (reply) message. The *replyTo* endpoint reference includes:

- *attributes* , which includes namespaceName, localname, and their values.
- *address* and its attributes and values.
- *referenceParameters*
- *metadata* and its attributes and elements.
- *extensibleElements* , which are any other elements usually provided for future extensions.

*faultTo*

**Document** Conditional. The endpoint reference that specifies the address to which the SOAP fault messages are routed. The *faultTo* endpoint reference includes:

- *attributes*, which includes namespaceName, localname, and their values.
- *address* and its attributes and values.
- *referenceParameters*
- *metadata* and its attributes and elements.
- *extensibleElements*, which are any other elements usually provided for future extensions.

---

## pub.soap.handler:getProperty

WmPublic. Gets the value of a specified property from a message context.

### Input Parameters

---

*messageContext*

**Object** Message context from which to retrieve a property value.

A message context contains properties for the SOAP message and provides access to the SOAP message. Integration Server creates the message context and passes it to the header handler. All handlers invoked by a given instance of a SOAP request or SOAP response use the same message context, which enables you to use the message context to pass information among handlers. For example, in a chain of request header handlers, the first request header handler could set a message property that the second request header handler retrieves.

*key*

**String** Name of the selected property whose value to retrieve.

### Output Parameters

---

*value*

**Object** Value of the specified property.

### Usage Notes

To access the contents of the service pipeline, use the [pub.soap.handler:getServicePipeline](#) instead of the `pub.soap.handler:getProperty` service.

### See Also

[pub.soap.handler:getServicePipeline](#)

pub.soap.handler.removeProperty  
pub.soap.handler.setProperty

---

## pub.soap.handler:getServicePipeline

WmPublic. Gets the service pipeline from a given message context.

### Input Parameters

---

*messageContext*      **Object** Message context from which to get the service pipeline.

A message context contains properties for the SOAP message and provides access to the SOAP message. Integration Server creates the message context and passes it to the header handler. All handlers invoked by a given instance of a SOAP request or SOAP response use the same message context, which enables you to use the message context to pass information among handlers.

### Output Parameters

---

*servicePipeline*      **Document** Document (IData) containing the service pipeline.

The contents of *servicePipeline* depend on whether the `pub.soap.handler:getServicePipeline` service executes as part of a consumer handler chain or a provider handler chain and which type of handler service (request, response, or fault) executes the service.

#### For this consumer handler service...

#### servicePipeline contains...

Request handler service

Contents of the web service connector pipeline after any pipeline mapping or manipulation occurs during execution of the web service connector but just before Integration Server sends the SOAP request.

Response handler service

The pipeline that becomes the web service connector output pipeline. At this point, the pipeline does not contain data from SOAP response. Integration Server adds the data from the SOAP

	response after handler processing completes.
Fault handler service	The contents of the web service connector output pipeline. At this point, the pipeline does not contain data from the SOAP fault. Integration Server adds the data from the SOAP fault after handler processing completes.
<b><u>For this provider handler service...</u></b>	<b><u>servicePipeline contains...</u></b>
Request handler service	The pipeline passed as input to the endpoint service. At this point, the pipeline does not yet contain data from the SOAP request. Integration Server places data from the SOAP request in the pipeline after handler processing completes
Response handler service	The output pipeline of the endpoint service that corresponds to the invoked web service operation.  <i>servicePipeline</i> also contains data from the SOAP request if the endpoint service did not drop the data from the pipeline during service execution.
Fault handler service	The output pipeline of the endpoint service that corresponds to the invoked web service operation.  <i>servicePipeline</i> also contains data from the SOAP request if the endpoint service did not drop the data from the pipeline during service execution.

### Usage Notes

Use this service to give handler services access to the contents of the pipeline. The handler service can then pass pipeline contents to another service. For example, during execution of a handler service for a provider web service descriptor, you can use the `pub.soap.handler.getServicePipeline` service to:

- Pass pipeline data from the request handler service to the endpoint service.

- Pass pipeline data from the endpoint service to the response handler service or fault handler service.

During execution of a handler service for a consumer web service descriptor, you can use the `pub.soap.handler:getServicePipeline` service to:

- Pass pipeline data from the web service connector input to the request handler service.
- Pass pipeline data from the response handler service or fault handler service to the web service connector output

Use the `pub.soap.handler:getServicePipeline` service to access the pipeline instead of using the `pub.soap.handler:getProperty` service to access the `servicePipeline` property.

---

## pub.soap.handler:getSOAPMessage

WmPublic. Gets the SOAP message from a given message context.

### Input Parameters

---

*messageContext*      **Object** Message context from which to get the SOAP message.

A message context contains properties for the SOAP message and provides access to the SOAP message. Integration Server creates the message context and passes it to the header handler. All handlers invoked by a given instance of a SOAP request or SOAP response use the same message context, which enables you to use the message context to pass information among handlers.

### Output Parameters

---

*SOAPMessage*      **Object** Object of type `javax.xml.soap.SOAPMessage` that represents the SOAP message.

### Usage Notes

You can use the SOAP message retrieved by this service with any existing public service that takes an object of type `javax.xml.soap.SOAPMessage` as input. For example, you can use the SOAP message as input for the `pub.soap.utils:addBodyEntry` or `pub.soap.utils:addHeaderEntry` services.

### See Also

[pub.soap.handler:setSOAPMessage](#)

---

## pub.soap.handler:getWebServiceInvocationProperties

WmPublic. Fetches the web service invocation properties.

### Input Parameters

---

*messageContext*      **Object** Optional. Message context containing the SOAP message from which to retrieve the web service invocation properties.

A message context contains properties for the SOAP message and provides access to the SOAP message. Integration Server creates the message context and passes it to the handler. All handlers invoked by a given instance of a SOAP request or SOAP response use the same message context, which enables you to use the message context to pass information among handlers.

### Output Parameters

---

*soapAction*      **String** SOAP action associated with the SOAP message request.

*binderName*      **String** Name of the WSBinder being invoked.

*binding*      **String** Binding against which the service was invoked.

*portType*      **String** Port type against which the service was invoked.

*operationQName*      **Document** Qualified name for the web service wrapper. The service only returns *operationQName* for RPC/Encoded and RPC/Literal services.

Key	Description
<i>namespaceName</i>	<b>String</b> Namespace name for the web service operation.
<i>localName</i>	<b>String</b> Name of the web service operation that was invoked.

*style*      **String** Style of the SOAP message. Possible values are:

- document

---

	<ul style="list-style-type: none"><li>■ <code>rpc</code></li></ul>
<i>use</i>	<b>String</b> Use of the SOAP message. Possible values are: <ul style="list-style-type: none"><li>■ <code>literal</code></li><li>■ <code>encoded</code></li></ul>
<i>inbound</i>	<b>String</b> Flag indicating whether the message is incoming or outgoing. A value of: <ul style="list-style-type: none"><li>■ <code>true</code> indicates that the SOAP message is incoming.</li><li>■ <code>false</code> indicates that the SOAP message is outgoing.</li></ul>
<i>soapVersion</i>	<b>String</b> SOAP version of the message. Possible values are: <ul style="list-style-type: none"><li>■ <code>SOAP11</code></li><li>■ <code>SOAP12</code></li></ul>

---

## pub.soap.handler:handlerSpec

WmPublic. Specification to use as the signature for a service that acts as a header handler.

### Input Parameters

---

<i>messageContext</i>	<b>Object</b> Message context to be processed by the header handler.  A message context contains properties for the SOAP message and provides access to the SOAP message. Integration Server creates the message context and passes it to the header handler. All handlers invoked by a given instance of a SOAP request or SOAP response use the same message context, which enables you to use the message context to pass information among handlers.
-----------------------	--

### Output Parameters

---

<i>statusCode</i>	<b>String</b> Specifies the status of the handler service execution which indicates the action Integration Server will take next. The <i>statusCode</i> parameter must be an integer from 0-3. The action Integration Server takes for a particular status code depends on whether the service is registered as a request handler, response handler, or fault handler. For more
-------------------	---

---

information, see *Web Services Developer's Guide* or *Software AG Designer Online Help*.

If *statusCode* does not contain one of the specified values, Integration Server assumes a value of 0.

*faultMessage*

**String** Conditional. Text to be used in the fault message. Integration Server uses the *faultMessage* value for any handler service that returns a status of 2 and when a response handler service returns a status of 3. For more information, see *Web Services Developer's Guide* or *Software AG Designer Online Help*.

### Usage Notes

Services that act as header handlers do not need to use the `pub.handler:handlerSpec` specification to define the signature of the service. However, services that act as header handlers must take the input parameters and produce the output parameters identified in this specification.

This specification can be used as the signature for the any handler service (request, response, or fault).

Integration Server altered handler chain processing in version 8.0 SP1. If you want created in Integration Server version 7.x of 8.0 to use the handler chain processing behavior available in Integration Server 7.x and 8.0, set the `watt.server.ws.71xHandlerChainBehavior` server configuration parameter to true. For more information about setting server configuration parameters, see *webMethods Integration Server Administrator's Guide*.

### See Also

[pub.soap.handler:registerWmConsumer](#)  
[pub.soap.handler:registerWmProvider](#)

---

## pub.soap.handler:hasFaultMessage

WmPublic. Determines whether the SOAP message in a given message context contains a SOAP fault message.

### Input Parameters

---

*messageContext*

**Object** Message context from which to get the SOAP message.

A message context contains properties for the SOAP message and provides access to the SOAP message. Integration Server creates the message context and passes it to the header handler. All handlers invoked by a given instance

of a SOAP request or SOAP response use the same message context, which enables you to use the message context to pass information among handlers.

## Output Parameters

---

*hasFaultMessage*

**String** Flag indicating whether the SOAP message contains a SOAP fault. A value of:

- `True` indicates the SOAP message contains a SOAP fault.
- `False` indicates the SOAP message does not contain a SOAP fault.

## pub.soap.handler:listConsumer

---

WmPublic. Returns a list of the consumer handlers currently registered on Integration Server.

## Input Parameters

---

None.

## Output Parameters

---

*Handlers*

**Document List** A document list identifying the registered consumer handlers on Integration Server. Information about each consumer handler is contained in a separate document.

Key	Description
<i>descriptiveName</i>	<b>String</b> Descriptive name assigned to the consumer handler when it was registered.
<i>className</i>	<b>String</b> Class name of the handler.
<i>policyType</i>	<b>String</b> Conditional. Policy type of the handler. <i>policyType</i> does not apply to service handlers.
<i>handleRequestService</i>	<b>String</b> Conditional. Fully qualified name of the service used as the request header

handler. This parameter is returned only if a request handler service was registered with the consumer handler.

*handleResponseService* **String** Conditional. Fully qualified name of the service used as the response header handler. This parameter is returned only if a response handler service was registered with the consumer handler.

*handleFaultService* **String** Conditional. Fully qualified name of the service used as the fault header handler. This parameter is returned only if a fault handler service was registered with the consumer handler.

*Headers* **Document List** List of QNames for the headers (i.e., IS document types) registered with the consumer handler.

<u>Key</u>	<u>Description</u>
<i>namespace</i>	<b>String</b> Namespace portion of the header's qualified name.
<i>localName</i>	<b>String</b> Local portion of the header's qualified name.

---

## pub.soap.handler:listProvider

WmPublic. Returns a list of the provider handlers currently registered on Integration Server.

### Input Parameters

---

None.

### Output Parameters

---

*Handlers* **Document List** A document list identifying the registered consumer handlers on Integration Server. Information about each consumer handler is contained in a separate document.

<u>Key</u>	<u>Description</u>
<i>descriptiveName</i>	<b>String</b> Descriptive name given to the provider handler when it was registered.
<i>className</i>	<b>String</b> Class name of the handler.
<i>policyType</i>	<b>String</b> Conditional. Policy type of the handler. <i>policyType</i> does not apply to service handlers.
<i>handleRequestService</i>	<b>String</b> Conditional. Fully qualified name of the service used as the request header handler. This parameter is returned only if a request handler service was registered with the provider handler.
<i>handleResponseService</i>	<b>String</b> Conditional. Fully qualified name of the service used as the response header handler. This parameter is returned only if a response handler service was registered with the provider handler.
<i>handleFaultService</i>	<b>String</b> Conditional. Fully qualified name of the service used as the fault header handler. This parameter is returned only if a fault handler service was registered with the provider handler.
<i>Headers</i>	<b>Document List</b> List of QNames for the headers registered with the provider handler.
<u>Key</u>	<u>Description</u>
<i>namespace</i>	<b>String</b> Namespace portion of the header's qualified name.
<i>localName</i>	<b>String</b> Local portion of the header's qualified name.

---

## pub.soap.handler:registerConsumer

WmPublic. *Deprecated* - Replaced by [pub.soap.handler:registerWmConsumer](#).

Registers a handler based on JAX-RPC with a consumer Web service descriptor.

### Input Parameters

---

*descriptiveName*      **String** Name that you want to assign to the SOAP consumer handler.

*handler*                **Object** The instance of the handler object.

*handlerInfo*          **Object** Optional. The instance of the handlerInfo object.

### Output Parameters

---

None.

---

## pub.soap.handler:registerProvider

WmPublic. *Deprecated* - Replaced by [pub.soap.handler:registerWmProvider](#).

Registers a handler based on JAX-RPC with a provider Web service descriptor.

### Input Parameters

---

*descriptiveName*      **String** Name that you want to assign to the SOAP provider handler.

*handler*                **Object** The instance of the handler object.

*handlerInfo*          **Object** Optional. The instance of the handlerInfo object.

### Output Parameters

---

None.

---

## pub.soap.handler:registerWmConsumer

WmPublic. Registers a handler for use with consumer.

---

## Input Parameters

---

<i>descriptiveName</i>	<b>String</b> Name to assign to the consumer header handler. Each consumer header handler must have a unique name.
<i>QNameList</i>	<b>Document List</b> Optional. Qualified names of the headers on which the handler operates. In the document list, each document references the <a href="#">pub.soap.utils:QName</a> document type.
<i>handleRequestService</i>	<b>String</b> Optional. Fully qualified name of the service to use as the request header handler.
<i>handleResponseService</i>	<b>String</b> Optional. Fully qualified name of the service to use as the response header handler.
<i>handleFaultService</i>	<b>String</b> Optional. Fully qualified name of the service to use as the fault header handler.

## Output Parameters

---

None.

## Usage Notes

This service replaces [pub.soap.handler:registerConsumer](#), which is deprecated.

Before you register a consumer header handler, create the services that will act as the request, response, and fault header handlers.

Integration Server stores information about registered header handlers in memory. Integration Server does not persist registered header handler information across restarts. Consequently, you must register header handlers each time Integration Server starts. To accomplish this, create a service that registers a header handler and make that service a start up service for the package that contains the services that act as header handlers.

You can use a consumer header handler with consumer (WSD) only.

Specify a value for *QNameList* if you want to associate with handler with one or more QNames. Registering QNames with a handler provides the following benefits:

- Integration Server can perform mustUnderstand checking for the header with the QName at run time. If a service receives a SOAP message in which a header requires mustUnderstand processing by the recipient, Integration Server uses the header QName to locate the handler that processes the header. Note that the handler must be part of the handler chain for the WSD that contains the service.
- When adding headers to a WSD, Designer populate the list of IS document types that can be used as headers in the WSD with the IS document types whose QNames were registered with the handlers already added to the WSD. If you add a IS

document type as a header to a WSDL and the QName of that IS document type is not associated with a handler, Designer add the header but display a warning stating that there is not an associated handler.

- When consuming WSDL to create a provider or consumer WSDL, Integration Server automatically adds a handler to the resulting WSDL if the WSDL contains a QName supported by the handler.

Use the [pub.soap.handler:registerWmProvider](#) service to register a header handler for use with provider.

To unregister a consumer header handler, use the [pub.soap.handler:unregisterConsumer](#) service.

If you specify a service that does not exist for *handleRequest*, *handleResponse*, or *handleFaultService*, Integration Server throws this error:

```
[ISS.0088.9421] The service <serviceName> does not exist.
```

If a registered handler with the same *descriptiveName* already exists, Integration Server throws this error.

```
[ISS.0088.9423] Service handler <handlerName> is already registered.
```

### See Also

[pub.soap.handler:registerWmProvider](#)

---

## pub.soap.handler:registerWmProvider

WmPublic. Registers a header handler for use with provider.

### Input Parameters

---

<i>descriptiveName</i>	<b>String</b> Name to assign to the provider header handler. Each provider header handler must have a unique name.
<i>QNameList</i>	<b>Document List</b> Optional. Qualified names of the headers on which the handler operates. In the document list, each document references the <a href="#">pub.soap.utils:QName</a> document type.
<i>handleRequestService</i>	<b>String</b> Optional. Fully qualified name of the service to use as the request header handler.
<i>handleResponseService</i>	<b>String</b> Optional. Fully qualified name of the service to use as the response header handler.
<i>handleFaultService</i>	<b>String</b> Optional. Fully qualified name of the service to use as the fault header handler.

---

## Output Parameters

---

None.

## Usage Notes

This service replaces [pub.soap.handler:registerProvider](#), which is deprecated.

Before you register a provider header handler, create the services that will act as the request, response, and fault header handlers.

Integration Server stores information about registered header handlers in memory. Integration Server does not persist registered header handler information across restarts. Consequently, you need to register header handlers each time Integration Server starts. To accomplish this, create a service that registers a header handler and make that service a start up service for the package that contains the services that act as header handlers.

You can use a provider header handler with provider only.

Specify a value for *QNameList* if you want to associate with handler with one or more QNames. Registering QNames with a handler provides the following benefits:

- Integration Server can perform mustUnderstand checking for the header with the QName at run time. If a service receives a SOAP message in which a header requires mustUnderstand processing by the recipient, Integration Server uses the header QName to locate the handler that processes the header. Note that the handler must be part of the handler chain for the WSD that contains the service.
- When adding headers to a WSD, Designer populate the list of IS document types that can be used as headers in the WSD with the IS document types whose QNames were registered with the handlers already added to the WSD. If you add a IS document type as a header to a WSD and the QName of that IS document type is not associated with a handler, Designer add the header but display a warning stating that there is not an associated handler.
- When consuming WSDL to create a provider or consumer WSD, Integration Server automatically adds a handler to the resulting WSD if the WSDL contains a QName supported by the handler.

Use the [pub.soap.handler:registerProvider](#) service to register a header handler for use with consumer.

To unregister a provider header handler, use the [pub.soap.handler:unregisterProvider](#) service.

If you specify a service that does not exist for *handleRequest*, *handleResponse*, or *handleFaultService*, Integration Server throws this error:

```
[ISS.0088.9421] The service <serviceName> does not exist.
```

If a registered handler with the same *descriptiveName* already exists, Integration Server throws this error.

```
[ISS.0088.9423] Service handler <handlerName> is already registered.
```

**See Also**

[pub.soap.handler:registerProvider](#)

---

## pub.soap.handler:removeBodyBlock

WmPublic. Removes a body block from a SOAP message, leaving an empty SOAP body, <SOAP-ENV:Body></SOAP-ENV:Body>.

---

**Input Parameters**

*messageContext*

**Object** Message context that contains the SOAP message from which to remove the body block.

A message context contains properties for the SOAP message as well as providing access to the SOAP message. Integration Server creates the message context and passes it to the handler. All handlers invoked by a given instance of a SOAP request or SOAP response use the same message context. This enables you to use the message context to pass information between handlers.

---

**Output Parameters**

None.

**Usage Notes**

If the service encounters any error when removing the body block, it throws an exception.

If you execute the service against a SOAP message that already contains an empty SOAP body, the service performs no action.

**See Also**

[pub.soap.handler:addBodyBlock](#)

[pub.soap.handler:getBodyBlock](#)

---

## pub.soap.handler:removeHeaderBlock

WmPublic. Removes a header block from a SOAP message.

## Input Parameters

---

<i>messageContext</i>	<p><b>Object</b> Message context that contains the SOAP message from which to remove a header block.</p> <p>A message context contains properties for the SOAP message as well as providing access to the SOAP message. Integration Server creates the message context and passes it to the header handler. All handlers invoked by a given instance of a SOAP request or SOAP response use the same message context. This enables you to use the message context to pass information between handlers.</p>
<i>QName</i>	<p><b>Object</b> Optional. Qualified name of the header block to remove. The <i>QName</i> document references the <a href="#">pub.soap.util:QName</a> document type.</p> <p><b>Note:</b> Either <i>QName</i> or <i>documentType</i> must be supplied.</p>
<i>documentType</i>	<p><b>String</b> Optional. Fully qualified name of the IS document type that corresponds to the structure of the header block that you want to remove. Integration Server uses the <i>QName</i> of the first element in the IS document type to determine which header block to remove.</p> <p><b>Note:</b> Either <i>QName</i> or <i>documentType</i> must be supplied.</p>
<i>index</i>	<p><b>String</b> Optional. An integer representing the position of the header block entry with the specified <i>QName</i> to remove. A SOAP message can contain multiple header blocks with the same <i>QName</i>. If you specify an <i>index</i>, Integration Server removes that occurrence of the header block. Note that 0 (zero) represents the first header block with the specified <i>QName</i>.</p> <p>If <i>index</i> is not provided, Integration Server removes all the header blocks with a matching <i>QName</i>.</p> <p>For example, if <i>QName</i> is <i>myNSName:myLocalName</i> and <i>index</i> is 1, Integration Server removes the second occurrence of the header block with the <i>QName</i> <i>myNSName:myLocalName</i>.</p>

## Output Parameters

---

*status*      **String** Indicates whether Integration Server removed the header block.

<u>Key</u>	<u>Description</u>
------------	--------------------

True	<p>If <i>index</i> was specified, indicates that Integration Server removed the header block with the specified QName at the specified index.</p> <p>If <i>index</i> was not specified, indicates that Integration Server removed all the header blocks matching the specified QName</p>
False	<p>If <i>index</i> was specified, indicates that Integration Server did not remove the header block at the specified index.</p> <p>For example, if <i>QName</i> is myNSName:myLocalName and <i>index</i> is 1, and the SOAP message does not contain a second header with the QName myNSName:myLocalName, the <i>status</i> is false.</p> <p>If <i>index</i> was not specified, indicates that Integration Server did not remove at least one header block with the specified QName.</p>

### Usage Notes

This service replaces [pub.soap.handler:removeHeaderElement](#), which is deprecated.

Either *QName* or *documentType* must be supplied. If both are provided, Integration Server uses *QName* and ignores *documentType*.

---

## pub.soap.handler:removeHeaderElement

WmPublic. *Deprecated* - Replaced by [pub.soap.handler:removeHeaderBlock](#).

Removes a header element (block) from a SOAP message.

### Input Parameters

---

<i>messageContext</i>	<p><b>Object</b> Message context that contains the SOAP message from which to remove a header element (block).</p> <p>A message context contains properties for the SOAP message as well as providing access to the SOAP message. Integration Server creates the message context and passes it to the header handler. All handlers invoked by a given instance of a SOAP request or SOAP response use the same message context. This enables you to use the message context to pass information between handlers.</p>
-----------------------	---

---

*QName* **Object** Qualified name of the header element (block) to remove. The *QName* document references the [pub.soap.utils:QName](#) document type.

### Output Parameters

---

*status* **String** Indicates whether Integration Server successfully removed the header element. A value of:

- `True` indicates that Integration Server removed the header element.
- `False` indicates that Integration Server did not remove the header element because the SOAP message did not contain a header element that matched the provided *QName*.

### See Also

[pub.soap.handler:addHeaderElement](#)  
[pub.soap.handler:getHeaderElement](#)

---

## pub.soap.handler:removeProperty

WmPublic. Removes a property from a message context.

### Input Parameters

---

*messageContext* **Object** Message context from which to remove a property.

A message context contains properties for the SOAP message and provides access to the SOAP message. Integration Server creates the message context and passes it to the header handler. All handlers invoked by a given instance of a SOAP request or SOAP response use the same message context, which enables you to use the message context to pass information among handlers. For example, in a chain of request header handlers, the first request header handler could set a message property that the second request header handler retrieves.

*key* **String** Name of the property to remove.

### Output Parameters

---

None.

### Usage Notes

The SOAP message contains properties reserved for use by Integration Server. Do not use [pub.soap.handler:setProperty](#) to set any of these properties or use [pub.soap.handler:removeProperty](#) to remove any of these properties as it may result in unpredictable behavior. The reserved properties are:

ContentType	@MESSAGE_USER_FROM_USER_NAME_TOKEN
originalContext	@MESSAGE_USER_FROM_X509_TOKEN
servicePipeline	@TRANSPORT_URL
style	@TRANSPORT_USER
use	

### See Also

[pub.soap.handler:getProperty](#)  
[pub.soap.handler:setProperty](#)

---

## pub.soap.handler:setProperty

WmPublic. Sets the value of a specific property in a message context.

### Input Parameters

---

<i>messageContext</i>	<b>Object</b> Message context in which to set a property.  A message context contains properties for the SOAP message and provides access to the SOAP message. Integration Server creates the message context and passes it to the header handler. All handlers invoked by a given instance of a SOAP request or SOAP response use the same message context, which enables you to use the message context to pass information among handlers. For example, in a chain of request header handlers, the first request header handler could set a message property that the second request header handler retrieves.
<i>key</i>	<b>String</b> Name of the property to set.
<i>value</i>	<b>Object</b> Value to assign to the specified property.

---

## Output Parameters

---

None.

### Usage Notes

The SOAP message contains properties reserved for use by Integration Server. Do not use [pub.soap.handler:setProperty](#) to set any of these properties or use [pub.soap.handler:removeProperty](#) to remove any of these properties as it may result in unpredictable behavior. The reserved properties are:

ContentType            @MESSAGE\_USER\_FROM\_USER\_NAME\_TOKEN

originalContext        @MESSAGE\_USER\_FROM\_X509\_TOKEN

servicePipeline        @TRANSPORT\_URL

style                   @TRANSPORT\_USER

use

### See Also

[pub.soap.handler:getProperty](#)

[pub.soap.handler:removeProperty](#)

---

## pub.soap.handler:setSOAPMessage

WmPublic. Sets the SOAP message in a message context.

### Input Parameters

---

*messageContext*        **Object** Message context in which to set the SOAP message.  
A message context contains properties for the SOAP message and provides access to the SOAP message. Integration Server creates the message context and passes it to the header handler. All handlers invoked by a given instance of a SOAP request or SOAP response use the same message context, which enables you to use the message context to pass information among handlers.

*SOAPMessage*          **Object** of type `javax.xml.soap.SOAPMessage` to use to overwrite the existing SOAP message in the message context.

### Output Parameters

---

None.

### Usage Notes

Use this service with caution, as it overwrites the entire SOAP message, including the SOAP header, body, and fault.

### See Also

[pub.soap.handler:getSOAPMessage](#)

---

## pub.soap.handler:unregisterConsumer

WmPublic. Unregisters a consumer web service descriptor handler.

### Input Parameters

---

<i>descriptiveName</i>	<b>String</b> Name of the consumer web service descriptor handler that you want to unregister.
------------------------	--

### Output Parameters

---

None.

---

## pub.soap.handler:unregisterProvider

WmPublic. Unregisters a provider web service descriptor handler.

### Input Parameters

---

<i>descriptiveName</i>	<b>String</b> Name of the provider web service descriptor handler that you want to unregister.
------------------------	--

### Output Parameters

---

None.

---

## pub.soap.handler:updateFaultBlock

WmPublic. Updates the code, subcodes, reasons, node, and role in an existing fault block.

## Input Parameters

---

*messageContext* **Object** Message context containing the SOAP message that contains the fault block to be updated.

A message context contains properties for the SOAP message and provides access to the SOAP message. Integration Server creates the message context and passes it to the handler. All handlers invoked by a given instance of a SOAP request or SOAP response use the same message context, which enables you to use the message context to pass information among handlers.

If the SOAP Message in *messageContext* does not contain a SOAP fault, the service returns the *status* output parameter as `false`.

*soapFault* **Document** The IData instance to be used to update the values in the fault block.

<u>Key</u>	<u>Description</u>
------------	--------------------

<i>reasons</i>	<b>Document List</b> Optional. Reasons for the SOAP fault. Each Document in the Document List contains a human readable explanation of the cause of the fault.
----------------	--

<u>Key</u>	<u>Descriptions</u>
------------	---------------------

<i>*body</i>	<b>String</b> Text explaining the cause of the fault.
--------------	---

<i>@lang</i>	<b>String</b> Optional. Language for the human readable description.
--------------	--

<i>code</i>	<b>Document</b> Optional. Contains the fault code and possible subcodes.
-------------	--

<u>Key</u>	<u>Description</u>
------------	--------------------

<i>namespaceName</i>	<b>String</b> Namespace name for the SOAP fault code.
----------------------	---

<i>localName</i>	<b>String</b> Code that identifies the fault.
------------------	---

<i>subCodes</i>	<b>Document List</b> Optional. Subcodes that provide further detail.
-----------------	--

	<u>Key</u>	<u>Description</u>
	<i>namespaceName</i>	<b>String</b> Namespace name for the subcode.
	<i>localName</i>	<b>String</b> Code that identifies the subcode.
<i>node</i>		<b>String</b> Optional. URI to the SOAP node where the fault occurred.
<i>role</i>		<b>String</b> Optional. Role in which the node was operating at the point the fault occurred.

### Output Parameters

<i>status</i>	<p><b>String</b> Flag indicating whether the fault block was updated successfully. A value of:</p> <ul style="list-style-type: none"> <li>■ <code>true</code> indicates that the fault block was successfully updated.</li> <li>■ <code>false</code> indicates that either a fault block is not present in the <i>messageContext</i> parameter or that the fault block was not successfully updated.</li> </ul>
---------------	---

### Usage Notes

You can use this service on a provider response handler or fault handler to update the fault block so that the values for code, subcodes, reasons, node, and role that Integration Server generates can be overridden by the custom values you specify.

If you are using this service to override the information in a fault block in a SOAP 1.1 message, keep the following points in mind:

- If multiple reasons are specified, Integration Server uses only the first reason specified because SOAP 1.1 does not support multiple reasons.
- If any subcodes or node values are specified, Integration Server ignores them because subcodes and node elements are not available in the SOAP 1.1 fault structure.

### See Also

[pub.soap.handler:addFaultBlock](#)  
[pub.soap.handler:getFaultBlock](#)

---

## pub.soap.processor:list

WmPublic. *Deprecated* - There is no replacement service.

Returns a list of the SOAP processors that are currently registered on the Integration Server.

### Input Parameters

---

None.

### Output Parameters

---

*list*     **Document List** List of processors currently registered on the server. Each document in the list contains the following information:

<u>Key</u>	<u>Description</u>
<i>directive</i>	<b>String</b> Process directive that is assigned to the SOAP processor.
<i>svcName</i>	<b>String</b> Fully qualified name of the service that functions as the SOAP processor.
<i>descriptiveName</i>	<b>String</b> Descriptive comment that was given to the SOAP processor when it was registered. This element will be empty if the processor was not registered with a descriptive comment.
<i>validateSOAPMessage</i>	<p><b>String</b> Flag indicating whether the SOAP message handler validates the SOAP messages that this processor sends and receives. A value of:</p> <ul style="list-style-type: none"> <li>■ <code>true</code> indicates that messages are validated by the SOAP message handler. Be aware that the validation process checks only that the message envelope is structured correctly. For example, it checks the message has at least one body element and there is at most one header element. It does not validate any of the data carried by the message.</li> </ul> <p>This setting overrides the server's global <code>watt.server.SOAP.validateSOAPMessage</code> setting.</p>

- indicates that messages are not validated by the SOAP message handler.

This setting overrides the server's global `watt.server.SOAP.validateSOAPMessage` setting.

If `validateSOAPMessage` is null, message validation for the processor is determined by the server's `watt.server.SOAP.validateSOAPMessage` setting.

### Notes

This service is deprecated. There is not a replacement service.

### See Also

[pub.soap.processor:registerProcessor](#)  
[pub.soap.processor:unregisterProcessor](#)

---

## pub.soap.processor:processMessage

WmPublic. *Deprecated* - There is no replacement service.

Executes the Integration Server's default SOAP processor.

This service behaves exactly like the built-in default SOAP processor. However, this service can be wrapped in a flow service, which enables you to create an access-controlled SOAP processor.

### Input Parameters

---

*soapRequestData*      **Object** SOAP object containing the SOAP request submitted to the Integration Server by a client.

*soapResponseData*      **Object** Empty SOAP object that the service will use to compose the SOAP response message.

### Output Parameters

---

*soapResponseData*      **Object** SOAP object containing the SOAP response message that is to be returned to the client.

### Usage Notes

This service is deprecated. There is not a replacement service.

---

You invoke `processMessage` from a wrapper service that you create and register as a SOAP processor on the Integration Server. To impose access control on the processor, you assign an access control list (ACL) to the wrapper service.

---

## pub.soap.processor:processRPCMessage

WmPublic. *Deprecated* - There is no replacement service.

Executes the Integration Server's SOAP RPC processor.

This service behaves exactly like the built-in SOAP RPC processor. However, this service can be wrapped in a flow service, which enables you to create an access-controlled SOAP processor.

---

### Input Parameters

*soapRequestData*      **Object** SOAP object containing the SOAP request submitted to the Integration Server by a client.

*soapResponseData*      **Object** Empty SOAP object that the service will use to compose the SOAP response message.

---

### Output Parameters

*soapResponseData*      **Object** SOAP object containing the SOAP response message that is to be returned to the client.

### Usage Notes

This service is deprecated. There is not a replacement service.

You invoke `processRPCMessage` from a wrapper service that you create and register as a SOAP processor on the Integration Server. To impose access control on the processor, you assign an access control list (ACL) to the wrapper service.

---

## pub.soap.processor:registerProcessor

WmPublic. *Deprecated* - There is no replacement service.

Registers a service as a SOAP processor on the Integration Server.

---

## Input Parameters

---

<i>directive</i>	<p><b>String</b> Process directive that you want to assign to the SOAP processor.</p> <p><b>Note:</b> Use only letters, digits, or the characters <code>_.!~*'()</code> in the name you specify in <i>directive</i>.</p>
<i>svcName</i>	<p><b>String</b> Fully qualified name of the service that you are registering as a SOAP processor.</p>
<i>descriptiveName</i>	<p><b>String</b> Descriptive comment for this SOAP processor. This comment is shown when you run the utility service <a href="#">pub.soap.processor:list</a> to get a list of the registered SOAP processors.</p>
<i>validateSOAPMessage</i>	<p><b>String</b> Optional. Flag indicating whether the SOAP message handler validates the SOAP messages that this processor sends and receives. Set to:</p> <ul style="list-style-type: none"><li>■ <code>true</code> to validate messages sent and received by this SOAP processor. Be aware that the validation process checks only that the message envelope is structured correctly. For example, it checks the message has at least one body element and there is at most one header element. It does not validate any of the data carried by the message.  This setting overrides the server's global <code>watt.server.SOAP.validateSOAPMessage</code> setting.</li><li>■ <code>false</code> to bypass validation on messages sent and received by this SOAP processor.  This setting overrides the server's global <code>watt.server.SOAP.validateSOAPMessage</code> setting.</li></ul> <p>Or, leave <i>validateSOAPMessage</i> null to validate messages according to the Integration Server's <code>watt.server.SOAP.validateSOAPMessage</code> setting. This is the default.</p>

---

## Output Parameters

---

None.

## Usage Notes

This service is deprecated. There is not a replacement service.

---

**See Also**[pub.soap.processor:list](#)[pub.soap.processor:unregisterProcessor](#)

---

## pub.soap.processor:unregisterProcessor

WmPublic. *Deprecated* - There is no replacement service.

Unregisters a SOAP processor by removing it from the registry.

---

**Input Parameters**

*directive*

**String** Process directive that you want to remove from the registry. Directive names are case sensitive.

**Tip:** To obtain a list of the current SOAP processor directives registered on the server, run the [pub.soap.processor:list](#) service.

---

**Output Parameters**

None.

**Usage Notes**

This service is deprecated. There is not a replacement service.

If the directive specified in *directive* is not registered on the Integration Server, `unregisterProcessor` throws an exception.

**See Also**[pub.soap.processor:list](#)[pub.soap.processor:registerProcessor](#)

---

## pub.soap.schema:encoding

WmPublic. Schema that defines the data types SOAP supports.

---

## pub.soap.schema:encoding\_1\_2

WmPublic. Schema that defines the data types SOAP 1.2 supports.

---

## pub.soap.schema:envelope

WmPublic. Schema that defines the structure of a SOAP message.

---

## pub.soap.schema:envelope\_1\_2

WmPublic. Schema that defines the structure of a SOAP 1.2 message.

---

## pub.soap.utils:addBodyEntry

WmPublic. Inserts an entry into the body element of a SOAP message.

---

### Input Parameters

*soapData*

**Object** SOAP object to which you want the body entry added.

*bodyEntry*

**com.wm.lang.xml.Node** XML node containing the body entry that you want to add to *soapData*.

**Note:** An XML node is a parsable representation of a node in an XML document. You generate an XML node using services such as [pub.xml:xmlStringToXMLNode](#).

**Important:** This service adds a single body entry to a SOAP object. If you need to add more than one entry, execute [pub.soap.utils:addBodyEntry](#) once for each entry.

**Important:** In webMethods Integration Server versions 6.0.1 and later, this service expects the node in *bodyEntry* to be namespace qualified. If the node is not qualified, the service throws an exception. If you created solutions based on the earlier behavior of this service (which permitted non-qualified entries), you can disable namespace enforcement by setting the server's `watt.server.SOAP.EnforceMsgPartNS` parameter to `false`.

---

### Output Parameters

*soapData*

**Object** SOAP object to which the body entry was added.

## Usage Notes

A SOAP object is an object that represents a SOAP message.

If you are composing a new SOAP message, you must first create an empty SOAP object (called *soapData*) with the `createSoapData` service and then add your header entries to with `pub.soap.utils:addHeaderEntry`.

If you are composing a SOAP response, you use `pub.soap.utils:addBodyEntry` to populate the *soapResponseData* object that the SOAP message handler generates and puts in the pipeline.

## See Also

`pub.soap.utils:createSoapData`  
`pub.soap.utils:addBodyEntry`  
`pub.soap.utils:addHeaderEntry`  
`pub.soap.utils:addTrailer`  
`pub.soap.utils:getBody`

## Examples

See the following in the WmSamples packages in the certified samples area of the Knowledge Center on the Empower Product Support website at <https://empower.softwareag.com>:

`sample.soap:buildMsg_sendHTTP`

`sample.soap:customProc_msgQuwue`

`sample.soap:targetSvc_defaultProc`

---

## pub.soap.utils:addHeaderEntry

WmPublic. Inserts an entry into the header element of a SOAP message.

### Input Parameters

---

*soapData*                      **Object** SOAP object to which you want the header entry added.

*headerEntry*                      **com.wm.lang.xml.Node** XML node containing the entry that you want to add to *soapData*.

**Note:** An XML node is a parsable representation of a node in an XML document. You generate an XML node using services such as `pub.xml:xmlStringToXMLNode`.

**Important:** This service adds a single header entry to a SOAP object. If you need to add more than one entry, execute `addHeaderEntry` once for each entry.

**Important:** In webMethods Integration Server versions 6.0.1 and later, this service expects the node in `headerEntry` to be namespace qualified. If the node is not qualified, the service throws an exception. If you created solutions based on the earlier behavior of this service (which permitted non-qualified entries), you can disable namespace enforcement by setting the server's `watt.server.SOAP.EnforceMsgPartNS` parameter to `false`.

*mustUnderstand*

**String** Optional. Value to which you want the `mustUnderstand` attribute set.

The `mustUnderstand` attribute specifies whether recipients are required to process a header entry (that is, whether processing of the entry is mandatory or optional). Recipients that cannot process a mandatory header entry must reject the message and return a SOAP fault.

A value of:

- 0 indicates that the header is optional.
- 1 indicates that the header is mandatory.

For additional information about the `mustUnderstand` attribute, see the *Simple Object Access Protocol (SOAP) 1.1 - W3C Note 08 May 2000* at <http://www.w3.org/TR/SOAP/>.

**Note:** If you do not set `mustUnderstand`, the `mustUnderstand` attribute is omitted from the header entry, which is equivalent to setting `mustUnderstand` to 0.

*actor*

**String** Optional. Value to which you want the `actor` attribute set.

The `actor` attribute specifies a URI that identifies the recipient to which a header entry is targeted. For additional information about the `mustUnderstand` attribute, see the *Simple Object Access Protocol (SOAP) 1.1 - W3C Note 08 May 2000* at <http://www.w3.org/TR/SOAP/>.

## Output Parameters

---

*soapData*

**Object** SOAP object to which the header entry was added.

## Usage Notes

A SOAP object is an object that represents a SOAP message.

If you are composing a new SOAP message, you must first create an empty SOAP object (called *soapData*) with the `createSoapData` service and then add your header entries to with `pub.soap.utils:addHeaderEntry`.

If you are composing a SOAP response, you use `addHeaderEntry` to populate the *soapResponseData* object that the SOAP message handler generates and puts in the pipeline.

## See Also

`pub.soap.utils:createSoapData`  
`pub.soap.utils:addBodyEntry`  
`pub.soap.utils:addTrailer`  
`pub.soap.utils:getHeader`  
`pub.soap.utils:getHeaderEntries`

## Examples

See the following in the WmSamples packages in the certified samples area of the Knowledge Center on the Empower Product Support website at <https://empower.softwareag.com>:

`sample.soap:buildMsg_sendHTTP`

`sample.soap:targetSvc_defaultProc`

---

## pub.soap.utils:addTrailer

WmPublic. Inserts a trailer in a SOAP message.

(A trailer is an arbitrary element that follows the Body element in the SOAP envelope.)

**Important:** It appears likely that trailers will not be permitted in future versions of SOAP (versions 1.2 and later). If you are designing a completely new solution, we recommend that you avoid using trailers. However, if you exchange SOAP messages with older systems that already make use of trailers, this service allows you to insert them into a SOAP message.

---

## Input Parameters

<i>soapData</i>	<b>Object</b> SOAP object to which you want the trailer added.
<i>trailer</i>	<b>com.wm.lang.xml.Node</b> XML node containing the trailer that you want to add to <i>soapData</i> .

**Note:** An XML node is a parsable representation of a node in an XML document. You generate an XML node using services such as [pub.xml:xmlStringToXMLNode](#).

**Important:** This service adds a single trailer to a SOAP object. If you need to insert more than one trailer in the message, execute `addTrailer` once for each trailer that needs to be added.

**Note:** The SOAP specification states that trailers must be namespace qualified, so be sure that the node in *trailer* specifies a namespace.

---

## Output Parameters

*soapData*      **Object** SOAP object to which the trailer was added.

## Usage Notes

A SOAP object is an object that represents a SOAP message.

- If you are composing a new SOAP message, you must first create an empty SOAP object (called *soapData*) with the `createSoapData` service and then add your header entries to with [pub.soap.utils:addHeaderEntry](#).
- If you are composing a SOAP response, you use [pub.soap.utils:addHeaderEntry](#) to populate the *soapResponseData* object that the SOAP message handler generates and puts in the pipeline.

## See Also

[pub.soap.utils:createSoapData](#)  
[pub.soap.utils:addHeaderEntry](#)  
[pub.soap.utils:addBodyEntry](#)  
[pub.soap.utils:getTrailers](#)

---

# pub.soap.utils:callbackServiceSpec

WmPublic. Defines the input signature for an outbound callback service.

---

## Input Parameters

*messageContext*      **Object** Message context to be processed by the outbound callback service.

A message context contains properties for the SOAP message and provides access to the SOAP message. Integration Server creates the message context and passes it to the outbound callback service.

### Output Parameters

---

None.

## pub.soap.utils:convertToVersionSpecificSOAPFault

WmPublic. Converts the generic SOAP fault structure to the SOAP version-specific fault structure that is used by web service descriptors created in versions of Integration Server prior to 8.2.

### Input Parameters

---

*soapProtocol*     **String** Optional. SOAP protocol with which the SOAP fault to convert works. Valid values are SOAP 1.1 Protocol or SOAP 1.2 Protocol. The default is SOAP 1.2 Protocol.

*fault*             **Document** The generic SOAP fault that is to be converted. The *fault* document references the [pub.soap.utils:soapFault](#) document type.

### Output Parameters

---

*SOAP-FAULT*     **Document** Document containing the converted SOAP fault.

<u>Key</u>	<u>Description</u>
<i>soapProtocol</i>	<b>String</b> Indicates the SOAP protocol to which the SOAP object works. This is the same as the <i>soapProtocol</i> input parameter. Valid values are SOAP 1.1 Protocol or SOAP 1.2 Protocol.
<i>Fault_1_1</i>	<b>Document</b> Conditional. Converted fault information. <i>Fault_1_1</i> and its child variables are populated only when <i>soapProtocol</i> is SOAP 1.1 Protocol.

<u>Key</u>	<u>Description</u>
------------	--------------------

---

<i>faultcode</i>	<p><b>String</b> Conditional. A code that identifies the fault. This field corresponds to the SOAP 1.1 <code>faultcode</code> element.</p> <p>This field is set based on the value of the <code>code/namespaceName</code> field of the <code>fault</code> input parameter.</p> <p>When <code>namespaceName</code> contains:</p> <ul style="list-style-type: none"><li>■ <b>http://schemas.xmlsoap.org/soap/envelope/</b>, which is the standard namespace name for a SOAP 1.1 Envelope, <code>faultcode</code> is set to the following: <code>SOAP-ENV:localName</code></li><li>■ <b>Any other value</b>, <code>faultcode</code> is set to the following: <code>{namespaceName}localName</code></li></ul> <p>In the above, <code>localName</code> is the value of the <code>code/localName</code> field and <code>namespaceName</code> is value of the <code>code/namespaceName</code> field of the <code>fault</code> input parameter</p>
<i>faultstring</i>	<p><b>String</b> Conditional. A human readable explanation of the fault. This field corresponds to the SOAP 1.1 <code>faultstring</code> element.</p> <p>The service maps the <code>*body</code> value from the first <code>reasons</code> document of the <code>fault</code> input parameter.</p>
<i>faultactor</i>	<p><b>String</b> Conditional. Information about the cause of the fault. This field corresponds to the SOAP 1.1 <code>faultactor</code> element.</p> <p>The service maps the value from the <code>role</code> field of the <code>fault</code> input parameter.</p>
<i>detail</i>	<p><b>Document</b> Conditional. Application-specific details about the SOAP fault. This field corresponds to the SOAP 1.1 <code>detail</code> element.</p> <p>The service maps the value from the <code>detail</code> field of the <code>fault</code> input parameter.</p>

*Fault\_1\_2* **Document** Conditional. Converted fault information. Fault\_1\_2 and its child variables are populated only when *soapProtocol* is SOAP 1.2 Protocol.

Key	Description
SOAP-ENV:Code	<p><b>Document</b> Conditional. Contains the fault code.</p>
SOAP-ENV:Value	<p><b>String</b> A code that identifies the fault. This corresponds to the SOAP 1.2 <i>Code</i> element.</p> <p>This field is set based on the value of the <i>code/namespaceName</i> field of the <i>fault</i> input parameter. When <i>namespaceName</i> contains:</p> <ul style="list-style-type: none"> <li>■ <b>http://www.w3.org/2003/05/soap-envelope</b>, which is, the standard namespace name for a SOAP 1.2 Envelope, <i>faultcode</i> is set to the following: SOAP-ENV:<i>localName</i></li> <li>■ <b>Any other value</b>, <i>faultcode</i> is set to the following: {<i>namespaceName</i>}<i>localName</i></li> </ul> <p>In the above, <i>localName</i> is the value of the <i>code/localName</i> field and <i>namespaceName</i> is value of the <i>code/namespaceName</i> field of the <i>fault</i> input parameter</p>
SOAP-ENV:Reason	<p><b>Document</b> Conditional. Document containing the reason for the SOAP fault. This corresponds to the SOAP 1.2 <i>Reason</i> element.</p>
SOAP-Env:Text	<p><b>Document</b> Conditional. Document containing the human readable explanation of the cause of the fault. The service maps the first</p>

document of the *reasons* field of the *fault* input parameter.

*@XML:lang* **String** Conditional. Specifies the language for the human readable description. This field corresponds to the `xml:lang` attribute of the `Text` child element of the SOAP1.2 `Reason` element.

*\*body* **String** Conditional. Text explaining the cause of the fault. This field corresponds to the `Text` child element of the SOAP 1.2 `Reason` element.

*SOAP-ENV:Node* **String** Conditional. URI to the SOAP node where the fault occurred. This field corresponds to the SOAP 1.2 `Node` element.

The service maps the value from the *node* field of the *fault* input parameter.

*SOAP-ENV:Role* **String** Conditional. Role in which the node was operating at the point the fault occurred. This field corresponds to the SOAP 1.2 `Role` element.

The service maps the value from the *role* field of the *fault* input parameter.

*SOAP-ENV:Detail* **Document** Conditional. Application- specific details about the SOAP fault. This field corresponds to the SOAP 1.2 `Detail` element.

The service maps the value from the *detail* field of the *fault* input parameter.

### Usage Notes

The following are instances where Integration Server generates a generic SOAP fault structure that you might want to convert to a SOAP version-specific fault structure:

- The *fault* output parameter from a web service connector generated from a web service descriptor created in Integration Server 8.2.
- The *soapFault* output parameter from the [pub.soap.handler:getFaultBlock](#) service.

The following lists instances where the service might not be able to convert all data in the generic SOAP fault structure to a corresponding field in the output.

- The data in the *code/subcodes* field of the *fault* input parameter does not map to any output parameter. As a result, the service does not convert the data for either the SOAP 1.1 or SOAP 1.2 protocol.
- The service might not be able to map all the data in the *reasons* field of the *fault* input parameter to the corresponding output parameters. The *reasons* field is a Document List that can represent the reason in multiple languages. However, the output can represent only a single value.
  - For the SOAP 1.1 protocol, the service maps the *\*body* from the first Document of the *fault/reasons* input parameter to the *Fault\_1\_1/faultstring* output parameter.
  - For the SOAP 1.2 protocol, the service maps only the first Document of the *fault/reasons* to the output parameter *Fault\_1\_2/SOAP-ENV:Reason/SOAP-Env:Text*.
- The data in the *node* field of the *fault* input parameter does not map to any element of the *Fault\_1\_1* output parameter. As a result, the service does not convert the value for the SOAP 1.1 protocol.

### See Also

[pub.soap.handler:getFaultBlock](#)

---

## pub.soap.utils:createSoapData

WmPublic. Creates a SOAP object consisting of SOAP envelope, body, and header entries.

### Input Parameters

---

*encoding* **String** Optional. Specifies the encoding method. Default value is UTF-8.

*addEmptyHeader*

**String** Optional. Specifies whether to create an empty header entry along with the SOAP envelope and body entries in the SOAP message. Set to:

- `True` to create an empty header entry in the SOAP message.
- `False` to create only the SOAP envelope and body entries.

This setting overrides the global `watt.server.SOAP.addEmptyHeader` setting of Integration Server.

**Important** There is no default value. If you do not specify a value for the *addEmptyHeader* parameter, the service adds an empty header entry, which is equivalent to setting *addEmptyHeader* to `True`.

*soapProtocol*

**String** Optional. Indicates the SOAP protocol that the SOAP object works with. The default value is read from the `watt.server.SOAP.defaultProtocol` property. Set to:

- `SOAP 1.1 protocol` to indicate the SOAP object works with SOAP 1.1.
- `SOAP 1.2 protocol` to indicate the SOAP object works with SOAP 1.2.

## Output Parameters

---

*soapData*

**Object** SOAP object.

## Usage Notes

The *encoding* parameter can support incoming SOAP messages in any encoding. Outgoing messages, however, are always encoded in UTF-8.

## See Also

[pub.soap.utils:addHeaderEntry](#)  
[pub.soap.utils:addBodyEntry](#)  
[pub.soap.utils:addTrailer](#)

## Examples

See the following in the `WmSamples` package in the certified samples area of the Knowledge Center on the Empower Product Support website at <https://empower.softwareag.com>:

`sample.soap:buildMsg_sendHTTP`

---

## pub.soap.utils:createXOPObject

WmPublic. Generates a com.wm.util.XOPObject instance from a base64Binary string, a byte array, or an input stream.

### Input Parameters

---

*contentType*                      **String** Optional. MIME type of the input data.

*data*                                **Document** Data from which you want to generate a com.wm.util.XOPObject instance.

Key	Description
<i>base64String</i>	<b>String</b> Optional. The base64-encoded string from which you want to generate the com.wm.util.XOPObject instance.
<i>bytes</i>	<b>byte [ ]</b> Optional. The byte array from which you want to generate the com.wm.util.XOPObject instance.
<i>stream</i>	<b>Object</b> Optional. The InputStream from which you want to generate the com.wm.util.XOPObject instance.

### Output Parameters

---

*xopObject*                      **Object** Conditional. An instance of com.wm.util.XOPObject generated from the input data.  
The value of *xopObject* will be null if you do not specify any value for the *data* input parameter.

### Usage Notes

You use the object of type com.wm.util.XOPObject to send or receive data as an MTOM stream. For more information about MTOM streaming, see *Web Services Developer's Guide*.

If you specify values for more than one key of the *data* input parameter, Integration Server uses only one value in the following order or precedence:

- *base64String*

- *bytes*
- *stream*

For example, if you provide values for *base64String*, *bytes*, and *stream* keys, Integration Server will execute the `pub.soap.utils:createXOPObject` service with the *base64String* value and will ignore the values provided for *bytes* and *stream* keys.

---

## pub.soap.utils:exitUnableToUnderstand

WmPublic. Terminates processing and returns a `mustUnderstand` fault to the client.

You execute this service when your SOAP processor detects a mandatory header entry that it cannot process.

### Input Parameters

---

*headerEntry*                      **com.wm.lang.xml.Node** XML node containing the header entry that cannot be understood.

### Output Parameters

---

None.

### Usage Notes

This service throws an exception, which is meant to be caught by the message handler so that the appropriate SOAP fault will be returned to the client. Your processor should not catch this exception.

---

## pub.soap.utils:getActor

WmPublic. Retrieves the value of the `actor` attribute (for SOAP 1.1) or the `role` attribute (for SOAP 1.2) from a given header entry.

### Input Parameters

---

*headerEntry*                      **com.wm.lang.xml.Node** The header entry whose `actor` value you want to retrieve.

- If you use [pub.soap.utils:getHeaderEntries](#) to retrieve header entries, you can loop over the list of header nodes to retrieve the `actor` value from each entry.
- If you use [pub.soap.utils:getHeader](#) to retrieve header entries, you must query the node returned by that service (using the [pub.xml:queryXMLNode](#) service) to extract a node for an

individual header entry. Then you can run `getActor` on the resulting node.

---

## Output Parameters

---

*actor* **String** Value of the header entry's `actor` attribute (for SOAP 1.1) or the `role` attribute (for SOAP 1.2). If the header entry does not have an `actor` attribute, *actor* will be null.

### See Also

[pub.soap.utils:addHeaderEntry](#)  
[pub.soap.utils:getMustUnderstand](#)  
[pub.soap.utils:getHeader](#)  
[pub.soap.utils:getHeaderEntries](#)

---

## pub.soap.utils:getBody

WmPublic. Retrieves the body from a SOAP message as a single node object.

---

### Input Parameters

---

*soapData* **Object** SOAP object containing the message whose Body node you want to retrieve.

---

### Output Parameters

---

*body* **com.wm.lang.xml.Node** The Body node from the SOAP message (that is, `<SOAP-ENV:Body>` to `</SOAP-ENV:Body>`).

### Usage Notes

This service returns the entire Body element in *body*. To extract data from the Body element, query *body* with the [pub.xml:queryXMLNode](#) service.

If you want to extract the body of the message as an array of nodes, use the [pub.soap.utils:getBodyEntries](#) service.

### See Also

[pub.soap.utils:getBodyEntries](#)  
[pub.soap.utils:addBodyEntry](#)

## Examples

See the following in the WmSamples packages in the certified samples area of the Knowledge Center on the Empower Product Support website at <https://empower.softwareag.com>:

sample.soap:buildMsg\_sendHTTP

sample.soap:customProc\_msgQueue

sample.soap:targetSvc\_defaultProc

---

## pub.soap.utils:getBodyEntries

WmPublic. Retrieves the body entries from a SOAP message as an array of node objects.

### Input Parameters

---

*soapData*                      **Object** The SOAP object containing the message whose body entries you want to retrieve.

### Output Parameters

---

*bodyEntries*                      **com.wm.lang.xml.Node[ ]** An array of XML nodes, where each node represents a body entry from the message.

### Usage Notes

This service returns each body entry as a separate node. You can loop over *bodyEntries* and extract data from each node with the [pub.xml:queryXMLNode](#) service.

If you want to extract the body of the message as a single node, use the [pub.soap.utils:getBody](#) service.

### See Also

[pub.soap.utils:getBody](#)  
[pub.soap.utils:addBodyEntry](#)

---

## pub.soap.utils:getDocument

WmPublic. Retrieves an entire SOAP message as a node object.

This service is useful when you want to use [pub.xml:queryXMLNode](#) to query an entire SOAP message. Since [queryXMLNode](#) requires a node as input, you cannot use it to query

---

a SOAP object directly. Instead, you must convert the SOAP object to a node and then query the resulting node.

### Input Parameters

---

*soapData*                      **Object** SOAP object for which you want a node representation.

### Output Parameters

---

*node*                              **com.wm.lang.xml.Node** Node representation of the entire SOAP message in *soapData* (that is, <SOAP-ENV:Envelope> to </SOAP-ENV:Envelope>).

### See Also

[pub.soap.utils:getBody](#)  
[pub.soap.utils:getBodyEntries](#)  
[pub.soap.utils:getHeader](#)  
[pub.soap.utils:getHeaderEntries](#)  
[pub.soap.utils:getTrailers](#)

---

## pub.soap.utils:getEncoding

WmPublic. Retrieves the encoding from a SOAP message as a single string.

### Input Parameters

---

*soapData*                      **Object** SOAP object containing the message whose encoding you want to retrieve.

### Output Parameters

---

*encoding*                        **com.wm.lang.xml.Node** Encoding from the SOAP message.

### See Also

[pub.soap.utils:getHeaderEntries](#)  
[pub.soap.utils:getBody](#)  
[pub.soap.utils:getBodyEntries](#)  
["pub.soap.utils:getDocument" on page 832](#)  
[pub.soap.utils:getTrailers](#)  
[pub.soap.utils:addHeaderEntry](#)

---

## pub.soap.utils:getHeader

WmPublic. Retrieves the header from a SOAP message as a single node object.

### Input Parameters

---

*soapData*                      **Object** SOAP object containing the message whose Header node you want to retrieve.

### Output Parameters

---

*header*                              **com.wm.lang.xml.Node** Header node from the SOAP message (that is, <SOAP-ENV:Header> to </SOAP-ENV:Header>).

### Usage Notes

This service returns the entire Header element in *header*. To extract data from the Header element, query *header* with the [pub.xml:queryXMLNode](#) service. If you want to extract the contents of the header as an array of nodes, use the [pub.soap.utils:getHeaderEntries](#) service.

### See Also

- [pub.soap.utils:getHeaderEntries](#)
- [pub.soap.utils:getBody](#)
- [pub.soap.utils:getBodyEntries](#)
- [pub.soap.utils:getTrailers](#)
- [pub.soap.utils:addHeaderEntry](#)

### Examples

See the following in the WmSamples package in the certified samples area of the Knowledge Center on the Empower Product Support website at <https://empower.softwareag.com>:

```
sample.soap:customProc_msgQueue
```

---

## pub.soap.utils:getHeaderEntries

WmPublic. Retrieves the header entries from a SOAP message as an array of node objects.

This service is useful when you want to build a process that loops through all the header entries in a message and identify entries with specific QNames (using the [pub.soap.utils:getQName](#) service) or actor attributes (using the [pub.soap.utils:getActor](#) service).

---

## Input Parameters

---

*soapData*                      **Object** SOAP object containing the message whose header entries you want to retrieve.

## Output Parameters

---

*headerEntries*                **com.wm.lang.xml.Node[ ]** Header entries from the SOAP message. Each node in the array represents a header entry from the message.

## Usage Notes

This service returns each header entry as a separate node. You can loop over *headerEntries* and extract data from each node with the [pub.xml:queryXMLNode](#) service or get the entry's QName and/or actor value using the [pub.soap.utils:getQName](#) and [pub.soap.utils:getActor](#) services.

If you want to extract the message header as a single node, use the [pub.soap.utils:getHeader](#) service.

## See Also

[pub.soap.utils:getHeader](#)  
[pub.soap.utils:getBody](#)  
[pub.soap.utils:getBodyEntries](#)  
[pub.soap.utils:getDocument](#)  
[pub.soap.utils:addHeaderEntry](#)  
[pub.soap.utils:getActor](#)  
[pub.soap.utils:getQName](#)

---

## pub.soap.utils:getMustUnderstand

WmPublic. Returns the mustUnderstand status for a given header entry.

The mustUnderstand status specifies whether recipients are *required* to process a header entry (that is, whether processing of the entry is mandatory or optional). Recipients that cannot process a mandatory header entry must reject the message and return a SOAP fault. (See the [pub.soap.utils:exitUnableToUnderstand](#) service.)

---

## Input Parameters

---

*headerEntry*                    **com.wm.lang.xml.Node** The header entry whose mustUnderstand status you want to retrieve.

- If you use [pub.soap.utils:getHeaderEntries](#) to retrieve header entries, you can loop over the list of header nodes to check the status of each entry.
- If you use [pub.soap.utils:getHeader](#) to retrieve header entries, you will need to query the node returned by that service (using the [pub.xml:queryXMLNode](#) service) to extract a node for an individual header entry. Then you can run [pub.soap.utils:getMustUnderstand](#) on the resulting node.

---

## Output Parameters

---

*mustUnderstand*

**String** Header entry's `mustUnderstand` status. If the header entry has a `mustUnderstand` attribute, *mustUnderstand* will return one of the following values:

- 0 indicates that the header is optional.
- 1 indicates that the header is mandatory.

If the header entry does not have a *mustUnderstand* attribute, `mustUnderstand` will return 0.

## Usage Notes

For additional information about the `mustUnderstand` attribute, see the *Simple Object Access Protocol (SOAP) 1.1 - W3C Note 08 May 2000* at <http://www.w3.org/TR/SOAP/> and for SOAP 1.2, see the *SOAP 1.2 W3C Recommendation 27 April 2007* at <http://www.w3.org/TR/soap12-part1/>.

## See Also

[pub.soap.utils:addHeaderEntry](#)  
[pub.soap.utils:getActor](#)  
[pub.soap.utils:getHeader](#)  
[pub.soap.utils:getHeaderEntries](#)  
[pub.soap.utils:exitUnableToUnderstand](#)

---

## pub.soap.utils:getQName

WmPublic. Returns the qualified name for a given node.

---

## Input Parameters

---

*node*

**com.wm.app.b2b.server.saaj.SOAPElement** The XML node whose qualified name you want to discover.

---

## Output Parameters

---

*Qname* **Document** The node's qualified name. *Qname* will contain the following keys:

<u>Key</u>	<u>Description</u>
<i>namespaceName</i>	<b>String</b> Namespace portion of the node's qualified name.
<i>localName</i>	<b>String</b> Local portion of the node's qualified name.

### Usage Notes

Generally, you use this service in conjunction with the [pub.soap.utils:getHeaderEntries](#) or [pub.soap.utils:getBodyEntries](#) service to loop over the message's header or body entries and identify entries with a particular qualified name.

### See Also

[pub.soap.utils:getBodyEntries](#)  
[pub.soap.utils:getHeaderEntries](#)

---

## pub.soap.utils:getTrailers

WmPublic. Retrieves the trailers from a SOAP message.

(A trailer is an arbitrary element that follows the Body element in the SOAP envelope.)

**Important:** It appears likely that trailers will not be permitted in future versions of SOAP (versions 1.2 and later). If you are designing a completely new solution, we recommend that you avoid using trailers. However, if you exchange SOAP messages with older systems that already make use of trailers, this service allows you to retrieve them from a SOAP message.

---

## Input Parameters

---

*soapData* **Object** SOAP object containing the message whose trailers you want to retrieve.

---

## Output Parameters

---

*trailers* **com.wm.lang.xml.Node[ ]** Array of nodes wherein each node represents a trailer from the message. If the message does not contain trailers, *trailers* will be null.

### See Also

[pub.soap.utils:addTrailer](#)  
[pub.soap.utils:getHeader](#)  
[pub.soap.utils:getHeaderEntries](#)  
[pub.soap.utils:getBody](#)  
[pub.soap.utils:getBodyEntries](#)  
[pub.soap.utils:getDocument](#)

---

## pub.soap.utils:getXOPObjectContent

WmPublic. Retrieves the contents of a com.wm.util.XOPObject instance as a base64Binary string, a byte array, or an InputStream.

### Input Parameters

---

*xopObject* **Object** Optional. The object of type com.wm.util.XOPObject.

*getAs* **String** Optional. The object type in which you want to retrieve the contents in the com.wm.util.XOPObject instance.

<u>Select...</u>	<u>To...</u>
base64String	Default. Retrieve the contents of the com.wm.util.XOPObject instance as a base64-encoded string.
bytes	Retrieve the contents of the com.wm.util.XOPObject instance as a byte array.
stream	Retrieve the contents of the com.wm.util.XOPObject instance as an InputStream.

## Output Parameters

---

*contentType* **String** Conditional. MIME type of the contents in the input `com.wm.util.XOPObject` instance. The *contentType* parameter is returned only if you have specified a value for the *xopObject* input parameter.

*data* **Document** Contents of the input `com.wm.util.XOPObject` instance.  
If the *xopObject* input parameter is null, this parameter has a null value.

Value	Description
<i>base64String</i>	<b>String</b> Conditional. Returns the contents of the <code>com.wm.util.XOPObject</code> instance as a base64-encoded string if the <i>getAs</i> input parameter is set to <code>base64String</code> .
<i>bytes</i>	<b>byte []</b> Conditional. Returns the contents of the <code>com.wm.util.XOPObject</code> instance as a byte array if the <i>getAs</i> input parameter is set to <code>bytes</code> .
<i>stream</i>	<b>Object</b> Conditional. Returns the contents of the <code>com.wm.util.XOPObject</code> instance as an <code>InputStream</code> if the <i>getAs</i> input parameter is set to <code>stream</code> .

## Usage Notes

You use the object of type `com.wm.util.XOPObject` to send or receive data as an MTOM stream. For more information about MTOM streaming, see *Web Services Developer's Guide*.

The content of the `XOPObject` can only be read one time. After you use the `pub.soap.utils:getXOPObjectContent` service to read the `XOPObject` content one time, subsequent attempts to re-read the `XOPObject` content will fail. It is recommended that after the `XOPObject` content is read that you drop it from the pipeline to make it clear to programming logic downstream that it is no longer available to be read.

When you set the *getAs* input parameter to `stream` so that the service returns the contents of the `com.wm.util.XOPObject` instance as a stream, the `pub.soap.utils:getXOPObjectContent` service does not automatically close the stream object. You can close the stream using the `pub.io:close` service.

---

## pub.soap.utils:QName

WmPublic. Document type that defines the structure of a qualified name.

### Parameters

---

*namespaceName*      **String** The namespace portion of a qualified name.

*localName*            **String** The local portion of a qualified name.

---

## pub.soap.utils:removeBodyEntry

WmPublic. Deletes a body entry from a SOAP message.

### Input Parameters

---

*soapData*            **Object** SOAP object containing the body entry that you want to delete.

*bodyEntry*           **com.wm.lang.xml.Node** Optional. The entry that you want to remove from *soapData*. (You would obtain the node with the [pub.soap.utils:getBodyEntries](#) service.)

**Note:** You can use *bodyEntry* or *index* to specify the entry that you want `removeBodyEntry` to delete.

*index*                **String** Optional. Index of the entry that you want to remove (where index 0 represents the first body entry). *index* is ignored if *bodyEntry* is specified.

### Output Parameters

---

None.

### Usage Notes

When you use the *bodyEntry* parameter, be sure that it specifies the correct node. This service deletes whatever node is specified in *bodyEntry*, even if the node is not a body entry. For example, if *bodyEntry* contains the whole Body element, `removeBodyEntry` will delete the body of the message.

Be aware that if you use the *index* parameter to delete an entry, you will change the index numbers (positions) of all entries following the one you deleted. For example,

if your message contains four body entries (0, 1, 2, 3) and you delete entry 1, then the entries originally at positions 2 and 3 will subsequently occupy positions 1 and 2.

### See Also

[pub.soap.utils:removeHeaderEntry](#)  
[pub.soap.utils:removeTrailer](#)  
[pub.soap.utils:addBodyEntry](#)  
[pub.soap.utils:getBody](#)  
[pub.soap.utils:getBodyEntries](#)

---

## pub.soap.utils:removeHeaderEntry

WmPublic. Deletes a header entry from a SOAP message.

### Input Parameters

---

<i>soapData</i>	<b>Object</b> SOAP object containing the header entry that you want to delete.
<i>headerEntry</i>	<b>com.wm.lang.xml.Node</b> Optional. The header entry that you want to remove from <i>soapData</i> . (You would obtain the node with the <a href="#">pub.soap.utils:getHeaderEntries</a> service.)  <b>Note:</b> You can use <i>headerEntry</i> or <i>index</i> to specify the entry that you want <code>removeHeaderEntry</code> to delete.
<i>index</i>	<b>String</b> Optional. Index of the entry that you want to remove (where index 0 represents the first header entry). <i>index</i> is ignored if <i>headerEntry</i> is specified.

### Output Parameters

---

None.

### Usage Notes

When you use the *headerEntry* parameter, be sure that it specifies the correct node. This service deletes whatever node is specified in *headerEntry*, even if the node is not a header entry. For example, if *headerEntry* contains the whole Header element, `removeHeaderEntry` will delete the entire header from the message.

**Note:** Be aware that if you use the *index* parameter to delete an entry, you will change the index numbers (positions) of all entries following the one you deleted. For example, if your header contains four entries (0, 1, 2, 3) and

you delete entry 1, then the entries originally at positions 2 and 3 will subsequently occupy positions 1 and 2.

### See Also

[pub.soap.utils:removeBodyEntry](#)  
[pub.soap.utils:removeTrailer](#)  
[pub.soap.utils:addHeaderEntry](#)  
[pub.soap.utils:getHeader](#)  
[pub.soap.utils:getHeaderEntries](#)

---

## pub.soap.utils:removeTrailer

WmPublic. Deletes a trailer from a SOAP message.

### Input Parameters

---

*soapData*

**Object** SOAP object containing the trailer that you want to delete.

*trailer*

**com.wm.lang.xml.Node** Optional. The trailer that you want to remove from *soapData*. (You would obtain the node with the [pub.soap.utils:getTrailers](#) service.)

**Note:** You can use *trailer* or *index* to specify the trailer that you want `removeTrailer` to delete.

*index*

**String**Optional. Index of the trailer that you want to remove (where index 0 represents the first trailer). *index* is ignored if *trailer* is specified.

### Output Parameters

---

None.

### Usage Notes

When you use the *trailer* parameter, be sure that it specifies the correct node. This service deletes whatever node is specified in *trailer*, even if the node is not a trailer. For example, if *trailer* contains the Body element, `removeTrailer` will delete the entire body of the message.

**Note:** Be aware that if you use the index parameter to delete a trailer, you will change the index numbers (positions) of all trailers following the one you deleted. For example, if your message contains four trailers (0, 1, 2, 3) and

you delete trailer 1, then the trailers originally at positions 2 and 3 will subsequently occupy positions 1 and 2.

### See Also

[pub.soap.utils:removeHeaderEntry](#)  
[pub.soap.utils:removeBodyEntry](#)  
[pub.soap.utils:addTrailer](#)  
[pub.soap.utils:getTrailers](#)

---

## pub.soap.utils:requestResponseSpec

WmPublic. Defines the input/output signature for a custom processor and a target service for the default processor.

### Input Parameters

---

*soapRequestData*      **Object** SOAP object containing the SOAP request submitted to the Integration Server by the client.

*soapResponseData*      **Object** Empty SOAP object that the custom processor or target service uses to compose the SOAP response message.

### Output Parameters

---

*soapResponseData*      **Object** SOAP object containing the message that is to be returned to the client.

---

## pub.soap.utils:resetWSDEffectivePolicy

WmPublic. Returns the effective policy for a handler in a web service descriptor to the policy set in the **Policy** name property in Software AG Designer.

### Input Parameters

---

*wsdName*      **String** The name of the web service descriptor for which you want to reset the effective policy.

**Note:** The `resetWSDEffectivePolicy` service applies only to web services that run in pre-8.2 compatibility mode (i.e., the **Pre-8.2 compatibility mode** property is set to `true`).

*handlerName*      **String** The name of the handler for which you want to reset the effective policy.

### Output Parameters

---

None.

### Usage Notes

You can also use Designer to reset the effective policy. In Designer, open the web service descriptor, select the handler on the **Handlers** view, and modify the value of **Effective policy name** property assigned to the handler.

The [pub.soap.utils:resetWSDEffectivePolicy](#) service throws an exception if the provided inputs are invalid.

### See Also

[pub.soap.utils.setWSDEffectivePolicy](#)

---

## pub.soap.utils.setWSDEffectivePolicy

WmPublic. Sets the effective policy for a handler in a web service descriptor.

### Input Parameters

---

*wsdName*      **String** The name of the web service descriptor for which you want to set the effective policy.

**Note:** The `setWSDEffectivePolicy` service applies only to web services that run in pre-8.2 compatibility mode (i.e., the **Pre-8.2 compatibility mode** property is set to `true`).

*handlerName*      **String** The name of the handler for which you want to set the effective policy.

*effectivePolicyID*      **String** The unique identifier for the policy that you want to use with the handler in the web service descriptor.

### Output Parameters

---

None.

### Usage Notes

The [pub.soap.utils.setWSDEffectivePolicy](#) service overrides the policy originally assigned to the handler in the web service descriptor.

---

The [pub.soap.utils.setWSEffectivePolicy](#) service applies to provider web service descriptors as well as consumer web service descriptors.

The [pub.soap.utils.setWSEffectivePolicy](#) service throws an exceptions if the provided input is incorrect. The service also verifies that the provided effective policy actually exists.

You can also use Designer to set the effective policy. In Designer, open the web service descriptor, select the handler in the **Handlers** view, and modify the value of **Effective policy name** property assigned to the handler.

You can reset the effective policy using the [pub.soap.utils:resetWSEffectivePolicy](#) service.

#### See Also

[pub.soap.utils:resetWSEffectivePolicy](#)

---

## pub.soap.utils:soapDataToBytes

WmPublic. Converts a SOAP object to a Byte Array.

This is useful when you want to use the message with a process that requires the message to be in the form of a Byte Array.

#### Input Parameters

---

*soapData*                      **Object** SOAP object that you want to convert to a Byte Array.

#### Output Parameters

---

*bytes*                              **Object** Entire SOAP message.

#### See Also

[pub.soap.utils:soapDataToString](#)  
[pub.soap.utils:streamToSoapData](#)  
[pub.soap.utils:stringToSoapData](#)

---

## pub.soap.utils:soapDataToString

WmPublic. Converts a SOAP object to a String.

This is useful when you want to use the message with a process that requires the message to be in the form of a String.

---

## Input Parameters

---

*soapData*                    **Object** SOAP object that you want to convert to a String.

## Output Parameters

---

*string*                        **String** Entire SOAP message.

## See Also

[pub.soap.utils:soapDataToBytes](#)  
[pub.soap.utils:streamToSoapData](#)  
[pub.soap.utils:stringToSoapData](#)

---

# pub.soap.utils:soapFault

WmPublic. Document type that defines the generic SOAP fault structure used by web service descriptors created in Integration Server 8.2 and later.

## Parameters

---

*code*                        **Document** Contains the fault code and possible subcodes.

<u>Key</u>	<u>Description</u>
<i>namespaceName</i>	<b>String</b> Optional. Namespace name for the SOAP fault code.
<i>localName</i>	<b>String</b> Code that identifies the fault.
<i>subCodes</i>	<b>Document List</b> Optional. Subcodes that provide further detail. Each Document in the <i>subCodes</i> Document List contains: <ul style="list-style-type: none"> <li>■ <i>namespaceName</i> for the subcode</li> <li>■ <i>localName</i> that identifies the subcode</li> </ul>

*reasons*                    **Document List** Reasons for the SOAP fault. Each Document in the Document List contains a human readable explanation of the cause of the fault.

	<b>Key</b>	<b>Description</b>
	<i>*body</i>	<b>String</b> Text explaining the cause of the fault.
	<i>@lang</i>	<b>String</b> Optional. Language for the human readable description.
<i>node</i>	<b>String</b>	Optional. URI to the SOAP node where the fault occurred.
<i>role</i>	<b>String</b>	Optional. Role in which the node was operating at the point the fault occurred.
<i>detail</i>	<b>Document</b>	Optional. Application-specific details about the SOAP fault.

## pub.soap.utils:streamToSoapData

WmPublic. Converts an `InputStream` containing a SOAP message to a SOAP object.

(A SOAP message must be represented as a SOAP object to be used with the data-retrieval services such as [pub.soap.utils:getHeader](#) and [pub.soap.utils:getBody](#)).

**Note:** This service is a convenient way to produce a SOAP object during development and testing. It is not meant to be used for production purposes because *it does not ensure that a valid SOAP message is produced*. For production purposes, we recommend that you create SOAP objects with the [pub.soap.utils:createSoapData](#) service and populate them with the message-composition services (for example, [pub.soap.utils:addBodyEntry](#) and [pub.soap.utils:addHeaderEntry](#)).

### Input Parameters

<i>stream</i>	<b>java.io.InputStream</b> SOAP message that is to be converted to a SOAP object.
<i>soapProtocol</i>	<p><b>String</b> Optional. Indicates the SOAP protocol that the resulting SOAP object will work with. The default value is read from the <code>watt.server.SOAP.defaultProtocol</code> property. Set to:</p> <ul style="list-style-type: none"> <li>■ SOAP 1.1 Protocol to indicate the SOAP object works with SOAP 1.1.</li> <li>■ SOAP 1.2 Protocol to indicate the SOAP object works with SOAP 1.2.</li> </ul>

---

## Output Parameters

---

*soapData*                    **Object** SOAP object representation of the SOAP message in *stream*.

### Usage Notes

Be aware that if *stream* does not contain a valid SOAP message, this service does not throw an exception. Instead, it produces a *soapData* that contains a representation of whatever it received in *stream* (which might not even be an XML document). This will cause problems later when you attempt to use the *soapData* with other SOAP utilities or pass it to the message handler. To determine whether *soapData* represents a valid SOAP message, we recommend that you always execute the [pub.soap.utils:validateSoapData](#) service immediately after using `streamToSoapData`.

### See Also

[pub.soap.utils:soapDataToBytes](#)  
[pub.soap.utils:stringToSoapData](#)  
[pub.soap.utils:validateSoapData](#)

---

## pub.soap.utils:stringToSoapData

WmPublic. Converts a String containing a SOAP message to a SOAP object.

(A SOAP message must be represented as a SOAP object to be used with the data-retrieval services such as [pub.soap.utils:getHeader](#) and [pub.soap.utils:getBody](#)).

**Note:** This service is a convenient way to produce a SOAP object during development and testing. It is not meant to be used for production purposes because *it does not ensure that a valid SOAP message is produced*. Additionally, producing a SOAP object from a String is a very time-consuming process. For production purposes, we recommend that you create SOAP objects with the such as [pub.soap.utils:getHeader](#) and [pub.soap.utils:getBody](#)).

---

## Input Parameters

---

*string*                        **String** SOAP message that is to be converted to a SOAP object.

*soapProtocol*                **String** Optional. Indicates the SOAP protocol that the resulting SOAP object will work with. The default value is read from the `watt.server.SOAP.defaultProtocol` property. Set to:

- SOAP 1.1 Protocol to indicate the SOAP object works with SOAP 1.1.

- SOAP 1.2 Protocol to indicate the SOAP object works with SOAP 1.2.

*addEmptyHeader*

**String** Optional. Specifies whether to create an empty header entry along with the SOAP envelope and body entries in the SOAP message. Set to:

- `True` to create an empty header entry in the SOAP message.
- `False` to create only the SOAP envelope and body entries.

This setting overrides the global `watt.server.SOAP.addEmptyHeader` setting of Integration Server.

**Important** There is no default value. If you do not specify a value for the `addEmptyHeader` parameter, the service uses the value specified in the `watt.server.SOAP.addEmptyHeader` server configuration parameter. For more information about `watt.server.SOAP.addEmptyHeader`, see *webMethods Integration Server Administrator's Guide*.

## Output Parameters

*soapData*

**Object** SOAP object representation of the SOAP message in *string*.

## See Also

[pub.soap.utils:soapDataToBytes](#)  
[pub.soap.utils:streamToSoapData](#)  
[pub.soap.utils:validateSoapData](#)

## pub.soap.utils:validateSoapData

WmPublic. Verifies that a SOAP object represents a valid SOAP message.

You can use this service to validate a SOAP object that was generated directly from an `InputStream` or `String` with [pub.soap.utils:stringToSoapData](#) or [pub.soap.utils:streamToSoapData](#). If `soapData` does not contain a valid SOAP message, `validateSoapData` will throw an exception.

This service validates the SOAP object against the schema in [pub.soap.schema:envelope](#).

---

## Input Parameters

---

*soapData*                      **Object** SOAP object that you want to validate.

## Output Parameters

---

None.

## Usage Notes

If you create SOAP objects using the standard message-composition services (for example, [pub.soap.utils:createSoapData](#), [pub.soap.utils:addBodyEntry](#), [pub.soap.utils:addHeaderEntry](#)) there is no need to use this service. This service is only necessary when you generate a SOAP object directly from an `InputStream` or a `String`.

When validating SOAP, Integration Server uses the W3C recommendation *XML Schema Part 2: Datatypes*. If you want to validate the input of this service for illegal values in the SOAP envelope and header, set the `watt.core.validation.w3cConformant` configuration parameter to `true`. For information about setting this configuration parameter, see *webMethods Integration Server Administrator's Guide*.

## See Also

[pub.soap.utils:stringToSoapData](#)  
[pub.soap.utils:streamToSoapData](#)

---

## pub.soap.wsa:action

`WmPublic`. Document type that defines the contents of the `wsa:Action` WS-Addressing header.

## Parameters

---

*wsa:Action*                      **Document** Contains the WS-Addressing action.

<u>Key</u>	<u>Description</u>
<i>*body</i>	<b>String</b> Value of the WS-Addressing action.

## Usage Notes

To add, retrieve, or remove the `wsa:Action` header of a SOAP message, use `pub.soap.wsa:action` as the value for the *documentType* input parameter of the [pub.soap.handler:addHeaderBlock](#), [pub.soap.handler:getHeaderBlock](#), and [pub.soap.handler:removeBodyBlock](#) services.

For more details about how Integration Server implements WS-Addressing, see the *Web Services Developer's Guide*.

## pub.soap.wsa:faultTo

WmPublic. Document type that defines the contents of the `wsa:FaultTo` WS-Addressing header.

### Parameters

*wsa:FaultTo* **Document** Contains the address of the intended receiver of the fault message.

Key	Description
<i>wsa:Address</i>	<b>Document</b> Contains the end point URI for the fault message.

Key	Description
<i>*body</i>	String The end point URI for the fault message.

*wsa:ReferenceParameters* **Document** Contains the set of reference parameter elements.

Key	Description
<i>*any</i>	<b>Object List</b> The reference parameter elements.

*wsa:Metadata* **Document** Contains the set of metadata elements.

Key	Description
<i>*any</i>	<b>Object List</b> The metadata elements.

*\*any* **Object List** Contains other extensible elements, if any.

## Usage Notes

To add, retrieve, or remove the `wsa:FaultTo` header of a SOAP message, use `pub.soap.wsa:faultTo` as the value for the `documentType` input parameter of the `pub.soap.handler:addHeaderBlock`, `pub.soap.handler:getHeaderBlock`, and `pub.soap.handler:removeBodyBlock` services.

For more details about how Integration Server implements WS-Addressing, see the *Web Services Developer's Guide*.

---

## pub.soap.wsa:from

WmPublic. Document type that contains the details of the source of the message.

### Parameters

---

*wsa:From*      **Document** Contains the details about the source of the message.

<u>Key</u>	<u>Description</u>				
<i>wsa:Address</i>	<b>Document</b> Contains the address of the source of the message.				
	<table> <thead> <tr> <th><u>Key</u></th> <th><u>Description</u></th> </tr> </thead> <tbody> <tr> <td><i>*body</i></td> <td><b>String</b> The address of the source of the message.</td> </tr> </tbody> </table>	<u>Key</u>	<u>Description</u>	<i>*body</i>	<b>String</b> The address of the source of the message.
<u>Key</u>	<u>Description</u>				
<i>*body</i>	<b>String</b> The address of the source of the message.				
<i>wsa:ReferenceParameters</i>	<b>Document</b> Contains the set of reference parameter elements.				
	<table> <thead> <tr> <th><u>Key</u></th> <th><u>Description</u></th> </tr> </thead> <tbody> <tr> <td><i>*any</i></td> <td><b>Object List</b> The reference parameter elements.</td> </tr> </tbody> </table>	<u>Key</u>	<u>Description</u>	<i>*any</i>	<b>Object List</b> The reference parameter elements.
<u>Key</u>	<u>Description</u>				
<i>*any</i>	<b>Object List</b> The reference parameter elements.				
<i>wsa:Metadata</i>	<b>Document</b> Contains the set of metadata elements.				
	<table> <thead> <tr> <th><u>Key</u></th> <th><u>Description</u></th> </tr> </thead> <tbody> <tr> <td><i>*any</i></td> <td><b>Object List</b> The metadata elements.</td> </tr> </tbody> </table>	<u>Key</u>	<u>Description</u>	<i>*any</i>	<b>Object List</b> The metadata elements.
<u>Key</u>	<u>Description</u>				
<i>*any</i>	<b>Object List</b> The metadata elements.				

*\*any* **Object List** Contains other extensible elements, if any.

### Usage Notes

To add, retrieve, or remove the `wsa:From` header of a SOAP message, use `pub.soap.wsa:from` as the value for the `documentType` input parameter of the `pub.soap.handler:addHeaderBlock`, `pub.soap.handler:getHeaderBlock`, and `pub.soap.handler:removeBodyBlock` services.

For more details about how Integration Server implements WS-Addressing, see the *Web Services Developer's Guide*.

---

## pub.soap.wsa:messageID

WmPublic. Document type that defines the contents of the `wsa:MessageID` WS-Addressing header.

### Parameters

---

*wsa:MessageID* **Document** Unique identifier of the SOAP message.

Key	Description
<i>*body</i>	<b>String</b> The unique identifier of the SOAP message.

### Usage Notes

To add, retrieve, or remove the `wsa:MessageID` header of a SOAP message, use `pub.soap.wsa:messageID` as the value for the `documentType` input parameter of the `pub.soap.handler:addHeaderBlock`, `pub.soap.handler:getHeaderBlock`, and `pub.soap.handler:removeBodyBlock` services.

For more details about how Integration Server implements WS-Addressing, see the *Web Services Developer's Guide*.

---

## pub.soap.wsa:problemAction

WmPublic. Document type that captures additional information about faults.

---

## Parameters

---

*wsa:ProblemAction* **Document** Contains additional information about faults.

<u>Key</u>	<u>Description</u>
<i>wsa:Action</i>	<b>Document</b> Optional. Element that provides the details about the [action] that caused the problem.
<u>Key</u>	<u>Description</u>
<i>*body</i>	<b>String</b> Optional. The [action] that caused the problem.
<i>wsa:SoapAction</i>	<b>String</b> Optional. Element that contains the SOAPAction IRI that caused the problem.

## Usage Notes

To add, retrieve, or remove the `wsa:ProblemAction` header of a SOAP message, use `pub.soap.wsa:problemAction` as the value for the *documentType* input parameter of the `pub.soap.handler:addHeaderBlock`, `pub.soap.handler:getHeaderBlock`, and `pub.soap.handler:removeBodyBlock` services.

For more details about how Integration Server implements WS-Addressing, see the *Web Services Developer's Guide*.

---

## pub.soap.wsa:problemHeaderQName

WmPublic. Document type that captures additional information about faults.

## Parameters

---

*wsa:ProblemHeaderQName* **Document** Contains additional information about faults.

<u>Key</u>	<u>Description</u>
<i>*body</i>	<b>String</b> Optional. The QName representing the name of the root element of the problem header block.

### Usage Notes

To add, retrieve, or remove the `wsa:ProblemHeaderQName` header of a SOAP message, use `pub.soap.wsa:problemHeaderQName` as the value for the *documentType* input parameter of the `pub.soap.handler:addHeaderBlock`, `pub.soap.handler:getHeaderBlock`, and `pub.soap.handler:removeBodyBlock` services.

For more details about how Integration Server implements WS-Addressing, see the *Web Services Developer's Guide*.

---

## pub.soap.wsa:problemIRI

WmPublic. Document type that captures the IRI that caused the problem.

### Parameters

---

*wsa:ProblemIRI*      **Document** Contains the IRI that caused the problem.

<u>Key</u>	<u>Description</u>
<i>*body</i>	<b>String</b> Optional. The string representing the IRI that caused the problem.

### Usage Notes

To add, retrieve, or remove the `wsa:ProblemIRI` header of a SOAP message, use `pub.soap.wsa:problemIRI` as the value for the *documentType* input parameter of the `pub.soap.handler:addHeaderBlock`, `pub.soap.handler:getHeaderBlock`, and `pub.soap.handler:removeBodyBlock` services.

For more details about how Integration Server implements WS-Addressing, see the *Web Services Developer's Guide*.

---

## pub.soap.wsa:relatesTo

WmPublic. Document type that defines the contents of the `wsa:RelatesTo` WS-Addressing header.

### Parameters

---

*wsa:RelatesTo*      **Document** Contains the relationship information.

<u>Key</u>	<u>Description</u>
------------	--------------------

<i>@Relationship Type</i>	<b>String</b> Optional. The relationship type.
<i>*body</i>	<b>String</b> <wsa:MessageID> of the related SOAP message.

### Usage Notes

To add, retrieve, or remove the `wsa:RelatesTo` header of a SOAP message, use `pub.soap.wsa:relatesTo` as the value for the *documentType* input parameter of the `pub.soap.handler:addHeaderBlock`, `pub.soap.handler:getHeaderBlock`, and `pub.soap.handler:removeBodyBlock` services.

For more details about how Integration Server implements WS-Addressing, see the *Web Services Developer's Guide*.

## pub.soap.wsa:replyTo

WmPublic. Document type that defines the contents of the `wsa:ReplyTo` WS-Addressing header.

### Parameters

*wsa:ReplyTo* **Document** Contains the address of the intended receiver of the response message.

<u>Key</u>	<u>Description</u>				
<i>wsa:Address</i>	<b>Document</b> Contains the end point URI for the response message.				
	<table> <thead> <tr> <th><u>Key</u></th> <th><u>Description</u></th> </tr> </thead> <tbody> <tr> <td><i>*body</i></td> <td><b>String</b> Optional. The end point URI for the response message.</td> </tr> </tbody> </table>	<u>Key</u>	<u>Description</u>	<i>*body</i>	<b>String</b> Optional. The end point URI for the response message.
<u>Key</u>	<u>Description</u>				
<i>*body</i>	<b>String</b> Optional. The end point URI for the response message.				
<i>wsa:Reference Parameters</i>	<b>Document</b> Optional. Contains the set of reference parameter elements.				
	<table> <thead> <tr> <th><u>Key</u></th> <th><u>Description</u></th> </tr> </thead> <tbody> </tbody> </table>	<u>Key</u>	<u>Description</u>		
<u>Key</u>	<u>Description</u>				

	<i>*any</i>	<b>Object List</b> Optional. The reference parameter elements.
<i>wsa:Metadata</i>	<b>Document</b>	Optional. Contains the set of metadata elements.
	<b>Key</b>	<b>Description</b>
	<i>*any</i>	<b>Object List</b> Optional. The metadata elements.
<i>*any</i>	<b>Object List</b>	Optional. Contains other extensible elements, if any.

### Usage Notes

To add, retrieve, or remove the `wsa:ReplyTo` header of a SOAP message, use `pub.soap.wsa:replyTo` as the value for the `documentType` input parameter of the `pub.soap.handler:addHeaderBlock`, `pub.soap.handler:getHeaderBlock`, and `pub.soap.handler:removeBodyBlock` services.

For more details about how Integration Server implements WS-Addressing, see the *Web Services Developer's Guide*.

---

## pub.soap.wsa:retryAfter

WmPublic. Document type that you can use to retrieve the `wsa:RetryAfter` header of a SOAP message.

### Parameters

<i>wsa:RetryAfter</i>	<b>Document</b>	Contains the <code>wsa:RetryAfter</code> header of a SOAP message.
	<b>Key</b>	<b>Description</b>
	<i>*body</i>	<b>String</b> Optional. The retry after duration retrieved from the <code>wsa:RetryAfter</code> SOAP header.

### Usage Notes

To add, retrieve, or remove the `wsa:RetryAfter` header of a SOAP message, use `pub.soap.wsa:retryAfter` as the value for the `documentType` input parameter

of the [pub.soap.handler:addHeaderBlock](#), [pub.soap.handler:getHeaderBlock](#), and [pub.soap.handler:removeBodyBlock](#) services.

For more details about how Integration Server implements WS-Addressing, see the *Web Services Developer's Guide*.

---

## pub.soap.wsa:to

WmPublic. Document type that defines the contents of the `wsa:To` WS-Addressing header.

### Parameters

---

*wsa:To*                      **Document** Contains the address of the intended receiver of the message.

Key	Description
<i>*body</i>	<b>String</b> The address of the intended receiver of the message.

### Usage Notes

To add, retrieve, or remove the `wsa:To` header of a SOAP message, use `pub.soap.wsa:to` as the value for the *documentType* input parameter of the [pub.soap.handler:addHeaderBlock](#), [pub.soap.handler:getHeaderBlock](#), and [pub.soap.handler:removeBodyBlock](#) services.

For more details about how Integration Server implements WS-Addressing, see the *Web Services Developer's Guide*.

---

## pub.soap.wsa:schema\_wsa

WmPublic. A schema containing the elements from `http://www.w3.org/2005/08/addressing` namespace.

---

## pub.soap.wsa.submission:action

WmPublic. Document type that defines the contents of the `wsa:Action` WS-Addressing header.

### Parameters

---

*wsa:Action*                      **Document** Contains the WS-Addressing action.

<u>Key</u>	<u>Description</u>
<i>*body</i>	<b>String</b> Value of the WS-Addressing action.

### Usage Notes

To add, retrieve, or remove the `wsa:Action` header of a SOAP message, use `pub.soap.wsa.submission:action` as the value for the `documentType` input parameter of the `pub.soap.handler:addHeaderBlock`, `pub.soap.handler:getHeaderBlock`, and `pub.soap.handler:removeBodyBlock` services.

The `pub.soap.wsa.submission:action` document type relates to the W3C WS-Addressing Submission version of WS-Addressing specification.

For more details about how Integration Server implements WS-Addressing, see the *Web Services Developer's Guide*.

---

## pub.soap.wsa.submission:faultTo

WmPublic. Document type that defines the contents of the `wsa:FaultTo` WS-Addressing header.

### Parameters

---

*wsa:FaultTo* **Document** Contains the address of the intended receiver of the fault message.

<u>Key</u>	<u>Description</u>
<i>wsa:Address</i>	<b>Document</b> Contains the end point URI for the fault message.

<u>Key</u>	<u>Description</u>
<i>*body</i>	<b>String</b> Optional. The end point URI for the fault message.
<i>wsa:Reference Properties</i>	<b>Document</b> Optional. The properties that are required to identify the entity or resource being conveyed.

<u>Key</u>	<u>Description</u>
------------	--------------------

	<i>*any</i>	<b>Object List</b> Optional. The property elements.
<i>wsa:Reference Parameters</i>		<b>Document</b> Optional. Contains the set of reference parameter elements.
	<b>Key</b>	<b>Description</b>
	<i>*any</i>	<b>Object List</b> Optional. The reference parameter elements.
<i>wsa:PortType</i>		<b>Document</b> Optional. Contains the QName of the primary <code>portType</code> of the endpoint being conveyed.
	<b>Key</b>	<b>Description</b>
	<i>*body</i>	<b>Object List</b> Optional. The QName of the primary <code>portType</code> of the endpoint being conveyed.
<i>wsa:Service Name</i>		<b>Document</b> Optional. Represents the QName identifying the WSDL service element that contains the definition of the endpoint being conveyed.
	<b>Key</b>	<b>Description</b>
	<i>@PortName</i>	<b>String</b> Optional. The name of the <code>&lt;wsdl:port&gt;</code> definition that corresponds to the endpoint being referenced.
	<i>*body</i>	<b>String</b> Optional. The <code>&lt;wsdl:service&gt;</code> definition that contains a WSDL description of the endpoint being referenced.
<i>*any</i>		<b>Object List</b> Optional. Contains other extensible elements, if any.

## Usage Notes

To add, retrieve, or remove the `wsa:FaultTo` header of a SOAP message, use `pub.soap.wsa.submission:faultTo` as the value for the `documentType` input parameter of the `pub.soap.handler:addHeaderBlock`, `pub.soap.handler:getHeaderBlock`, and `pub.soap.handler:removeBodyBlock` services.

The `pub.soap.wsa.submission:faultTo` document type relates to the W3C WS-Addressing Submission version of WS-Addressing specification.

For more details about how Integration Server implements WS-Addressing, see the *Web Services Developer's Guide*.

---

## pub.soap.wsa.submission:from

WmPublic. Document type that contains the details about the source of the message.

### Parameters

---

*wsa:From*

**Document** Contains the details about the source of the message.

<u>Key</u>	<u>Description</u>
<i>wsa:Address</i>	<b>Document</b> Contains the details about the source of the message.

<u>Key</u>	<u>Description</u>
<i>*body</i>	<b>String</b> Optional. The address of the source of the message.

*wsa:Reference Properties* **Document** Optional. The properties that are required to identify the entity or resource being conveyed.

<u>Key</u>	<u>Description</u>
<i>*any</i>	<b>Object List</b> Optional. The property elements.

*wsa:Reference Parameters* **Document** Optional. Contains the set of reference parameter elements.

<u>Key</u>	<u>Description</u>
------------	--------------------

	<i>*any</i>	<b>Object List</b> Optional. The reference parameter elements.						
<i>wsa:PortType</i>		<b>Document</b> Optional. Contains the QName of the primary <code>portType</code> of the endpoint being conveyed.						
		<table border="0"> <thead> <tr> <th style="border-bottom: 1px solid black; text-decoration: underline;">Key</th> <th style="border-bottom: 1px solid black; text-decoration: underline;">Description</th> </tr> </thead> <tbody> <tr> <td><i>*body</i></td> <td><b>Object List</b> Optional. The QName of the primary <code>portType</code> of the endpoint being conveyed.</td> </tr> </tbody> </table>	Key	Description	<i>*body</i>	<b>Object List</b> Optional. The QName of the primary <code>portType</code> of the endpoint being conveyed.		
Key	Description							
<i>*body</i>	<b>Object List</b> Optional. The QName of the primary <code>portType</code> of the endpoint being conveyed.							
<i>wsa:Service Name</i>		<b>Document</b> Optional. Represents the QName identifying the WSDL service element that contains the definition of the endpoint being conveyed.						
		<table border="0"> <thead> <tr> <th style="border-bottom: 1px solid black; text-decoration: underline;">Key</th> <th style="border-bottom: 1px solid black; text-decoration: underline;">Description</th> </tr> </thead> <tbody> <tr> <td><i>@PortName</i></td> <td><b>String</b> Optional. The name of the <code>&lt;wsdl:port&gt;</code> definition that corresponds to the endpoint being referenced.</td> </tr> <tr> <td><i>*body</i></td> <td><b>String</b> Optional. The <code>&lt;wsdl:service&gt;</code> definition that contains a WSDL description of the endpoint being referenced.</td> </tr> </tbody> </table>	Key	Description	<i>@PortName</i>	<b>String</b> Optional. The name of the <code>&lt;wsdl:port&gt;</code> definition that corresponds to the endpoint being referenced.	<i>*body</i>	<b>String</b> Optional. The <code>&lt;wsdl:service&gt;</code> definition that contains a WSDL description of the endpoint being referenced.
Key	Description							
<i>@PortName</i>	<b>String</b> Optional. The name of the <code>&lt;wsdl:port&gt;</code> definition that corresponds to the endpoint being referenced.							
<i>*body</i>	<b>String</b> Optional. The <code>&lt;wsdl:service&gt;</code> definition that contains a WSDL description of the endpoint being referenced.							
	<i>*any</i>	<b>Object List</b> Optional. Contains other extensible elements, if any.						

### Usage Notes

To add, retrieve, or remove the `wsa:From` header of a SOAP message, use `pub.soap.wsa.submission:from` as the value for the *documentType* input parameter of the `pub.soap.handler:addHeaderBlock`, `pub.soap.handler:getHeaderBlock`, and `pub.soap.handler:removeBodyBlock` services.

The `pub.soap.wsa.submission:from` document type relates to the W3C WS-Addressing Submission version of WS-Addressing specification.

For more details about how Integration Server implements WS-Addressing, see the *Web Services Developer's Guide*.

---

## pub.soap.wsa.submission:messageID

WmPublic. Document type that defines the contents of the `wsa:MessageID` WS-Addressing header.

### Parameters

---

*wsa:MessageID*      **Document** Unique identifier of the SOAP message.

<u>Key</u>	<u>Description</u>
<i>*body</i>	<b>String</b> The unique identifier of the SOAP message.

### Usage Notes

To add, retrieve, or remove the `wsa:MessageID` header of a SOAP message, use `pub.soap.wsa.submission:messageID` as the value for the *documentType* input parameter of the `pub.soap.handler:addHeaderBlock`, `pub.soap.handler:getHeaderBlock`, and `pub.soap.handler:removeBodyBlock` services.

The `pub.soap.wsa.submission:messageID` document type relates to the W3C WS-Addressing Submission version of WS-Addressing specification.

For more details about how Integration Server implements WS-Addressing, see the *Web Services Developer's Guide*.

---

## pub.soap.wsa.submission:relatesTo

WmPublic. Document type that defines the contents of the `wsa:RelatesTo` WS-Addressing header.

### Parameters

---

*wsa:RelatesTo*      **Document** Contains the relationship information.

<u>Key</u>	<u>Description</u>
<i>@RelationshipType</i>	<b>String</b> Optional. The relationship type.

*\*body* **String**<wsa:MessageID> of the related SOAP message.

### Usage Notes

To add, retrieve, or remove the `wsa:RelatesTo` header of a SOAP message, use `pub.soap.wsa.submission:relatesTo` as the value for the *documentType* input parameter of the `pub.soap.handler:addHeaderBlock`, `pub.soap.handler:getHeaderBlock`, and `pub.soap.handler:removeBodyBlock` services.

The `pub.soap.wsa.submission:relatesTo` document type relates to the W3C WS-Addressing Submission version of WS-Addressing specification.

For more details about how Integration Server implements WS-Addressing, see the *Web Services Developer's Guide*.

---

## pub.soap.wsa.submission:replyTo

WmPublic. Document type that specifies the destination to which the response message is to be sent.

### Parameters

---

*wsa:FaultTo* **Document** Contains the address of the intended receiver of the response message.

<u>Key</u>	<u>Description</u>
<i>wsa:Address</i>	<b>Document</b> Contains the end point URI for the response.
<i>*body</i>	<b>String</b> Optional. The end point URI for the fault message.
<i>wsa:Reference Properties</i>	<b>Document</b> Optional. The properties that are required to identify the entity or resource being conveyed.
<u>Key</u>	<u>Description</u>

	<i>*any</i>	<b>Object List</b> Optional. The property elements.						
<i>wsa:Reference Parameters</i>		<b>Document</b> Optional. Contains the set of reference parameter elements.						
		<table border="0"> <thead> <tr> <th style="text-align: left;"><u>Key</u></th> <th style="text-align: left;"><u>Description</u></th> </tr> </thead> <tbody> <tr> <td><i>*any</i></td> <td><b>Object List</b> Optional. The reference parameter elements.</td> </tr> </tbody> </table>	<u>Key</u>	<u>Description</u>	<i>*any</i>	<b>Object List</b> Optional. The reference parameter elements.		
<u>Key</u>	<u>Description</u>							
<i>*any</i>	<b>Object List</b> Optional. The reference parameter elements.							
<i>wsa:PortType</i>		<b>Document</b> Optional. Contains the QName of the primary <code>portType</code> of the endpoint being conveyed.						
		<table border="0"> <thead> <tr> <th style="text-align: left;"><u>Key</u></th> <th style="text-align: left;"><u>Description</u></th> </tr> </thead> <tbody> <tr> <td><i>*body</i></td> <td><b>Object List</b> Optional. The QName of the primary <code>portType</code> of the endpoint being conveyed.</td> </tr> </tbody> </table>	<u>Key</u>	<u>Description</u>	<i>*body</i>	<b>Object List</b> Optional. The QName of the primary <code>portType</code> of the endpoint being conveyed.		
<u>Key</u>	<u>Description</u>							
<i>*body</i>	<b>Object List</b> Optional. The QName of the primary <code>portType</code> of the endpoint being conveyed.							
<i>wsa:Service Name</i>		<b>Document</b> Optional. Represents the QName identifying the WSDL service element that contains the definition of the endpoint being conveyed.						
		<table border="0"> <thead> <tr> <th style="text-align: left;"><u>Key</u></th> <th style="text-align: left;"><u>Description</u></th> </tr> </thead> <tbody> <tr> <td><i>@PortName</i></td> <td><b>String</b> Optional. The name of the <code>&lt;wsdl:port&gt;</code> definition that corresponds to the endpoint being referenced.</td> </tr> <tr> <td><i>*body</i></td> <td><b>String</b> Optional. The <code>&lt;wsdl:service&gt;</code> definition that contains a WSDL description of the endpoint being referenced.</td> </tr> </tbody> </table>	<u>Key</u>	<u>Description</u>	<i>@PortName</i>	<b>String</b> Optional. The name of the <code>&lt;wsdl:port&gt;</code> definition that corresponds to the endpoint being referenced.	<i>*body</i>	<b>String</b> Optional. The <code>&lt;wsdl:service&gt;</code> definition that contains a WSDL description of the endpoint being referenced.
<u>Key</u>	<u>Description</u>							
<i>@PortName</i>	<b>String</b> Optional. The name of the <code>&lt;wsdl:port&gt;</code> definition that corresponds to the endpoint being referenced.							
<i>*body</i>	<b>String</b> Optional. The <code>&lt;wsdl:service&gt;</code> definition that contains a WSDL description of the endpoint being referenced.							
	<i>*any</i>	<b>Object List</b> Optional. Contains other extensible elements, if any.						

## Usage Notes

To add, retrieve, or remove the `wsa:ReplyTo` header of a SOAP message, use `pub.soap.wsa.submission:replyTo` as the value for the *documentType* input parameter of the [pub.soap.handler:addHeaderBlock](#), [pub.soap.handler:getHeaderBlock](#), and [pub.soap.handler:removeBodyBlock](#) services.

The `pub.soap.wsa.submission:replyTo` document type relates to the W3C WS-Addressing Submission version of WS-Addressing specification.

For more details about how Integration Server implements WS-Addressing, see the *Web Services Developer's Guide*.

---

## pub.soap.wsa.submission:retryAfter

WmPublic. Document type that you can use to retrieve the `wsa:RetryAfter` header of a SOAP message.

### Parameters

---

*wsa:RetryAfter*

**Document** Contains you can use to retrieve the `wsa:RetryAfter` header of a SOAP message.

Key	Description
<i>*body</i>	<b>String</b> Optional. The retry after duration retrieved from the <code>wsa:RetryAfter</code> SOAP header.

## Usage Notes

To add, retrieve, or remove the `wsa:RetryAfter` header of a SOAP message, use `pub.soap.wsa.submission:retryAfter` as the value for the *documentType* input parameter of the [pub.soap.handler:addHeaderBlock](#), [pub.soap.handler:getHeaderBlock](#), and [pub.soap.handler:removeBodyBlock](#) services.

The `pub.soap.wsa.submission:retryAfter` document type relates to the W3C WS-Addressing Submission version of WS-Addressing specification.

For more details about how Integration Server implements WS-Addressing, see the *Web Services Developer's Guide*.

---

## pub.soap.wsa.submission:to

WmPublic. Document type that defines the contents of the `wsa:To` WS-Addressing header.

### Parameters

---

*wsa:To*                      **Document** Contains the address of the intended receiver of the message.

Key	Description
<i>*body</i>	<b>String</b> The address of the intended receiver of the message.

### Usage Notes

To add, retrieve, or remove the `wsa:To` header of a SOAP message, use `pub.soap.wsa.submission:to` as the value for the *documentType* input parameter of the `pub.soap.handler:addHeaderBlock`, `pub.soap.handler:getHeaderBlock`, and `pub.soap.handler:removeBodyBlock` services.

The `pub.soap.wsa.submission:to` document type relates to the W3C WS-Addressing Submission version of WS-Addressing specification.

For more details about how Integration Server implements WS-Addressing, see the *Web Services Developer's Guide*.

---

## pub.soap.wsa.submission:schema\_wsa\_submission

WmPublic. A schema containing the elements from `http://schemas.xmlsoap.org/ws/2004/08/addressing` namespace.

---

## pub.soap.wsrn:closeSequence

WmPublic. Closes a reliable messaging sequence.

### Input Parameters

---

*serverSequenceId*            **String** Unique identifier associated with the reliable messaging sequence that you want to close.

**Note:** The *serverSequenceId* parameter is returned as the output parameter of the `pub.soap.wsrn:createSequence` service or as the *reliableMessagingInfo/responseReliableMessagingProperties/server SequenceId* output parameter of a web service connector.

## Output Parameters

- fault*      **Document** Conditional. Contents of the fault block.  
The fault document references the `pub.soap.util:soapFault` document type.
- transportInfo*      **Document** Conditional. Headers from response and request messages.  
The contents of *transportInfo* vary depending on the actual transport used by the service.

**Note:** The transport information returned by this service is similar to the transport information returned by a web service connector. For more information, see *Web Services Developer's Guide*.

The *transportInfo* parameter contains the following keys:

<u>Key</u>	<u>Description</u>
<i>requestHeaders</i>	<b>Document</b> Conditional. Header fields from the request message issued by the reliable messaging source.
<i>responseHeaders</i>	<b>Document</b> Conditional. Header fields from the response.  Each key in <i>responseHeaders</i> represents a field (line) of the response header. Key names represent the names of header fields. The key values are Strings containing the values of the fields.  Whether or not the service returns the <i>responseHeaders</i> parameter depends on the success or failure of the service. In the case of failure, the point at which the failure occurs determines the presence of the <i>responseHeaders</i> parameter.
<i>status</i>	<b>Document</b> Conditional. Status code from the request.

---

*statusMessage*      **Document** Conditional. Description of the status code returned by the underlying transport.

---

## pub.soap.wsrn:createSequence

WmPublic. Sends a request to a reliable messaging destination to create a new reliable messaging sequence.

### Input Parameters

---

*consumer  
WebService  
DescriptorName*      **String** Fully qualified name of the consumer web service descriptor for which the reliable message sequence is to be created. That is, the consumer web service descriptor that contains the reliable messaging destination endpoint information.

**Note:** The consumer web service descriptor must have a reliable messaging policy attached or must be created from a reliable messaging policy annotated WSDL.

*\_port*      **String** Specifies the port that Integration Server uses to resolve the endpoint address with which the reliable messaging sequence is to be established.

*sequenceKey*      **String** Optional. Unique key to identify the message sequence. A reliable messaging client associates a sequence key to a message sequence based on the endpoint URL to which the message sequence is directed. In cases where there are several message sequences directed to the same endpoint URL, you can specify a custom sequence key to identify each sequence. Each sequence is then uniquely identified by the endpoint URL and the user-specified sequence key.

**Important**The user-specified sequence key should not exceed 32 characters in length.

*acksTo*      **Document** Optional. Consumer response endpoint address to which the reliable message destination must send the acknowledgment. To specify the consumer response endpoint address, use the **Response endpoint address template** binder property of the consumer web service descriptor for which the reliable message sequence is to be created, as the address template and replace the placeholders <server> and <port> with appropriate values.

If no address is specified as *acksTo*, the acknowledgment messages are sent back to the requester.

<u>Key</u>	<u>Description</u>
<i>address</i>	<b>Document</b> Document specifying the address to which the acknowledgment is to be sent.

<u>Key</u>	<u>Description</u>
<i>value</i>	<b>String</b> Address to which the acknowledgment is to be sent.

*auth* **Document** Optional. Transport-level credentials to include in the request. Integration Server uses the information provided in *auth* to create the SOAP request.

**Note:** Information specified in *auth* overwrites any authentication credentials specified in the consumer endpoint alias that is assigned to the binder.

<u>Key</u>	<u>Description</u>
<i>transport</i>	<b>Document</b> Optional. Transport level authorization parameters to include in the HTTP request. Integration Server uses the information specified in the <i>transport</i> variable to populate the Authorization header in the HTTP request.

You only need to provide credentials in *transport* if the endpoint URL specifies HTTPS and you want to overwrite the credentials specified in the consumer endpoint alias assigned to the binder.

<u>Key</u>	<u>Description</u>
<i>type</i>	<b>String</b> Optional. Type of authentication required by the reliable messaging client. If <i>type</i> is not specified, Integration Server uses <b>Basic</b> .

**Note:** any value other than **Basic** is specified, Integration Server ignores the credentials

provided in *user*, *pass*, and *serverCerts*.

<i>user</i>	<b>String</b> Optional. User name used to authenticate the reliable messaging client at the HTTP or HTTPS transport level on the reliable messaging server.
<i>pass</i>	<b>String</b> Optional. Password used to authenticate the reliable messaging client to the reliable messaging server.
<i>serverCerts</i>	<p><b>Document</b> Optional. The private key and certificate chain of the message signer.</p> <ul style="list-style-type: none"> <li>■ <i>keyStoreAlias</i> <b>String</b> Alias to the keystore that contains the private key used to securely connect to the reliable messaging server.</li> <li>■ <i>keyAlias</i> <b>String</b> Alias to the key in the keystore that contains the private key used to connect to the reliable messaging server securely. The key must be in the keystore specified in <i>keyStoreAlias</i>.</li> </ul>

*timeout* **String** Optional. Time (in milliseconds) to wait for a response from the reliable messaging server before timing out and terminating the request.

If *timeout* is not specified, or a value less than 0 is specified, Integration Server uses the value of the `watt.server.SOAP.request.timeout` server property.

For more information about server configuration properties, see *webMethods Integration Server Administrator's Guide*.

A *timeout* value of 0 means Integration Server waits for a response indefinitely. If the connection to the reliable messaging server ends before Integration Server receives a response, the service ends with an exception and a status code of 408.

<i>_url</i>	<b>String</b> Optional. URL to use as the endpoint URL for the web service. If supplied, the value of <i>_url</i> overwrites the endpoint URL in the original WSDL.
<i>transportHeaders</i>	<b>Document</b> Optional. Transport header fields that you want to explicitly set in the request issued by the reliable messaging client.  Specify a key in <i>transportHeaders</i> for each header field that you want to set. The key name represents the name of the header field and the key value represents the value of that header field. The names and values supplied to <i>transportHeaders</i> must be of type String. For information about using <i>transportHeaders</i> , including a description of the default Integration Server behavior, see <i>Web Services Developer's Guide</i> .

## Output Parameters

---

<i>serverSequenceId</i>	<b>String</b> Unique identifier returned by Integration Server and associated with each message sequence.
-------------------------	---

**Note:** The *serverSequenceId* is used as the input parameter of `pub.soap.wsrn:closeSequence`, `pub.soap.wsrn:sendAcknowledgementRequest`, `pub.soap.wsrn:terminateSequence`, and `pub.soap.wsrn:waitUntilSequenceCompleted` services.

<i>fault</i>	<b>Document</b> Conditional. Contents of the fault block.  The fault document references the " <a href="#">pub.soap.utils:soapFault</a> " on <a href="#">page 846</a> document type.
--------------	--

<i>transportInfo</i>	<b>Document</b> Conditional. Headers from response and request messages.
----------------------	--

The contents of *transportInfo* vary depending on the actual transport used by the service.

**Note:** The transport information returned by this service is similar to the transport information returned by a web service connector. For more information, see *Web Services Developer's Guide*.

The *transportInfo* parameter contains the following keys:

<u>Key</u>	<u>Description</u>
------------	--------------------

<i>requestHeaders</i>	<b>Document</b> Conditional. Header fields from the request message issued by the reliable messaging source.
<i>responseHeaders</i>	<p><b>Document</b> Conditional. Header fields from the response.</p> <p>Each key in <i>responseHeaders</i> represents a field (line) of the response header. Key names represent the names of header fields. The key values are Strings containing the values of the fields.</p> <p>Whether or not the service returns the <i>responseHeaders</i> parameter depends on the success or failure of the service. In the case of failure, the point at which the failure occurs determines the presence of the <i>responseHeaders</i> parameter.</p>
<i>status</i>	<b>Document</b> Conditional. Status code from the request.
<i>statusMessage</i>	<b>Document</b> Conditional. Description of the status code returned by the underlying transport.

---

## pub.soap.wsrn:sendAcknowledgementRequest

WmPublic. Requests an acknowledgment for a message sequence.

### Input Parameters

---

*serverSequenceId* **String** Unique identifier associated with the reliable messaging sequence for which you want an acknowledgment.

**Note:** The *serverSequenceId* parameter is returned as the output parameter of the `pub.soap.wsrn:createSequence` service or as the `reliableMessagingInfo/responseReliableMessagingProperties/serverSequenceId` output parameter of the web service connector.

### Output Parameters

---

*fault* **Document** Conditional. Contents of the fault block.

The fault document references the "[pub.soap.utils:soapFault](#)" on [page 846](#) document type.

*transportInfo*

**Document** Conditional. Headers from response and request messages.

The contents of *transportInfo* vary depending on the actual transport used by the service.

**Note:** The transport information returned by this service is similar to the transport information returned by a web service connector. For more information, see *Web Services Developer's Guide*.

The *transportInfo* parameter contains the following keys:

<u>Key</u>	<u>Description</u>
<i>requestHeaders</i>	<b>Document</b> Conditional. Header fields from the request message issued by the reliable messaging source.
<i>responseHeaders</i>	<p><b>Document</b> Conditional. Header fields from the response.</p> <p>Each key in <i>responseHeaders</i> represents a field (line) of the response header. Key names represent the names of header fields. The key values are Strings containing the values of the fields.</p> <p>Whether or not the service returns the <i>responseHeaders</i> parameter depends on the success or failure of the service. In the case of failure, the point at which the failure occurs determines the presence of the <i>responseHeaders</i> parameter.</p>
<i>status</i>	<b>Document</b> Conditional. Status code from the request.
<i>statusMessage</i>	<b>Document</b> Conditional. Description of the status code returned by the underlying transport.

---

## pub.soap.wsrn:terminateSequence

WmPublic. Terminates a reliable messaging sequence.

### Input Parameters

---

*serverSequenceId* **String** Unique identifier associated with the reliable messaging sequence that you want to terminate.

**Note:** The *serverSequenceId* parameter is returned as the output parameter of the `pub.soap.wsrn:createSequence` service or as the `reliableMessagingInfo/responseReliableMessagingProperties/serverSequenceId` output parameter of the web service connector.

### Output Parameters

---

*fault* **Document** Conditional. Contents of the fault block.  
The fault document references the "[pub.soap.utils:soapFault](#)" on [page 846](#) document type.

*transportInfo* **Document** Conditional. Headers from response and request messages.

The contents of *transportInfo* vary depending on the actual transport used by the service.

**Note:** The transport information returned by this service is similar to the transport information returned by a web service connector. For more information, see *Web Services Developer's Guide*.

The *transportInfo* parameter contains the following keys:

Key	Description
<i>requestHeaders</i>	<b>Document</b> Conditional. Header fields from the request message issued by the reliable messaging source.
<i>responseHeaders</i>	<b>Document</b> Conditional. Header fields from the response.

Each key in *responseHeaders* represents a field (line) of the response header. Key names represent the names of header fields. The key values are Strings containing the values of the fields.

Whether or not the service returns the *responseHeaders* parameter depends on the success or failure of the service. In the case of failure, the point at which the failure occurs determines the presence of the *responseHeaders* parameter.

*status*

**Document** Conditional. Status code from the request.

*statusMessage*

**Document** Conditional. Description of the status code returned by the underlying transport.

---

## pub.soap.wsrn:waitUntilSequenceCompleted

WmPublic. Instructs Integration Server to wait for a reliable messaging sequence to complete before terminating it.

### Input Parameters

---

*serverSequenceId* **String** Unique identifier associated with a reliable messaging sequence. Integration Server waits for all the messages in the specified message sequence to be sent and acknowledged before terminating the sequence.

**Note:** The *serverSequenceId* parameter is returned as the output parameter of the `pub.soap.wsrn:createSequence` service or as the *reliableMessagingInfo/responseReliableMessagingProperties/serverSequenceId* output parameter of the web service connector.

*maxWaitingTime* **String** Optional. Maximum time (in milliseconds) to wait for the reliable messaging sequence to complete before terminating it. If no value is specified, the service waits indefinitely until it receives a reply.

## Output Parameters

---

<i>fault</i>	<p><b>Document</b> Conditional. Contents of the fault block.</p> <p>The fault document references the "<a href="#">pub.soap.util:soapFault</a>" on <a href="#">page 846</a> document type.</p>
<i>transportInfo</i>	<p><b>Document</b> Conditional. Headers from response and request messages.</p> <p>The contents of <i>transportInfo</i> vary depending on the actual transport used by the service.</p> <div style="background-color: #f0f0f0; padding: 10px; margin-top: 10px;"> <p><b>Note:</b> The transport information returned by this service is similar to the transport information returned by a web service connector. For more information, see <i>Web Services Developer's Guide</i>.</p> </div>

The *transportInfo* parameter contains the following keys:

Key	Description
<i>requestHeaders</i>	<p><b>Document</b> Conditional. Header fields from the request message issued by the reliable messaging source.</p>
<i>responseHeaders</i>	<p><b>Document</b> Conditional. Header fields from the response.</p> <p>Each key in <i>responseHeaders</i> represents a field (line) of the response header. Key names represent the names of header fields. The key values are Strings containing the values of the fields.</p> <p>Whether or not the service returns the <i>responseHeaders</i> parameter depends on the success or failure of the service. In the case of failure, the point at which the failure occurs determines the presence of the <i>responseHeaders</i> parameter.</p>
<i>status</i>	<p><b>Document</b> Conditional. Status code from the request.</p>

*statusMessage*

**Document** Conditional. Description of the status code returned by the underlying transport.

---

# 32 Storage Folder

---

■ About the Storage Elements .....	880
■ Locking Considerations .....	881
■ Sample Flow Service for Checkpoint Restart .....	883
■ Summary of Elements in this Folder .....	884

You use the elements in the storage folder to create, close, and delete data stores in the Integration Server short-term store.

## About the Storage Elements

You use the elements in the storage folder to create, close, and delete data stores in the Integration Server short-term store. Integration Server uses the short-term store for information that needs to persist across server restarts. For example, if the Integration Server on which your flow service is executing becomes unavailable and then restarts, the flow service can check the state information in the short-term store and begin processing at the point where the flow service was interrupted. The short-term store exists as the IS\_DATASTORE table in an external database identified to Integration Server through the ISInternal functional alias.

When using the `pub.storage` services, keep in mind that the short-term store is not intended to be used as a general-purpose storage engine. Rather, it is primarily provided to support shared storage of application resources and transient data in an Integration Server clustered environment. Consequently, Software AG recommends that you do not use the short-term store to process high volumes, large data records, or to permanently archive records.

**Important:** These services are a tool for maintaining state information in the short-term store. It is up to the developer of the flow service to make sure the flow service keeps track of its state and correctly handles restarts.

In Release 7.1, the Integration Server 6.1 Repository Server was replaced by a set of database tables collectively called IS Internal. During Integration Server installation, you can choose to use the embedded IS Internal database, or you can choose to use an external RDBMS in which you have created or will create the IS Internal database component. If you choose the external RDBMS, data associated with the `pub.storage` services will be stored in the IS\_DATASTORE table in the IS Internal database component. For DB2, the size of a BLOB column is defined when the table is created; you might find that the VALUE column in the IS\_DATASTORE table is not wide enough to accommodate your `pub.storage` data. If you have not yet created the IS Internal database component, open the appropriate table creation script below in a text editor and modify the width of the VALUE column in the IS\_DATASTORE table:

- `Software AG_directory\common\db\scripts\db2\isinternal\version \create\db2_isi_c_ddl.sql`
- `Software AG_directory\common\db\scripts\db2as400\isinternal\version \create\db2as400_isi_c_ddl.sql`

Where *version* is the directory that contains the latest version of the external RDBMS.

**Tip:** The directory with the highest number corresponds to the latest version of the external RDBMS.

If you have already created the IS Internal database component and the VALUE column is not wide enough to accommodate your pub.storage data, use DB2 commands to modify the width of the VALUE column in the IS\_DATASTORE table.

## Locking Considerations

---

The following sections describe in general how the pub.storage services handle locking requests. See the individual service descriptions for more detailed information.

### Entry Locking

To maintain data integrity, the short-term store uses locking to ensure that multiple threads do not modify the same entry at the same time. For insertions and removals, the short-term store sets and releases the lock. For updates, the client must set and release the lock. Using locking improperly, that is, creating a lock but not releasing it, can cause deadlocks in the short-term store.

The following guidelines can help you avoid short-term store deadlock:

- Release locks in the thread through which they were set. In other words, you cannot set a lock in one thread and release it in another. The safest way to do this is to release each lock in the flow service that acquired it.
- Unlock entries before the flow completes. Entries remain locked until released via a put ([pub.storage:put](#)) or an explicit unlock ([pub.storage:unlock](#)). To accomplish this, always pair a call to [pub.storage:get](#) or [pub.storage:lock](#) with a call to [pub.storage:put](#) or [pub.storage:unlock](#) so that every lock is followed by an unlock. In addition, use a Try-Catch pattern in your flow service so that an exception does not prevent the flow service from continuing and releasing the lock.
- Set limits on how long the threads will wait for a lock to be set or released. By setting finite limits, you allow the pub.storage service to release locks after a set amount of time and thereby avoid a deadlock situation. For more information, see "[Wait Time and Duration](#)" on page 882.

Following these guidelines might not be sufficient in some situations. For example, an Integration Server or hardware crash might result in a prematurely terminated flow service, thereby leaving an outstanding lock on the pub.storage service. Or, a client flow service might hang while requesting a lock. In these situations, having limits on how long a lock can exist (*duration*) or how long a lock request will wait (*wait time*) can prevent an application deadlock while using pub.storage services. For more information about lock duration and wait time, see "[Wait Time and Duration](#)" on page 882.

### Data Store Locking

When a pub.storage service locks an entry, the service also implicitly locks the data store in which the entry resides. This behavior prevents another thread from deleting the

---

entire data store and the entries it contains while your thread is working with the entry. When the locked entry is unlocked, the implicit lock on the data store is also released.

Be careful when explicitly unlocking data stores. Consider the following example:

1. User\_A locks an item. This creates two locks: an explicit lock on the entry, and an implicit lock on the data store.
2. User\_A later unlocks the data store explicitly while still holding the lock on the entry.
3. User\_B locks, then deletes the data store, including the entry locked by User\_A in the first step.

When User\_A explicitly unlocked the data store in step 2, User\_B was able to delete the entry the User\_A was working with.

## Automatic Promotion to Exclusive Lock

If a `pub.storage` service tries to acquire an exclusive lock on an object, but finds a shared lock from the same thread already in place on the object, the service will try to promote the lock to an exclusive lock.

If a `pub.storage` service that requires an exclusive lock encounters a shared or exclusive lock held by another thread, it will wait until the object becomes available. If the object remains locked for the period specified by the `waitlength` parameter passed by the service, or the value configured on the `watt.server.storage.lock.maxWait` property, the service will fail.

## Wait Time and Duration

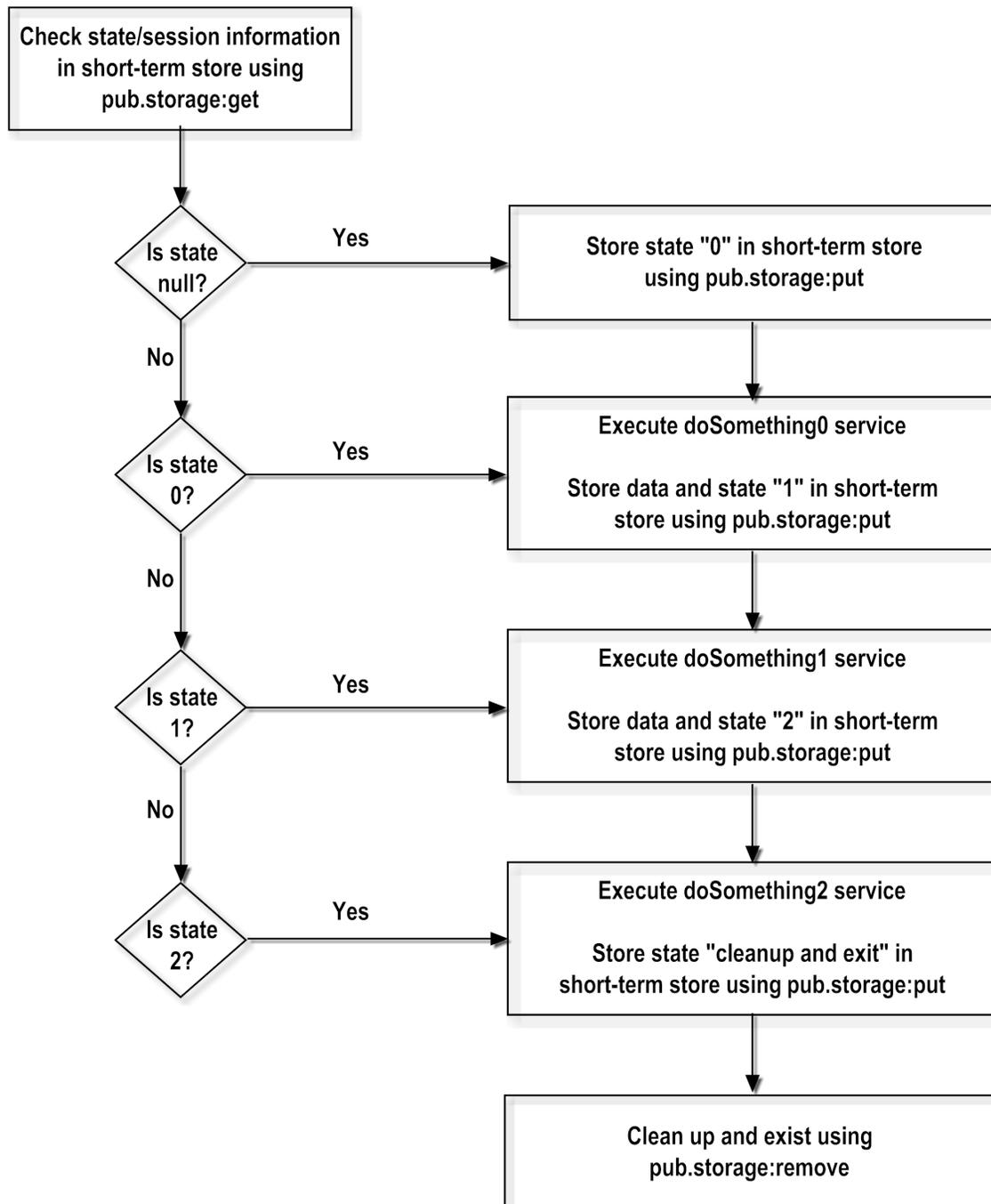
You can control how long Integration Server will wait to obtain a lock and how long it will hold a lock by using the following server properties:

- You can change the lock wait by using the `watt.server.storage.lock.maxWait` property from the **Settings > Extended Settings** screen in Integration Server Administrator. By default, a lock request will wait 240000 milliseconds (4 minutes) to obtain a lock. If a `pub.storage` service specifies a lock wait through the `waitlength` parameter, Integration Server uses this value instead of the value specified on the `watt.server.storage.lock.maxWait` property.
- You can change the lock duration by using the `watt.server.storage.lock.maxDuration` property from the **Settings > Extended Settings** screen in Integration Server Administrator. By default, a lock can exist for 180000 milliseconds (3 minutes). After 3 minutes, the server forcibly releases the lock.

## Sample Flow Service for Checkpoint Restart

---

The following diagram shows how to code checkpoint restart into your services. The following diagram explains the logic of a flow and shows where the various `pub.storage` services are used to achieve checkpoint restart.

**Logic to achieve checkpoint restart**

## Summary of Elements in this Folder

The following elements are available in this folder:

Element	Package and Description
<a href="#">pub.storage:add</a>	WmPublic. Inserts a new entry into a data store.
<a href="#">pub.storage:closeStore</a>	WmPublic. <i>Obsolete</i> – Closes a data store and unregisters the data store with the server.
<a href="#">pub.storage:deleteStore</a>	WmPublic. Deletes a data store and all its contents. Any data in the data store is deleted. If the data store does not exist, the service takes no action.
<a href="#">pub.storage:get</a>	WmPublic. Retrieves a value from a data store and locks the entry and the data store on behalf of the thread that invoked the service.
<a href="#">pub.storage:keys</a>	WmPublic. Obtains a list of all the keys in a data store.
<a href="#">pub.storage:listLocks</a>	WmPublic. Lists all pub.storage locks held by the supplied lock holder or target. If no input is supplied, the service returns a list of all pub.storage locks.
<a href="#">pub.storage:lock</a>	WmPublic. Locks an entry and/or data store on behalf of the thread invoking this service.
<a href="#">pub.storage:put</a>	WmPublic. Inserts or updates an entry in a data store. If the key does not exist in the data store, the entry is inserted.
<a href="#">pub.storage:registerStore</a>	WmPublic. <i>Obsolete</i> – Opens or creates a data store and registers the store with the server.
<a href="#">pub.storage:releaseLocks</a>	WmPublic. Releases all pub.storage locks held by the identified lock holders and ids. If both holders and ids are specified, the service ignores the holders and uses ids.
<a href="#">pub.storage:remove</a>	WmPublic. Removes an entry from a data store.
<a href="#">pub.storage:shutdown</a>	WmPublic. Releases internal resources used by the pub.storage services. This service is run automatically when the WmPublic package is unloaded and should <i>not</i> be explicitly invoked by a client.

Element	Package and Description
<a href="#">pub.storage:startup</a>	WmPublic. Performs initialization of internal facilities used by the <code>pub.storage</code> services. This service is run automatically when the WmPublic package is loaded and should <i>not</i> be explicitly invoked by a client.
<a href="#">pub.storage:unlock</a>	WmPublic. Unlocks an entry or a data store.

## pub.storage:add

WmPublic. Inserts a new entry into a data store.

If the key already exists in the data store, the `pub.storage:add` service does nothing.

### Input Parameters

<i>storeName</i>	<b>String</b> Name of the data store in which to insert the entry.
<i>key</i>	<b>String</b> Key under which the entry is to be inserted.
<i>value</i>	<b>Document</b> Value (IData object) to be inserted.

### Output Parameters

<i>result</i>	<p><b>String</b> Flag indicating whether the entry was successfully added. A value of:</p> <ul style="list-style-type: none"> <li>■ <code>true</code> indicates that the new entry was inserted successfully.</li> <li>■ <code>false</code> indicates that the entry was not inserted (usually because an entry for <i>key</i> already exists).</li> </ul>
<i>error</i>	<b>String</b> Error message generated while inserting the new entry into the data store.

## pub.storage:closeStore

WmPublic. *Obsolete* – Closes a data store and unregisters the data store with the server.

If the data store is not registered with the server, an exception will be thrown. A data store cannot be accessed after it has been unregistered. If you want to access the data in the data store, you need to register the data store again using [pub.storage:registerStore](#).

### Input Parameters

---

*storeName*                    **String** Name of the data store to close and unregister.

### Output Parameters

---

None.

### Usage Notes

This service is obsolete. When the repository was removed for Integration Server version 7.1.2, this service became a NOP (no operation).

---

## pub.storage:deleteStore

WmPublic. Deletes a data store and all its contents. Any data in the data store is deleted. If the data store does not exist, the service takes no action.

### Input Parameters

---

*storeName*                    **String** Name of the data store to delete.

*waitLength*                    **String** Optional. Length of time, in milliseconds, that you want to wait for this data store to become available for deletion if it is already locked by another thread. The default is the default Maximum Lock Wait value, which is specified on the `watt.server.storage.lock.maxWait` property. You can update this property by using the **Settings > Extended Settings** screen on the Integration Server Administrator.

### Output Parameters

---

*count*                            **String** Number of data store entries that were deleted. If the store does not exist, this value is 0.

### Usage Notes

This service obtains an exclusive lock on the data store, but no locks on the individual entries in the data store.

If this service finds a shared lock from the same thread on the data store, the service will automatically promote the lock to an exclusive lock.

The exclusive lock prevents other threads from acquiring locks on the data store or entries within the data store during the delete operation.

---

## pub.storage:get

WmPublic. Retrieves a value from a data store and locks the entry and the data store on behalf of the thread that invoked the service.

**Important:** This service does not automatically release the lock on the data store or entry after performing the get operation, so you need to make sure the lock is released by calling the [pub.storage:put](#) or [pub.storage:unlock](#) service. If you do not release the lock, other threads will not be able to access the resource until Integration Server automatically releases the lock after the amount of time specified on the `watt.server.storage.lock.maxDuration` property has passed.

---

### Input Parameters

<i>storeName</i>	<b>String</b> Name of the data store from which you want to retrieve the entry.
<i>key</i>	<b>String</b> Key of the entry whose value you want to retrieve.
<i>waitLength</i>	<b>String</b> Optional. Length of time, in milliseconds, that you want to wait for this entry to become available if it is already locked by another thread. The default is the default Maximum Lock Wait value, which is specified on the <code>watt.server.storage.lock.maxWait</code> property. You can update this property by using the <b>Settings &gt; Extended Settings</b> screen on the Integration Server Administrator.
<i>lockMode</i>	<b>String</b> Optional. Type of lock you want to place on the entry. Set to: <ul style="list-style-type: none"><li>■ <b>Exclusive</b> to prevent other threads from reading or updating the entry while you are using it. The service also obtains a shared lock on the data store. An exclusive lock on an entry allows you to modify the entry.</li><li>■ <b>Read</b> is obsolete. If this value is specified, the service obtains a shared lock.</li><li>■ <b>Share</b> to prevent other threads from obtaining an exclusive lock on the entry. The service also obtains a shared lock on the data store. A shared lock on an entry allows you to read, but not modify, the entry. This is the default.</li></ul>

## Output Parameters

---

*value*                      **Document** Retrieved entry (IData object). If the requested entry does not exist, the value of this parameter is null.

## Usage Notes

If you request an exclusive lock and the service finds a shared lock from the same thread on the entry, the service will automatically promote the shared lock on the entry to an exclusive lock.

When this service locks an entry, it also acquires a shared lock on the associated data store to prevent another thread from deleting the data store, and the entries it contains, while your thread has the entry locked.

When storing and retrieving the flow state in the short-term store for checkpoint restart purposes, be sure the value of *key* is unique to the transaction.

---

## pub.storage:keys

WmPublic. Obtains a list of all the keys in a data store.

## Input Parameters

---

*storeName*                      **String** Name of the data store from which you want to obtain a list of keys.

## Output Parameters

---

*keys*                              **String List** Keys for the data store specified in *storeName* .

---

## pub.storage:listLocks

WmPublic. Lists all pub.storage locks held by the supplied lock holder or target. If no input is supplied, the service returns a list of all pub.storage locks.

## Input Parameters

---

*holder*                              **String** Optional. Identifies the holder whose pub.storage locks are to be listed. The format is "\_DataStore\_<sessionId:threadId>", where:

- *sessionId* is a unique, internally generated identifier for the client's session in Integration Server.
- *threadId* is a unique, internally generated identifier for the client's thread in Integration Server.

*target* **String** Optional. Identifies the target whose pub.storage locks are to be listed.

## Output Parameters

*locks* **Document List** The list of pub.storage locks. This output variable can be null.

Value	Description
<i>id</i>	<b>String</b> The internal ID of the lock
<i>target</i>	<b>String</b> Item that is locked, specified as a data store name or the key for an entry
<i>holder</i>	<b>String</b> Holder of the lock. This value is generated internally by the pub.storage services.
<i>type</i>	<b>String</b> "EXCLUSIVE" or "SHARE"
<i>count</i>	<b>String</b> Number of lock holders sharing this lock
<i>time</i>	<b>String</b> The time the lock was created.

## pub.storage:lock

WmPublic. Locks an entry and/or data store on behalf of the thread invoking this service.

**Important:** When you lock an entry or data store using this service, you must release the lock by using a put ([pub.storage:put](#)) or an explicit unlock ([pub.storage:unlock](#)). If you do not release the lock, other threads will not be able to access the resource until Integration Server automatically releases the lock after the amount of time specified on the `watt.server.storage.lock.maxDuration` parameter has passed.

**Important:** Be careful when releasing locks with the `pub.storage:unlock` service. If you release a lock on a data store, another thread can obtain a lock on the data store and delete it, and the entries it contains, even if your thread still has locks on one or more of the entries.

## Input Parameters

---

<i>storeName</i>	<b>String</b> Name of the data store containing the entry.
<i>key</i>	<b>String</b> Optional. Key of the entry that you want to lock.  If <i>key</i> is not supplied and you request: <ul style="list-style-type: none"><li>■ A shared lock, the service obtains a shared lock on the data store, allowing other threads to read and modify entries, but not to delete them.</li><li>■ An exclusive lock, the service obtains an exclusive lock on the data store, preventing other threads from locking the data store and the entries, thereby preventing those threads from reading, modifying, or deleting the entries or the data store.</li></ul> If both <i>storeName</i> and <i>key</i> are specified and you request: <ul style="list-style-type: none"><li>■ A shared lock, the service obtains a shared lock on the data store and the entry.</li><li>■ An exclusive lock, the service obtains a shared lock on the data store and an exclusive lock on the entry.</li></ul>
<i>waitLength</i>	<b>String</b> Optional. Length of time, in milliseconds, that you want to wait for this entry to become available if it is already locked by another thread. The default is the default Maximum Lock Wait value, which is specified on the <code>watt.server.storage.lock.maxWait</code> property. You can update this property by using the <b>Settings &gt; Extended Settings</b> screen on the Integration Server Administrator.
<i>lockMode</i>	<b>String</b> Optional. Type of lock you want to place on the entry or data store. Set to: <ul style="list-style-type: none"><li>■ <b>Exclusive</b> to prevent other threads from obtaining a lock on the data store or entry.  An exclusive lock on an entry allows you to modify the entry, and prevents other threads from reading or modifying the entry.</li></ul>

An exclusive lock on a data store also locks the entries in the data store. In addition, an exclusive lock on a data store allows you to delete the data store.

- `Read` is obsolete. If this value is specified, the service obtains a shared lock.
- `Share` to prevent other threads from obtaining an exclusive lock on an entry or a data store. A shared lock on an entry allows you to read, but not modify, the entry. A shared lock on a data store prevents another thread from deleting the data store. This is the default.

### Output Parameters

---

None.

### Usage Notes

If you have not specified a *key*, and your flow service does not invoke `pub.storage:put` or `pub.storage:unlock`, or your service throws an exception before invoking `pub.storage:put` or `pub.storage:unlock`, the entire data store remains locked until the amount of time specified on the `watt.server.storage.lock.maxDuration` parameter has passed.

If the key does not exist in the data store at the time your flow service executes, the `pub.storage:lock` service is a NOP (no operation). Set the `watt.server.storage.addKeyToStoreIfNotPresent` parameter to true if you want the `pub.storage:lock` service to add the specified key to the data store if the key does not exist at the time the service executes. When the `watt.server.storage.addKeyToStoreIfNotPresent` parameter is set to true, the `pub.storage:lock` service creates the specified key, assigns it a NULL value, and then locks the entry in the data store.

If you request an exclusive lock on an entry, the service obtains an exclusive lock on the entry and a shared lock on the data store. If this service finds a shared lock from the same thread on the entry, the service will automatically promote the shared lock on the entry to an exclusive lock.

If you request a shared lock on an entry, the service obtains a shared lock on the entry and a shared lock on the data store.

If you request a shared lock on an entry or a data store and this service finds an exclusive lock from the same thread, the existing exclusive lock will be reused. The exclusive lock will not be demoted to a shared lock.

If you request an exclusive lock on a data store, and this service finds a shared lock from the same thread on the data store, the service will automatically promote the shared lock on the data store to an exclusive lock.

---

## pub.storage:put

WmPublic. Inserts or updates an entry in a data store. If the key does not exist in the data store, the entry is inserted.

If the requested entry is not currently locked by the thread that invoked this service, the `pub.storage:put` service will automatically attempt to lock the entry for the duration of the put operation.

The service obtains an exclusive lock on the entry and a shared lock on the data store. If the service finds a shared lock from the same thread on the entry, the service will automatically promote the shared lock to an exclusive lock.

This service releases the lock when the put operation has completed.

### Input Parameters

---

*storeName*                    **String** Name of the data store into which you want to insert or update the entry.

*value*                        **Document** Value (IData object) to be inserted or updated.

*waitLength*                **String** Optional. Length of time, in milliseconds, that you want to wait for this entry to become available if it is already locked by another thread. If the wait length expires before a lock is obtained, the service fails and throws an exception. The default is the default Maximum Lock Wait value, which is specified on the `watt.server.storage.lock.maxWait` property. You can update this property by using the **Settings > Extended Settings** screen on the Integration Server Administrator.

This parameter is used only when your service did not explicitly lock the entry beforehand.

*key*                         **String** Key where you want to insert or update the entry.

### Output Parameters

---

*error*                        **String** Error message generated while inserting the new entry into the data store.

### Usage Notes

When storing and retrieving the flow state in the short-term store for checkpoint restart purposes, be sure the value of *key* is unique to the transaction.

---

## pub.storage:registerStore

WmPublic. *Obsolete* – Opens or creates a data store and registers the store with the server.

A data store must be registered before it can be accessed. If the store is already registered with the server, this service does nothing.

### Input Parameters

---

*storeName*                      **String** Name of the data store to register.

### Output Parameters

---

None.

---

## pub.storage:releaseLocks

WmPublic. Releases all pub.storage locks held by the identified lock holders and ids. If both holders and ids are specified, the service ignores the holders and uses IDs.

This service is intended primarily for administrators. It is most useful when used in combination with pub.storage:listLocks. You can map the locks/holder string list from that service to the holders input variable in this service or the locks/id string list to the ids input variables. If neither ids nor holders are supplied, no locks are released.

**Important:** Use this service with care. It will release locks held by active threads and could cause their processing to fail. In addition, if you release a lock on a data store, another thread can obtain a lock on the data store and delete it, and the entries it contains, even if the original thread still has locks on one or more of the entries.

### Input Parameters

---

*holders*                      **String List** Optional. Holders whose pub.storage locks are to be released.

*ids*                              **String** Optional. Ids whose pub.storage locks are to be released.

### Output Parameters

---

*count*                              **String List** Number of locks that were released.

---

## pub.storage:remove

WmPublic. Removes an entry from a data store. This service obtains an exclusive lock on the entry and a shared lock on the data store.

### Input Parameters

---

<i>storeName</i>	<b>String</b> Name of the data store from which to remove an entry.
<i>key</i>	<b>String</b> Key of the entry that you want to remove.
<i>waitLength</i>	<b>String</b> Optional. Length of time, in milliseconds, that you want to wait for this entry to become available for deletion if it is already locked by another thread. The default is the default Maximum Lock Wait value, which is specified on the <code>watt.server.storage.lock.maxWait</code> property. You can update this property by using the <b>Settings &gt; Extended Settings</b> screen on the Integration Server Administrator.

### Output Parameters

---

<i>result</i>	<b>String</b> Flag indicating whether the entry was successfully removed. A value of: <ul style="list-style-type: none"><li>■ <code>true</code> indicates that the entry was removed successfully.</li><li>■ <code>false</code> indicates that the entry was not removed (usually because an entry for key does not exist).</li></ul>
---------------	---

---

## pub.storage:shutdown

WmPublic. Releases internal resources used by the `pub.storage` services. This service is run automatically when the WmPublic package is unloaded and should *not* be explicitly invoked by a client.

---

## pub.storage:startup

WmPublic. Performs initialization of internal facilities used by the `pub.storage` services. This service is run automatically when the WmPublic package is loaded and should *not* be explicitly invoked by a client.

## pub.storage:unlock

WmPublic. Unlocks an entry or a data store.

When a flow service retrieves an entry using the [pub.storage:get](#) service, the entry is locked to prevent modification by other users before the flow completes. The entry remains locked until the lock owner invokes a [pub.storage:put](#) service. To unlock a service without using the [pub.storage:put](#) service, use the [pub.storage:unlock](#) service.

In addition, if a flow service uses the [pub.storage:lock](#) service to lock an entry or data store, you must use the [pub.storage:unlock](#) or [pub.storage:put](#) service to release the lock.

**Important:** Be careful when releasing locks with this service. If you release a lock on a data store, another thread can obtain a lock on the data store and delete it, and the entries it contains, even if the original thread still has locks on one or more of the entries.

### Input Parameters

---

<i>storeName</i>	<b>String</b> Name of the data store in which to unlock an entry.
<i>key</i>	<b>String</b> Optional. Key of the entry that you want to unlock. If <i>key</i> is not supplied, the lock will be removed from the data store specified in <i>storeName</i> , but any locks on entries in the data store will remain.

### Output Parameters

---

None.

# 33 String Folder

---

■ Summary of Elements in this Folder .....	898
--	-----

You use the elements in the string folder to perform string manipulation and substitution operations.

## Summary of Elements in this Folder

The following elements are available in this folder:

Element	Package and Description
<a href="#">pub.string:base64Decode</a>	WmPublic. Decodes a Base-64 encoded string into a sequence of bytes.
<a href="#">pub.string:base64Encode</a>	WmPublic. Converts a sequence of bytes into a Base64-encoded String.
<a href="#">pub.string:bytesToString</a>	WmPublic. Converts a sequence of bytes to a String.
<a href="#">pub.string:compareStrings</a>	WmPublic. Performs a case-sensitive comparison of two strings, and indicates whether the strings are identical.
<a href="#">pub.string:concat</a>	WmPublic. Concatenates two strings.
<a href="#">pub.string:HTMLDecode</a>	WmPublic. Replaces HTML character entities with native characters.
<a href="#">pub.string:HTMLEncode</a>	WmPublic. Replaces HTML-sensitive characters with equivalent HTML character entities.
<a href="#">pub.string:indexOf</a>	WmPublic. Returns the index of the first occurrence of a sequence of characters in a string.
<a href="#">pub.string:isAlphanumeric</a>	WmPublic. Determines whether a string consists entirely of alphanumeric characters (in the ranges A-Z, a-z, or 0-9).
<a href="#">pub.string:isDate</a>	WmPublic. Determines whether a string follows a specified date pattern.
<a href="#">pub.string:isNullOrBlank</a>	WmPublic. Checks a string for a null or a blank value.

Element	Package and Description
<a href="#">pub.string:isNumber</a>	WmPublic. Determines whether the contents of a string can be converted to a float value.
<a href="#">pub.string:length</a>	WmPublic. Returns the length of a string.
<a href="#">pub.string:lookupDictionary</a>	WmPublic. Looks up a given key in a hash table and returns the string to which that key is mapped.
<a href="#">pub.string:lookupTable</a>	WmPublic. Locates a key in a String Table and returns the string to which that key is mapped.
<a href="#">pub.string:makeString</a>	WmPublic. Builds a single string by concatenating the elements of a String List.
<a href="#">pub.string:messageFormat</a>	WmPublic. Formats an array of strings into a given message pattern.
<a href="#">pub.string:numericFormat</a>	WmPublic. Formats a number into a given numeric pattern.
<a href="#">pub.string:objectToString</a>	WmPublic. Converts an object to string representation using the Java toString() method of the object.
<a href="#">pub.string:padLeft</a>	WmPublic. Pads a string to a specified length by adding pad characters to the beginning of the string.
<a href="#">pub.string:padRight</a>	WmPublic. Pads a string to a specified length by adding pad characters to the end of the string.
<a href="#">pub.string:replace</a>	WmPublic. Replaces all occurrences of a specified substring with a substitute string.
<a href="#">pub.string:stringToBytes</a>	WmPublic. Converts a string to a byte array.
<a href="#">pub.string:substitutePipelineVariables</a>	WmPublic. Replaces a pipeline variable with its corresponding value.
<a href="#">pub.string:substring</a>	WmPublic. Returns a substring of a given string.

Element	Package and Description
<a href="#">pub.string:tokenize</a>	WmPublic. Tokenizes a string using specified delimiter characters and generates a String List from the resulting tokens.
<a href="#">pub.string:toLower</a>	WmPublic. Converts all characters in a given string to lowercase.
<a href="#">pub.string:toUpper</a>	WmPublic. Converts all characters in a given string to uppercase.
<a href="#">pub.string:trim</a>	WmPublic. Trims leading and trailing white space from a given string.
<a href="#">pub.string:URLDecode</a>	WmPublic. Decodes a URL-encoded string.
<a href="#">pub.string:URLEncode</a>	WmPublic. URL-encodes a string.

## pub.string:base64Decode

WmPublic. Decodes a Base-64 encoded string into a sequence of bytes.

### Input Parameters

*string*                      **String** A Base64-encoded String to decode into bytes.

### Output Parameters

*value*                      **byte[ ]** The sequence of bytes decoded from the Base64-encoded String.

*encoding*                      **String** Optional. Specifies the encoding method. Default value is ASCII.

## pub.string:base64Encode

WmPublic. Converts a sequence of bytes into a Base64-encoded String.

---

## Input Parameters

---

<i>bytes</i>	<b>byte[ ]</b> Sequence of bytes to encode into a Base64-encoded String.
<i>useNewLine</i>	<b>String</b> Optional. Flag indicating whether to retain or remove the line breaks. Set to: <ul style="list-style-type: none"> <li>■ <code>true</code> to retain the line breaks. This is the default.</li> <li>■ <code>false</code> to remove the line breaks.</li> </ul>
<i>encoding</i>	<b>String</b> Optional. Specifies the encoding method. Default value is <code>ASCII</code> .

---

## Output Parameters

---

<i>value</i>	<b>String</b> Base64-encoded String encoded from the sequence of bytes.
--------------	---

### Usage Notes

By default, the `pub.string:base64Encode` service inserts line breaks after 76 characters of data, which is not the canonical lexical form expected by implementations such as MTOM. You can use the `useNewLine` parameter to remove the line breaks. For more information about MTOM implementations, refer to *Web Services Developer's Guide*.

---

## pub.string:bytesToString

WmPublic. Converts a sequence of bytes to a String.

---

### Input Parameters

---

<i>bytes</i>	<b>byte[ ]</b> Sequence of bytes to convert to a String.
<i>encoding</i>	<b>String</b> Optional. Name of a registered, IANA character set (for example, <code>ISO-8859-1</code> ). If you specify an unsupported encoding, the system throws an exception. <p>To use the default encoding, set <code>encoding</code> to <code>autoDetect</code>.</p>
<i>ignoreBOMChars</i>	<b>String</b> Optional. Flag indicating whether or not Integration Server removes the byte order mark (BOM) characters in the

input sequence of bytes before converting the byte array to string. Set to:

- `true` to remove the byte order mark (BOM) characters before converting the input sequence of bytes to string, if the byte array contains BOM characters.
- `false` to include the byte order mark (BOM) characters while converting the input sequence of bytes to string. The default is `false`.

### Output Parameters

*string*                      **String** String representation of the contents of *bytes* .

## pub.string:compareStrings

WmPublic. Performs a case-sensitive comparison of two strings, and indicates whether the strings are identical.

### Input Parameters

*inString1*                      **String** Optional. String to compare against *inString2* . This input variable can be null.

*inString2*                      **String** Optional. String to compare against *inString1* . This input variable can be null.

### Output Parameters

*isEqual*                      **String** Indicates whether or not *inString1* and *inString2* are identical.

- `true` indicates that *inString1* and *inString2* are identical.
- `false` indicates that *inString1* and *inString2* are not identical.

**Note:** If both *inString1* and *inString2* are null, the service considers the strings to be identical and returns `true`.

## pub.string:concat

WmPublic. Concatenates two strings.

---

### Input Parameters

<i>inString1</i>	<b>String</b> String to which you want to concatenate another string.
<i>inString2</i>	<b>String</b> String to concatenate to <i>inString1</i> .

---

### Output Parameters

<i>value</i>	<b>String</b> Result of concatenating <i>inString1</i> with <i>inString2</i> ( <i>inString1</i> + <i>inString2</i> ).
--------------	---

---

## pub.string:HTMLDecode

WmPublic. Replaces HTML character entities with native characters.

Specifically, the service:

<u>Replaces this HTML character entity...</u>	<u>With...</u>
&gt;	>
&lt;	<
&amp;	&
&quot;	"

---

### Input Parameters

<i>inString</i>	<b>String</b> An HTML-encoded String.
-----------------	---------------------------------------

---

### Output Parameters

<i>value</i>	<b>String</b> Result from decoding the contents of <i>inString</i> . Any HTML character entities that existed in <i>inString</i> will appear as native characters in <i>value</i> .
--------------	---

---

## pub.string:HTMLEncode

WmPublic. Replaces HTML-sensitive characters with equivalent HTML character entities.

Specifically, this service:

<u>Replaces this native language character...</u>	<u>With...</u>
>	&gt;
<	&lt;
&	&amp;
"	&quot;
'	&#39

These translations are useful when displaying text in an HTML context.

### Input Parameters

*inString*                      **String** The character you want to encode in HTML.

### Output Parameters

*value*                              **String** Result from encoding the contents of *inString*. Any HTML-sensitive characters that existed in *inString* (for example, > or &) will appear as the equivalent HTML character entities in *value*.

---

## pub.string:indexOf

WmPublic. Returns the index of the first occurrence of a sequence of characters in a string.

---

### Input Parameters

---

<i>inString</i>	<b>String</b> String in which you want to locate a sequence of characters.
<i>subString</i>	<b>String</b> Sequence of characters to locate.
<i>fromIndex</i>	<b>String</b> Optional. Index of <i>inString</i> from which to start the search. If no value is specified, this parameter contains 0 to indicate the beginning of the string.

### Output Parameters

---

<i>value</i>	<b>String</b> Index of the first occurrence of <i>subString</i> in <i>inString</i> . If no occurrence is found, this parameter contains -1.
--------------	---

---

## pub.string:isAlphanumeric

WmPublic. Determines whether a string consists entirely of alphanumeric characters (in the ranges A–Z, a–z, or 0–9).

### Input Parameters

---

<i>inString</i>	<b>String</b> Optional. String to be checked for alphanumeric characters.
-----------------	---

### Output Parameters

---

<i>isAlphanumeric</i>	<b>String</b> Indicates whether or not all the characters in <i>inString</i> are alphanumeric. <ul style="list-style-type: none"><li>■ <code>true</code> indicates that all the characters in <i>inString</i> are alphanumeric.</li><li>■ <code>false</code> indicates that <i>not all</i> the characters in <i>inString</i> are alphanumeric.</li></ul> The service returns <code>false</code> if <i>inString</i> is not specified.
-----------------------	--

---

## pub.string:isDate

WmPublic. Determines whether a string follows a specified date pattern.

### Input Parameters

---

<i>inString</i>	<b>String</b> Optional. String to be checked for adherence to the specified date <i>pattern</i> .
<i>pattern</i>	<b>String</b> Date format for specifying the <i>inString</i> parameter (for example, yyyyMMdd HH:mm:ss.SSS).  For more information about the pattern strings that can be specified for the date, see " <a href="#">Pattern String Symbols</a> " on page 190.

### Output Parameters

---

<i>isDate</i>	<b>String</b> Indicates whether or not <i>inString</i> follows the specified date pattern. <ul style="list-style-type: none"><li>■ <code>true</code> indicates that <i>inString</i> follows the specified date pattern.</li><li>■ <code>false</code> indicates that <i>inString</i> does not follow the specified date pattern.</li></ul> The service returns <code>false</code> if <i>inString</i> is not specified.
---------------	---

### Usage Notes

The service returns an error if both *inString* and *pattern* are not specified.

You can specify any random string (for example, 111212) as both *inString* and *pattern* . The service returns `true` if the same user-defined string is specified as both *inString* and *pattern* . This is because the `java.text.SimpleDateFormat` class parses the user-defined input string and pattern to a valid date when the particular input values are identical.

---

## pub.string:isNullOrBlank

WmPublic. Checks a string for a null or a blank value.

---

### Input Parameters

---

*inString* **String** Optional. String to be checked for a null or a blank value.

### Output Parameters

---

*isNullorBlank* **String** Indicates whether or not *inString* has a null or a blank value.

- `true` indicates that *inString* has either a null or a blank value.
- `false` indicates that *inString* contains a value that is not null.

**Note:** If *inString* is not specified, the service considers the string to be blank and returns `true`.

---

## pub.string:isNumber

WmPublic. Determines whether the contents of a string can be converted to a float value.

### Input Parameters

---

*inString* **String** Optional. String to be checked for conversion to float.

### Output Parameters

---

*isNumber* **String** Indicates whether or not *inString* can be converted to a float value.

- `true` indicates that *inString* can be converted to a float value.
- `false` indicates that *inString* cannot be converted to a float value.

The service returns `false` if *inString* is not specified.

---

## pub.string:length

WmPublic. Returns the length of a string.

---

### Input Parameters

---

*inString*                    **String** String whose length you want to discover.

### Output Parameters

---

*value*                        **String** The number of characters in *inString* .

---

## pub.string:lookupDictionary

WmPublic. Looks up a given key in a hash table and returns the string to which that key is mapped.

### Input Parameters

---

*hashtable*                    **java.util.Hashtable** Hash table that uses String objects for keys and values.

*key*                            **String** Key in *hashtable* whose value you want to retrieve.

**Note:** The key is case sensitive.

### Output Parameters

---

*value*                        **String** Value of the string to which *key* is mapped. If the requested key in *hashtable* is null or if *key* is not mapped to any value in *hashtable* , the service returns null.

---

## pub.string:lookupTable

WmPublic. Locates a key in a String Table and returns the string to which that key is mapped.

### Input Parameters

---

*lookupTable*                    **String [ ] [ ]** A multi-row, multi-column string table in which to search.

---

<i>keyColumnIndex</i>	<b>String</b> Index of the "key" column. Default is 0.
<i>valueColumnIndex</i>	<b>String</b> Index of the "value" column. Default is 1.
<i>key</i>	<b>String</b> Key to locate.  <b>Note:</b> The key is case sensitive.
<i>ignoreCase</i>	<b>String</b> Optional. Flag indicating whether to perform a case-sensitive or case-insensitive search. Set to: <ul style="list-style-type: none"><li>■ <code>true</code> to perform a case-insensitive search.</li><li>■ <code>false</code> to perform a case-sensitive search. This is the default.</li></ul>
<i>useRegex</i>	<b>String</b> Optional. Flag indicating whether the values in the table are to be interpreted as regular expressions.  <b>Note:</b> The regular expressions in the table should not include slashes. For example, use <code>hello.*</code> , not <code>/hello.*</code> .  Set to: <ul style="list-style-type: none"><li>■ <code>true</code> to interpret the key column values in the table as regular expressions.</li><li>■ <code>false</code> to interpret the key column values in the table as literal values (that is, not regular expressions). This is the default.</li></ul>

## Output Parameters

---

<i>value</i>	<b>String</b> First value in the "value" column whose key matches <i>key</i> . If no match is found, this parameter is null.
--------------	--

---

## pub.string:makeString

WmPublic. Builds a single string by concatenating the elements of a String List.

### Input Parameters

---

<i>elementList</i>	<b>String List</b> Strings to concatenate.
--------------------	--

---

*separator* **String** String to insert between each non-null element in *elementList* .

### Output Parameters

---

*value* **String** Result from concatenating the strings in *elementList* . Strings are separated by the characters specified in *separator* .

---

## pub.string:messageFormat

WmPublic. Formats an array of strings into a given message pattern.

### Input Parameters

---

*pattern* **String** Message that includes "placeholders" where elements from *argumentList* are to be inserted. The message can contain any sequence of characters. Use the {*n*} placeholder to insert elements from *argumentList* , where *n* is the index of the element that you want to insert. For example, the following pattern string inserts elements 0 and 1 into the message:

```
Test results: {0} items passed, {1} items failed.
```

**Note:** Do not use any characters except digits for *n* .

*argumentList* **String List** Optional. List of strings to use to populate *pattern* . If *argumentList* is not supplied, the service will not replace placeholders in *pattern* with actual values.

### Output Parameters

---

*value* **String** Result from substituting *argumentList* into *pattern* . If *pattern* is empty or null, this parameter is null.

---

## pub.string:numericFormat

WmPublic. Formats a number into a given numeric pattern.

**Note:** The rounding mode for `pub.string:numericFormat` service is `HALF_EVEN`. For more rounding modes, use the `pub.math:roundNumber` service.

## Input Parameters

---

*num* **String** The number to format.

*pattern* **String** A pattern string that describes the way in which *num* is to be formatted:

<u>This symbol...</u>	<u>Indicates...</u>
0	A digit.
#	A digit. Leading zeroes will not be shown.
.	A placeholder for a decimal separator.
,	A placeholder for a grouping separator.
;	A separation in format.
-	The default negative prefix.
%	That <i>num</i> will be multiplied by 100 and shown as a percentage.
x	Any character used as a prefix or suffix (for example, A, \$).
'	That special characters are to be used as literals in a prefix or suffix. Enclose the special characters within " (for example, '#').

The following are examples of pattern strings:

<u>Pattern</u>	<u>Description</u>
#,###	Use commas to separate into groups of three digits. The pound sign denotes a digit and the comma is a placeholder for the grouping separator.

<code>#,####</code>	Use commas to separate into groups of four digits.
<code>\$#.00</code>	Show digits before the decimal point as needed and exactly two digits after the decimal point. Prefix with the \$ character.
<code>'#'#.0</code>	Show digits before the decimal point as needed and exactly one digit after the decimal point. Prefix with the # character. The first character in a pattern is the dollar sign (\$). The pound sign denotes a digit and the period is a placeholder for decimal separator.

### Output Parameters

---

*value*      **String***num* formatted according to *pattern*. If *pattern* is an empty (not null) string, the default pattern of comma separators is used and the number of digits after the decimal point remains unchanged.

## pub.string:objectToString

---

WmPublic. Converts an object to string representation using the Java `toString()` method of the object.

### Input Parameters

---

*object*      **Object** The object to be converted to string representation.

### Output Parameters

---

*string*      **String** String representation of the input object converted using the Java `toString()` method of the object.

---

## pub.string:padLeft

WmPublic. Pads a string to a specified length by adding pad characters to the beginning of the string.

### Input Parameters

---

<i>inString</i>	<b>String</b> String that you want to pad.
<i>padString</i>	<b>String</b> Characters to use to pad <i>inString</i> .
<i>length</i>	<b>String</b> Total length of the resulting string, including pad characters.

### Output Parameters

---

<i>value</i>	<b>String</b> Contents of <i>inString</i> preceded by as many pad characters as needed so that the total length of the string equals <i>length</i> .
--------------	--

### Usage Notes

If *padString* is longer than one character and does not fit exactly into the resulting string, the beginning of *padString* is aligned with the beginning of the resulting string. For example, suppose *inString* equals `shipped` and *padString* equals `x9y`.

<b>If <i>length</i> equals...</b>	<b>Then <i>value</i> will contain...</b>
-----------------------------------	--

---

7	<code>shipped</code>
10	<code>x9yshipped</code>
12	<code>x9x9yshipped</code>

If *inString* is longer than *length* characters, only the last *length* characters from *inString* are returned. For example, if *inString* equals `acct1234` and *length* equals 4, *value* will contain `1234`.

---

## pub.string:padRight

WmPublic. Pads a string to a specified length by adding pad characters to the end of the string.

### Input Parameters

---

<i>inString</i>	<b>String</b> String that you want to pad.
<i>padString</i>	<b>String</b> Characters to use to pad <i>inString</i> .
<i>length</i>	<b>String</b> Total length of the resulting string, including pad characters.

### Output Parameters

---

<i>value</i>	<b>String</b> Contents of <i>inString</i> followed by as many pad characters as needed so that the total length of the string equals <i>length</i> .
--------------	--

### Usage Notes

If *padString* is longer than one character and does not fit exactly into the resulting string, the end of *padString* is aligned with the end of the resulting string. For example, suppose *inString* equals `shipped` and *padString* equals `x9y`.

<b>If <i>length</i> equals...</b>	<b>Then <i>value</i> will contain...</b>
7	<code>shipped</code>
10	<code>shippedx9y</code>
12	<code>shippedx9y9y</code>

If *inString* is longer than *length* characters, only the first *length* characters from *inString* are returned. For example, if *inString* equals `1234acct` and *length* equals 4, *value* will contain `1234`.

---

## pub.string:replace

WmPublic. Replaces all occurrences of a specified substring with a substitute string.

---

### Input Parameters

---

<i>inString</i>	<b>String</b> String containing the substring to replace.
<i>searchString</i>	<b>String</b> Substring to replace within <i>inString</i> .
<i>replaceString</i>	<b>String</b> Character sequence that will replace <i>searchString</i> . If this parameter is null or empty, the service removes all occurrences of <i>searchString</i> from <i>inString</i> .
<i>useRegex</i>	<b>String</b> Optional. Flag indicating whether <i>searchString</i> is a regular expression. When regular expressions are used to specify a search string, <i>replaceString</i> may also contain interpolation variables (for example, "\$1") that match parenthetical subexpressions in <i>searchString</i> .  Set to: <ul style="list-style-type: none"> <li>■ <code>true</code> to indicate that <i>searchString</i> is a regular expression.</li> <li>■ <code>false</code> to indicate that <i>searchString</i> is not a regular expression. This is the default.</li> </ul>

### Output Parameters

---

<i>value</i>	<b>String</b> Contents of <i>inString</i> with replacements made.
--------------	---

---

## pub.string:stringToBytes

WmPublic. Converts a string to a byte array.

### Input Parameters

---

<i>string</i>	<b>String</b> String to convert to a byte[ ].
<i>encoding</i>	<b>String</b> Optional. Name of a registered, IANA character set that specifies the encoding to use when converting the String to an array of bytes (for example: ISO-8859-1).  To use the default encoding, set this value to <code>autoDetect</code> . If you specify an unsupported encoding, an exception will be thrown.

---

## Output Parameters

---

*bytes*                      **byte[ ]** Contents of *string* represented as a byte[ ].

---

## pub.string:substitutePipelineVariables

WmPublic. Replaces a pipeline variable with its corresponding value.

---

### Input Parameters

---

*inString*                      **String** Optional. String containing the pipeline variable to replace. Specify the name of the pipeline variable between the % symbols (for example, %phone%).

---

### Output Parameters

---

*value*                              **String** Contents of *inString* with the pipeline variable replaced.

### Usage Notes

The service returns an error if *inString* is not specified.

If *inString* does not contain any variable between the % symbols, or contains a value other than the pipeline variable between the % symbols, the service does not perform any variable substitution from the pipeline.

If you want to include the % symbol in the output, you can specify it as \% in *inString*. To specify the value of the pipeline variable as a percentage in the output, append \% after the variable name in *inString*. For example, suppose a pipeline variable *revenueIncreasePercent* has a value of 100.

---

#### If *inString* equals...

---

%revenueIncreasePercent%\%

---

#### Then *value* will contain...

---

100%

The service cannot be used for substitution of global variables.

---

## pub.string:substring

WmPublic. Returns a substring of a given string.

### Input Parameters

---

<i>inString</i>	<b>String</b> String from which to extract a substring.
<i>beginIndex</i>	<b>String</b> Beginning index of the substring to extract (inclusive).
<i>endIndex</i>	<b>String</b> Ending index of the substring to extract (exclusive). If this parameter is null or empty, the substring will extend to the end of <i>inString</i> .

### Output Parameters

---

<i>value</i>	<b>String</b> Substring from <i>beginIndex</i> and extending to the character at <i>endIndex</i> - 1.
--------------	---

---

## pub.string:tokenize

WmPublic. Tokenizes a string using specified delimiter characters and generates a String List from the resulting tokens.

This service does not return delimiters as tokens.

### Input Parameters

---

<i>inString</i>	<b>String</b> String you want to tokenize (that is, break into delimited chunks).
<i>delim</i>	<b>String</b> Delimiter characters. If null or empty, the service uses the default delimiters <code>\t\n\r</code> , where t, n, and r represent the white space characters tab, new line, and carriage return).

### Output Parameters

---

<i>valueList</i>	<b>String List</b> Strings containing the tokens extracted from <i>inString</i> .
------------------	---

---

## pub.string:toLower

WmPublic. Converts all characters in a given string to lowercase.

### Input Parameters

---

<i>inString</i>	<b>String</b> String to convert.
<i>language</i>	<b>String</b> Optional. Lowercase, two-letter ISO-639 code. If this parameter is null, the system default is used.
<i>country</i>	<b>String</b> Optional. Uppercase, two-letter ISO-3166 code. If this parameter is null, the system default is used.
<i>variant</i>	<b>String</b> Optional. Vendor and browser-specific code. If null, this parameter is ignored.

### Output Parameters

---

<i>value</i>	<b>String</b> Contents of <i>inString</i> , with all uppercase characters converted to lowercase.
--------------	---

---

## pub.string:toUpper

WmPublic. Converts all characters in a given string to uppercase.

### Input Parameters

---

<i>inString</i>	<b>String</b> String to convert.
<i>language</i>	<b>String</b> Optional. Lowercase, two-letter ISO-639 code. If this parameter is null, the system default is used.
<i>country</i>	<b>String</b> Optional. Uppercase, two-letter ISO-3166 code. If this parameter is null, the system default is used.
<i>variant</i>	<b>String</b> Optional. Vendor and browser-specific code. If null, this parameter is ignored.

### Output Parameters

---

<i>value</i>	<b>String</b> Contents of <i>inString</i> , with all lowercase characters converted to uppercase.
--------------	---

---

## pub.string:trim

WmPublic. Trims leading and trailing white space from a given string.

### Input Parameters

---

*inString*                    **String** String to trim.

### Output Parameters

---

*value*                        **String** Contents of *inString* with white space trimmed from both ends.

---

## pub.string:URLDecode

WmPublic. Decodes a URL-encoded string.

### Input Parameters

---

*inString*                    **String** URL-encoded string to decode.

### Output Parameters

---

*value*                        **String** Result from decoding *inString*. If *inString* contained plus (+) signs, they will appear in *value* as spaces. If *inString* contained *%hex* encoded characters, they will appear in *value* as the appropriate native character.

---

## pub.string:URLEncode

WmPublic. URL-encodes a string.

Encodes characters the same way that data posted from a WWW form is encoded (that is, the `application/x-www-form-urlencoded` MIME type).

### Input Parameters

---

*inString*                    **String** String to URL-encode.

## Output Parameters

---

*value*

**String** Result from URL-encoding *inString*. If *inString* contained non-alphanumeric characters (except [-\_.\*@]), they will appear in *value* as their URL-encoded equivalents (% followed by a two-digit hex code). If *inString* contained spaces, they will appear in *value* as plus (+) signs.

# 34 Sync Folder

---

■ Summary of Elements in this Folder .....	922
--	-----

You use the elements in the sync folder to coordinate the execution of services. You can coordinate services so that a waiting service will execute if and only if a notifying service produces the input required by the waiting service within a specified time period. The synchronization services wait for and send notification using a key. A notifying service only delivers input to waiting services with the same key.

## Summary of Elements in this Folder

The following elements are available in this folder:

Element	Package and Description
<code>pub.sync:notify</code>	WmPublic. Notifies services waiting on the specified key and delivers the input document to the services.
<code>pub.sync:shutdown</code>	WmPublic. Releases internal resources used by the <code>pub.sync</code> services. This service is run automatically when the WmPublic package is unloaded and should <i>not</i> be explicitly invoked by a client.
<code>pub.sync:wait</code>	WmPublic. Allows one or more services to wait for delivery of data from a notifying service.

### pub.sync:notify

WmPublic. Notifies services waiting on the specified key and delivers the input document to the services.

#### Input Parameters

<i>key</i>	<b>String</b> Name of the key. Waiting services with the same <i>key</i> will receive notification and input from this service.
<i>value</i>	<b>Document</b> Input for the waiting services.

#### Output Parameters

<i>notified</i>	<b>String</b> Number of waiting services that received the notification. The <i>notified</i> count only includes services waiting at the time the <code>pub.sync:notify</code> service was called. Wait requests
-----------------	--

---

that start after `pub.sync:notify` executes are not included in the *notified* count.

### Usage Notes

The value of the server property `watt.server.sync.timeout` determines the maximum length of time that the notification can exist. However, if a service with an exclusive wait is registered for the notification *key*, the notification ends as soon as the exclusive wait receives the notification.

---

## pub.sync:shutdown

WmPublic. Releases internal resources used by the `pub.sync` services. This service is run automatically when the WmPublic package is unloaded and should *not* be explicitly invoked by a client.

---

## pub.sync:wait

WmPublic. Allows one or more services to wait for delivery of data from a notifying service.

---

### Input Parameters

<i>key</i>	<b>String</b> Name of the key for which the service is waiting notification. The service receives notification and data from a notifying service with the same <i>key</i> .
<i>time</i>	<b>String</b> Length of time, in seconds, the service waits for notification. If the request times out, an exception is thrown.  <b>Note:</b> A <i>time</i> value of -1 or 0 results in undefined behavior.
<i>exclusive</i>	<b>String</b> Optional. Flag indicating whether this service waits exclusively for the notification and prevents other services from waiting for a notification for the specified <i>key</i> . Set to: <ul style="list-style-type: none"><li>■ <code>yes</code> to specify that this service waits exclusively for the notification from the specified <i>key</i>. Integration Server prevents other services from waiting for the specified notification.</li><li>■ <code>no</code> to allow multiple services to wait for notification. This is the default.</li></ul>

---

## Output Parameters

---

*value*                      **Document** Input delivered by the notifying service.

### Usage Notes

Any service that is waiting for the *key* notification, receives the notification as long as the lifespan of the wait request overlaps with the lifespan of the notification. However, if a service with an exclusive wait registers for the notification *key*, the notification ends as soon as the exclusive wait is notified.

An exclusive wait might not be the only wait that receives the notification. For example, an exclusive wait might be registered after other non-exclusive waits have been notified. However, once an exclusive wait is registered, it will be the last wait to be notified.

Notification must occur within the time period specified by the *time* parameter. If the wait request expires before receiving a notification, Integration Server throws a `ServiceException`: [ISS.0086.9067] wait timed out.

If the `pub.sync.wait` service specifies a *key* for which an exclusive wait already exists, Integration Server returns a `ServiceException`: [ISS.0086.9065] already in exclusive wait.

If the `pub.sync.wait` service specifies an exclusive wait for a *key* for which regular wait threads already exist, Integration Server throws a `ServiceException`: [ISS.0086.9066] cannot obtain exclusive wait.

# 35 Synchronization Folder

---

- Summary of Elements in this Folder ..... 926

You use the elements in the synchronization folder to perform latching and cross-referencing operations in a publish-and-subscribe integration.

## Summary of Elements in this Folder

The following elements are available in this folder:

Element	Package and Description
<a href="#">pub.synchronization.latch:closeLatch</a>	WmPublic. Closes the latch for a resource.
<a href="#">pub.synchronization.latch:isLatchClosed</a>	WmPublic. Checks a resource's latch status.
<a href="#">pub.synchronization.latch:openLatch</a>	WmPublic. Opens the latch for a resource.
<a href="#">pub.synchronization.xref:createXReference</a>	WmPublic. Creates a cross-reference between a canonical key and a native ID.
<a href="#">pub.synchronization.xref:deleteByObjectId</a>	WmPublic. Removes all cross-reference records associated with a particular process or synchronization.
<a href="#">pub.synchronization.xref:deleteXReference</a>	WmPublic. Deletes a cross-reference record from the cross-reference table.
<a href="#">pub.synchronization.xref:getCanonicalKey</a>	WmPublic. Retrieves the canonical key for a specified native ID.
<a href="#">pub.synchronization.xref:getNativeId</a>	WmPublic. Retrieves the native ID of a resource record associated with a canonical key.
<a href="#">pub.synchronization.xref:insertXReference</a>	WmPublic. Inserts a cross-reference between a native ID and a canonical key.

### pub.synchronization.latch:closeLatch

WmPublic. Closes the latch for a resource.

The resource cannot be acted upon while the latch is closed. By closing a latch, you can prevent a circular update between the source and target resources.

### Input Parameters

---

<i>appId</i>	<b>String</b> A unique identifier for the target resource for which you want to close a latch. Typically, the <i>appId</i> is the name of the adapter or the resource.
<i>canonicalKey</i>	<b>String</b> The canonical key. A unique identifier for the canonical document used in the synchronization.
<i>objectId</i>	<b>String</b> A unique identifier for the object or process being synchronized. Typically, the <i>objectId</i> field is set to the name of the business process for which you are performing synchronization, such as "order" or "customer."

### Output Parameters

---

None.

---

## pub.synchronization.latch:isLatchClosed

WmPublic. Checks a resource's latch status.

By checking the latch status, you can determine whether a resource has been updated.

### Input Parameters

---

<i>appId</i>	<b>String</b> A unique identifier for the resource for which you want to check the latch status. Typically, the <i>appId</i> is the name of the adapter or the resource.
<i>canonicalKey</i>	<b>String</b> The canonical key. A unique identifier for the canonical document used in the synchronization.
<i>objectId</i>	<b>String</b> A unique identifier for the object or process being synchronized. Typically, the <i>objectId</i> field is set to the name of the business process for which you are performing synchronization, such as "order" or "customer."

### Output Parameters

---

*isLatchClosed* **String** The status of the latch. A value of:

- `true` indicates that the latch is closed. The resource has been updated.
- `false` indicates that the latch is open. The resource has not been updated.

### Usage Notes

Use the latch status to determine whether or not to update the resource.

- If the latch is closed (*isLatchClosed* is true), the resource is already updated. Use the [pub.synchronization.latch:openLatch](#) service to end execution of the update and open the latch in preparation for the next update to the resource.
- If the latch is open (*isLatchClosed* is false), the resource has not yet been updated. Invoke services to locate and update the record in the target resource. Then invoke the [pub.synchronization.latch:closeLatch](#) service to close the latch and prevent circular updates.

For more information about using the `pub.synchronization.latch` services to prevent echo suppression, see the *Publish-Subscribe Developer's Guide*.

### See Also

[pub.synchronization.latch:closeLatch](#)  
[pub.synchronization.latch:openLatch](#)

---

## pub.synchronization.latch:openLatch

WmPublic. Opens the latch for a resource.

By opening the latch, you can end propagation of the update and make the resource available for future updates.

### Input Parameters

---

<i>appId</i>	<b>String</b> A unique identifier for the resources for which you want to open the latch. Typically, the <i>appId</i> is the name of the adapter or the resource.
<i>canonicalKey</i>	<b>String</b> The canonical key. A unique identifier for the canonical document used in the synchronization.
<i>objectId</i>	<b>String</b> A unique identifier for the object or process being synchronized. Typically, the <i>objectId</i> field is set to the name of the business process for which you are performing synchronization, such as "order" or "customer."

---

## Output Parameters

---

None.

---

## pub.synchronization.xref:createXReference

WmPublic. Creates a cross-reference between a canonical key and a native ID.

---

### Input Parameters

---

<i>appId</i>	<b>String</b> A unique identifier for the resource (application) for which you want to create a cross-reference to a canonical key.
<i>nativeId</i>	<b>String</b> A unique identifier for the resource record for which you want to create a cross-reference to a canonical key.
<i>canonicalKey</i>	<b>String</b> Optional. A canonical key. If a canonical key is not provided as input, <code>createXReference</code> creates the canonical key and the cross-reference.
<i>objectId</i>	<b>String</b> A unique identifier for the object or process being synchronized. Typically, the <i>objectId</i> field is set to the name of the business process for which you are performing synchronization, such as "order" or "customer."

---

### Output Parameters

---

<i>canonicalKey</i>	<b>String</b> The canonical key. This key correlates native IDs of records from different resources. This will be a new, unique key if <i>canonicalKey</i> was not provided as an input parameter. If <i>canonicalKey</i> was provided as input, this output parameter returns the same value.
---------------------	--

### Usage Notes

The canonical document is the standard format that a document assumes while it travels through webMethods components. A source resource will convert or map data from its proprietary data format into the canonical format before publishing the document. A target resource (a subscriber to the canonical document) will map the canonical document to the target resource's proprietary data format before processing the document. The canonical document acts as the intermediary data format between resources.

On the source side of the synchronization, use the `createXReference` service to create the canonical key for the canonical document and establish a cross-reference between the

record in the source application and the canonical document. Before publishing the canonical document, link the generated *canonicalKey* to the canonical document.

On the target side of synchronization, use the [pub.synchronization.xref:insertXReference](#) service to insert the cross-reference between a canonical key and the native ID for the record in the target resource.

For more information about using the `createXReference` service to create synchronizations, see the *Publish-Subscribe Developer's Guide*

### See Also

[pub.synchronization.xref:insertXReference](#)

---

## pub.synchronization.xref:deleteByObjectId

WmPublic. Removes all cross-reference records associated with a particular process or synchronization.

### Input Parameters

---

*objectId*     **String** A unique identifier for the object or process for which you want to delete all cross-reference records. Typically, the *objectId* field is set to the name of the business process for which you are performing synchronization, such as "order" or "customer."

### Output Parameters

---

None.

### Usage Notes

You can use this service to purge unwanted cross-reference records from the cross-reference table. For example, if you wanted to delete all cross-reference records for the `purchaseOrder` synchronization, specify "purchaseOrder" as the *objectId*.

---

## pub.synchronization.xref:deleteXReference

WmPublic. Deletes a cross-reference record from the cross-reference table.

This service deletes only one cross-reference record.

### Input Parameters

---

*appId*     **String** A unique identifier for the resource (application) for which you want to delete a cross-reference record.

---

<i>canonicalKey</i>	<b>String</b> The canonical key. A unique identifier for the canonical document for which you want to delete a cross-reference.
<i>objectId</i>	<b>String</b> A unique identifier for the object or process for which you want to delete a cross-reference. Typically, the <i>objectId</i> field is set to the name of the business process for which you are performing synchronization, such as "order" or "customer."

### Output Parameters

---

None.

---

## pub.synchronization.xref:getCanonicalKey

WmPublic. Retrieves the canonical key for a specified native ID.

### Input Parameters

---

<i>appId</i>	<b>String</b> A unique identifier for the resource (application) that contains the native ID for which you want to retrieve a canonical key.
<i>nativeId</i>	<b>String</b> A unique identifier for the resource record for which you want to obtain the canonical key.
<i>objectId</i>	<b>String</b> A unique identifier for the object or process being synchronized. Typically, the <i>objectId</i> field is set to the name of the business process for which you are performing synchronization, such as "order" or "customer."

### Output Parameters

---

<i>canonicalKey</i>	<b>String</b> The canonical key for the provided native ID. If the requested key cannot be found or does not exist in the cross-reference table, an empty string is returned.
---------------------	---

### Usage Notes

You can use this service to determine whether you need to insert or update a record in the resource.

- If the canonical key exists (*canonicalKey* contains a value), a cross-reference between the native ID and the canonical key already exists. The record with the specified *nativeId* is not a new record. You can then invoke the

[pub.synchronization.latch.isLatchClosed](#) service to determine whether the resource needs to be updated.

- If the canonical key does not exist (*canonicalKey* contains an empty string), then the record with the native ID is a new record. You can use the [pub.synchronization.xref.createXReference](#) service to generate the canonical key and create the cross-reference to the native ID.

For more information about using the `getCanonicalKey` service in synchronizations, see the *Publish-Subscribe Developer's Guide*.

### See Also

[pub.synchronization.latch.isLatchClosed](#)  
[pub.synchronization.xref.createXReference](#)

---

## pub.synchronization.xref:getNativeId

WmPublic. Retrieves the native ID of a resource record associated with a canonical key.

### Input Parameters

---

<i>appId</i>	<b>String</b> A unique identifier for the resource from which you want to retrieve the native ID associated with the provided canonical key.
<i>canonicalKey</i>	<b>String</b> The canonical key for which you want to obtain the corresponding native ID.
<i>objectId</i>	<b>String</b> A unique identifier for the object or process being synchronized. Typically, the <i>objectId</i> field is set to the name of the business process for which you are performing synchronization, such as "order" or "customer."

### Output Parameters

---

<i>nativeId</i>	<b>String</b> A unique identifier for the resource record associated with the provided canonical key. If the requested <i>nativeId</i> cannot be found in the cross-reference table, an empty string is returned.
-----------------	---

### Usage Notes

You can use the `getNativeId` service on the target side of a synchronization to determine if the record in the target resource needs to be inserted or just updated.

- If the native ID does not exist (the *nativeId* field contains an empty string) and you specified the correct input values, then the record does not exist in the resource. You will need to insert the record in the resource to generate the native ID. Then use the

[pub.synchronization.xref:insertXReference](#) service to insert a cross-reference between the native ID and the canonical key.

- If the native ID exists (the *nativeId* field contains a value), then a cross-reference between the canonical key and the record already exists. The record already exists in the resource and only needs to be updated.

After you insert or update the record in the resource, make sure to use [pub.synchronization.latch:closeLatch](#) to close the latch for the record to prevent circular updates (echoes).

For more information about using the `getNativeId` service in synchronizations, see the *Publish-Subscribe Developer's Guide*.

### See Also

[pub.synchronization.latch:closeLatch](#)  
[pub.synchronization.xref:insertXReference](#)

---

## pub.synchronization.xref:insertXReference

WmPublic. Inserts a cross-reference between a native ID and a canonical key.

### Input Parameters

---

<i>appId</i>	<b>String</b> A unique identifier for the resource for which you want to establish a cross-reference between a native ID and a canonical key.
<i>nativeId</i>	<b>String</b> A unique identifier for the resource record with which you want to establish a cross-reference to <i>canonicalKey</i> .
<i>canonicalKey</i>	<b>String</b> The canonical key with which you want to establish a cross-reference to <i>nativeId</i> .
<i>objectId</i>	<b>String</b> A unique identifier for the object or process being synchronized. Typically, the <i>objectId</i> field is set to the name of the business process for which you are performing synchronization, such as "order" or "customer."

### Output Parameters

---

None.

### Usage Notes

Use this service on the target side of a synchronization to create a cross-reference between the new record in the target resource and the canonical document.

Most resources generate a unique ID for a new record. Invoke the `insertXReference` service after you add the new record in the resource.

After you insert the cross-reference between the new native ID and the canonical key, use [pub.synchronization.latch:closeLatch](#) to close the latch for the record to prevent circular updates (echoes).

For more information about using the `insertXReference` service in synchronizations, see the *Publish-Subscribe Developer's Guide*.

**See Also**

[pub.synchronization.latch:closeLatch](#)

[pub.synchronization.xref:createXReference](#)

# 36 Trigger Folder

---

■ Summary of Elements in this Folder .....	936
--	-----

You can use the services in the trigger folder to create and delete webMethods messaging trigger and JMS triggers. You can also use services to manage document retrieval and document processing for individual webMethods messaging triggers and change the state of one or more JMS triggers.

**Note:** A webMethods messaging trigger is a trigger that subscribes to and processes documents published to a webMethods messaging provider (webMethods Broker or webMethods Universal Messaging) or published locally within the Integration Server. A JMS trigger is a trigger that receives messages from a Destination (queue or topic) on a JMS provider and then processes those messages.

## Summary of Elements in this Folder

The following elements are available in this folder:

Element	Package and Description
<a href="#">pub.trigger:createJMSTrigger</a>	WmPublic. Creates a JMS trigger.
<a href="#">pub.trigger:createTrigger</a>	WmPublic. Creates a webMethods messaging trigger.
<a href="#">pub.trigger:deleteJMSTrigger</a>	WmPublic. Deletes a JMS trigger.
<a href="#">pub.trigger:deleteTrigger</a>	WmPublic. Deletes a webMethods messaging trigger.
<a href="#">pub.trigger:disableJMSTriggers</a>	WmPublic. Disables one or more JMS triggers.
<a href="#">pub.trigger:enableJMSTriggers</a>	WmPublic. Enables one or more JMS triggers.
<a href="#">pub.trigger:resourceMonitoringSpec</a>	WmPublic. Specification for the signature of a resource monitoring service.
<a href="#">pub.trigger:resumeProcessing</a>	WmPublic. Resumes document processing for the specified webMethods messaging trigger.

Element	Package and Description
<code>pub.trigger:resumeRetrieval</code>	WmPublic. Resumes retrieval of documents from the messaging provider for a specific webMethods messaging trigger.
<code>pub.trigger:suspendJMSTriggers</code>	WmPublic. Suspends one or more JMS triggers.
<code>pub.trigger:suspendProcessing</code>	WmPublic. Suspends document processing for the specified webMethods messaging trigger.
<code>pub.trigger:suspendRetrieval</code>	WmPublic. Suspends retrieval of documents from the messaging provider for a specific webMethods messaging trigger.

## pub.trigger:createJMSTrigger

WmPublic. Creates a JMS trigger.

### Input Parameters

*triggerName* **String** Fully qualified name for the JMS new trigger. Names use any combination of letters, and/or the underscore character. Make sure to specify the name of the folder and subfolder in which you want to save the JMS trigger.

**Note:** For a list of reserved words and symbols for element names, see *webMethods Service Development Help*.

*package* **String** Name of the package in which you want to save the trigger.

*aliasName* **String** Name of the JMS connection alias that you want this JMS trigger to use to receive messages from the JMS provider.

The JMS connection alias must already exist at the time this service executes. Although a JMS connection alias does not need to be enabled at the time you create the JMS trigger, the JMS connection alias must be enabled for the JMS trigger to execute at run time.

A transacted JMS connection alias cannot be assigned to a JMS trigger if a cluster policy is applied to the connection factory used by the JMS connection alias.

*jmsTriggerType* **String** Type of JMS trigger. Specify:

- `Standard` to create a standard JMS trigger. This is the default.
- `SOAPJMS` to create a SOAP-JMS trigger.

*properties* **Document** Optional. Properties that you want to assign to the JMS trigger.

Key	Description
<i>enabled</i>	<p><b>String</b> Flag indicating whether the new JMS trigger is enabled or disabled. Set to:</p> <ul style="list-style-type: none"> <li>■ <code>true</code> to create the JMS trigger in an enabled state.</li> <li>■ <code>false</code> to create the JMS trigger in a disabled state. This is the default.</li> </ul>
<i>joinTimeout</i>	<p><b>String</b> Optional. Number of milliseconds Integration Server waits for additional messages to fulfill the join. Integration Server starts the join time-out period when it receives the first message that satisfies the join.</p> <p>Set <i>joinTimeout</i> to <code>-1</code> to indicate that the join condition never expires.</p> <p>The default is one day (86400000 milliseconds).</p> <p>You need to specify a <i>joinTimeout</i> only when the <i>joinType</i> is <code>AND</code> or <code>XOR</code>. You do not need to specify a join time-out for an <code>OR</code> join.</p> <div style="background-color: #f0f0f0; padding: 5px;"> <p><b>Note:</b>You can specify a <i>joinTimeout</i> for a standard JMS trigger only. SOAP-JMS triggers cannot have joins.</p> </div>
<i>joinType</i>	<p><b>String</b> Specifies the join type for this standard JMS trigger. The join type indicates whether Integration Server needs to receive messages from all, any, or only one of destinations to execute the trigger service.</p>

You only need to set *joinType* if the JMS trigger receives messages from multiple destinations.

**Note:** You can specify a *joinType* for a standard JMS trigger only. SOAP-JMS triggers can receive messages from one destination only and therefore cannot have joins.

Set to:

- N/A to indicate that this JMS trigger does not have a join. That is, the JMS trigger receives messages from one Destination only.
- AND to invoke the trigger service when the standard JMS trigger receives a message from every destination within the join time-out period. The messages must have the same activation.

For more information about activation IDs, see *Using webMethods Integration Server to Build a Client for JMS*.

- OR to invoke the trigger service when the standard JMS trigger receives a message from any of the specified destinations.

**Note:** Using an Any (OR) join is similar to creating multiple JMS triggers that listen to different destinations. While a JMS trigger with an Any (OR) join will use fewer resources (a single thread will poll each destination for messages), it may cause a decrease in performance (it may take longer for one thread to poll multiple destinations).

- XOR to invoke the trigger service when it receives a message from any of the specified destinations. For the duration of the join time-out period, the Integration Server discards any messages with the same activation that the trigger receives from the specified destinations.

*maxRetryAttempts* **String** Optional. Maximum number of times Integration Server should re-execute the trigger service when the trigger service ends because of a transient error that causes an `ISRuntimeException`. The default is 0 attempts (indicating the trigger service does not retry)

**Note** *maxRetryAttempts* applies to non-transacted JMS triggers only.

*retryInterval* **String** Optional. Length of time Integration Server waits between retry attempts. The default is 10 seconds.

**Note** *retryInterval* applies to non-transacted JMS triggers only.

*onTransientError* **String** Flag indicating how Integration Server handles transient errors for the JMS trigger.

For a non-transacted JMS trigger, indicates how Integration Server handles a retry failure for a JMS trigger. A retry failure occurs when Integration Server reaches the maximum number of retry attempts and the trigger service still fails because of an `ISRuntimeException`.

For a transacted JMS trigger, indicates how Integration Server handles a transient error that occurs during service execution, resulting in the entire transaction being rolled back.

Specify one of the following:

- `Throw Exception/ Recover Only`

This is the default.

For a non-transacted JMS trigger, indicate that Integration Server throws a service exception when the last allowed retry attempt ends because of an `ISRuntimeException`.

For a transacted JMS trigger, indicate that Integration Server recovers the message back to the JMS provider. Integration

Server receives the message again almost immediately.

- Suspend and Retry Later/ Suspend and Recover

For a non-transacted JMS trigger, indicate that Integration Server suspends the trigger when the last allowed retry attempt ends because of an `ISRuntimeException`. Integration Server retries the trigger service at a later time when the resources needed by the trigger service become available.

For a transacted JMS trigger, indicate that Integration Server suspends the JMS trigger and then recovers the message back to the JMS provider. Integration Server executes the trigger service at a later time when the resources needed by the trigger service become available.

Selecting `Suspend and Retry Later/ Suspend and Recover` also instructs Integration Server to suspend the trigger when a transient error occurs during trigger preprocessing. For more information about transient error handling during trigger preprocessing, see *webMethods Service Development Help*.

*resumeTaskSvcName*

**String** Optional. Fully qualified name of the service that Integration Server executes when one of the following occurs:

- The trigger service ends because of a retry failure and `onTransientError` is set to `Suspend and Retry Later/Suspend and Recover`.
- The trigger service is part of a transacted JMS trigger and `onTransientError` property is set to `Suspend and Retry Later/ Suspend and Recover`.
- The document resolver service used for exactly-once processing (*dupResolverSvcName*) ends because of a run-time exception and the

`watt.server.trigger.preprocess.suspendAndRetryOnError` is set to true.

*isConcurrent*

**String** Flag indicating whether the JMS trigger uses a concurrent processing mode or a serial processing mode. Set to:

- `true` to specify a concurrent processing mode. Integration Server processes multiple messages for this trigger at one time.
- `false` to specify a serial processing mode. Integration Server processes messages received by this trigger one after the other. This is the default.

*suspendOnError*

**String** Flag indicating whether Integration Server suspends the JMS trigger when an exception occurs during trigger service execution. Set to:

- `true` to suspend the trigger when a trigger service ends with a fatal error.
- `false` to not suspend the JMS trigger when a trigger service ends with a fatal error. This is the default.

*maxExecutionThreads*

**String** Optional. Maximum number of messages that Integration Server can process concurrently on each connection for this trigger. *maxExecutionThreads* must be greater than or equal to *connectionCount*.

The default is 1.

**Note:**This setting applies to concurrent JMS triggers only.

**Note:**If the JMS provider from which the JMS trigger retrieves messages does not support concurrent access by durable subscribers, set the value of *maxExecutionThreads* to 1.

*maxBatchSize*

**String** Optional. Maximum number of messages that the trigger service can receive at one time. If you do not want the trigger

to perform batch processing, leave this property set to 1. The default is 1.

A transacted JMS trigger can be used for batch processing if the JMS connection alias used by the trigger connects to a JMS provider that supports the reuse of transacted JMS sessions. If the JMS provider does not support the reuse of transacted JMS sessions, set *maxBatchSize* to 1. Consult the documentation for your JMS provider to determine whether or not the JMS provider supports the reuse of transacted JMS sessions. Note that webMethods Broker version 8.2 and higher and webMethods Universal Messaging version 9.5 support the reuse of transacted JMS sessions.

**Note** For a SOAP-JMS trigger the *maxBatchSize* must be set to 1.

#### *dupDetection*

**String** Flag indicating whether exactly-once processing is enabled for the JMS trigger.

Set to:

- `true` to specify that exactly-once processing is provided for messages received by this trigger.
- `false` to specify that exactly-once processing is not provided for messages received by this trigger. This is the default.

#### *dupHistory*

**String** Flag indicating whether a document history database will be maintained and used to determine whether a message is a duplicate. Set to:

- `true` to indicate that Integration Server uses a document history database as part of exactly-once processing.
- `false` to indicate that Integration Server does not use a document history database as part of exactly-once processing. This is the default.

#### *dupHistoryTTL*

**String** Optional Number of milliseconds that the document history database maintains

an entry for a document processed by this trigger.

The default is 2 hours (7200000 milliseconds).

*dupResolverSvcName* **String** Optional. Specifies the service that you created to determine whether message's status is New, Duplicate, or In Doubt.

*prefetchSize* **String** Optional. Specifies the maximum number of messages Integration Server attempts to retrieve for this JMS trigger when it requests more messages from the webMethods Broker.

The default is 10.

**Note:** This parameter applies only when working with the webMethods Broker as a JMS provider.

*acknowledgeMode* **String** Indicates how the JMS trigger acknowledges messages it receives to the JMS provider.

**Note:** *acknowledgeMode* applies to non-transacted JMS triggers only. When creating a transacted JMS trigger, Integration Server ignores *acknowledgeMode*. The JMS connection alias specified for *aliasName* determines whether or not the created trigger is transacted or non-transacted.

Set to:

- **CLIENT\_ACKNOWLEDGE** to acknowledge or recover the message only after the JMS trigger processes the message completely. This is the default.
- **AUTO\_ACKNOWLEDGE** to automatically acknowledge the message when it is received by the JMS trigger. The Integration Server will acknowledge the message before the trigger completes processing. The JMS provider cannot redeliver the message if Integration Server

becomes unavailable before message processing completes.

- `DUPS_OK_ACKNOWLEDGE` to lazily acknowledge the delivery of messages. This may result in the delivery of duplicate messages.

*executeUser*

**String** Optional. Name of the user account whose credentials Integration Server uses to execute a service associated with the JMS trigger. You can specify a locally defined user account or a user account defined in a central or external directory. The default is Administrator.

*connectionCount*

**String** Optional. Specifies the number of connections a concurrent JMS trigger can use to retrieve messages from the JMS provider. *connectionCount* must be less than or equal to *maxExecutionThreads*. The default is 1.

**Note:** *connectionCount* applies only when the JMS connection alias specified for *aliasName* is configured to create a separate connection for each JMS trigger.

**Note:** When using multiple connections to the webMethods Broker acting as the JMS provider, Integration Server uses a different client ID for each JMS trigger that uses the JMS connection alias. However, when Integration Server connects to other JMS providers, it uses the same client ID for each connection. Some JMS providers do not permit multiple connections to use the same client ID to retrieve messages from a Topic with a durable subscriber. Review the JMS provider documentation before configuring the use of multiple connections for a JMS connection alias and any concurrent JMS triggers that use the JMS connection alias.

*destinations*

**Document List** Destinations from which the JMS trigger receives messages.

**Note:** For a SOAP-JMS trigger, you can specify one destination only.

<b>Key</b>	<b>Description</b>
<i>destination</i>	<p><b>String</b> Name or lookup name of the Destination from which you want the JMS trigger to receive messages.</p> <p>Specify the lookup name of the Destination object when the JMS connection alias uses JNDI to retrieve administered objects.</p> <p>Specify the provider-specific name of the Destination when the JMS connection alias uses the native webMethods API to connect directly to the webMethods Broker.</p>
<i>destinationType</i>	<p><b>String</b> Optional. Type of destination from which the JMS trigger receives messages. Set to:</p> <ul style="list-style-type: none"> <li>■ <code>Queue</code> to specify that the destination is a queue. This is the default.</li> <li>■ <code>Topic</code> to specify that the destination is a topic.</li> </ul>
<i>messageSelector</i>	<p><b>String</b> Optional. Filter used to receive a subset of messages from the specified destination. A <i>message selector</i> allows a client to filter the messages it wants to receive by use of a SQL92 string expression in the message header. That expression is applied to properties in the message header (not to the message body content) containing the value to be filtered.</p>
<i>durableSubscriberName</i>	<p><b>String</b> Optional. Name of the durable subscriber that you want to create for this JMS trigger on the JMS provider. A durable subscriber creates a durable subscription on the JMS provider. A durable subscription allows the subscriber to receive all the messages published on a topic, including those published while the subscriber is inactive.</p>

**Note:** *durableSubscriberName* applies when *destinationType* is set to `Topic` only.

*durableSubscriber  
NoLocal*

**String** Optional. Flag indicating whether the JMS trigger ignores messages sent by the same Integration Server on which the JMS trigger resides.

Set to:

- `true` to indicate that the JMS trigger ignores messages sent by the same Integration Server on which the JMS trigger resides.
- `false` to indicate that the JMS trigger receives and processes messages sent by the same Integration Server on which the JMS trigger resides. This is the default.

**Note**  *durableSubscriberNoLocal* applies when  *destinationType* is set to `Topic` only.

**Note** If the JMS connection alias specified for this trigger has the **Create New Connection per Trigger** option enabled, then set  *durableSubscriberNoLocal* to `false`. For the JMS trigger to ignore locally published messages, the publisher and subscriber must share the same connection. When the JMS connection alias uses multiple connections per trigger, the publisher and subscriber will not share the same connection.

*routingRules*

**Document List** Optional. Routing rules for messages received by this standard JMS trigger.

**Note:** You only need to specify routing rules for standard JMS triggers. SOAP-JMS triggers do not use routing rules.

Key	Description
<i> ruleName</i>	<b>String</b> Name for the routing rule.
<i> serviceName</i>	<b>String</b> Fully qualified name of the service Integration Server invokes when it receives a message from one of the specified destinations.

*filter*

**String** Optional. Filter that you want Integration Server to apply to messages the JMS trigger receives. A filter specifies criteria for the contents of the message body. Integration Server applies a local filter to message after the JMS trigger receives the message from the JMS provider.

**Note:** Integration Server evaluates the routing rules in the same order in which the rules appear in the *routingRules* document list. It is possible that a message could satisfy more than one routing rule. However, Integration Server executes only the service associated with the first satisfied routing rule and ignores the remaining routing rules. Therefore, the order in which you list routing rules is important.

## Output Parameters

---

None.

## Usage Notes

You can use `pub.trigger:createJMSTriggers` to create standard JMS triggers or SOAP-JMS triggers.

Even though WS endpoint triggers are SOAP-JMS triggers, you can create WS endpoint triggers via Integration Server Administrator only. For more information about WS endpoint triggers, see *webMethods Integration Server Administrator's Guide*.

If you use a JNDI provider to store JMS administered objects, the Connection Factories and Destinations (queues and topics) that you want this JMS trigger to use to consume messages must already exist. If they do not exist, the JMS trigger will be created but will not start. The JMS trigger will appear disabled in Designer and Integration Server Administrator.

If you use the native webMethods API to connect directly to the webMethods Broker, the Destinations from which you want the JMS trigger to receive messages must exist on the Broker. However, if you intend to use a durable subscriber to receive messages, it can be created by Integration Server when the `pub.trigger:createJMSTrigger` executes successfully. For more information about creating Destinations on the Broker, see *Administering webMethods Broker*.

The transaction type of the JMS connection alias determines whether or not the JMS trigger is transacted (that is, it receives and processes messages as part of a transaction). Transacted JMS triggers have slightly different properties and operate differently than non-transacted JMS triggers.

For a standard JMS trigger, the trigger service that you want to specify in the routing rule must already exist on the same Integration Server on which you create the JMS trigger.

A standard JMS trigger can contain multiple routing rules. Each routing rule must have a unique name.

A standard JMS trigger that contains an All (AND) or Only one (XOR) join can only have one routing rule and cannot have a batch processing size (**Max batch messages** property) greater than 1. A standard JMS trigger with an Any (Or) join can have multiple routing rules.

When you select `Topic` as the *destinationType* and specify a value for *durableSubscriberName*, Integration Server creates a durable subscriber for the JMS trigger on the JMS provider. A *durable subscriber* establishes a durable subscription with a unique identity on the JMS provider. A *durable subscription* allows subscribers to receive all the messages published on a topic, including those published while the subscriber is inactive (for example, if the JMS trigger is disabled). When the associated JMS trigger is disabled, the JMS provider holds the messages in nonvolatile storage. If a durable subscription already exists for the specified durable subscriber on the JMS provider, this service resumes the subscription.

When you select `Topic` as the *destinationType*, but do not specify a durable subscriber name, Integration Server creates a non-durable subscriber for the JMS trigger. A non-durable subscription allows subscribers to receive messages on their chosen topic only if the messages are published while the subscriber is inactive. A non-durable subscription lasts the lifetime of its message consumer. Note that non-durable subscribers cannot receive messages in a load-balanced fashion.

Integration Server uses a consumer to receive messages for a JMS trigger. This consumer encapsulates the actual `javax.jms.MessageConsumer` and `javax.jms.Session`.

Triggers and services can both be configured to retry. When a standard trigger invokes a service (that is, the service functions as a trigger service), Integration Server uses the trigger retry properties instead of the service retry properties. For a SOAP-JMS trigger, Integration Server uses the retry properties of the SOAP-JMS trigger instead of the retry properties of the service used as an operation in the web service descriptor.

When Integration Server retries a trigger service and the trigger service is configured to generate audit data on error, Integration Server adds an entry to the audit log for each failed retry attempt. Each of these entries will have a status of "Retried" and an error message of "Null". However, if Integration Server makes the maximum retry attempts and the trigger service still fails, the final audit log entry for the service will have a status of "Failed" and will display the actual error message. Integration Server makes the audit log entry regardless of which retry failure option the trigger uses.

Integration Server generates the following journal log message between retry attempts:

```
[ISS.0014.0031D] Service serviceName failed with ISRuntimeException.  
Retry x of y will begin in retryInterval milliseconds.
```

If you do not configure service retry for a trigger, set the *maxRetryAttempts* to 0. Because managing service retries creates extra overhead, setting this property to 0 can improve the performance of services invoked by the trigger.

You can invoke the `pub.flow:getRetryCount` service within a trigger service to determine the current number of retry attempts made by Integration Server and the maximum

number of retry attempts allowed for the trigger service. For more information about the `pub.flow:getRetryCount` service, see the *webMethods Integration Server Built-In Services Reference*.

Before a standard JMS trigger can be enabled, the trigger service must already exist on the same Integration Server.

The signature for a standard JMS trigger service must reference one of the following specifications:

- Use `pub.jms:triggerSpec` as the specification reference if the trigger service will process one message at a time.
- Use `pub.jms:batchTriggerSpec` as the specification reference if the trigger service will process multiple messages at one time. That is, the trigger service will receive a batch of messages as input and process all of those messages in a single execution. A trigger that receives and processes a batch of messages is sometimes referred to as a *batch trigger*.

If you create a concurrent JMS trigger that uses multiple connections to receive messages from the JMS provider, (you specified a value greater than 0 for `connectionCount`), keep the following points in mind:

- The JMS connection alias associated with this trigger must be configured to create an individual connection for each trigger. That is, the **Create New Connection per Trigger** option must be set to **Yes** for the JMS connection alias.
- If the JMS connection alias specifies a connection to the webMethods Broker, the following must be true:
  - The webMethods Broker must be webMethods Broker version 7.1 or higher.
  - The versions of following three Broker jar files installed on Integration Server must be the 8.0 SP1 or higher versions of the files.
    - *Software AG\_directory/common/lib/wm-jmsclient.jar*
    - *Software AG\_directory/common/lib/wm-brokerclient.jar*
    - *Software AG\_directory/Integration Server\_directory/instances/instance\_name/lib/jars/wm-jmsnaming.jar*
- The JMS trigger must be configured for concurrent processing (`isConcurrent` is set to true). You cannot use multiple connections with JMS triggers that perform serial processing.
- The JMS trigger must receive messages from Queues or from Topics using a durable subscriber. You cannot use multiple connections with JMS triggers that receive messages from Topics using a non-durable subscriber.
- The `connectionCount` value must be less than or equal to the `maxExecutionThreads` value.

SOAP-JMS triggers do not use routing rules. For SOAP-JMS triggers, Integration Server processes the SOAP message contained in the JMS message by executing an operation in a web service descriptor.

To use a SOAP-JMS trigger as a listener for provider web service descriptors, do the following:

- Create a provider web service endpoint alias for the JMS transport in which the SOAP-JMS trigger is specified as the JMS trigger that acts as a listener.
- Assign the web service endpoint alias to the JMS binder in the web service descriptor for which you want the SOAP-JMS trigger to listen for messages.

For more information about creating JMS triggers, see *webMethods Service Development Help*.

### See Also

[pub.trigger:deleteJMSTrigger](#)  
[pub.jms:triggerSpec](#)  
[pub.jms:batchTriggerSpec](#)

---

## pub.trigger:createTrigger

WmPublic. Creates a webMethods messaging trigger.

### Input Parameters

---

*triggerName* **String** Fully qualified name for the new trigger that uses any combination of letters, and/or the underscore character. Make sure to specify the name of the folder and subfolder in which you want to save the trigger.

**Note:** For a list of reserved words and symbols for element names, see *webMethods Service Development Help*.

*package* **String** Name of the package in which you want to save the trigger.

*properties* **Document** Optional. Properties that you want to assign to the trigger.

Key	Description
<i>joinTimeOut</i>	<b>String</b> Number of milliseconds Integration Server waits for the other documents in the join condition. Integration Server starts the join time-out period when it pulls the first document that satisfies the join condition from the trigger queue.

You need to specify a join time-out only when your condition is an `AND` or `XOR` join type. You do not need to specify a join time-out for an `OR` join condition or a condition that does not use joins.

Set *joinTimeOut* to `-1` to indicate that the join condition never expires.

The default is 1 day.

*queueCapacity*

**String** Maximum number of documents that Integration Server maintains in the queue for this trigger.

The default is 10.

*queueRefillLevel*

**String** Number of unprocessed documents that must remain in the trigger queue before Integration Server retrieves more documents for the trigger from the Broker.

The default is 4.

The *queueRefillLevel* value must be less than or equal to the *queueCapacity* value.

**Note** The *queueRefillLevel* parameter applies to a `webMethods` messaging trigger that receives documents from the Broker only. At run time, Integration Server ignores the *queueRefillLevel* if the trigger receives documents from Universal Messaging.

*ackQueueSize*

**String** Maximum number of pending document acknowledgments for the trigger. The value must be greater than zero.

The default is 1.

*maxRetryAttempts*

**String** Maximum number of times Integration Server should attempt to re-execute the trigger service. If you

want the trigger service to retry until it executes successfully, specify `-1`.

The default is 5 retries.

*retryInterval*

**String** Number of seconds Integration Server waits between retry attempts.

The default is 10 seconds.

*onRedeliveryFailure*

**String** Specifies how Integration Server handles retry failure for the trigger. Retry failure occurs when Integration Server reaches the maximum number of retry attempts and the trigger service still fails because of a run-time exception.

Specify one of the following values:

- `Throw Exception` to indicate that Integration Server throws a service exception when the last allowed retry attempt ends because of a run-time exception.

This is the default.

- `Suspend and Retry Later` to indicate that Integration Server suspends the trigger when the last allowed retry attempt ends because of a run-time exception. Integration Server retries the trigger service at a later time.

**Note** If you set *onRedeliveryFailure* to `Suspend and Retry later`, you must specify a service for the *resumeTaskSvcName* parameter. If you do not specify a service and the trigger suspends because of retry failure, Integration Server will not resume the trigger automatically. You must resume the trigger manually.

Selecting `Suspend and Retry Later` also instructs Integration Server to suspend the trigger when a

---

transient error occurs during trigger preprocessing. For more information about transient error handling during trigger preprocessing, see *webMethods Service Development Help*.

*resumeTaskSvcName*

**String** Fully qualified name of the service that Integration Server executes when one of the following occurs:

- During exactly-once processing, the document resolver service ends because of a transient error. Integration Server suspends the trigger and invokes the *resumeTaskSvcName* to determine when the resources associated with the document resolver service are available. After the resources become available, Integration Server resumes document retrieval and document processing for the trigger.
- A trigger ends because of retry failure and the *onRedeliveryFailure* variable is set to `Suspend` and `Retry Later`. Integration Server executes the *resumeTaskSvcName* to determine whether the resources associated with a trigger service are available. If the resources are available, Integration Server resumes document retrieval and document processing for the trigger.

*isPriorityEnabled*

**Boolean** Indicates whether the trigger receives messages in order of priority or in the order in which they are published. Specify one of the following values:

- `true` to indicate that documents should reach the trigger in order of priority. The higher the priority the faster the document will be received.
- `false` to indicate that documents should reach the trigger in the order

in which they are published. This is the default.

**Note** The *isPriorityEnabled* parameter applies to a webMethods messaging trigger that receives documents from the Broker only. At run time, Integration Server ignores the *isPriorityEnabled* value if the trigger receives a locally published documents or receives documents from Universal Messaging.

All webMethods messaging triggers that receive documents from Universal Messaging receive higher priority documents in an expedited fashion.

*isConcurrent*

**String** Indicates whether the trigger uses a concurrent processing mode or a serial processing mode. Specify one of the following values:

- *true* to specify a concurrent processing mode. Integration Server processes as many documents in the trigger queue as it can at once.
- *false* to specify a serial processing mode. Integration Server processes documents in the trigger queue one after the other. This is the default.

*serialSuspendOnError*

**String** Indicates whether Integration Server suspends document processing and document retrieval automatically when a trigger service ends with an error. Set to:

- *true* to indicate that Integration Server suspends the trigger automatically if an error occurs during trigger service execution.
- *false* to indicate that Integration Server should not suspend a trigger if an error occurs during trigger service execution. This is the default.

---

<i>maxExecutionThreads</i>	<b>String</b> Maximum number of documents that Integration Server can process concurrently for this trigger. Integration Server uses one server thread to process each document in the trigger queue.
<i>dupDetection</i>	<b>String</b> Indicates whether Integration Server performs exactly-once processing for guaranteed documents received by this trigger. Set to: <ul style="list-style-type: none"><li>■ <code>true</code> to indicate that Integration Server performs exactly-once processing for guaranteed documents received by this trigger.</li><li>■ <code>false</code> to indicate that exactly-once processing is not performed. This is the default.</li></ul>
<i>dupHistory</i>	<b>String</b> Indicates whether Integration Server uses a document history database as part of performing exactly-once processing. Set to: <ul style="list-style-type: none"><li>■ <code>true</code> to indicate that Integration Server uses a document history database as part of exactly-once processing.</li><li>■ <code>false</code> to indicate that Integration Server does not use a document history database as part of exactly-once processing. This is the default.</li></ul>
<i>dupHistoryTTL</i>	<b>String</b> Number of milliseconds that the document history database maintains an entry for a document processed by this trigger.  The default is 2 hours.
<i>dupResolverSvcName</i>	<b>String</b> Fully qualified name of the service used to determine conclusively whether a document's status is New, Duplicate, or In Doubt.

*executeUser*

**String** Optional. Name of the user account whose credentials Integration Server uses to execute a service associated with the trigger. You can specify a locally defined user account or a user account defined in a central or external directory. The default is Administrator.

**Note** The *executeUser* parameter applies to webMethods messaging triggers that receive message from Universal Messaging only. At run time, Integration Server ignores the *executeUser* value if the trigger receives documents from the Broker or locally published documents. To set the execution user for webMethods messaging triggers that receive documents from Broker or locally published documents, use Integration Server Administrator to set the **User** property on the **Settings > Resources > Store Settings > Edit Document Store Settings** screen.

*conditions*

**Document List** Optional. Specifies the conditions for the trigger. A condition associates one or more publishable document types with a single service. The publishable document type acts as the subscription piece of the trigger. The service is the processing piece. When the trigger receives documents to which it subscribes, the Integration Server processes the document by invoking the service specified in the condition. Triggers can contain multiple conditions; however, a trigger can contain only one join condition.

**Note:** The order in which you list conditions in the *conditions* list is important because it indicates the order in which Integration Server evaluates the conditions at run time. When Integration Server receives a document, it invokes the service specified in the first condition that is satisfied by the document. The remaining conditions are ignored. For more information about the order in which conditions are evaluated, see the *Publish-Subscribe Developer's Guide*.

**Key****Description**

---

<i>conditionName</i>	<p><b>String</b> Name you want to assign to the condition.</p> <p>By default, Integration Server assigns each condition a default name such as Condition1 or Condition2.</p>
<i>serviceName</i>	<p><b>String</b> Fully qualified name of the service that to be invoked when the trigger receives documents or messages to which it subscribes.</p>
<i>joinType</i>	<p><b>String</b> The join type for the condition. The join type determines whether Integration Server needs to receive all, any, or only one of the documents or messages in the condition to execute the trigger service.</p> <p>You must specify a <i>joinType</i> if the condition subscribes to more than one document type or message. That is, if <i>messageTypeFilterPairs</i> contains more than one pair, you must select a <i>joinType</i> . Specify one of the following:</p> <ul style="list-style-type: none"><li>■ N/A to indicate this is not a join condition.</li><li>■ AND to indicate that Integration Server invokes the trigger service when the server receives an instance of each specified message type within the join time-out period. The instance documents must have the same activation ID.</li></ul> <p>This is the default join type.</p> <ul style="list-style-type: none"><li>■ OR to indicate that Integration Server invokes the associated trigger service when it receives an instance of any one of the specified publishable document types.</li><li>■ XOR to indicate that Integration Server invokes the associated trigger service when it receives an instance of any of the specified document types. For the duration of the join time-out period, Integration Server</li></ul>

discards (blocks) any instances of the specified publishable document types with the same activation ID.

*messageTypeFilterPairs* **Document List** Specifies the messages and document types to which a trigger subscribes and the filter that must be applied to instances of the message or document type

<u>Key</u>	<u>Description</u>
<i>messageType</i>	<b>String</b> Fully qualified name of the publishable document type or message to which the trigger subscribes.
<i>filter</i>	<p><b>String</b> Optional. Filter that you want applied to each received document. The trigger processes only those documents that meet the filter criteria.</p> <p>If the publishable document type uses Broker as the messaging provider, specify a filter that you want Integration Server and/or Broker to apply to each instance of this publishable document type.</p> <p>If the publishable document type uses Universal Messaging as the messaging provider, specify the local filter that you want Integration Server to apply to each instance of the publishable document type received by the trigger.</p>

*providerFilter***String** Optional. Filter that you want Universal Messaging to apply to each instance of the publishable document type. Universal Messaging enqueues the document for the trigger only if the filter criteria is met.

**Note:** run time, Integration Server ignores the *providerFilter* if the trigger receives documents from Broker or locally published documents.

**Note:** If you specify multiple *messageType* values in one condition, you need to select a *joinType* .

## Output Parameters

None.

## Usage Notes

The client executing this service must have write access to the folders and packages in which the client wants to save the new webMethods messaging trigger. If the client does not have write access, Integration Server throws a write permissions error. For more information about assigning access permissions to folders and packages, see *webMethods Service Development Help*.

The messaging connection alias assigned to the publishable document type determines the messaging provider used by any triggers that subscribe to the publishable document type.

Integration Server validates the webMethods messaging trigger before saving it. If Integration Server determines that the webMethods messaging trigger does not meet the following requirements, Integration Server throws a `ServiceException` and does not create the trigger.

- The webMethods messaging trigger contains at least one condition.
- Each condition in the webMethods messaging trigger specifies a unique name.
- Each condition in the webMethods messaging trigger specifies a service.

- Each condition in the webMethods messaging trigger specifies at least one publishable document type.
- If more than one condition in the webMethods messaging trigger specifies the same publishable document type and the trigger receives messages from the Broker, the contents of *conditions/messageTypeFilterPairs/filter* must be the same for each condition. Software AG does not recommend using the same publishable document type in more than one condition in the same trigger.
- If more than one condition in the webMethods messaging trigger specifies the same publishable document type and the trigger receives messages from Universal Messaging, the contents of *conditions/messageTypeFilterPairs/providerFilter* must be the same for each condition. The contents of *conditions/messageTypeFilterPairs/filter* can be different for each condition.
- The webMethods messaging trigger contains no more than one join condition.
- The webMethods messaging trigger receives messages using the same messaging connection alias. That is, the publishable document types to which the trigger subscribes must all specify the same messaging connection alias. For the publishable document types to which the trigger subscribes, the value of the **Connection alias name** property can be:
  - The name of a specific messaging connection alias.
  - DEFAULT where the default messaging connection alias is the same as the alias specified for the other publishable document types to which the trigger subscribes.
- The syntax of a filter in *conditions/messageTypeFilterPairs/filter* must be correct. Create this filter using the conditional expression syntax described in *webMethods Service Development Help*. Integration Server validates the filter syntax when the `pub.trigger:createTrigger` service executes.

**Note:** Integration Server does not validate the syntax of the filter in *conditions/messageTypeFilterPairs/providerFilter*.

A trigger service for a webMethods messaging trigger must meet the following requirements:

- Before you can enable a webMethods messaging trigger, the trigger service must already exist on the same Integration Server.
- The input signature for the trigger service needs to have a document reference to the publishable document type.
- The name for this document reference must be the fully qualified name of the publishable document type. The fully qualified name of a publishable document type conforms to the following format: `folder.subfolder:PublishableDocumentTypeName`

For example, suppose that you want a webMethods messaging trigger to associate the `Customers:customerInfo` publishable document type with the

Customers:addToCustomerStore service. On the **Input/Output** tab of the service, the input signature must contain a document reference named Customers:customerInfo.

- If you intend to use the service in a join condition (a condition that associates multiple publishable document types with a service), the service's input signature must have a document reference for each publishable document type. The names of these document reference fields must be the fully qualified names of the publishable document type they reference.

If the pub.trigger:createTrigger service executes successfully, Integration Server establishes the subscription locally by creating a trigger queue for the webMethods messaging trigger on the Integration Server.

If the trigger subscribes to one or more publishable document types that specify a Broker connection alias, one of the following occurs when the pub.trigger:createTrigger service executes:

- If the Broker connection alias is enabled, Integration Server registers the trigger subscription with the Broker by creating a client for the trigger on the Broker. Integration Server also creates a subscription for each *messageType* specified in the trigger conditions and saves the subscriptions with the trigger client. Broker validates the filters in the webMethods messaging trigger conditions when Integration Server creates the subscriptions.
- If the Broker connection alias is not enabled, the trigger will only receive documents published locally. When Integration Server reconnects to Broker, the next time Integration Server restarts, Integration Server will create a client for the trigger on the Broker and create subscriptions for the publishable document types identified in the trigger conditions. Broker validates the filters in the trigger conditions when Integration Server creates the subscriptions.

If the trigger subscribes to a publishable document type that uses Universal Messaging as the messaging provider, one of the following happens upon saving the trigger.

- If Integration Server is currently connected to Universal Messaging, Integration Server creates a named object on the channel that corresponds to the publishable document type.
- If Integration Server is not currently connected to Universal Messaging, you need to synchronize the publishable document type with the provider when the connection becomes available. Synchronizing the document type will create the named object on the channel that corresponds to the publishable document type.

If *messageType* specifies a publishable document type that does not exist on the Broker (that is, there is no associated Broker document type), Integration Server still creates the trigger and the trigger client on the Broker, but does not create any subscriptions on the Broker. Integration Server creates the subscriptions when you synchronize (push) the publishable document type with the Broker.

If *messageType* specifies a publishable document type that does not have a corresponding provider definition (channel) on Universal Messaging, Integration Server creates the trigger. On the **Settings > Messaging > webMethods Messaging Trigger Management** page,

Integration Server Administrator displays an error that indicates the channel does not exist beneath the trigger name.

If a webMethods messaging trigger subscribes to a publishable document type that is not in the same package as the trigger or uses a triggers service that is not in the same package as the trigger, create package dependencies on the package containing the publishable document type and/or trigger service from the package containing the trigger. This ensures that Integration Server loads the package containing the publishable document type before loading the trigger.

If a webMethods messaging trigger subscribes to publishable document types associated with a Universal Messaging connection alias, you can create the following types of filters:

- A provider filter that Universal Messaging applies to the documents that it receives. Universal Messaging saves the filter along with the subscription to the document type. When Universal Messaging receives an instance of the publishable document type, Universal Messaging applies the filter to the document and enqueues the document for the trigger only if the filter criteria is met. Use the *conditions/messageTypeFilterPairs/providerFilter* field to specify a provider filter. For information about the syntax for provider filters for Universal Messaging, see the Universal Messaging documentation.

**Note:** The scope of the provider filter depends on the encoding type for the publishable document type. When *IData* is the encoding type, Universal Messaging applies the filter to the custom header fields added to a published document via the *\_properties* field. When protocol buffers is the encoding type, Universal Messaging applies the filter to the body of the document only. Because Integration Server includes the headers in the body of the published document as well as in the document header, you can still filter on the document headers when the encoding type is protocol buffers.

- A local filter that Integration Server applies to the published document header or document contents after the trigger receives the document. Use the *conditions/messageTypeFilterPairs/filter* to specify a local filter. Create the local filter using the conditional expression syntax described in *webMethods Service Development Help*.

When the encoding type for a publishable document type is *IData*, it is optional to include *\_properties* in the provider filter for a webMethods messaging trigger. However, when the encoding type is protocol buffers, you need to include *\_properties* in the provider filter. If you want a provider filter that operates on the contents of *\_properties* to work regardless of the encoding type, always include *\_properties* in the filter expression.

If a webMethods messaging trigger subscribes to publishable document types associated with a Broker connection alias, you can specify a single filter that can be used by Broker and/or Integration Server. Use the *conditions/messageTypeFilterPairs/filter* to specify the filter. Create the filter using the conditional expression syntax described in *webMethods Service Development Help*.

---

For more information about creating webMethods messaging triggers, see *webMethods Service Development Help*.

**See Also**

[pub.trigger:deleteTrigger](#)

---

## pub.trigger:deleteJMSTrigger

WmPublic. Deletes a JMS trigger.

**Input Parameters**

---

*triggerName*                    **String** Fully qualified name of the JMS trigger to delete.

**Output Parameters**

---

None

**See Also**

[pub.trigger:createJMSTrigger](#)

---

## pub.trigger:deleteTrigger

WmPublic. Deletes a webMethods messaging trigger.

**Input Parameters**

---

*triggerName*                    **String** Fully qualified name of the webMethods messaging trigger that you want to delete.

**Output Parameters**

---

None.

**Usage Notes**

The trigger must be unlocked for this service to execute successfully. If the trigger is locked when this service executes, Integration Server throws an error stating "Trigger is locked, change not permitted."

When you delete a webMethods messaging trigger that uses a Broker connection alias that specifies a shared client prefix, deleting the trigger will not delete the corresponding client queue on the Broker. When the client prefix is shared, the client queue might be used by other Integration Servers. The publishable document type to which a

webMethods messaging trigger subscribes determines the connection alias used by the trigger.

When you delete a webMethods messaging trigger that uses a Universal Messaging connection alias that specifies a shared client prefix, deleting the trigger will not delete the corresponding NamedObject. When the client prefix is shared, the NamedObject might be used by other Integration Servers. The publishable document type to which a webMethods messaging trigger subscribes determines the connection alias used by the trigger.

### See Also

[pub.trigger:createTrigger](#)

---

## pub.trigger:disableJMSTriggers

WmPublic. Disables one or more JMS triggers.

### Input Parameters

---

<i>triggerNameList</i>	<b>String List</b> Specifies the JMS triggers that you want to disable.
<i>applyChangeAcrossCluster</i>	<b>String</b> Optional. Flag indicating whether the specified JMS triggers should be disabled across all the servers in the cluster. Set to: <ul style="list-style-type: none"><li>■ <code>true</code> to disable the specified JMS triggers on all the nodes in the cluster.<div data-bbox="574 1241 1365 1514" style="background-color: #f0f0f0; padding: 10px; margin: 10px 0;"><b>Note:</b> To make the state change on all the servers in a cluster, Integration Server must be configured to synchronize trigger changes across the cluster. For more information about configuring an Integration Server to synchronize trigger management changes across a cluster, see <i>webMethods Integration Server Administrator's Guide</i>.</div></li><li>■ <code>false</code> to disable the JMS triggers on the local Integration Server only. This is the default.</li></ul>

### Output Parameters

---

None.

## Usage Notes

When a JMS trigger is disabled, the JMS trigger is stopped. Integration Server neither retrieves nor processes messages for the JMS trigger. The JMS trigger remains in this state until you enable the trigger.

When you disable a JMS trigger that has a non-durable subscriber, the JMS provider will remove any messages for the JMS trigger.

If you disable a SOAP-JMS trigger that acts as a listener for one or more provider web service descriptors, Integration Server will not retrieve any messages for those web service descriptors.

When you disable a JMS trigger, Integration Server does the following:

- If the JMS trigger is waiting before making a retry attempt, Integration Server interrupts processing for the JMS trigger.
- If the JMS trigger is currently processing messages, Integration Server waits a specified amount of time before forcing the JMS trigger to stop processing messages. If it does not complete in the allotted time the message consumer used to receive messages for the JMS trigger is stopped and the JMS session is closed. At this point the server thread for the JMS trigger continues to run to completion. However, the JMS trigger will not be able to acknowledge the message when processing completes. If the message is guaranteed (PERSISTENT), this can lead to duplicate messages.

The time Integration Server waits between the request to disable the JMS trigger and forcing the trigger to stop is specified by the `watt.server.jms.trigger.stopRequestTimeout` property

Because administered objects, like destinations, are configured outside of Integration Server, disabling a JMS trigger has no impact on the subscription.

Use the [pub.trigger:enableJMSTriggers](#) service to enable one or more JMS triggers.

Use the [pub.trigger:suspendJMSTriggers](#) service to suspend one or more JMS triggers.

You can also use the **Settings > Messaging > JMS Trigger Management** screens in Integration Server Administrator to disable, enable, and suspend JMS triggers. For more information, see *webMethods Integration Server Administrator's Guide*.

If you set `applyChangeAcrossCluster` to `true` and the synchronization is not successful, the following occurs:

- If Integration Server does not update all the Integration Servers in the cluster successfully, the Integration Server writes the following server log entry for each server that could not be updated:

```
[ISS.0098.0107E] Error occurred during cluster invoke:
Alias = remoteAliasName; Service = serviceName; Exception = exceptionName
```

The Integration Server Administrator also displays the following message:

```
[ISS.0085.9203] Errors occurred while updating remote aliases
(x of y updates failed). See server logs for more details.
```

- If Integration Server cannot update the Integration Servers in the cluster because the change could not be made locally, the Integration Server Administrator displays the following message:

```
[ISS.0085.9204] Local update failed: Exception providing reason for failure.
(Note: The cluster synchronization will not run until all local errors are
resolved.)
```

- If Integration Server cannot update the other Integration Servers in the cluster because cluster synchronization is not configured, the Integration Server writes the following server log entry:

```
[ISS.0033.0156W] Cluster invoke did not complete successfully.
Cluster Synchronization feature is not configured.
```

### See Also

[pub.trigger:enableJMSTriggers](#)  
[pub.trigger:suspendJMSTriggers](#)

---

## pub.trigger:enableJMSTriggers

WmPublic. Enables one or more JMS triggers.

### Input Parameters

---

<i>triggerNameList</i>	<b>String List</b> Specifies the JMS triggers that you want to enable.
<i>applyChangeAcrossCluster</i>	<p><b>String</b> Optional. Flag indicating whether the specified JMS triggers should be enabled across all the servers in the cluster. Set to:</p> <ul style="list-style-type: none"> <li>■ <code>true</code> to enable the specified JMS triggers on all the nodes in the cluster.</li> </ul> <p><b>Note:</b> To make the state change on all the servers in a cluster, the Integration Server must be configured to synchronize trigger changes across the cluster. For more information about configuring an Integration Server to synchronize trigger management changes across a cluster, see <i>webMethods Integration Server Administrator's Guide</i>.</p> <ul style="list-style-type: none"> <li>■ <code>false</code> to enable the JMS triggers on the local Integration Server only. This is the default.</li> </ul>

### Output Parameters

---

None.

## Usage Notes

When a JMS trigger is enabled, the JMS trigger is running and connected to the JMS provider. Integration Server retrieves and processes messages for the JMS trigger.

You can also use the **Settings > Messaging > JMS Trigger Management** screens in Integration Server Administrator to disable, enable, and suspend JMS triggers. For more information, see *webMethods Integration Server Administrator's Guide*.

You can use the [pub.trigger:disableJMSTriggers](#) service to disable one or more JMS triggers.

Use the [pub.trigger:suspendJMSTriggers](#) service to suspend one or more JMS triggers.

If you set *applyChangeAcrossCluster* to `true` and the synchronization is not successful, the following occurs:

- If Integration Server does not update all the Integration Servers in the cluster successfully, the Integration Server writes the following server log entry for each server that could not be updated:

```
[ISS.0098.0107E] Error occurred during cluster invoke:
Alias = remoteAliasName; Service = serviceName; = exceptionName
```

The Integration Server Administrator also displays the following message:

```
[ISS.0085.9203] Errors occurred while updating remote aliases
(x of y updates failed). See server logs for more details.
```

- If Integration Server cannot update the Integration Servers in the cluster because the change could not be made locally, the Integration Server Administrator displays the following message:

```
[ISS.0085.9204] Local update failed: Exception providing reason for failure.
(Note: The cluster synchronization will not run until all local errors
are resolved.)
```

- If the Integration Server cannot update the Integration Servers in the cluster because cluster synchronization is not configured, the Integration Server writes the following server log entry:

```
[ISS.0033.0156W] Cluster invoke did not complete successfully.
Cluster Synchronization feature is not configured.
```

You can use the Integration Server Administrator to view and change cluster synchronization status for triggers. For more information, see *webMethods Integration Server Administrator's Guide*.

## See Also

[pub.trigger:disableJMSTriggers](#)  
[pub.trigger:suspendJMSTriggers](#)

---

## pub.trigger:resourceMonitoringSpec

WmPublic. Specification for the signature of a resource monitoring service.

---

## Input Parameters

---

None.

## Output Parameters

---

*isAvailable*

**String** Indicates whether the resources needed by the trigger (webMethods messaging trigger or JMS) to perform exactly-once processing or to execute the trigger service are available. The value of this field determines whether Integration Server resumes the trigger or re-executes the resource monitoring service. Integration Server continues to execute a resource monitoring service until the value of *isAvailable* is "true". The *isAvailable* field must have one of the following values:

- `true` to indicate that the resources associated with the trigger are available. For a webMethods messaging trigger, Integration Server resumes document retrieval and document processing for the trigger. For a JMS trigger, Integration Server enables the trigger.
- `false` to indicate that the resources associated with the trigger are not available. Integration Server will not resume the trigger.

## Usage Notes

The `pub.trigger:resourceMonitoringSpec` must be used as the service signature for any service used as a resource monitoring service. A *resource monitoring service* determines whether the resources associated with a trigger (webMethods messaging trigger or JMS) are available for exactly-once processing or document pre-processing. Integration Server executes a resource monitoring service after retry failure occurs for the trigger or when the document resolver service fails because of a run-time exception. For more information about building a resource monitoring service, see the *Publish-Subscribe Developer's Guide*.

---

## pub.trigger:resumeProcessing

WmPublic. Resumes document processing for the specified webMethods messaging trigger.

## Input Parameters

---

*triggerName*

**String** Fully qualified name of the webMethods messaging trigger for which you want to resume document processing.

*persistChange*

**String** Optional. Flag indicating whether the document processing change should be permanent or temporary. Set to:

- `true` to save the change to file. Integration Server persists the change across server restarts, package reloads, and changes to trigger properties. The trigger will continue to process documents until it is actively suspended via the Integration Server Administrator or by execution of the `pub.trigger:suspendProcessing` service.
- `false` to indicate that the change is temporary and will not be maintained when the server restarts, the trigger is enabled or disabled, or the package containing the trigger reloads. This is the default.

*applyChangeAcrossCluster*

**String** Optional. Flag indicating whether document processing should be resumed for this trigger across all the servers in the cluster. Set to:

- `true` to resume document processing for the specified trigger on all the nodes in the cluster.

**Note:** To make the document processing change on all the servers in a cluster, the Integration Server must be configured to synchronize trigger changes across the cluster. For more information about configuring an Integration Server to synchronize trigger management changes across a cluster, see *webMethods Integration Server Administrator's Guide*.

- `false` to indicate that document processing for this trigger should be resumed on the local Integration Server only. This is the default.

## Output Parameters

---

None.

## Usage Notes

This service affects all documents in the specified trigger queue on the Integration Server, including documents retrieved from the messaging provider and from local publishing.

If you do not persist the change, the trigger reverts to the previously saved document processing state when the Integration Server restarts, the trigger is enabled or disabled, or the package containing the trigger reloads.

After this service executes, the Integration Server resumes document processing for this trigger at the percentage specified in the **Execution Threads Throttle** field on the **Settings**

> **Messaging > webMethods Messaging Trigger Management** page in the Integration Server Administrator.

Integration Server resumes document processing for the specified trigger even if document processing is suspended for all triggers on the Integration Server (that is, the **Processing State** for all triggers is set to Suspended).

Integration Server will not resume document processing for the specified trigger if the trigger is locked by a user. For more information about locking elements, see *webMethods Service Development Help*.

If you set *applyChangeAcrossCluster* to `true` and the synchronization is not successful, the following occurs:

- If the Integration Server does not update all the Integration Servers in the cluster successfully, the Integration Server writes the following server log entry for each server that could not be updated:

```
[ISS.0098.0107E] Error occurred during cluster invoke:
Alias = remoteAliasName; Service = serviceName; Exception = exceptionName
```

The Integration Server Administrator also displays the following message:

```
[ISS.0085.9203] Errors occurred while updating remote aliases
(x of y updates failed). See server logs for more details.
```

- If the Integration Server cannot update the Integration Servers in the cluster because the change could not be made locally, the Integration Server Administrator displays the following message:

```
[ISS.0085.9204] Local update failed: Exception providing reason for failure.
(Note: The cluster synchronization will not run until all local errors are
resolved.)
```

- If the Integration Server cannot update the Integration Servers in the cluster because cluster synchronization is not configured, the Integration Server writes the following server log entry:

```
[ISS.0033.0156W] Cluster invoke did not complete successfully.
Cluster Synchronization feature is not configured.
```

You can use the Integration Server Administrator to view and change cluster synchronization status for triggers. For more information, see *webMethods Integration Server Administrator's Guide*.

In a Java service, you can resume document processing using `com.wm.app.b2b.server.dispatcher.trigger.TriggerFacade.setProcessingSuspended()`. For more information about this method, see the *webMethods Integration Server Java API Reference* for the `com.wm.app.b2b.server.dispatcher.trigger.TriggerFacade` class.

You can resume and suspend document processing for an individual trigger or all triggers using the Integration Server Administrator. For more information, see *webMethods Integration Server Administrator's Guide*.

## See Also

[pub.trigger:suspendProcessing](#)

---

## pub.trigger:resumeRetrieval

WmPublic. Resumes retrieval of documents from the messaging provider for a specific webMethods messaging trigger.

### Input Parameters

---

<i>triggerName</i>	<b>String</b> Fully qualified name of the webMethods messaging trigger for which you want to resume document retrieval.
<i>persistChange</i>	<b>String</b> Optional. Flag indicating whether the document retrieval change should be permanent or temporary. Set to: <ul style="list-style-type: none"><li>■ <code>true</code> to save the change to file. Integration Server persists the change across server restarts, package reloads, and changes to trigger properties. The trigger will continue to retrieve documents until it is actively suspended via the Integration Server Administrator or by execution of the <code>pub.trigger:suspendRetrieval</code> service.</li><li>■ <code>false</code> to indicate that the change is temporary and will not be maintained when the server restarts, the trigger is enabled or disabled, or the package containing the trigger reloads. This is the default.</li></ul>
<i>applyChangeAcrossCluster</i>	<b>String.</b> Optional. Flag indicating whether document retrieval should be resumed for this trigger across all the servers in the cluster. Set to: <ul style="list-style-type: none"><li>■ <code>true</code> to resume document retrieval for the specified trigger on all the servers in the cluster.</li></ul> <div data-bbox="571 1388 1367 1661" style="background-color: #f0f0f0; padding: 10px;"><p><b>Note:</b> To make the document retrieval change on all the servers in a cluster, the Integration Server must be configured to synchronize trigger changes across the cluster. For more information about configuring an Integration Server to synchronize trigger management changes across a cluster, see <i>webMethods Integration Server Administrator's Guide</i>.</p></div> <ul style="list-style-type: none"><li>■ <code>false</code> to indicate that document retrieval for this trigger should be resumed on the local Integration Server only. This is the default.</li></ul>

## Output Parameters

---

None.

## Usage Notes

This service does not affect document retrieval for locally published documents to which this trigger subscribes.

If you do not persist the change, the trigger reverts to the previously saved document retrieval state when the server restarts, the trigger is enabled or disabled, or the package containing the trigger reloads.

After this service executes, the Integration Server resumes document retrieval for this trigger at the percentage specified in the **Queue Capacity Throttle** field on the **Settings > Messaging > webMethods Messaging Trigger Management** page in the Integration Server Administrator.

The Integration Server resumes document retrieval for the specified trigger even if document retrieval is suspended for all the triggers on the Integration Server (that is, the **Retrieval State** for all triggers is set to Suspended).

The Integration Server will not resume document retrieval for the specified trigger if the trigger is locked by a user. For more information about locking elements, see *webMethods Service Development Help*.

If you set *applyChangeAcrossCluster* to `true` and the synchronization is not successful, the following occurs:

- If the Integration Server does not update all the Integration Servers in the cluster successfully, the Integration Server writes the following server log entry for each server that could not be updated:

```
[ISS.0098.0107E] Error occurred during cluster invoke:
Alias = remoteAliasName; Service = serviceName; Exception = exceptionName
```

The Integration Server Administrator also displays the following message:

```
[ISS.0085.9203] Errors occurred while updating remote aliases
(x of y updates failed). See server logs for more details.
```

- If the Integration Server cannot update the Integration Servers in the cluster because the change could not be made locally, the Integration Server Administrator displays the following message:

```
[ISS.0085.9204] Local update failed: Exception providing reason for failure.
(Note: The cluster synchronization will not run until all local errors are
resolved.)
```

- If the Integration Server cannot update the Integration Servers in the cluster because cluster synchronization is not configured, the Integration Server writes the following server log entry:

```
[ISS.0033.0156W] Cluster invoke did not complete successfully.
Cluster Synchronization feature is not configured.
```

You can use the Integration Server Administrator to view and change cluster synchronization status for triggers. For more information, see *webMethods Integration Server Administrator's Guide*.

In a Java service, you can resume document retrieval by calling `com.wm.app.b2b.server.dispatcher.trigger.TriggerFacade.setRetrievalSuspended()`. For more information about this method, see the *webMethods Integration Server Java API Reference* for the `com.wm.app.b2b.server.dispatcher.trigger.TriggerFacade` class.

You can resume and suspend document retrieval for an individual trigger or all triggers using the Integration Server Administrator. For more information, see *webMethods Integration Server Administrator's Guide*.

### See Also

[pub.trigger:suspendRetrieval](#)

---

## pub.trigger:suspendJMSTriggers

WmPublic. Suspends one or more JMS triggers.

### Input Parameters

---

<i>triggerNameList</i>	<b>String List</b> Specifies the JMS triggers that you want to suspend.
<i>applyChangeAcrossCluster</i>	<p><b>String</b> Optional. Flag indicating whether the specified JMS triggers should be suspended across all the servers in the cluster. Set to:</p> <ul style="list-style-type: none"> <li>■ <code>true</code> to suspend the specified JMS triggers on all the nodes in the cluster.</li> </ul> <div style="background-color: #f0f0f0; padding: 10px; margin: 10px 0;"> <p><b>Note:</b> To make the status change on all the servers in a cluster, the Integration Server must be configured to synchronize trigger changes across the cluster. For more information about configuring an Integration Server to synchronize trigger management changes across a cluster, see <i>webMethods Integration Server Administrator's Guide</i>.</p> </div> <ul style="list-style-type: none"> <li>■ <code>false</code> to suspend the JMS triggers on the local Integration Server only. This is the default.</li> </ul>

### Output Parameters

---

None.

## Usage Notes

When a JMS trigger is suspended, the JMS trigger is running and connected to the JMS provider. Integration Server has stopped message retrieval, but continues processing any messages it has already retrieved. Integration Server enables the JMS trigger automatically upon server restart or when the package containing the JMS trigger reloads.

If you suspend a SOAP-JMS trigger that acts as a listener for one or more provider web service descriptors, Integration Server will not retrieve any messages for those web service descriptors.

If a JMS trigger is processing messages when this service executes, the JMS trigger will complete processing. JMS trigger also acknowledges the messages to the JMS provider.

After a suspending a JMS trigger, Integration Server will not start processing for any additional messages already received by the JMS trigger.

Use the [pub.trigger:disableJMSTriggers](#) service to disable one or more JMS triggers.

Use the [pub.trigger:enableJMSTriggers](#) service to enable one or more JMS triggers.

You can also use the **Settings > Messaging > JMS Trigger Management** screens in Integration Server Administrator to disable, enable, and suspend JMS triggers. For more information, see *webMethods Integration Server Administrator's Guide*.

If you set *applyChangeAcrossCluster* to `true` and the synchronization is not successful, the following occurs:

- If Integration Server does not update all the Integration Servers in the cluster successfully, the Integration Server writes the following server log entry for each server that could not be updated:

```
[ISS.0098.0107E] Error occurred during cluster invoke:  
Alias = remoteAliasName; Service = serviceName; Exception = exceptionName
```

The Integration Server Administrator also displays the following message:

```
[ISS.0085.9203] Errors occurred while updating remote aliases  
(x of y updates failed). See server logs for more details.
```

- If Integration Server cannot update the Integration Servers in the cluster because the change could not be made locally, the Integration Server Administrator displays the following message:

```
[ISS.0085.9204] Local update failed: Exception providing reason for failure.  
(Note: The cluster synchronization will not run until all local errors  
are resolved.)
```

- If Integration Server cannot update the other Integration Servers in the cluster because cluster synchronization is not configured, the Integration Server writes the following server log entry:

```
[ISS.0033.0156W] Cluster invoke did not complete successfully.  
Cluster Synchronization feature is not configured.
```

---

**See Also**

[pub.trigger:disableJMSTriggers](#)  
[pub.trigger:enableJMSTriggers](#)

---

## pub.trigger:suspendProcessing

WmPublic. Suspends document processing for the specified webMethods messaging trigger.

---

**Input Parameters**

---

<i>triggerName</i>	<b>String</b> Fully qualified name of the webMethods messaging trigger for which you want to suspend document processing.
<i>persistChange</i>	<b>String</b> Optional. Flag indicating whether the document processing change should be permanent or temporary. Set to: <ul style="list-style-type: none"><li>■ <code>true</code> to save the change to file. Integration Server persists the change across server restarts, package reloads, and changes to trigger properties. The trigger will not process documents until processing is actively resumed via the Integration Server Administrator or by execution of the <code>pub.trigger:resumeProcessing</code> service.</li><li>■ <code>false</code> to indicate that the change is temporary and will not be maintained when the server restarts, the trigger is enabled or disabled, or the package containing the trigger reloads. This is the default.</li></ul>
<i>applyChangeAcross Cluster</i>	<b>String</b> Optional. Flag indicating whether document processing should be suspended for this trigger across all the servers in the cluster. Set to: <ul style="list-style-type: none"><li>■ <code>true</code> to suspend document processing for the specified trigger on all the servers in the cluster.</li></ul>

**Note:** To make the document processing change on all the servers in a cluster, the Integration Server must belong to a properly configured cluster and it must be configured to synchronize trigger changes across the cluster. For more information about configuring an Integration Server to synchronize trigger management changes across a cluster, see *webMethods Integration Server Administrator's Guide*.

- `false` to indicate that document processing for this trigger should be suspended on the local Integration Server only. This is the default.

## Output Parameters

---

None.

## Usage Notes

This service affects all documents in the specified trigger queue on the Integration Server, including documents retrieved from the messaging provider and from local publishing.

When you suspend document processing, the Integration Server will not dispatch any more server threads to process documents in the trigger's queue. Any server threads currently processing documents for the trigger will execute to completion. This includes documents that are being retried.

When you suspend document processing, documents that the trigger retrieves will collect in the trigger queue until the trigger resumes document processing. If the server restarts before document processing resumes, volatile documents are discarded.

If you do not persist the change, the trigger reverts to the previously saved document processing state when the server restarts, the trigger is enabled or disabled, or the package containing the trigger reloads.

The Integration Server will not suspend document processing for the specified trigger if the trigger is locked by a user. For more information about locking elements, see *webMethods Service Development Help*.

If you suspend document processing, but do not suspend document retrieval for a trigger, the trigger queue fills to capacity and Integration Server stops retrieving documents for this trigger from the messaging provider.

If you set `applyChangeAcrossCluster` to `true` and the synchronization is not successful, the following occurs:

- If the Integration Server does not update all the Integration Servers in the cluster successfully, the Integration Server writes the following server log entry for each server that could not be updated:

```
[ISS.0098.0107E] Error occurred during cluster invoke:
Alias = remoteAliasName; Service = serviceName; Exception = exceptionName
```

The Integration Server Administrator also displays the following message:

```
[ISS.0085.9203] Errors occurred while updating remote aliases
(x of y updates failed). See server logs for more details.
```

- If the Integration Server cannot update the Integration Servers in the cluster because the change could not be made locally, the Integration Server Administrator displays the following message:

```
[ISS.0085.9204] Local update failed: Exception providing reason for failure.
(Note: The cluster synchronization will not run until all local errors
```

are resolved.)

- If the Integration Server cannot update the Integration Servers in the cluster because cluster synchronization is not configured, the Integration Server writes the following server log entry:

```
[ISS.0033.0156W] Cluster invoke did not complete successfully.
Cluster Synchronization feature is not configured.
```

You can use the Integration Server Administrator to view and change cluster synchronization status for triggers. For more information, see *webMethods Integration Server Administrator's Guide*.

In a Java service, you can suspend document processing by calling `com.wm.app.b2b.server.dispatcher.trigger.TriggerFacade.setProcessingSuspended()`. For more information about this method, see the *webMethods Integration Server Java API Reference* for the `com.wm.app.b2b.server.dispatcher.trigger.TriggerFacade` class.

You can resume and suspend document processing for an individual trigger or all triggers using the Integration Server Administrator. For more information, see *webMethods Integration Server Administrator's Guide*.

### See Also

[pub.trigger:resumeProcessing](#)

## pub.trigger:suspendRetrieval

WmPublic. Suspends retrieval of documents from the messaging provider for a specific webMethods messaging trigger.

### Input Parameters

<i>triggerName</i>	<b>String</b> Fully qualified name of the webMethods messaging trigger for which you want to suspend document retrieval.
<i>persistChange</i>	<p><b>String</b> Optional. Flag indicating whether the document retrieval change should be permanent or temporary. Set to:</p> <ul style="list-style-type: none"> <li>■ <code>true</code> to save the change to file. Integration Server persists the change across server restarts, package reloads, and changes to trigger properties. The trigger will not retrieve documents until retrieval is actively resumed via the Integration Server Administrator or by execution of the <code>pub.trigger:resumeProcessing</code> service.</li> <li>■ <code>false</code> to indicate that the change is temporary and will not be maintained when the server restarts, the trigger is enabled or disabled, or the package containing the trigger reloads. This is the default.</li> </ul>

*applyChangeAcrossCluster*

**String** Optional. Flag indicating whether document retrieval should be suspended for this trigger across all the servers in the cluster. Set to:

- `true` to suspend document retrieval for the specified trigger on all the servers in the cluster.

**Note:** To make the document retrieval change on all the servers in a cluster, the Integration Server be configured to synchronize trigger changes across the cluster. For more information about configuring an Integration Server to synchronize trigger management changes across a cluster, see *webMethods Integration Server Administrator's Guide*.

- `false` to indicate that document retrieval for this trigger should be suspended on the local Integration Server only. This is the default.

## Output Parameters

---

None.

## Usage Notes

This service does not affect document retrieval for locally published documents to which the specified trigger subscribes.

When you suspend document retrieval, the specified trigger will continue to receive documents delivered to the default client.

The Integration Server will not suspend document processing for the specified trigger if the trigger is locked by a user. For more information about locking elements, see *webMethods Service Development Help*.

When you suspend document retrieval, the Integration Server will not dispatch any server threads to retrieve documents from the messaging provider for the trigger. Any server threads currently retrieving documents for the trigger will execute to completion.

When you suspend document retrieval, documents to which this trigger subscribes will collect on the messaging provider. Documents remain in the trigger's client queue until document retrieval resumes for the trigger or the documents expire.

If you do not resume document retrieval before the server restarts, the trigger package reloads, or the trigger properties are modified, the messaging provider discards any volatile documents for the trigger.

If you do not persist the change, the trigger reverts to the previously saved document retrieval state when the server restarts, the trigger is enabled or disabled, or the package containing the trigger reloads.

If you suspend document retrieval for a trigger, but do not suspend document processing for the trigger, the trigger eventually processes all the documents that were retrieved from the messaging provider for the trigger.

If you set *applyChangeAcrossCluster* to true and the synchronization is not successful, the following occurs:

- If the Integration Server does not update all the Integration Servers in the cluster successfully, the Integration Server writes the following server log entry for each server that could not be updated:

```
[ISS.0098.0107E] Error occurred during cluster invoke:  
Alias = remoteAliasName; Service = serviceName; Exception = exceptionName
```

The Integration Server Administrator also displays the following message:

```
[ISS.0085.9203] Errors occurred while updating remote aliases (x of y  
updates failed). See server logs for more details.
```

- If the Integration Server cannot update the Integration Servers in the cluster because the change could not be made locally, the Integration Server Administrator displays the following message:

```
[ISS.0085.9204] Local update failed: Exception providing reason for failure.  
(Note: The cluster synchronization will not run until all local errors  
are resolved.)
```

- If the Integration Server cannot update the Integration Servers in the cluster because cluster synchronization is not configured, the Integration Server writes the following server log entry:

```
[ISS.0033.0156W] Cluster invoke did not complete successfully.  
Cluster Synchronization feature is not configured.
```

You can use the Integration Server Administrator to view and change cluster synchronization status for triggers. For more information, see *webMethods Integration Server Administrator's Guide*.

In a Java service, you can suspend document retrieval by calling `setRetrievalSuspended()`. For more information about this method, see the *webMethods Integration Server Java API Reference* for the `com.wm.app.b2b.server.dispatcher.trigger.TriggerFacade` class.

You can resume and suspend document retrieval for an individual trigger or all triggers using the Integration Server Administrator. For more information, see *webMethods Integration Server Administrator's Guide*.

## See Also

[pub.trigger:resumeRetrieval](#)

# 37 TX Folder

---

■ Summary of Elements in this Folder .....	982
--	-----

Use services in the tx folder to perform administrative tasks for guaranteed delivery transactions. For more information about guaranteed delivery, see the *Guaranteed Delivery Developer's Guide* and *webMethods Integration Server Administrator's Guide*.

## Summary of Elements in this Folder

The following elements are available in this folder:

Element	Package and Description
<a href="#">pub.tx:init</a>	WmPublic. Initializes the processing of inbound guaranteed delivery requests.
<a href="#">pub.tx:shutdown</a>	WmPublic. Stops the processing of inbound guaranteed delivery requests.
<a href="#">pub.tx:resetOutbound</a>	WmPublic. Reinitializes the processing of outbound guaranteed delivery requests.

### pub.tx:init

WmPublic. Starts processing inbound guaranteed delivery requests. These are guaranteed delivery transactions sent to Integration Server from client applications.

#### Input Parameters

None.

#### Output Parameters

*Operation*                      **String** A message indicating whether the service completed successfully. The service returns an exception if an error is encountered.

#### Usage Notes

If you shut down the guaranteed delivery capabilities of Integration Server to correct a configuration problem or to make an administrative change, use this service to reinitialize guaranteed delivery.

You can also use this service to reinitialize guaranteed delivery if it becomes disabled due to an error (for example, because of a disk full condition or if the job store database becomes inaccessible). Reinitialize guaranteed delivery after you correct the problem.

---

## pub.tx:shutdown

WmPublic. Stops processing inbound guaranteed delivery requests. These are guaranteed delivery transactions sent to Integration Server from client applications.

### Input Parameters

---

None.

### Output Parameters

---

<i>Operation</i>	<b>String</b> A message indicating whether the service completed successfully. The service returns an exception if an error is encountered.
------------------	---

### Usage Notes

You might want to shut down guaranteed delivery to perform some administration functions, such as correcting configuration errors or starting a new audit-trail log. (To start a new audit-trail log, move or rename the existing log; the server automatically starts a new log if one does not already exist.)

---

## pub.tx:resetOutbound

WmPublic. Reinitializes the processing of outbound guaranteed delivery requests. Outbound guaranteed delivery requests are those sent to another Integration Server.

### Input Parameters

---

None.

### Output Parameters

---

<i>Operation</i>	<b>String</b> A message indicating whether the service completed successfully. The service returns an exception if an error is encountered.
------------------	---

### Usage Notes

If guaranteed delivery capabilities for outbound transactions become disabled due to an error (for example, if the server encounters a disk full condition or if the job store database becomes inaccessible), use this service to reinitialize guaranteed delivery after you correct the problem. If you invoke this service while outbound guaranteed delivery

is functioning normally, the service will throw an exception and outbound guaranteed delivery will not be reinitialized.

# 38 UniversalName Folder

---

■ Summary of Elements in this Folder .....	986
--	-----

You use the elements in the universalName folder to list the contents of the Universal Name Registry and to look up services or document types by their universal names.

## Summary of Elements in this Folder

The following elements are available in this folder:

Element	Package and Description
<a href="#">pub.universalName:find</a>	WmPublic. Returns the fully qualified service name for an explicit universal name.
<a href="#">pub.universalName.findDocumentType</a>	WmPublic. Returns the fully qualified document type name for a provided explicit universal name
<a href="#">pub.universalName:list</a>	WmPublic. Returns a list of services in the current universal-name registry.
<a href="#">pub.universalName:listAll</a>	WmPublic. Returns the contents of the current universal-name registry, including services and document types.

### pub.universalName:find

WmPublic. Returns the fully qualified service name for an explicit universal name.

#### Input Parameters

<i>namespaceName</i>	<b>String</b> Namespace portion of the universal name.
<i>localName</i>	<b>String</b> Local portion of the universal name.

#### Output Parameters

<i>svcName</i>	<b>String</b> Conditional. Fully qualified name of the service associated with the universal name in <i>namespaceName</i> and <i>localName</i> . If the specified universal name is not in the registry, <i>svcName</i> will be null.
----------------	---

---

## pub.universalName:findDocumentType

WmPublic. Returns the fully qualified document type name for a provided explicit universal name

### Input Parameters

---

*namespaceName*      **String** Namespace portion of the universal name.

*localName*            **String** Local name portion of the universal name.

### Output Parameters

---

*svcName*              **String** Conditional. Fully qualified name of the document type associated with the universal name in *namespaceName* and *localName*. If the specified universal name is not in the registry, *svcName* will be null.

---

## pub.universalName:list

WmPublic. Returns a list of services in the current universal-name registry.

### Input Parameters

---

None.

### Output Parameters

---

*names*            **Document List** Service entries in the universal name registry. Each document (IData object) in the list represents a service entry in the universal-name registry. (There is one entry for every *explicit* universal name that has been defined on the server. Implicit universal names are not maintained in the registry.)

Each document in the list contains the following information:

<b>Key</b>	<b>Description</b>
<i>universalName</i>	<b>Document</b> The universal name associated with the entry. This document contains the following information:

	<u>Key</u>	<u>Description</u>
	<i>namespaceName</i>	<b>String</b> Namespace portion of the universal name.
	<i>localName</i>	<b>String</b> Local portion of the universal name.
<i>svcName</i>		<b>String</b> Fully qualified webMethods service name associated with the entry (for example, <code>gl.post:postEntry</code> ).

### Usage Notes

To return the entire contents of the universal-name registry, use the [pub.universalName:listAll](#) service.

---

## pub.universalName:listAll

WmPublic. Returns the contents of the current universal-name registry, including services and document types.

### Input Parameters

---

None.

### Output Parameters

---

*names* **Document List** Entries in the universal name registry. Each document (IData object) in the list represents an entry in the universal-name registry.

There is one entry for every *explicit* universal name that has been defined on the server. Implicit universal names are not maintained in the registry.

Each document in the list contains the following information:

<u>Key</u>	<u>Description</u>
<i>universalName</i>	<b>Document</b> Universal name associated with the entry. This document contains the following information:

<u>Key</u>	<u>Description</u>
------------	--------------------

<i>namespaceName</i>	<b>String</b> Namespace portion of the universal name.
<i>localName</i>	<b>String</b> Local portion of the universal name.
<i>svcName</i>	<b>String</b> Fully qualified Integration Server service name or document type name associated with the entry (for example, <code>gl.post:postEntry</code> ).

**Usage Notes**

To return a list of the services in the universal-name registry only, use the `pub.universalName:listservice`.



# 39

## Utils Folder

---

■ Summary of Elements in this Folder .....	992
--	-----

The utils folder contains utility services.

## Summary of Elements in this Folder

The following elements are available in this folder:

Element	Package and Description
<code>pub.utils:deepClone</code>	WmPublic. Clones an object using the default Java serialization mechanism.
<code>pub.utils:executeOSCommand</code>	WmPublic. Executes an operating system command such as <code>dir</code> in Windows or <code>ls</code> in UNIX.
<code>pub.utils:generateUUID</code>	WmPublic. Generates a random Universally Unique Identifier (UUID).
<code>pub.utils:getServerProperty</code>	WmPublic. Retrieves the value of a specified server property.
<code>pub.utils.messaging:migrateDocTypesTriggersToUM</code>	WmPublic. Migrates the messaging provider for one or more publishable document types and webMethods messaging trigger from Broker to Universal Messaging.
<code>pub.utils:transcode</code>	WmPublic. Transcodes data from one encoding to another.
<code>pub.utils.ws:setCompatibilityModeFalse</code>	WmPublic. Changes the value of the <b>Pre-8.2 compatibility mode</b> property for a web service descriptor to false.

### `pub.utils:deepClone`

WmPublic. Clones an object using the default Java serialization mechanism.

The *originalObject* and all its members must support the `java.io.Serializable` interface.

### Input Parameters

*originalObject*            **java.io.Serializable** Object to be cloned.

### Output Parameters

*clonedObject*            **Object** Copy of the *originalObject* .

## pub.utils:executeOSCommand

WmPublic. Executes an operating system command such as `dir` in Windows or `ls` in UNIX.

**Caution:** Use the `pub.utils:executeOSCommand` service with extreme caution; the commands can affect the production systems where Integration Server is running.

### Parameter Settings for OSCommands.cnf file

The `OSCommands.cnf` configuration file in the *Integration Server\_directory*\instances\*instance\_name* \packages\WmPublic\config directory contains parameters that Integration Server uses to provide validation checks to make the `pub.utils:executeOSCommand` service secure.

**Note:** If you make any changes to the `OSCommands.cnf`, you must reload the WmPublic package or restart Integration Server for the changes to take effect.

For security reasons, the `pub.utils:executeOSCommand` service checks the input *command* parameter against the list of *allowedOSCommands* in the `OSCommands.cnf` file. The service also checks the input *workingDirectory* parameter against the list of *allowedWorkingDirectories* . If the input command or directory is not on the allowed list, the service throws an exception.

### Parameter Settings

The following table shows the parameter settings for the `OSCommands.cnf` file:

Parameter	Description
<i>allowedOSCommands</i>	List of commands that can be executed using the <code>pub.utils.executeOSCommand</code> service. You can specify commands and command parameters. If you specify a command (example, <code>ps</code> ), Integration Server will run any

Parameter	Description
	command parameter with that command (such as <code>ps -ef</code> or <code>ps -aux</code> ). If you specify a particular command parameter (example, <code>ps -ef</code> ) Integration Server will run <i>only</i> that command parameter, not any other parameter associated with the command (such as <code>ps -aux</code> ).
<i>allowedWorkingDirectories</i>	List of directories where the allowed commands can be executed.

When modifying the parameters in the `OSCommands.cnf` file, keep the following points in mind:

- Use semicolon (;) as the delimiter for the *allowedOSCommands* and *allowedWorkingDirectories* parameters.
- If a command name or working directory has a semicolon (;), use backslashes (\\) before the semicolon while specifying the allowed paths.

For example, if the allowed command is `cmd.exe /c c:/temp/ab;c.txt`, specify it as `cmd.exe /c c:/temp/ab\\;c.txt` when specifying it as a parameter for the `OSCommands.cnf` file.

## Input Parameters

<i>command</i>	<p><b>String</b> Command to be executed on the target operating system. You can include command parameters only if the command parameters do not contain spaces.</p> <p><b>Important:</b> You must use <i>arguments</i> to specify all of the command parameters if any one of the parameters contains spaces.</p>
<i>arguments</i>	<p><b>String List</b> Optional. All of the command parameters for the <i>command</i>. Add one element to the parameter for each command parameter that you specify.</p>
<i>environment</i>	<p><b>String List</b> Optional. An array of strings of environment variable settings in the <code>name=value</code> format. Set the value to null if you want the service to use the environment of the current process as the environment for its subprocesses.</p>
<i>workingDirectory</i>	<p><b>String List</b> Optional. The working directory from which the command is to be executed. Set the value to null if you want the service to use the working directory of the current process as the working directory of its subprocesses.</p>

---

## Output Parameters

---

<i>status</i>	<b>String</b> The exit value of the executed command. The value 0 (zero) indicates normal termination.
<i>outputMessage</i>	<b>String</b> Output of the executed command.
<i>errorMessage</i>	<b>String</b> Errors that occurred during command execution.

## Usage Notes

To execute the `dir` command on Windows XP using the `pub.utils:executeOSCommand`, the *command* parameter is passed as `cmd.exe /c dir` and the *working directory* parameter is passed as `c:/temp`. The *outputMessage* parameter will contain the files of `c:/temp` directory.

---

## pub.utils:generateUUID

WmPublic. Generates a random Universally Unique Identifier (UUID).

### Input Parameters

---

None.

### Output Parameters

---

<i>UUID</i>	<b>String</b> A randomly generated Universally Unique Identifier (UUID).
-------------	--

---

## pub.utils:getServerProperty

WmPublic. Retrieves the value of a specified server property.

### Input Parameters

---

<i>propertyName</i>	<b>String</b> The name of the server property whose value you want to retrieve (for example, <code>watt.server.SOAP.directive</code> ).
<i>defaultValue</i>	<b>String</b> Optional. The default value to return if the server property specified in <i>propertyName</i> does not exist. If the server

property does exist, the `getServerProperty` service ignores this value.

## Output Parameters

---

*propertyValue* **String** The value of the requested server property. If the property does not exist, and you did not set a *defaultValue*, the `getServerProperty` service returns null.

## pub.utils.messaging:migrateDocTypesTriggersToUM

---

WmPublic. Changes one or more publishable document types to use a Universal Messaging server as the messaging provider.

The `pub.utils.messaging.migrateDocTypesTriggersToUM` service does one or more of the following depending on the provided input:

- Changes the messaging connection alias assigned to the publishable document type to one specified in the service.
- Sets the encoding type of the publishable document type to protocol buffers.
- Synchronizes the updated publishable document types with the messaging provider.
- Converts the filters used by the `webMethods` messaging trigger that subscribe to the publishable document types. Specifically, the service migrates the filter expressions that can be evaluated by the Universal Messaging server from the **Filter** field for a trigger condition to the **Provider Filter (UM)** field. This may include some syntax changes.

## Input Parameters

---

*reportOnly* **java.lang.Boolean** Optional. Indicates whether the service changes the document types and `webMethods` messaging triggers and returns a summary of the changes or if the service returns only a summary without making any changes. Set to:

- `true` to return a summary of the document types and `webMethods` messaging trigger that the service would successfully and unsuccessfully update, including the errors and warnings that would be encountered. The service does not modify any document types and triggers. This is the default.
- `false` to update the publishable document types and `webMethods` messaging triggers, synchronize the publishable document types with the messaging provider, and return a summary of the changes.

- backupPackages* **java.lang.Boolean** Optional. Indicates whether the service creates a package archive for each package affected by this service prior to making any updates to the publishable document types and webMethods messaging trigger. Set to:
- `true` to create an archive for each package that contains a publishable document type or webMethods messaging trigger that will be updated by this service. The archive will be named *packageName.zip\_timeStamp* and placed in the following directory: and will be located in the following directory: *Integration Server\_directory \instances\instance\_name \replicate \archive*. This is the default.
  - `false` to skip creating an archive of each affected package before making updates.

**Important** Software AG recommends that you set *backupPackages* to `true` when *reportOnly* is set to `false` if you have not taken other measures to back up the packages containing the publishable document types and triggers to be migrated prior to executing the `pub.utils.messaging:migrateDocTypesTriggersToUM` service. After the service executes successfully, restoring from a backup file is the only way to revert a change made by the service.

- packageNames* **String List** Optional. Name of the packages containing the publishable document types that you want to migrate. Integration Server also migrates any webMethods messaging triggers in packages that reference publishable document types contained in the specified packages.

The *packageNames* and *documentTypesNames* input parameters are mutually exclusive. If you specify a value for both parameters, the service fails with an exception. If you do not specify a value for either, the service updates publishable document types and webMethods messaging triggers in all packages that do not start with the letters “wm” in any case combination.

- documentTypeNames* **String List** Optional. Fully qualified names of the publishable document types that you want to update.

The *packageNames* and *documentTypeNames* input parameters are mutually exclusive. If you specify a value for both parameters, the service fails with an exception. If you do not specify a value for either, the service updates publishable document types and webMethods messaging triggers in all packages that do not start with the letters “wm” in any case combination.

*newConnectionAlias* **String** Optional. Name of the Universal Messaging (UM) connection alias to assign to the publishable document types. Any webMethods messaging trigger that subscribe to updated publishable document types will use this alias to receive published documents.

You must specify a *newConnectionAlias* to update any publishable document types that currently specify the Broker connection alias or are publishable locally only. If you do not specify a *newConnectionAlias*, the service skips these publishable document types.

If the default connection alias is the Broker connection alias and you do not specify a *newConnectionAlias*, the service skips any publishable document types that use the default connection alias. However, if you change the default connection alias to be Universal Messaging connection alias and then execute the `pub.utils.messaging:migrateDocTypesTriggersToUM` service, the service migrates any publishable document types that use the default messaging connection alias as well as any triggers that subscribe to those publishable document types.

If a publishable document types already specifies a Universal Messaging connection alias, you do not need to specify a *newConnectionAlias*. However, you can use the service to change the Universal Messaging connection alias used by one or more publishable document types.

## Output Parameters

---

*affectedPackages* **Document** Conditional. An IData containing key=value pairs that identify the packages impacted by the migration and the name of the archive created for the package. The service returns an archive name only if *reportOnly* is set to false and *backupPackages* is set to true. The archive will be named *packageName.zip\_timeStamp* and will be located in the following directory: *Integration Server\_directory \instances \instance\_name \replicate\archive*.

The service returns *affectedPackages* only if the service finds at least one publishable document type or webMethods messaging trigger to update.

*failedPackages* **Document List** Conditional. List of the packages that could not be archived. If Integration Server cannot create a package archive for any of the affected packages, the service ends with an error and nothing is migrated.

The service attempts to archive a package only when *backupPackages* is set to true.

The service returns the *failedPackages* parameter only if one of the following occurs:

- The *backupPackages* input parameter is set to true and an error prevented the service from archiving at least one package.
- A package listed in the *packageNames* input parameter did not contain any publishable document types or webMethods messaging triggers to update.

Key	Description
<i>package</i>	<b>String</b> Name of the package that the service could not archive because of an error.
<i>errors</i>	<b>String</b> Text of the error message that caused package archiving to fail.

*updatedDocument Types* **Document List** Conditional. List of the publishable document types updated by the service.

The service returns the *updatedDocumentTypes* parameter only if the service updated one or more publishable document types.

Key	Description
<i>package</i>	<b>String</b> Name of the package containing the updated publishable document type.
<i>name</i>	<b>String</b> Fully qualified name of the updated publishable document type.

*skippedDocument Types* **Document List** Conditional. List of the publishable document types that were not updated because there were no changes to make. Some reasons the service might skip a publishable document type include, but are not limited to:

- The publishable document type already uses the messaging connection alias specified in *newConnectionAlias* and the encoding type for the publishable document type is already set to protocol buffers.
- The publishable document type uses a Broker connection alias and a Universal Messaging connection alias was not specified in *newConnectionAlias*.

- Values were not specified for either the *packageName*s or the *documentTypeNames* input parameters and Integration Server found publishable document types in packages whose names started with the letters “wm” in any case combination

The service returns the *skippedDocumentTypes* parameter only if the service skipped updating one or more publishable document types.

Key	Description
<i>package</i>	<b>String</b> Name of the package containing the skipped publishable document type.
<i>name</i>	<b>String</b> Fully qualified name of the skipped publishable document type.
<i>reason</i>	<b>String</b> Message indicating why the service did not update the publishable document type.

#### *failedDocumentTypes*

**Document List** Conditional. List of the publishable document types that the service could not update because of an error.

The service returns the *failedDocumentTypes* parameter only if there were one or more publishable document types that the service could not update because of an error.

Key	Description
<i>package</i>	<b>String</b> Name of the package containing the publishable document type that the service could not update because of an error.
<i>name</i>	<b>String</b> Name of the publishable document type that the service could not update because of an error.
<i>errors</i>	<b>String</b> Text of the error message that prevented the service from updating the publishable document type.

#### *providerSyncResults*

**Document List** Conditional. Results of synchronizing the updated publishable document types with the messaging provider.

The service only produces this output parameter when *reportOnly* is set to false and the service updated at least one publishable document type.

<u>Key</u>	<u>Description</u>
<i>successfulPDTs</i>	<b>String List</b> Fully qualified names of the publishable document types that the service synchronized successfully with the messaging provider.
<i>unsuccessfulPDTs</i>	<b>String List</b> Fully qualified names of the publishable document types that the service did not synchronize successfully with the messaging provider.
<i>errors</i>	<b>Document List</b> List of any errors that occurred during document type synchronization.
<i>warnings</i>	<b>Document List</b> List of any warnings that occurred during document type synchronization
<i>updatedTriggers</i>	<p><b>Document List</b> Conditional. List of the webMethods messaging triggers updated without warnings by the service. When warnings do not occur when updating the trigger, it indicates that Integration Server migrated the entire contents of the <b>Filter</b> field to the <b>Provider Filter (UM)</b> field for all the conditions in the trigger.</p> <p>The service returns the <i>updatedTriggers</i> parameter only if the service updated at least one trigger without warnings.</p>
<u>Key</u>	<u>Description</u>
<i>package</i>	<b>String</b> Name of the package containing the updated webMethods messaging trigger.
<i>name</i>	<b>String</b> Fully qualified name of the updated webMethods messaging trigger.
<i>reason</i>	<b>String</b> Identifies any filters that were converted. For a trigger that was updated without warnings, <i>reason</i> displays the contents of the <b>Filter</b> field prior to migration of the trigger filter, the contents of the <b>Provider Filter (UM)</b> field after migration of the trigger filter, and an indication that the filter was converted successfully.
<i>updatedTriggers WithWarning</i>	<b>Document List</b> Conditional. List of the webMethods messaging triggers updated but with warnings. Warnings indicate that the service could not migrate the entire contents of the <b>Filter</b> field to the <b>Provider Filter (UM)</b> field for all the conditions in the trigger.

Integration Server might have migrated some expressions in the trigger conditions, but not the entire filter.

When a trigger is updated with warnings, the service keeps the original filter in the **Filter** field and places the filter expressions that were migrated in the **Provider Filter (UM)** field

The service returns the *updatedTriggersWithWarning* parameter only if the service updated at least one trigger for which a warning occurred.

<b>Key</b>	<b>Description</b>
<i>package</i>	<b>String</b> Name of the package containing the updated webMethods messaging trigger.
<i>name</i>	<b>String</b> Fully qualified name of the updated webMethods messaging trigger.
<i>reason</i>	<b>String</b> Identifies any filters that were converted. For a trigger that was updated with warnings, <i>reason</i> displays the expressions in the <b>Filter</b> field that were migrated, the converted expression in the <b>Provider Filter (UM)</b> , and an indication that the filter was converted successfully.
<i>warning</i>	<b>String</b> Warning message indicating why the complete original filter in the <b>Filter</b> field could not be migrated to the <b>Provider Filter (UM)</b> field.

*skippedTriggers* **Document List** Conditional. List of the webMethods messaging triggers that the service did not update because no changes were needed.

The service returns the *skippedTriggers* parameter only if the service skipped updating at least one trigger.

<b>Key</b>	<b>Description</b>
<i>package</i>	<b>String</b> Name of the package containing the skipped webMethods messaging trigger.
<i>name</i>	<b>String</b> Fully qualified name of the skipped webMethods messaging trigger.
<i>reason</i>	<b>String</b> Message indicating why the service did not make any updates for the webMethods messaging

trigger. For example, if the trigger conditions did not contain any filters, the reason will be “Trigger did not contain any filters.”

### *failedTriggers*

**Document List** Conditional. List of the webMethods messaging triggers that the service could not update because of an error.

The service returns the *failedTriggers* parameter only if at least one trigger failed migration.

<u>Key</u>	<u>Description</u>
<i>package</i>	<b>String</b> Name of the package containing the webMethods messaging trigger that the service could not update because of an error.
<i>name</i>	<b>String</b> Name of the webMethods messaging trigger that the service could not update because of an error.
<i>errors</i>	<b>String</b> Text of the error message that prevented the service from updating the webMethods messaging trigger

### Usage Notes

To execute the `pub.utils.messaging.migrateDocTypesTriggersToUM`, Integration Server must be connected to a Broker. Integration Server must have a Broker connection alias that configures a connection to the Broker that contains the provider definitions for the publishable document types and the trigger subscriptions that you want to updated. The Broker connection alias must be enabled at the time the `pub.utils.messaging.migrateDocTypesTriggersToUM` executes.

Make sure the publishable document types that use a Broker connection alias are synchronized with the associated Broker document types prior to executing the `pub.utils.messaging.migrateDocTypesTriggersToUM` service. You can synchronize publishable document types using Designer or by executing the `pub.publish.syncToProvider` service.

When using Universal Messaging as the messaging provider, webMethods messaging trigger require more server threads and system threads than needed when using Broker as the messaging provider. Before migrating publishable document types and webMethods messaging triggers to work with Universal Messaging, Software AG recommends increasing the size of your server thread pool by at least two additional threads per trigger that will be migrated. For example, if you intend to migrate 100 triggers, increase the server thread pool by 200 threads.

As part of updating publishable document types and webMethods messaging triggers to work with Universal Messaging, first run the `pub.utils.messaging.migrateDocTypesTriggersToUM` service in report only mode, that is with `reportOnly` set to true. Use the service output

to find potential problems in the migration, such as identifying publishable document types and triggers that will not be migrated successfully. When you are confident that all the publishable document types and webMethods messaging triggers that you want to migrate will do so successfully, run the `pub.utils.messaging.migrateDocTypesTriggersToUM` service with `reportOnly` set to false.

Prior to executing the `pub.utils.messaging:migrateDocTypesTriggersToUM` service, if you have not taken other measures to back up the packages containing the publishable document types and triggers to be migrated, Software AG recommends that you set `backupPackages` to true when `reportOnly` is set to false. After the service executes successfully, restoring from a backup file is the only way to revert a change made by the service.

Publishable document types that use a Universal Messaging connection alias cannot be published locally. Do not migrate a publishable document type that is used in local publishing.

When determining which publishable document types to migrate, make sure that you migrate all of the document types subscribed to by a group of webMethods messaging triggers at one time. This ensures that the webMethods messaging triggers will subscribe to document types that use the same messaging connection alias. Integration Server considers a trigger to be invalid if it subscribes to publishable document types that use different messaging connection aliases. The `pub.utils.messaging.migrateDocTypesTriggersToUM` service will save a webMethods messaging trigger that uses mixed connection aliases but will disable the trigger. Additionally, the service will not attempt to convert any of the filters in conditions of the disabled trigger. To resolve this issue, you can modify the publishable document types used by the trigger so that the publishable document types all use the same messaging connection alias. You can then re-execute the `pub.utils.messaging.migrateDocTypesTriggersToUM` service to migrate the encoding type of the publishable document types as well as the trigger filters. Alternatively, you can restore the publishable document types that the trigger subscribes to by replacing the migrated packages with the archive packages created by the service when `backupPackages` is set to true. For example, suppose that trigger `triggerAB` subscribes to `documentA` and `documentB` both of which use a Broker connection alias. In `documentTypeNames`, you specify only `documentA`. When the service executes it migrates `documentA`. However, the service does not migrate `triggerAB` because the trigger subscribes to publishable document types that use different messaging connection aliases. This configuration is invalid. The service saves `triggerAB` in a disabled state.

If you do not specify a `newConnectionAlias`, the publishable document type already uses a Universal Messaging connection alias, and the publishable document type has an encoding type of `IData`, the `pub.utils.messaging.migrateDocTypesTriggersToUM` service changes the encoding type of the publishable document type to protocol buffers. The service then updates the filters for any webMethods messaging triggers that subscribe to the document types so that the filters are compatible with protocol buffers.

If publishable document type use the default messaging connection alias and the default messaging connection alias is the Broker connection alias, you can migrate your document types and triggers to work with Universal Messaging and use the protocol buffer encoding type by first switching the default connection alias to a Universal Messaging connection alias and then running the `pub.utils.messaging:migrateDocTypesTriggersToUM` service with `reportOnly` set to false. Use

Integration Server Administrator to change the default messaging connection alias. Note that a publishable document type uses the default messaging connection alias if the **Connection alias name** property for the document type is set to DEFAULT.

The `pub.utils.messaging:migrateDocTypesTriggersToUM` service automatically attempts to migrate the `webMethods` messaging triggers that subscribe to the publishable document types updated by the service. Migrating a trigger consists of migrating a filter. To migrate a filter, the service does the following:

1. The `pub.utils.messaging:migrateDocTypesTriggersToUM` service examines the expressions in the **Filter** field in a trigger condition and determines which expressions can become a provider filter on Universal Messaging. Some filter expressions that are valid as local filters on Integration Server or valid as filters on Broker contain syntax that is not supported by protocol buffers or Universal Messaging
2. The `pub.utils.messaging:migrateDocTypesTriggersToUM` service migrates expressions that are valid filters for protocol buffers to the **Provider Filter (UM)** field.
3. The service then does one of the following:
  - If the `pub.utils.messaging:migrateDocTypesTriggersToUM` service can migrate all the expressions in the **Filter** field to the **Provider Filter (UM)** field, the service deletes the contents of the **Filter** field. The service lists the trigger in the `updatedTriggers/name` output parameter. The service lists the successfully migrated filter expressions in `updatedTriggers/reason` output parameter.
  - If the `pub.utils.messaging:migrateDocTypesTriggersToUM` service can migrate only some of the expressions in the **Filter** field, the service migrates the filter expressions that it can in the **Provider Filter (UM)** field. The service leaves the original filter in the **Filter** field. The service lists the trigger in the `updatedTriggersWithWarning/name` output parameter. The service lists the successfully migrated filter expressions in `updatedTriggersWithWarning/reason` output parameter and lists the filter expressions that it could not migrate in the `updatedTriggersWithWarning/warning` output parameter.

For a `webMethods` messaging trigger that receives messages from Universal Messaging, the contents of the **Filter** field are used as a local filter. Integration Server applies the local filter after it receives the message from Universal Messaging

- If the `pub.utils.messaging:migrateDocTypesTriggersToUM` service cannot migrate any of the expressions in the **Filter** field, the service leaves the original filter in the **Filter** field. The service lists the trigger in the `updatedTriggersWithWarning/name` output parameter. The service lists the filter expressions that it could not migrate in the `updatedTriggersWithWarning/warning` output parameter.

The `pub.utils.messaging:migrateDocTypesTriggersToUM` updates publishable document types contained in packages that begin with “Wm” when the document type is specified in `documentTypeNames` or the package is specified in `packageNames` only.

The `pub.utils.messaging:migrateDocTypesTriggersToUM` service synchronizes updated publishable document types with the provider definition on the messaging provider. Be sure to review the contents of `providerSyncResults/errors` and

*providerSyncResults/warnings* for errors and warnings that Integration Server generated during synchronization

If Integration Server did not successfully synchronize an updated publishable document type use the returned errors in *providerSyncResults/errors* to determine why Integration Server could not synchronize the publishable document type. Then, fix the cause of the error and synchronize the publishable document type with the messaging provider. You can synchronize documents using Designer or by executing the `pub.publish:syncToProvider` service.

The `pub.utils.messaging:migrateDocTypesTriggersToUM` service will not modify any publishable document types or `webMethods` messaging triggers that are locked for edit by another user or checked out at the time the service executes.

To use the `pub.utils.messaging:migrateDocTypesTriggersToUM` service to update a publishable document type and `webMethods` messaging trigger, the user or client invoking the service must have Write access to the publishable document type and `webMethods` messaging triggers.

Use the *webMethods Error Message Reference* to find out more information about the error and warning messages returned by `pub.utils.messaging:migrateDocTypesTriggersToUM` service.

To generate additional logging information in the Sever log when the `pub.utils.messaging:migrateDocTypesTriggersToUM` service executes, set the logging level for the server log facilities **0153 Dispatcher (Universal Messaging)** and **0154 Protocol Buffer Encoding (Universal Messaging)** to Debug or Trace. Increased logging can help you to locate problems that occur during service execution.

---

## pub.utils:transcode

WmPublic. Transcodes data from one encoding to another.

### Input Parameters

---

*inputData*

**Document** Data to be transcoded. Depending on the data type of the source data, use *string* or *bytes* variable to specify the data.

You must provide input data to *string* or *bytes*. If you do not specify input data in either *string* or *bytes*, the `pub.utils:transcode` service ends with an exception. If you specify input data for both *string* and *bytes*, the `pub.utils:transcode` service uses the contents of *string*, ignoring *bytes*.

Key	Description
<i>string</i>	<b>String</b> Optional. String containing the data to convert to another encoding

	<i>bytes</i>	<b>byte[]</b> Optional. Sequence of bytes to convert to another encoding.
<i>sourceEncoding</i>		<b>String</b> Encoding used for the source data. This must be an encoding supported by the JRE used with Integration Server.
<i>targetEncoding</i>		<b>String</b> Encoding to which the source data needs to be transcoded. This must be an encoding supported by the JRE used with Integration Server.
<i>onTranscodingError</i>		<p><b>String</b> Optional. Specifies the action to take when encountering unmappable characters. When transcoding data from one character set to another, it is possible that a character in the source encoding cannot be mapped to a character in the target encoding. Specify one of the following:</p> <ul style="list-style-type: none"> <li>■ <code>replace</code> to replace an unmappable character with a replacement character. If you select <code>replace</code>, specify a replacement character in <i>replaceWith</i>.</li> <li>■ <code>ignore</code> to drop any unmappable characters.</li> <li>■ <code>report</code> to throw a <code>ServiceException</code> if the service encounters any unmappable characters. This is the default.</li> </ul>
<i>replaceWith</i>		<p><b>Document</b> Optional. Character used to replace an unmappable character found during transcoding. Specify a replacement character for characters that cannot be mapped in <i>string</i> or <i>bytes</i>.</p> <p>The <code>pub.utils:transcode</code> service uses a replacement character only when <i>onTranscodingError</i> is set to <code>replace</code>.</p> <p>If <i>onTranscodingError</i> is set to <code>replace</code> and a replacement character is not specified, the <code>pub.utils:transcode</code> service uses the default replacement character of space (“\u0020”).</p> <p>If you specify a replacement character for both <i>string</i> and <i>bytes</i>, the <code>pub.utils:transcode</code> service uses the contents of <i>string</i>, ignoring <i>bytes</i>.</p>
	<b>Key</b>	<b>Description</b>
	<i>string</i>	<b>String</b> Optional. The replacement character to use for unmappable characters in the input data.

	<i>bytes</i>	<b>byte[]</b> Optional. The replacement character to use for unmappable characters in the input data.
<i>outputAs</i>		<p><b>String</b> Optional. The data type to use for the output data. Specify one of the following:</p> <ul style="list-style-type: none"> <li>■ <code>string</code></li> <li>■ <code>bytes</code></li> </ul> <p>If you do not specify a value for <i>outputAs</i>, the <code>pub.utils:transcode</code> service returns the output in the same data type used for <i>inputData</i>. For example, if you supplied the source data to <i>inputData/string</i>, the service returns the target data to the <i>outputData/string</i> output parameter.</p>
<i>normalizationForm</i>		<p><b>String</b> Optional. The Unicode normalization form to use during transcoding. Specify one of the following:</p> <ul style="list-style-type: none"> <li>■ <code>none</code> indicates that normalization is not done.</li> <li>■ <code>NFC</code> (Normalization Form C). Canonical Decomposition, followed by Canonical Composition. This is the default.</li> <li>■ <code>NFD</code> (Normalization Form D). Canonical Decomposition.</li> <li>■ <code>NFKC</code> (Normalization Form KC). Compatibility Decomposition, followed by Canonical Composition</li> <li>■ <code>NFKD</code> (Normalization Form KD) Compatibility Decomposition.</li> </ul>

## Output Parameters

---

*outputData* **Document** Transcoded data. The `pub.utils:transcode` service returns the transcoded data in *outputData/string* or *outputData/bytes*, depending on the value of the *outputAs* input parameter. If a value was not specified for *outputAs*, the service returns the transcoded data in the same data type as the supplied input data.

Key	Description
<i>string</i>	<b>String</b> Conditional. Transcoded contents of <i>inputData</i> as a String. The <code>pub.utils:transcode</code> service returns this output parameter if the input data was supplied in <i>inputData/string</i> or <i>outputAs</i> was set to <code>string</code> .

*bytes* **bytes[]** Conditional. Transcoded contents of *inputData* as a `byte[]`. The `pub.utils.transcode` service returns this output parameter if the input data was supplied in *inputData/bytes* or *outputAs* was set to `bytes`.

---

## pub.utils.ws:setCompatibilityModeFalse

WmPublic. Changes the value of the **Pre-8.2 compatibility mode** property for a web service descriptor to false.

### Input Parameters

---

*reportOnly* **java.lang.Boolean** Optional. Indicates whether the service actually changes the **Pre-8.2 compatibility mode** property for the web service descriptors and returns a summary of the changes or if the service returns only a summary without making any changes. Set to:

- `true` to return a summary of the web service descriptors that the service would successfully and unsuccessfully change, including the errors and warnings that would be encountered. The service does not modify any web service descriptors.
- `false` to change the **Pre-8.2 compatibility mode** property to false for the specified web service descriptors, save the web service descriptor, and return a summary of the changes. This is the default.

*convertAllPackages* **java.lang.Boolean** Optional. Indicates whether the service changes the **Pre-8.2 compatibility mode** property for web service descriptors in all of the packages on Integration Server or only web service descriptors in specific packages. Set to:

- `true` to change the web service descriptors in all of the packages on Integration Server.

**Note:** When *convertAllPackages* is set to `true`, the `pub.utils.ws.setCompatibilityModeFalse` service changes the **Pre-8.2 compatibility mode** property for those web service descriptors in enabled packages only. Additionally, the service skips any packages whose names begin with "Wm".

- `false` to change the web service descriptors in the packages specified in *packageNames* only. This is the default.

**Note:** If you set *convertAllPackages* to `false`, you must specify packages in *packageNames* .

*packageNames*

**String List** Optional. Names of the packages containing the web service descriptors for which you want to change the **Pre-8.2 compatibility mode** property to `false`. The packages you specify must be enabled. Note that package names are not case-sensitive.

**Note:** If you do not specify any packages in *packageNames* , you must set *convertAllPackages* to `true`.

## Output Parameters

*updatedWarningCount*

**String** Conditional. Number of web service descriptors for which changing the value of the **Pre-8.2 compatibility mode** property to `false` resulted in a warning. If *reportOnly* is set to `true`, *updatedWarningCount* indicates the number of web service descriptors for which a warning would be generated. The service returns *updatedWarningCount* only if Integration Server encountered a warning when successfully changing the value of the **Pre-8.2 compatibility mode** property from `true` to `false` for at least one web service descriptor.

If the service did not (or would not if *reportOnly* is set to `true`) update any web service descriptors, *updatedWarningCount* is not returned.

**Note:** Some warnings require further action to be taken, such as regenerating web service connectors. Be sure to review all warnings.

*updated*

**Document List** Conditional. The web service descriptors for which the service successfully changed the value of the **Pre-8.2 compatibility mode** property from `true` to `false`, including any warnings that occurred. If *reportOnly* is set to `true`, *updated* indicates the number of web service descriptors that the service would successfully change and any warnings that the service would encounter.

The service returns the *updated* parameter only if the service successfully changed the value of the **Pre-8.2 compatibility mode** property from `true` to `false` for at least one web service descriptor.

Key	Description
-----	-------------

<i>package</i>	<b>String</b> Package that contains the updated web service descriptor.
<i>name</i>	<b>String</b> Fully qualified name of the updated web service descriptor.
<i>warnings</i>	<b>String List</b> List of any warning that occurred when the service changed the <b>Pre-8.2 compatibility mode</b> property from true to false.

*failed*

**Document List** Conditional. The web service descriptors for which the service could not change the value of the **Pre-8.2 compatibility mode** from true to false because an error occurred.

The service returns the *failed* parameter only if an error prevented the service from changing the value of the **Pre-8.2 compatibility mode** property from true to false for at least one web service descriptor.

<u>Key</u>	<u>Description</u>
<i>package</i>	<b>String</b> Package containing the web service descriptor that could not be changed.
<i>name</i>	<b>String</b> Fully qualified name of the web service descriptor that could not be changed because of an error.
<i>warnings</i>	<b>String List</b> List of the warnings that occurred when the service attempted to change the <b>Pre-8.2 compatibility mode</b> property from true to false.
<i>errors</i>	<b>String List</b> List of the errors that prevented the service from changing the value of the <b>Pre-8.2 compatibility mode</b> property from true to false.

*skipped*

**Document List** Conditional. The web service descriptors for which the **Pre-8.2 compatibility mode** property is already set to false.

The service returns the *skipped* parameter only if the service encountered web service descriptors for which **Pre-8.2 compatibility mode** property was already set to false.

<u>Key</u>	<u>Description</u>
------------	--------------------

---

<i>package</i>	<b>String</b> Package containing the web service descriptor.
<i>name</i>	<b>String</b> Fully qualified name of the web service descriptors for which <b>Pre-8.2 compatibility mode</b> property was already set to false.

### Usage Notes

Even though the *convertAllPackages* and *packageNames* input parameters are optional, you must either set *convertAllPackages* to `true` or specify packages in *packageNames*.

Use the `pub.utils.ws:setCompatibilityModeFalse` service to change the **Pre-8.2 compatibility mode** property value for multiple web service descriptors at one time.

The **Pre-8.2 compatibility mode** property indicates the version of the web service stack with which the web service descriptor is compatible:

- When the **Pre-8.2 compatibility mode** property is set to `true`, the web service descriptor runs on the earlier version of the web services stack, specifically the web services stack available in Integration Server versions 7.x, 8.0, and 8.0 SP1. web service descriptors running in pre-8.2 compatibility mode have the same design-time features and run-time behavior as web service descriptors run in versions of Integration Server prior to version 8.2.
- When the **Pre-8.2 compatibility mode** property is set to `false`, the web service descriptor runs on the current version of the web services stack. web service descriptors that do not run in pre-8.2 compatibility mode have the design-time features and run-time behavior available in the current version of the web services stack.

For more details about which features are impacted by the compatibility mode of the web service descriptor, see the *Web Services Developer's Guide*.

Before changing the web service descriptor, the service verifies that the web service descriptor can be deployed to the web services stack that corresponds to the chosen compatibility mode. Warnings indicate that the web service descriptor can be deployed to the web services stack successfully but some run-time behavior might change. If warnings occur for a particular web service descriptor, the service changes the compatibility mode and lists warnings in the *updated* output parameter. Errors identify the functionality that is incompatible with the web services stack. If errors occur for a particular web service descriptor, the service does not change the compatibility mode for that web service descriptor. the service identifies the errors in the *failed* output parameter.

Some warnings require further action to be taken, such as regenerating web service connectors. Be sure to review all warnings.

The `pub.utils.ws:setCompatibilityModeFalse` service will not modify any web service descriptors that are locked for edit or checked out at the time the service executes.

To use the `pub.utils.ws:setCompatibilityModeFalse` service to change the Pre-8.2 compatibility mode property for a web service descriptor, you or whichever client is invoking the service must have Write access to the web service descriptors.



# 40 VCS Folder

---

■ Summary of Elements in this Folder .....	1016
--	------

You use the elements in the VCS folder to manage user associations for the Version Control System Integration feature.

- Note:** You can also manage user associations between Designer and a VCS server from the **Solutions > VCS > User Mapping** page in Integration Server Administrator. For more information about the Version Control System Integration feature, see *Configuring the VCS Integration Feature*.
- Important:** The WmVCS package, which provides the functionality for using the VCS Integration Feature, is deprecated as of Integration Server version 9.9. Software AG recommends that you use the local service development feature (Local Version Control Integration) to check package elements and their supporting files into and out of a version control system (VCS) directly from Designer.

## Summary of Elements in this Folder

The following elements are available in this folder:

Element	Package and Description
<a href="#">pub.vcs.admin:getUsers</a>	WmVCS. <i>Deprecated</i> - Use the local service development feature (Local Version Control Integration) to check package elements and their supporting files into and out of a version control system (VCS) directly from Designer. Returns a list of the Designer user accounts that are associated with a corresponding version control system (VCS) user account on the VCS server.
<a href="#">pub.vcs.admin:removeCurrentUser</a>	WmVCS. <i>Deprecated</i> - Use the local service development feature (Local Version Control Integration) to check package elements and their supporting files into and out of a version control system (VCS) directly from Designer. Removes the currently logged in Designer user account from the list of users associated with a version control system (VCS) user account on the VCS server.
<a href="#">pub.vcs.admin:removeMultipleUsers</a>	WmVCS. <i>Deprecated</i> - Use the local service development feature (Local Version Control Integration) to check package elements and their supporting files into and out of a version

Element	Package and Description
	control system (VCS) directly from Designer. Removes the specified Designer user accounts from the list of users associated with a version control system (VCS) user account.
<a href="#">pub.vcs.admin:setCurrentUser</a>	WmVCS. <i>Deprecated</i> - Use the local service development feature (Local Version Control Integration) to check package elements and their supporting files into and out of a version control system (VCS) directly from Designer. Associates a version control system (VCS) user account with the currently logged in Designer user account.
<a href="#">pub.vcs.admin:setMultipleUsers</a>	WmVCS. <i>Deprecated</i> - Use the local service development feature (Local Version Control Integration) to check package elements and their supporting files into and out of a version control system (VCS) directly from Designer. Associates a version control system (VCS) user account with the specified Designer user accounts.

## pub.vcs.admin:getUsers

WmVCS. *Deprecated* - Use the local service development feature (Local Version Control Integration) to check package elements and their supporting files into and out of a version control system (VCS) directly from Designer.

Returns a list of the Designer user accounts that are associated with a corresponding version control system (VCS) user account on the VCS server.

### Input Parameters

None.

### Output Parameters

*users* **Document List** List of Designer user accounts with associations to a VCS user account on the VCS server.

Key	Description
-----	-------------

---

<i>devName</i>	<b>String</b> The name of the Designer user account.
<i>vcsName</i>	<b>String</b> The name of the VCS user account associated with the <i>devName</i> account.

### Usage Notes

This service is available only to administrator users.

This service is for use with the VCS Integration feature only.

---

## pub.vcs.admin:removeCurrentUser

WmVCS. *Deprecated* - Use the local service development feature (Local Version Control Integration) to check package elements and their supporting files into and out of a version control system (VCS) directly from Designer.

Removes the currently logged in Designer user account from the list of users associated with a version control system (VCS) user account on the VCS server.

### Input Parameters

None.

### Output Parameters

None.

### Usage Notes

This service is available to all users. After running this service, the currently logged in Designer user account is no longer associated with a VCS user account. Nominally, this prevents the Designer user from checking elements in to and out of the VCS repository. However, on Windows operating systems, VCS actions will still be submitted by the VCS client with the user's current Windows user name. If the credentials of the Windows user account match the credentials of a VCS account on the VCS server, the VCS actions will be completed successfully.

This service is for use with the VCS Integration feature only.

The user is advised to check in all checked out elements before running this service. Version Control System Integration feature will not permit an element to be checked in by other than the user who checked it out. This service has no effect on the VCS user account.

---

## pub.vcs.admin:removeMultipleUsers

WmVCS. *Deprecated* - Use the local service development feature (Local Version Control Integration) to check package elements and their supporting files into and out of a version control system (VCS) directly from Designer.

Removes the specified Designer user accounts from the list of users associated with a version control system (VCS) user account.

### Input Parameters

---

*devNames*    **String List** The names of the Designer user accounts.

### Output Parameters

---

None.

### Usage Notes

This service is available only to administrator users. The user account name is case-sensitive. After running this service, the Designer user accounts specified in the input parameters are no longer associated with a VCS user account. Nominally, this prevents the specified Designer users from checking elements in to and out of the VCS repository. However, on Windows operating systems, VCS actions will still be submitted by the VCS client with the user's current Windows user name. If the credentials of the Windows user account match the credentials of a VCS account on the VCS server, the VCS actions will be completed successfully.

This service is for use with the VCS Integration feature only.

Administrators are advised to verify that all elements checked out by the specified Designer users are checked in before running this service. The Version Control System Integration feature will not permit an element to be checked in by other than the user who checked it out. This service has no effect on the VCS user account.

---

## pub.vcs.admin:setCurrentUser

WmVCS. *Deprecated* - Use the local service development feature (Local Version Control Integration) to check package elements and their supporting files into and out of a version control system (VCS) directly from Designer.

Associates a version control system (VCS) user account with the currently logged in the Designer user account. For information about restrictions on account association, see the [Usage Notes](#).

---

### Input Parameters

*vcsName*      **String** The name of an existing VCS user account on the VCS server.

*vcsPassword*      **String** The password of the VCS user account specified in *vcsName* .

---

### Output Parameters

None.

### Usage Notes

This service is available to all users. A Developer user name does not have to be the same as the associated VCS server user name, and all user account credentials are case-sensitive. Each Designer user can have one VCS user account associated with the Designer user account. Although it is possible for more than one Designer user to be associated with the same VCS user account, Software AG recommends that you avoid this configuration as it may result in errors or unpredictable results.

This service is for use with the VCS Integration feature only.

After running this service, the currently logged in Designer user account is associated with a user account on the VCS server, enabling the Designer user to check elements in to and out of the VCS repository (with proper ACL permissions). This association is maintained until it is removed with the [pub.vcs.admin:removeCurrentUser](#) or [pub.vcs.admin:removeMultipleUsers](#) services. This service does not validate, create, or modify VCS server accounts.

---

## pub.vcs.admin:setMultipleUsers

WmVCS. *Deprecated* - Use the local service development feature (Local Version Control Integration) to check package elements and their supporting files into and out of a version control system (VCS) directly from Designer.

Associates a version control system (VCS) user account with the specified Designer user accounts.

---

### Input Parameters

*devNames*      **Document List** Information required to associate each Designer user account with a VCS user account on the VCS server:

<b>Key</b>	<b>Description</b>
<i>devName</i>	<b>String</b> The name of the Designer user account.

<i>vcsName</i>	<b>String</b> The name of an existing VCS user account on the VCS server.
<i>vcsPassword</i>	<b>String</b> The password of the VCS user account specified in <i>vcsName</i> .

### Output Parameters

---

None.

### Usage Notes

This service is available only to administrator users. A Designer user name does not have to be the same as the associated VCS server user name, and the user account name is case-sensitive. Each Designer user can have one VCS user account associated with the Designer user account. Although it is possible for more than one Designer user to be associated with the same VCS user account, Software AG recommends that you avoid this configuration as it may result in errors or unpredictable results.

This service is for use with the VCS Integration feature only.

After running this service, the Designer user accounts specified in the input parameters are associated with a corresponding user account on the VCS server, enabling the Designer users to check elements in to and out of the VCS repository (with proper ACL permissions). This association is maintained until it is removed with the [pub.vcs.admin:removeCurrentUser](#) or [pub.vcs.admin:removeMultipleUsers](#) services. This service does not validate, create, or modify VCS server accounts.



# 41 XML Folder

---

■ Summary of Elements in this Folder .....	1024
--	------

You use the elements in the xml folder to perform operations on XML documents.

## Summary of Elements in this Folder

The following elements are available in this folder:

Element	Package and Description
<a href="#">pub.xml:documentToXMLString</a>	WmPublic. Converts a document (IData object) to an XML string.
<a href="#">pub.xml:freeXMLNode</a>	WmPublic. Frees the resources allocated to a given XML node.
<a href="#">pub.xml:getNextXMLNode</a>	WmPublic. Gets the next XML node from a NodeIterator.
<a href="#">pub.xml:getXMLNodeIterator</a>	WmPublic. Creates and returns a NodeIterator.
<a href="#">pub.xml:getXMLNodeType</a>	WmPublic. Returns information about an XML node.
<a href="#">pub.xml:loadEnhancedXMLNode</a>	WmPublic. Retrieves an XML document via HTTP or HTTPS, parses it using the enhanced XML parser, and produces an org.w3c.dom.Node object.
<a href="#">pub.xml:loadXMLNode</a>	WmPublic. Retrieves an XML document via HTTP or HTTPS, parses it using the legacy XML parser, and produces an XML node.
<a href="#">pub.xml:queryXMLNode</a>	WmPublic. Queries an XML node.
<a href="#">pub.xml:xmlNodeToDocument</a>	WmPublic. Converts an XML node to a document (an IData object).
<a href="#">pub.xml:xmlStringToEnhancedXMLNode</a>	WmPublic. Converts an XML document (represented as a String, byte[], or InputStream) to an org.w3c.dom.Node object using the enhanced XML parser.
<a href="#">pub.xml:xmlStringToXMLNode</a>	WmPublic. Converts an XML document (represented as a String, byte[], or

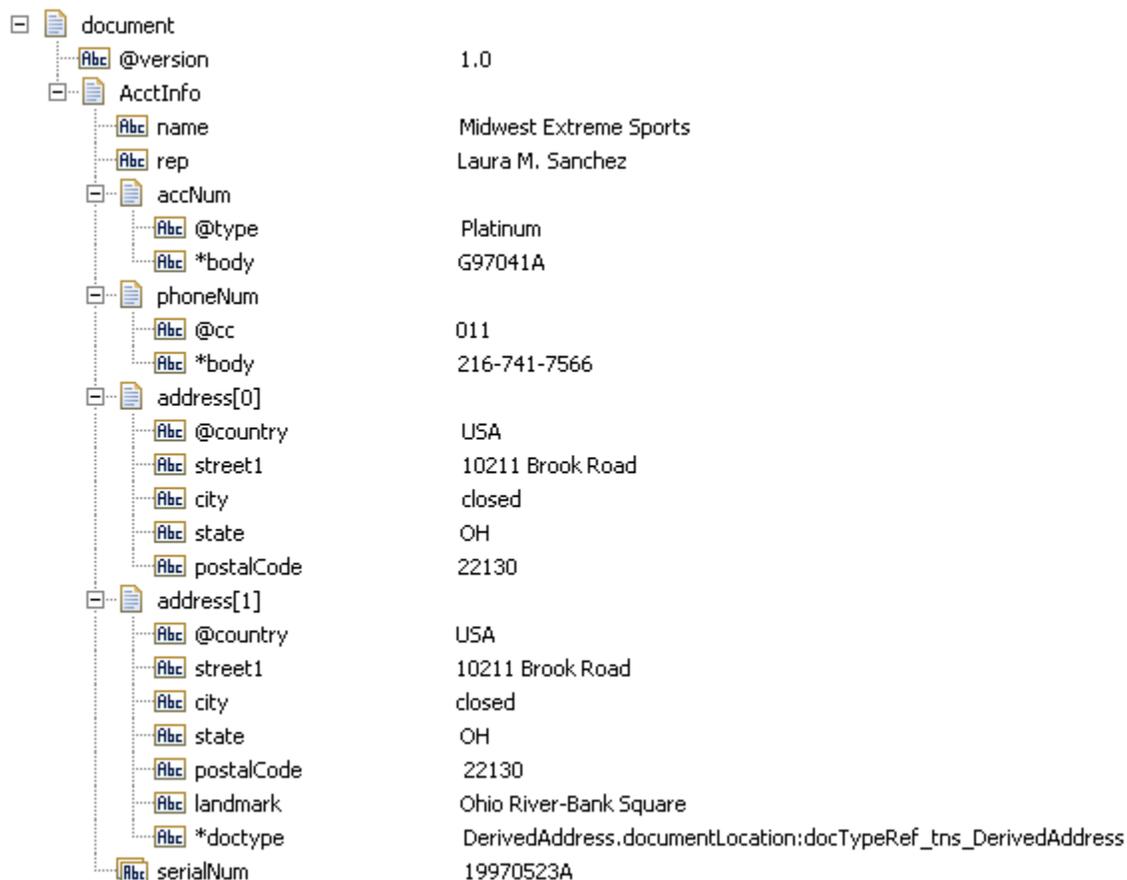
Element	Package and Description
	InputStream) to an XML node using the legacy XML parser.

## pub.xml:documentToXMLString

WmPublic. Converts a document (IData object) to an XML string.

This service recurses through a given document, building an XML representation from the elements within it. Key names are turned into XML elements, and the key values are turned into the contents of those elements.

This service would convert this document (IData object)...



To an XML document that looks like this...

```

<?xml version="1.0" ?>
<tns:AcctInfo>
  xmlns:tns="http://localhost/DerivedAddress/schema.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >
  <name>Midwest Extreme Sports</name>

```

```

<rep>Laura M. Sanchez</rep>
<acctNum type=platinum>G97041A</acctNum>
<phoneNum cc=011>216-741-7566</phoneNum>
<address country=USA>
  <street1>10211 Brook Road</street1>
  <city>Cleveland</city>
  <state>OH</state>
  <postalCode>22130</postalCode>
</address>
<address country=USA xsi:type="tns:DerivedAddress">
<street1>10211 Brook Road</street1>
  <city>Cleveland</city>
  <state>OH</state>
  <postalCode>22130</postalCode>
  <landMark>Besides Ohio River-Bank Square</landMark>
  <telNo>001222555</telNo>
</address>
<serialNum>19970523A</serialNum>
<serialNum>20001106G</serialNum>
<serialNum>20010404K</serialNum>
</tns:AcctInfo>

```

Note that:

- Key names that start with the attribute prefix (which, in this example, is the "@" character) are turned into attributes of the elements in which they occur. For example, the @type key in the acctNum element is converted to the type=platinum attribute of the <acctNum> element in the resulting XML String.
- Some modules, such as EDI and FlatFile, have fields that begin with the @ symbol that will be considered attributes regardless of whether the attrPrefix is @ or another character. For example, the FlatFile module always considers fields named @record-id and @delimiters to be attributes.
- When a document type contains a String variable that represents a required attribute (meaning that the variable name starts with the "@" symbol and the **Required** property is set to **True** in Designer) and the input document does not contain the required attribute, Integration Server adds an empty attribute during document encoding. For example, if the document type contains a required String variable named @myAttribute but @myAttribute is missing from the input document, Integration Server adds myAttribute="" to the XML document.

**Note:** Because empty xmlns attributes are invalid, if the document type contains a required String variable named @xmlns and the input document does not specify a value for the @xmlns attribute, Integration Server *does not* add xmlns="" to the XML document.

- Also note that the \*body key is used to represent the value of a simple element that contains both a text value and an attribute. See the acctNum and phoneNum keys for an example of this kind of element.
- The \*doctype field is used to represent the IS document type to which the IData object conforms. This field is used to populate the relevant value for the xsi:type attribute in the XML string. The document type referred by the \*doctype field should either be created by importing an XSD file (XML schema) or generated while consuming WSDL for a provider or consumer web service descriptor. You should

also select the **Register document types with schema type** option in the New Document Type wizard when generating a document type from an XML schema.

- Fields that are not String or Document based (for example, Floats or Integers) are converted to XML values using the underlying object's `toString` method.

## Input Parameters

### *attrPrefix*

**String** Optional. Prefix that designates keys containing attributes. The default prefix is "@".

For example, if you set *attrPrefix* to `ATT_` and *document* contained the following element:



`pub.xml:documentToXMLString` would convert the `ATT_currency` key to the attribute, `currency=dollars`, in the `<tx>` element as shown below:

```
<tx currency=dollars>
<acct>cash</acct>
<amt>120.00</amt>
</tx>
```

### *document*

**Document** IData object that is to be converted to XML. Note that if you want to produce a valid XML document (one with a single root node), *document* must contain only one top-level IData object (that is, a single document). The name of that document will serve as the name of the XML document's root element.

For example, *document* shown in the example in this service's description contains one top-level document named `AcctInfo`, which would result in one root element named `<AcctInfo>` in the resulting XML String.

If you needed to produce an XML fragment (for example, a loose collection of elements that are not encompassed within a single root element) then *document* can contain multiple top-level elements. To produce this type of output, you must also set the *addHeader* and *enforceLegalXML* parameters to `false`.

### *nsDecls*

**Document** Optional. Namespaces associated with any namespace prefixes that are used in the key names in *document*. Each entry in *nsDecls* represents a namespace prefix/URI pair, where a key name represents a prefix and the value of the key specifies the namespace URI.

For example, to define the URIs associated with two prefixes called `GSX` and `TxMon`, you would set `nsDecls` as follows:



For each prefix specified in `nsDecls`, `pub.xml:documentToXMLString` generates an `xmlns` attribute and inserts it into the top-most element of the resulting XML String. For example, if `nsDecls` had the two keys shown above, `pub.xml:documentToXMLString` would insert the following attributes into the root element of the XML String:

```
xmlns:gsx="http://www.gsx.com"
xmlns:TxMon="http://www.acutrak/txmonitor"
```

**Note:** Alternatively, you can declare a namespace by including an `@xmlns` key in `document`. (If you were not using the `@` character to designate attributes, use the correct attribute prefix in your code.)

*addHeader*

**String** Optional. Flag specifying whether the header element:

```
<?xml version="1.0"?>
```

is to be included in the resulting XML String.

Set to:

- `true` to include the header. This is the default.
- `false` to omit the header. (You would omit the header to generate an XML fragment or to insert a custom header.)

*encode*

**String** Optional. Flag indicating whether to HTML-encode the data. Set this parameter to `true` if your XML data contains special characters, including the following: `<` `>` `&` `"` `'`

Set to:

- `true` to HTML-encode the data.  
For example, the string expression `5 < 6` would be converted to `<expr>5 &lt; 6</expr>`, which is valid.
- `false` to not HTML-encode the data. This is the default.  
For example, the string expression `5 < 6` would be converted to `<expr>5 < 6</expr>`, which is invalid.

*documentTypeName*

**String** Optional. Fully qualified name of the document type that describes the structure and format of the output document (for example, `examples.rtd:exampleRecord1`).

You can use this parameter to ensure that the output includes elements that might not be present in *document* at run time, or to describe the order in which elements are to appear in the resulting XML String.

*generateRequiredTags* **String** Optional. Flag indicating whether empty tags are to be included in the output document if a mandatory element appears in the document type specified in *documentTypeName* but does not appear in *document* . Set to:

- `true` to include mandatory elements if they are not present in *document*.
- `false` to omit mandatory elements if they are not present in *document* . This is the default.

**Note:** The *generateRequiredTags* is only applicable if *documentTypeName* is provided.

*generateNilTags* **String** Optional. Flag indicating whether the resulting XML string includes the attribute `xsi:nil` for elements that are null. Set to:

- `true` to generate the `xsi:nil` attribute for an element if the **Allow null** property for the corresponding field is set to `true` and the field is null in *document* .

**Note:** *generateRequiredTags* must also be set to `true` to generate the `xsi:nil` attribute in the XML String.

- `false` to omit the `xsi:nil` attribute even if a nillable field is, in fact, null. This is the default.

*enforceLegalXML* **String** Optional. Flag indicating whether the service throws an exception when *document* contains multiple root elements or illegal XML tag names. Set to:

- `true` to throw an exception if *document* would produce an XML String containing multiple root elements and/or illegal XML tag names.
- `false` to allow the resulting XML String to contain multiple root elements and/or illegal XML tag names. You would use this setting, for example, to create an XML fragment composed of multiple elements that were not all enclosed within a root element. This is the default.

*dtdHeaderInfo* **Document** Optional. Contents of the DOCTYPE header to be inserted into the XML String. (You can retrieve

this information from an incoming document using [pub.xml:getXMLNodeType.](#))

Key	Description
<i>systemID</i>	<b>String</b> Optional. System identifier for the DTD, if any.
<i>publicID</i>	<b>String</b> Optional. Public identifier for the DTD, if any.
<i>rootNSPrefix</i>	<b>String</b> Optional. Namespace prefix of the <i>rootLocalName</i> , if any.
<i>rootLocalName</i>	<b>String</b> Optional. Local name (excluding the namespace prefix) of the root element.

*bufferSize*

**String** Optional. Initial size (in bytes) of the String buffer that `documentToXMLString` uses to assemble the output XML String. If the String buffer fills up before `documentToXMLString` is finished generating the XML String, it reallocates the buffer, expanding it by this amount each time the buffer becomes full.

If you do not set *bufferSize*, `documentToXMLString` looks to see whether a default buffer size is specified in the following parameter on the server:

```
watt.server.recordToDocument.bufferSize
```

If so, it uses this value to allocate the String buffer. If this parameter is not set, `documentToXMLString` uses a default buffer size of 4096 bytes.

For best performance, you should always set *bufferSize* to a value that closely matches the size of the XML String that you expect `documentToXMLString` to produce. This practice will spare the server from having to enlarge the buffer repeatedly if the XML String is many times larger than the default buffer or if you arbitrarily set *bufferSize* to a value that is too small.

Setting *bufferSize* to an appropriately sized value will also prevent your service from unnecessarily consuming more memory than it needs if the XML String is much smaller than the default buffer size or if you arbitrarily set *bufferSize* to a value that is too large.

## Output Parameters

*xmlData*                      **String** XML String produced from *document* .

### Usage Notes

If you are building an IData that will be converted to an XML String, keep the following points in mind:

- If you want to generate a simple element that contains only a character value, represent it with a String element in *document* as shown in the following:

```

Abc name           Midwest Extreme Sports

```

- If you want to generate an element that contains children, represent with an IData in *document* as shown in the following.

```

Abc address1
├── Abc @country      USA
├── Abc street1       10211 Brook Road
├── Abc city          closed
├── Abc state         OH
└── Abc postalCode    22130

```

To produce attributes, put the attribute values in keys whose name starts with the character(s) specified in *attrPrefix* . For example, if you use the default *attrPrefix* , the names of all keys containing attributes (and only those keys containing attributes) must start with the @ character (for example, @type, @xmlns).

Also, when you include attributes, make sure that keys representing attributes are direct children of the elements in which they are to be applied. For example, if you want to include an xmlns attribute in the <AcctInfo> element in the example shown in the description of this service, you must create a String field named @xmlns in the AcctInfo field within *document* .

- If you want to generate a simple element that contains a character value and one or more attributes, you must represent it as an IData that has one key for each attribute and a key named \*body that contains element's value. For example, if you wanted to produce the following element:

```
<phoneNum cc=011>216-741-7566</phoneNum>
```

You would include the following in *document* :

```

Abc phoneNum
├── Abc @cc          011
└── Abc *body        216-741-7566

```

- To include namespaces, make sure you do the following:

- Include the appropriate namespace prefix in the key names in *document*. For example, to produce an element called `acctNum` that belongs to a namespace that is represented by the "GSX" prefix, you would include a key named `GSX:acctNum` in *document*.
- Define the URIs for the prefixes that appear in *document*. You can do this through *nsDecls* or by including an `@xmlns` key in the element where you want the `xmlns` attribute to be inserted. See the *nsDecls* description above for more information about declaring namespaces.

To return the processed XML document to the client that originally submitted it, invoke [pub.flow:setResponse](#). Keep in mind that you may need to modify the encoding.

To generate the `xsi:nil` attribute for an element in the XML String, the following must be true:

- *documentTypeName* must be provided.
- *generateRequiredTags* and *generateNilTags* must be set to `true`.
- The element must correspond to a field that is nillable. That is, the **Allow null** property must be set to **True**.
- If the element corresponds to a document field, at run time, the document must not contain any content. However, if the document contains a *\*body* field, the [pub.xml:documentToXMLString](#) service generates the `xsi:nil` attribute for the corresponding element if the value of *\*body* is null.

If a document field is nillable and is null at run time, the resulting XML String does not contain any child elements for the element that corresponds to the document.

By default, the [pub.xml:documentToXMLString](#) service uses the prefix "xsi" for the nil attribute, where `xsi` refers to the namespace `http://www.w3.org/2001/XMLSchema-instance`. If *nsDecls* declares a different prefix for this namespaces, the service uses that prefix instead of `xsi`.

---

## pub.xml:freeXMLNode

`WmPublic`. Frees the resources allocated to a given XML node.

You can optionally call this service when you are using a `NodeIterator` to iterate over an XML node and you decide to stop processing the node before reaching the end. By explicitly calling `pub.xml:freeXMLNode`, you immediately free the resources associated with the node instead of waiting for Java garbage collection to do this. Although it is not mandatory to call this service when you finish processing an XML node with a `NodeIterator`, doing so can boost server performance. Note that after you have freed an XML node using this service, the node becomes unstable and should not be used by any subsequent processes.

---

## Input Parameters

*rootNode* XML node or enhanced XML node whose resources you want to release. This parameter supports the following types of input:

- **com.wm.lang.xml.Document** An XML node whose resources you want to release.
- **enhanced XML node** An enhanced XML node whose resources you want to release.

Specify the same type of input that you supplied to `pub.xml:getXMLNodeIterator`.

---

## Output Parameters

None.

---

## pub.xml:getNextXMLNode

WmPublic. Gets the next XML node from a NodeIterator.

---

## Input Parameters

*iterator* **com.wm.app.b2b.util.NodeIterator** NodeIterator from which to retrieve the next node.

---

## Output Parameters

*next* **Document** Conditional. The requested node. Will be null when the NodeIterator has no more nodes to return. Otherwise, *next* will contain the following:

Key	Description
<i>name</i>	<b>String</b> Element type name of the node. If the element belongs to a namespace and the namespace was declared at the time the NodeIterator was constructed, <i>name</i> will have the prefix declared for that namespace. If the namespace is not declared, <i>name</i> will use prefix that occurs in the XML.

*node*

XML node identified by the input criteria used to originally generate the `NodeIterator`. *node* will be one of the following types and depends on what was supplied to the *node* input parameter for the `pub.xml:getXMLNodeIterator` service:

- **com.wm.lang.xml.Node**
- **enhanced XML node**

It is possible that all calls to `getNextXMLNode` on a given `NodeIterator` will yield the same document instance, where the values of the instance's entries vary. For this reason, applications should assume that each call to `getNextXMLNode` invalidates the document returned by the previous call. This approach maximizes the speed of the server and minimizes the use of resources.

### Usage Notes

A `NodeIterator` is acquired via the service [pub.xml:getXMLNodeIterator](#). The output of that service is a document (`IData` object) containing the element type name of the node and the node itself. The instance of this document is only valid until the next `getNextXMLNode` call on the same `NodeIterator`, because `getNextXMLNode` uses the same document object for each call.

---

## pub.xml:getXMLNodeIterator

`WmPublic`. Creates and returns a `NodeIterator`.

A `NodeIterator` iterates over the element node descendants of an XML node and returns the element nodes that satisfy the given criteria. The client application or flow service uses the service [pub.xml:getNextXMLNode](#) to get each node in turn. `NodeIterators` can only be created for XML nodes (not for HTML nodes).

`getXMLNodeIterator` is useful for loading and parsing documents on demand. Large or slow documents need only be loaded as far as needed to get the desired data. `NodeIterators` are also useful for providing service as the pertinent information in the document arrives rather than first waiting for the entire document to load. This service is primarily intended to deal with large documents or documents that arrive slowly.

`NodeIterator` provides a moving-window mode, in which the only node that is resident in memory is the last node returned by [pub.xml:getNextXMLNode](#). In this mode, when [pub.xml:getNextXMLNode](#) is called, all nodes preceding the newly returned node become invalid, including all nodes previously returned by [pub.xml:getNextXMLNode](#). The client must fully complete processing preceding nodes before advancing the window by

calling `pub.xml:getNextXMLNode` again. In moving-window mode, the document consumes at least enough memory to hold the most recently returned node.

Moving-window mode allows the server to process multi-megabyte XML documents using very little memory. Moving-window mode may only be used on a node that represents an entire XML document and not on any descendant node.

**Note:** You can use moving-window mode if the input node is of type `com.wm.lang.xml.Node` only. Moving-window mode cannot be used with an enhanced XML node.

## Input Parameters

*node*

The XML node or enhanced XML node for which you want to produce a `NodeIterator`. This parameter supports the following types of input:

- **com.wm.lang.xml.Node** XML node for which you want to produce a `NodeIterator`. The node can represent either an XML document or an element of an XML document; however, if the `NodeIterator` will be used in moving-window mode, a whole XML document must be used. This is because moving window mode is only meaningful for managing the loading process of a document, and to operate on a node is to have already loaded the node
- **enhanced XML node** An enhanced XML node for which you want to produce a `NodeIterator`. An enhanced XML node can be produced by `pub.xml:loadEnhancedXMLNode`, `pub.xml:xmlStringToEnhancedXMLNode`, or a content handler that receives an XML document in a request for which `xmlFormat` is set to enhanced.

*criteria*

**String List** Optional. Pattern strings identifying the nodes that the iterator is to return. A pattern string may take either the form `<localName>` or the form `<prefix>:<localName>`.

When a pattern takes the first form, it identifies an element whose local name is `<localName>` and that belongs to the default XML namespace. When a pattern takes the second form, it identifies an element whose local name is `<localName>` and whose XML namespace is given by the prefix `<prefix>`. If the input parameter `nsDecls` declares this prefix, the namespace URI of the element must match the URI declared for the prefix. If the prefix is not declared in `nsDecls`, the prefix is matched against prefixes found in the XML.

`<prefix>` and `<localName>` can each optionally take the value `"*"` (asterisk) to match any namespace or local name.

A "\*" prefix also matches elements residing in the default namespace.

If you do not specify *criteria*, all element node children of the root element are returned.

### *nsDecls*

**Document** Optional. Namespaces associated with any namespace prefixes used in *criteria*. Each entry in *nsDecls* represents a namespace prefix/URI pair, where a key name represents a prefix and the value of the key specifies the namespace URI.

For example, to define the URIs associated with two prefixes called `GSX` and `TxMon`, you would set *nsDecls* as follows:



### *movingWindow*

**String** Optional. Flag indicating whether the `NodeIterator` is to iterate using a moving window, as described above. In moving-window mode, the entire document preceding the node most recently returned by `getXMLNodeIterator` is discarded. Subsequent attempts to return preceding portions of the document will return either the repeating text `*PURGED*` or the proper data, depending on whether the data falls within an area that the server was able to discard. When iterating with a moving window, the current node should be queried and completely examined prior to requesting the next node.

Set to:

- `true` to use the `NodeIterator` in moving-window mode.
- `false` to use the `NodeIterator` in normal mode. This is the default.

**Note:** If *node* is an enhanced XML node, set *movingWindow* to `false`.

## Output Parameters

### *iterator*

`com.wm.app.b2b.util.NodeIterator` `NodeIterator` for use with the service [pub.xml:getNextXMLNode](#).

---

## pub.xml:getXMLNodeType

WmPublic. Returns information about an XML node.

### Input Parameters

---

*rootNode*                    **com.wm.lang.xml.Document** XML node about which you want information.

### Output Parameters

---

*systemID*                    **String** Conditional. System identifier, as provided by the DTD associated with *rootNode*. If *rootNode* does not have a system identifier, this value is null.

*publicID*                    **String** Conditional. Public identifier, as provided by the DTD associated with *rootNode*. If *rootNode* does not have a public identifier, this value is null.

*rootNamespace*              **String** URI of the XML namespace to which *rootNode*'s root element belongs.

*rootNSPrefix*               **String** Conditional. Namespace prefix of root element in *rootNode*, if any.

*rootLocalName*              **String** Conditional. Local name (excluding the namespace prefix) of the root element in *rootNode*, if any.

---

## pub.xml:loadEnhancedXMLNode

WmPublic. Retrieves an XML document via HTTP or HTTPS, parses it using the enhanced XML parser, and produces an `org.w3c.dom.Node` object.

An DOM node is a special representation of an XML document that can be consumed by any program that uses standard DOM APIs. The `pub.xml:xmlNodeToDocument` service and the `pub.xml:data.domeNodeToXMLData` service can accept a DOM object as input.

### Input Parameters

---

*url*                            **String** The URL of the document you want to load. This string *must* begin with `http:` or `https:`. For example:

```
http://www.rubicon.com/orders/orders.html
```

—OR—

```
https://localhost:5555/WmPublic/index.html
```

You can include a query string (for example, a collection of "name=value" pairs) with the string that you specify. However, you might want to use the data variable for this type of information instead. It is usually a more practical place for "name=value" data, because it allows you to link individual variables in the query string.

#### *method*

**String** Optional. Set this value to specify the HTTP method (GET or POST) that you want the target server to execute on the resource specified in *url*. This value determines the way in which `pub.xml:loadEnhancedXMLNode` submits data values (if any) to the resource identified in *url*.

- If you set `method` to `get`, `pub.xml:loadEnhancedXMLNode` appends the values you specify in `data` to the value in *url*. (Note that only certain data elements are valid when you use the GET method). The default is `get`.
- If you set `method` to `post`, `pub.xml:loadEnhancedXMLNode` sends the values in `data` in the body of the HTTP or HTTPS request.

#### *auth*

**Document** Optional. Authentication and authorization information that `pub.xml:loadEnhancedXMLNode` will use if the requested resource is protected.

**Note:** If you include your data with the string in *url*, do not specify a value in *data*.

<b>Key</b>	<b>Description</b>
<i>type</i>	<b>String</b> Type of authentication <code>pub.xml:loadEnhancedXMLNode</code> will use to submit this request. Leave this field blank, as the only option currently available is basic HTTP authentication.
<i>user</i>	<b>String</b> User name that <code>pub.xml:loadEnhancedXMLNode</code> will submit if the requested resource is protected.  The user name must have authority to access the resource specified in <i>url</i> .

	<p>This value defaults to the value of <code>watt.net.httpUser</code> in the server's configuration file (<code>server.cnf</code>).</p>
<i>pass</i>	<p><b>String</b> Password associated with the user name specified in <i>user</i> . If the user does not require a password, leave <i>pass</i> empty.</p> <p>This value defaults to the value of <code>watt.net.httpPass</code> in the server's configuration file (<code>server.cf</code>).</p>
<i>data</i>	<p><b>Document</b> Optional. The data that you want <code>pub.xml:loadEnhancedXMLNode</code> to submit with the request. Specify data using one or more of the following elements.</p>

**Note:** When you use more than one element to submit data, *args* is appended first, *table* is appended second, and *string* is appended last.

<b>Key</b>	<b>Description</b>
<i>args</i>	<p><b>Document</b> Optional. Specifies name=value pairs that <code>pub.xml:loadEnhancedXMLNode</code> is to submit to the resource in <i>url</i> .</p> <p>You can use <i>args</i> to submit data via either the POST or GET method.</p> <p>To specify data using <i>args</i> , create one element for each name=value pair that you want to submit, where the key represents the name portion of the pair and the value represents the value portion of the pair.</p> <p>Note that when you use <i>args</i> , <code>pub.xml:loadEnhancedXMLNode</code> will automatically:</p> <ul style="list-style-type: none"> <li>■ URL-encode name=value pair, so you do not need to URL-encode the values you specify in <i>args</i> .</li> <li>■ Insert the "&amp;" character between pairs, so you do not need to include it in <i>args</i> .</li> </ul>

- Prefix the entire query string with the "?" character if it submits the data in *args* via a GET. You do not need to include this character in *args*.

When you submit data using the *args* variable, Integration Server automatically sets the value of the Content-Type header to `application/x-www-form-urlencoded`. If you want to explicitly specify a different Content-Type, you must submit your data using the *string* or *bytes* variable.

### *table*

**String Table** Optional. Specifies data that `pub.xml:loadEnhancedXMLNode` will use to construct a query string to submit to the resource specified in *url*.

*table* is similar to *args*, but it allows you to submit unnamed values in a query string, not just name=value pairs.

To specify data using *table*, create one row for each value that you want to submit, where:

- The contents of column 0 represent the name portion of the pair (leave this column null to submit an unnamed value, and...
- The contents of column 1 represent the value portion of the pair.

When you submit data using the *table* variable, the Integration Server automatically sets the value of the Content-Type header to `application/x-www-form-urlencoded`. If you want to explicitly specify a different Content-Type, you must submit your data using the *string* or *bytes* variable.

Note that when you use *table*, `pub.xml:loadEnhancedXMLNode` will automatically:

- URL-encode name=value pair, so you do not need to URL-encode the values you specify in *table*.

- Insert the "&" character between the pairs (or unnamed values) that it constructs, so you do not need to include it in *table* .
- Prefix the entire query string with the "?" character if it submits the data in *table* via the GET method. You do not need to include this character in *table* .

*string*

**String** Optional. Text that you want `pub.xml:loadEnhancedXMLNode` to submit to the resource in *url* .

You can use *string* to submit data via either the POST or GET method.

If you use *string* to specify your data, make sure that you specify the string *exactly* as you want it presented in the HTTP request. (If you are using the GET method, make sure you URL-encode the contents of *string* ). When performing a POST the *string* is submitted to the resource as the body of the document.

*bytes*

**byte[ ]** Optional. Data that `pub.xml:loadEnhancedXMLNode` is to submit to the resource in *url* . You can use *bytes* only to submit data via the POST method.

**Note** When you use *bytes* and another element (*args* , *table* , or *string* ) to submit data with `pub.xml:loadEnhancedXMLNode`, the service appends the data from the *args* , *table* , or *string* element to *url* . The service appends *args* to *url* first, *table* second, and *string* last. The service encodes the data from the *bytes* element in the body of the post.

*stream*

**java.io.InputStream** Optional. Data that `pub.xml:loadEnhancedXMLNode` is to submit to the resource in *url* . You can use *stream* only to submit data via the POST method.

**Note** When you use *stream* and another element (*args*, *table*, or *string*) to submit data with `pub.xml:loadEnhancedXMLNode`, the service appends the data from the *args*, *table*, or *string* element to *url*. The service appends *args* to *url* first, *table* second, and *string* last. The service encodes the data from the *stream* element in the body of the post. If *stream* is specified, *bytes* is ignored.

*encoding*

**String** Optional. Name of a registered IANA character set.

*headers*

**Document** Optional. Fields that you want to explicitly override in the HTTP request header issued by `pub.xml:loadEnhancedXMLNode`.

Specify one element for each header field that you want to set, where the element's name represents the name of the header field, and the element's value represents the value of that header field.

If *headers* is not set, `pub.xml:loadEnhancedXMLNode` will use its default header values.

**Note:** You do not need to type a colon after the field name because `pub.xml:loadEnhancedXMLNode` will automatically insert the colon when it inserts this field into the request header.

If you want to assign specific values to header fields used by `pub.xml:loadEnhancedXMLNode`, keep the following points in mind:

- When you specify the value of a header field, you override whatever default value webMethods Integration Server is configured to use for HTTP requests. For example, if you set the User-Agent header field to `B2B/3.0`, the server uses that value instead of the default value specified by the `watt.net.userAgent` parameter.
- The `pub.xml:loadEnhancedXMLNode` service automatically determines the value of the Content-Length header field. You cannot specify a value for Content-Length.
- Be aware that when you submit data using the *args* or *table* elements, `pub.xml:loadEnhancedXMLNode` automatically sets the Content-Type header field to `application/x-www-`

`formurlencoded`. You cannot override this setting using the *headers* variable. If you want to explicitly specify a content type in headers, make sure to use the *string* or *bytes* element to submit your data, not *args* or *table*.

- Certain header fields are automatically derived from other input parameters assigned to `pub.xml:loadEnhancedXMLNode`. For example, the Authorization header field is automatically derived from your *auth* parameter setting. Except for the Content-Length header field and the Content-Type header field (which, as described above, you cannot override when submitting data via *args* or *table*), a value that you specify in headers overrides the value that `pub.xml:loadEnhancedXMLNode` might otherwise derive from other parameter settings.
- The `pub.xml:loadEnhancedXMLNode` service does not validate data that you specify in *headers*. It simply passes it on to the target server in the request header. *Make sure you specify header field names and their values correctly.* For a complete list of valid request header fields, see <http://www.w3.org> for the latest HTTP specification published by the W3C.
- To specify request headers in *headers*, create a string element for each header that you want to specify, where:
  - The name of the element defines the name header field (for example, User-Agent, If-Modified-Since, Mail\_Address), and...
  - The value of the element specifies the value you want assigned to that field.

#### *encoding*

**String** Optional. Character set in which the returned document is encoded. The parser requires this value in order to interpret a document correctly. Set to:

- `autoDetect` to determine the document's character set from the document, where UTF-8 is the default character set for XML.
- The name of a registered IANA character set to decode the document using that character set (for example, ISO-8859-1).

#### *failOnHTTPError*

**String** Optional. Determines whether `pub.xml:loadEnhancedXMLNode` will fail (throw an exception) if the requested URL is not loaded correctly based on an HTTP status code. This parameter allows for customized error handling of the load failure. Set to:

- `true` to throw a service exception if the URL is not loaded as indicated by an HTTP status code between 400 and 599, inclusive.
- `false` to ignore HTTP errors. If there is an error, `pub.xml:loadEnhancedXMLNode` returns `status` and `statusMessage`. This is the default.

*inputProcessing*

**Document.** Optional. Contains a set of input parameters that instruct Integration Server how to read the XML document. The fields are comparable to those in the `javax.xml.stream.XMLInputFactory` class.

Key	Description
<i>isValidating</i>	<p><b>String</b> Optional. Determines whether Integration Server performs DTD validation. Set to:</p> <ul style="list-style-type: none"> <li>■ <code>true</code> to perform DTD validation.</li> <li>■ <code>false</code> to disable DTD validation. This is the default.</li> </ul>
<i>isNamespaceAware</i>	<p><b>String</b> Optional. Determines whether Integration Server provides namespace processing for XML 1.0 support while parsing the XML document. Set to:</p> <ul style="list-style-type: none"> <li>■ <code>true</code> to enable namespace processing. This is the default.</li> <li>■ <code>false</code> to disable namespace processing.</li> </ul>
<i>isCoalescing</i>	<p><b>String</b> Optional. Determines whether Integration Server coalesces adjacent character data while parsing the XML document. Set to:</p> <ul style="list-style-type: none"> <li>■ <code>true</code> to coalesce adjacent character data.</li> <li>■ <code>false</code> to indicate that Integration Server does not coalesce adjacent character data. This is the default.</li> </ul>
<i>isReplacingEntityReferences</i>	<p><b>String</b> Optional. Determines whether, while parsing the XML document, Integration Server replaces internal</p>

entity references with replacement text and treats them as characters. Set to:

- `true` to replace entity references. This is the default.
- `false` to indicate entity references will not be replaced.

*isSupportingExternalEntities*

**String** Optional. Determines whether Integration Server resolves external parsed entities while parsing the XML document. Set to:

- `true` to resolve external parsed entities.
- `false` to indicate Integration Server does not resolve external parsed entities.

The JVM in which Integration Server runs determines the default.

*supportDTD*

**String** Optional. Determines whether Integration Server supports DTDs while parsing the XML document. Set to:

- `true` to support DTDs while parsing the XML document. This is the default.
- `false` to disable support of DTDs while parsing the XML document.

*partitionSize*

**String** Optional. Specifies the size, measured in bytes, of the partitions on the heap where the enhanced XML parser stores parsed document information. Specify a suffix of “k” to indicate kilobytes or “m” to indicate megabytes. For example, 10k or 10m.

If you do not specify a value, Integration Server uses the default partition size value specified on the **Settings > Enhanced XML Parsing** screen in Integration Server Administrator.

## Output Parameters

---

*node*

**org.w3c.dom.Node** Conditional. XML node representing the returned ML document.

The `pub.xml:loadEnhancedXMLNode` service returns *node* only when Integration Server parses the XML document successfully.

*status*

**String** Conditional. The HTTP status code returned by the target server if an HTTP error occurs when loading the requested URL.

The `pub.xml:loadEnhancedXMLNode` service returns *status* when an HTTP error occurs and *failOnHTTPError* is set to false.

*statusMessage*

**String** Conditional. The HTTP status message returned by the target server if an HTTP error occurs when loading the requested URL.

The `pub.xml:loadEnhancedXMLNode` service returns *statusMessage* when an HTTP error occurs and *failOnHTTPError* is set to false.

## Usage Notes

If `pub.xml:loadEnhancedXMLNode` does not receive a response within the timeout period specified in the server's `watt.net.timeout` parameter, it will throw an exception. For information about the `watt.net.timeout` parameter, see *webMethods Integration Server Administrator's Guide*.

Use the `pub.xml:loadXMLNode` service to load an XML document and convert it to an XML node using the legacy XML parser. For more information about the legacy XML parser and the enhanced XML parser, see *webMethods Integration Server Administrator's Guide*.

Keep the following information in mind when specifying a *partitionSize*

- The *partitionSize* is a hint for the enhanced XML parser so that it can estimate the amount of heap space needed to parse the document. Often, it not possible to determine the size of an inbound XML document before parsing.
- As a general rule, Software AG recommends a *partitionSize* that is 1/2 the size of the unparsed XML document.
  - A *partitionSize* that is considerably larger than 1/2 the size of the unparsed XML document causes the enhanced XML parser to consume more heap space than necessary but might also improve throughput. However, this can impact the overall performance of Integration Server.
  - A *partitionSize* that is considerably smaller than 1/2 the size of the unparsed XML document causes the enhanced XML parser to create a large number of partitions to parse the document. While this might use less heap space, it may reduce the throughput of the parser.
  - A *partitionSize* that is three times smaller or three times larger than 1/2 the size of the unparsed XML document will likely have little impact on the performance.

- At run time, the enhanced XML parser overrides a *partitionSize* that consumes all of the available heap space.
- At run time, if the *partitionSize* results in an initial heap allocation that exceeds the single document limit set in the **Maximum heap allocation for any single document** field the limit for all documents set in the **Maximum heap allocation for all documents combined** field, the enhanced XML parser reduces the partition size automatically. For more information about heap allocation limits for the enhanced XML parser, see *webMethods Integration Server Administrator's Guide*
- If you do not specify *partitionSize*, the enhanced XML parser uses the default specified in the **Default partition size** field on the **Settings > Enhanced XML Parsing** page in Integration Server Administrator.

---

## pub.xml:loadXMLNode

WmPublic. Retrieves an XML document via HTTP or HTTPS, parses it using the legacy XML parser, and produces an XML node.

An XML node is a special representation of an XML document that can be consumed by the Integration Server. Most webMethods services that operate on XML documents require an XML node as input.

### Input Parameters

---

<i>url</i>	<p><b>String</b> The URL of the document you want to load. This string <i>must</i> begin with <code>http:</code> or <code>https:</code>. For example:</p> <pre>http://www.rubicon.com/orders/orders.html</pre> <p>—OR—</p> <pre>https://localhost:5555/WmPublic/index.html</pre> <p>You can include a query string (for example, a collection of "name=value" pairs) with the string that you specify. However, you might want to use the data variable for this type of information instead. It is usually a more practical place for "name=value" data, because it allows you to link individual variables in the query string.</p>
<i>method</i>	<p><b>String</b> Optional. Set this value to specify the HTTP method (GET or POST) that you want the target server to execute on the resource specified in <i>url</i>. This value determines the way in which <code>pub.xml:loadXMLNode</code> submits data values (if any) to the resource identified in <i>url</i>. The default is <code>get</code>.</p> <ul style="list-style-type: none"> <li>■ If you set <i>method</i> to <code>get</code>, <code>pub.xml:loadXMLNode</code> appends the values you specify in <i>data</i> to the value in <i>url</i>. (Note that</li> </ul>

only certain data elements are valid when you use the GET method).

- If you set `method` to `post`, `pub.xml:loadXMLNode` sends the values in `data` in the body of the HTTP or HTTPS request.

*auth*

**Document** Optional. Authentication and authorization information that `pub.xml:loadXMLNode` will use if the requested resource is protected.

**Note:** If you include your data with the string in `url`, do not specify a value in `data`.

Key	Description
<i>type</i>	<b>String</b> Type of authentication <code>pub.xml:loadXMLNode</code> will use to submit this request. Leave this field blank, as the only option currently available is basic HTTP authentication.
<i>user</i>	<b>String</b> User name that <code>pub.xml:loadXMLNode</code> will submit if the requested resource is protected.  The user name must have authority to access the resource specified in <code>url</code> .  This value defaults to the value of <code>watt.net.httpUser</code> in the server's configuration file ( <code>server.cnf</code> ).
<i>pass</i>	<b>String</b> Password associated with the user name specified in <code>user</code> . If the user does not require a password, leave <code>pass</code> empty.  This value defaults to the value of <code>watt.net.httpPass</code> in the server's configuration file ( <code>server.cf</code> ).

*data*

**Document** Optional. The data that you want `pub.xml:loadXMLNode` to submit with the request. Specify data using one or more of the following elements.

**Note:** When you use more than one element to submit data, `args` is appended first, `table` is appended second, and `string` is appended last.

Key	Description
<i>args</i>	<p><b>Document</b> Optional. Specifies name=value pairs that <code>pub.xml:loadXMLNode</code> is to submit to the resource in <i>url</i>.</p> <p>You can use <i>args</i> to submit data via either the POST or GET method.</p> <p>To specify data using <i>args</i>, create one element for each name=value pair that you want to submit, where the key represents the name portion of the pair and the value represents the value portion of the pair.</p> <p>Note that when you use <i>args</i>, <code>pub.xml:loadXMLNode</code> will automatically:</p> <ul style="list-style-type: none"> <li>■ URL-encode name=value pair, so you do not need to URL-encode the values you specify in <i>args</i>.</li> <li>■ Insert the "&amp;" character between pairs, so you do not need to include it in <i>args</i>.</li> <li>■ Prefix the entire query string with the "?" character if it submits the data in <i>args</i> via a GET. You do not need to include this character in <i>args</i>.</li> </ul> <p>When you submit data using the <i>args</i> variable, the Integration Server automatically sets the value of the Content-Type header to <code>application/x-www-form-urlencoded</code>. If you want to explicitly specify a different Content-Type, you must submit your data using the <i>string</i> or <i>bytes</i> variable.</p>
<i>table</i>	<p><b>String Table</b> Optional. Specifies data that <code>pub.xml:loadXMLNode</code> will use to construct a query string to submit to the resource specified in <i>url</i>.</p> <p><i>table</i> is similar to <i>args</i>, but it allows you to submit unnamed values in a query string, not just name=value pairs.</p>

To specify data using *table*, create one row for each value that you want to submit, where:

- The contents of column 0 represent the name portion of the pair (leave this column null to submit an unnamed value, and...
- The contents of column 1 represent the value portion of the pair.

When you submit data using the *table* variable, the Integration Server automatically sets the value of the Content-Type header to `application/x-www-form-urlencoded`. If you want to explicitly specify a different Content-Type, you must submit your data using the *string* or *bytes* variable.

Note that when you use *table*, `pub.xml:loadXMLNode` will automatically:

- URL-encode name=value pair, so you do not need to URL-encode the values you specify in *table*.
- Insert the "&" character between the pairs (or unnamed values) that it constructs, so you do not need to include it in *table*.
- Prefix the entire query string with the "?" character if it submits the data in *table* via the GET method. You do not need to include this character in *table*.

*string*

**String** Optional. Text that you want `pub.xml:loadXMLNode` to submit to the resource in *url*.

You can use *string* to submit data via either the POST or GET method.

If you use *string* to specify your data, make sure that you specify the string *exactly* as you want it presented in the HTTP request. (If you are using the GET method, make sure you URL-encode the contents of *string*). When performing

a POST the *string* is submitted to the resource as the body of the document.

*bytes*

**byte[ ]** Optional. Data that `pub.xml:loadXMLNode` is to submit to the resource in *url*. You can use *bytes* only to submit data via the POST method.

**Note**When you use *bytes* and another element (*args*, *table*, or *string*) to submit data with `pub.xml:loadXMLNode`, the service appends the data from the *args*, *table*, or *string* element to *url*. The service appends *args* to *url* first, *table* second, and *string* last. The service encodes the data from the *bytes* element in the body of the post.

*stream*

**java.io.InputStream** Optional. Data that `pub.xml:loadXMLNode` is to submit to the resource in *url*. You can use *stream* only to submit data via the POST method.

**Note**When you use *stream* and another element (*args*, *table*, or *string*) to submit data with `pub.xml:loadXMLNode`, the service appends the data from the *args*, *table*, or *string* element to *url*. The service appends *args* to *url* first, *table* second, and *string* last. The service encodes the data from the *stream* element in the body of the post. If *stream* is specified, *bytes* is ignored.

*encoding*

**String** Optional. Name of a registered IANA character set.

*headers*

**Document** Optional. Fields that you want to explicitly override in the HTTP request header issued by `pub.xml:loadXMLNode`.

Specify one element for each header field that you want to set, where the element's name represents the name of the header field, and the element's value represents the value of that header field.

If *headers* is not set, `pub.xml:loadXMLNode` will use its default header values.

**Note:** You do not need to type a colon after the field name because `pub.xml:loadXMLNode` will automatically insert the colon when it inserts this field into the request header.

If you want to assign specific values to header fields used by `pub.xml:loadXMLNode`, keep the following points in mind:

- When you specify the value of a header field, you override whatever default value webMethods Integration Server is configured to use for HTTP requests. For example, if you set the User-Agent header field to `B2B/3.0`, the server uses that value instead of the default value specified by the `watt.net.userAgent` parameter.
- The `pub.xml:loadXMLNode` service automatically determines the value of the Content-Length header field. You cannot specify a value for Content-Length.
- Be aware that when you submit data using the `args` or `table` elements, `pub.xml:loadXMLNode` automatically sets the Content-Type header field to `application/x-www-form-urlencoded`. You cannot override this setting using the `headers` variable. If you want to explicitly specify a content type in headers, make sure to use the `string` or `bytes` element to submit your data, not `args` or `table`.
- Certain header fields are automatically derived from other input parameters assigned to `pub.xml:loadXMLNode`. For example, the Authorization header field is automatically derived from your `auth` parameter setting. Except for the Content-Length header field and the Content-Type header field (which, as described above, you cannot override when submitting data via `args` or `table`), a value that you specify in headers overrides the value that `pub.xml:loadXMLNode` might otherwise derive from other parameter settings.
- The `pub.xml:loadXMLNode` service does not validate data that you specify in `headers`. It simply passes it on to the target server in the request header. *Make sure you specify header field names and their values correctly.* For a complete list of valid request header fields, see <http://www.w3.org> for the latest HTTP specification published by the W3C.
- To specify request headers in `headers`, create a string element for each header that you want to specify, where:
  - The name of the element defines the name header field (for example, User-Agent, If-Modified-Since, Mail\_Address), and...

- The value of the element specifies the value you want assigned to that field.

*encoding*

**String** Optional. Character set in which the returned document is encoded. The parser requires this value in order to interpret a document correctly. Set to:

- `autoDetect` to determine the document's character set based on document type, where:
  - ISO-8859-1 is used for HTML.
  - UTF-8 is used for XML.
- The name of a registered IANA character set to decode the document using that character set (for example, ISO-8859-1).

If you do not specify an *encoding* value, `pub.xml:loadXMLNode` decodes the returned document using the following defaults:

<u>If the document is...</u>	<u>It is decoded using...</u>
HTML	ISO-8859-1
XML	UTF-8

*expandDTD*

**String** Optional. Flag indicating whether or not `pub.xml:loadXMLNode` is to process references to parameter entities in the returned document's DTD. Set to:

- `true` to expand references to parameter entities to their full definition.
- `false` to ignore references to parameter entities. This is the default.

**Note:** You might want or need to use this variable in cases when you have a syntactically correct document that causes a parse error because it violates a definition in an external parameter-entity reference. By setting *expandDTD* to false, you can bypass the external definition so that `pub.xml:loadXMLNode` can parse the document successfully

*isXML*

**String** Optional. Flag indicating whether the returned document is XML or HTML. `pub.xml:loadXMLNode` must know this in order to parse a document correctly. Set to:

- `autoDetect` to parse the document based on its type. When you use this option, `pub.xml:loadXMLNode` senses the document's type based on its `<!DOCTYPE . . . \>` or `<?XML . . . \>` tag. If it cannot determine a document's type, it parses it as HTML. This is the default.
- `true` to parse the document as XML.
- `false` to parse the document as HTML.

**Note:** : If you know what type of document `pub.xml:loadXMLNode` will receive, Software AG recommends that you explicitly set `isXML` instead of using `autoDetect`. It will cut processing time, because the server will not have to examine the document to determine its type. The default value is `autoDetect`.

#### *loadAs*

**String** Optional. Flag that specifies the form in which you want `pub.xml:loadXMLNode` to make the parsed document available to subsequent services. Set to:

- `bytes` to make the document available as a byte array. This is the default.

Use this option if the document will be used as input to a service that operates on whole documents (for example, [pub.xml:queryXMLNode](#)).

- `stream` to make the document available as an `InputStream`.

Use this option if the document will be used as input to a service that can process a document incrementally (for example, [pub.xml:getXMLNodeIterator](#)).

#### *failOnHTTPError*

**String** Optional. Determines whether `pub.xml:loadXMLNode` will fail (throw an exception) if the requested URL is not loaded correctly based on an HTTP status code. This parameter allows for customized error handling of the load failure. Set to:

- `true` to throw a service exception if the URL is not loaded as indicated by an HTTP status code between 400 and 599, inclusive.
- `false` to ignore HTTP errors. If there is an error, the HTML page returned by the web server will be sent to the parser. This is the default.

#### *expandGeneralEntities*

**String** Optional. Flag indicating whether `pub.xml:pub.xml:loadXMLNode` should expand references to general entities in the returned document's DTD. Set to:

- `true` to expand references to general entities to their full definition. This is the default.
- `false` to ignore references to general entities.

---

## Output Parameters

*node* **com.wm.lang.xml.Node** XML node representing the returned HTML or XML document.

### Usage Notes

If `pub.xml:loadXMLNode` does not receive a response within the timeout period specified in the server's `watt.net.timeout` parameter, it will throw an exception. For information about the `watt.net.timeout` parameter, see *webMethods Integration Server Administrator's Guide*.

If `expandGeneralEntities` is not specified, Integration Server uses the value in `watt.core.xml.expandGeneralEntities`. If `watt.core.xml.expandGeneralEntities` is not set, the references to general entities are always expanded.

Use the `pub.xml:loadEnhancedXMLNode` service to load an XML document and convert it to an XML node using the enhanced XML parser. For more information about the legacy XML parser and the enhanced XML parser, see *webMethods Integration Server Administrator's Guide*.

---

## pub.xml:queryXMLNode

WmPublic. Queries an XML node.

The *fields* parameter specifies how data is extracted from the node to produce an output variable. This output variable is called a "binding," because the *fields* parameter binds a certain part of the document to a particular output variable. At run time, this service must include at least one *fields* entry. The service must include at least one entry in *fields*. The result of each query you specify in *fields* is returned in a variable whose name and type you specify.

---

### Input Parameters

*node* The XML node or enhanced XML node that you want to query. This parameter supports the following types of input:

- **com.wm.lang.xml.Node** XML node that you want to query. An XML node can be produced by `pub.xml:loadXMLNode`, `pub.xml:xmlStringToXMLNode` or an XML content handler.

- enhanced XML node** The enhanced XML node that you want to query. An enhanced XML node can be produced by `pub.xml:loadEnhancedXMLNode`, `pub.xml:xmlStringToEnhancedXMLNode`, or an XML content handler that receives a document with an `xmlFormat` is set to enhanced. If you supply an enhanced XML node, you must use XQL to query the node.

*nsDecls*

**Document** Optional. Namespaces associated with any namespace prefixes used element to specify elements in *fields/query*. Each entry in *nsDecls* represents a namespace prefix/URI pair, where a key name represents a prefix and the value of the key specifies the namespace URI.

For example, to define the URIs associated with two prefixes called `GSX` and `TxMon`, you would set *nsDecls* as follows:



*fields*

**Document List** Optional. Parameters describing how data is to be extracted from *node*. Each document in the list contains parameters for a single query, as follows:

<u>Key</u>	<u>Description</u>												
<i>name</i>	<b>String</b> Name to assign to resulting value.												
<i>resultType</i>	<b>String</b> Object type that the query is to yield. The following shows the allowed values for <i>resultType</i> .												
	<table border="1"> <thead> <tr> <th><u>Underlying Value</u></th> <th><u>Corresponding Data Type</u></th> </tr> </thead> <tbody> <tr> <td>Object</td> <td>Object</td> </tr> <tr> <td>Object[]</td> <td>Object List</td> </tr> <tr> <td>Record</td> <td>Document</td> </tr> <tr> <td>Record[]</td> <td>Document List</td> </tr> <tr> <td>String</td> <td>String</td> </tr> </tbody> </table>	<u>Underlying Value</u>	<u>Corresponding Data Type</u>	Object	Object	Object[]	Object List	Record	Document	Record[]	Document List	String	String
<u>Underlying Value</u>	<u>Corresponding Data Type</u>												
Object	Object												
Object[]	Object List												
Record	Document												
Record[]	Document List												
String	String												

	String[]	String List
	String[][]	String Table
<i>query</i>		<b>String</b> Query identifying the data to be extracted from <i>node</i> .
<i>queryType</i>		<b>String</b> Query language in which <i>query</i> is expressed. Valid values are WQL and XQL.  If the content of <i>node</i> is an enhanced XML node, you must set <i>queryType</i> to XQL.
<i>onnull</i>		<b>String</b> Code indicating what you want queryXmlNode to do when the result is null. Set to one of the following: <ul style="list-style-type: none"> <li>■ <code>continue</code> to indicate that all result values are acceptable for this query (including null).</li> <li>■ <code>fail</code> to indicate that the service should fail if the result of this query is null and continue in all other cases.</li> <li>■ <code>succeed</code> to indicate that the service should continue if the result of this query is null and fail in all other cases.</li> </ul>
<i>fields</i>		<b>Document List</b> Parameters that support recursive execution of bindings. Each <i>fields</i> list defines bindings for one level of the output with the top level being the pipeline and the first level down being contents of a document or document list in the pipeline.

## Output Parameters

Document Results from the queries specified in *fields*. This service returns one element for each query specified in *fields*. The specific names and types of the returned elements are determined by the *fields/name* and *field/resultType* parameters of the individual queries.

## Usage Notes

If `pub.xml:queryXmlNode` fails, it throws a server exception. Common reasons for `pub.xml:queryXmlNode` to fail include:

- A variable that has no query string assigned to it.
- A syntax error in a query string.
- A query fails the “Allows Nulls” test.
- The node variable does not exist or it is null.

## pub.xml:xmlNodeToDocument

`WmPublic`. Converts an XML node to a document (an `IData` object).

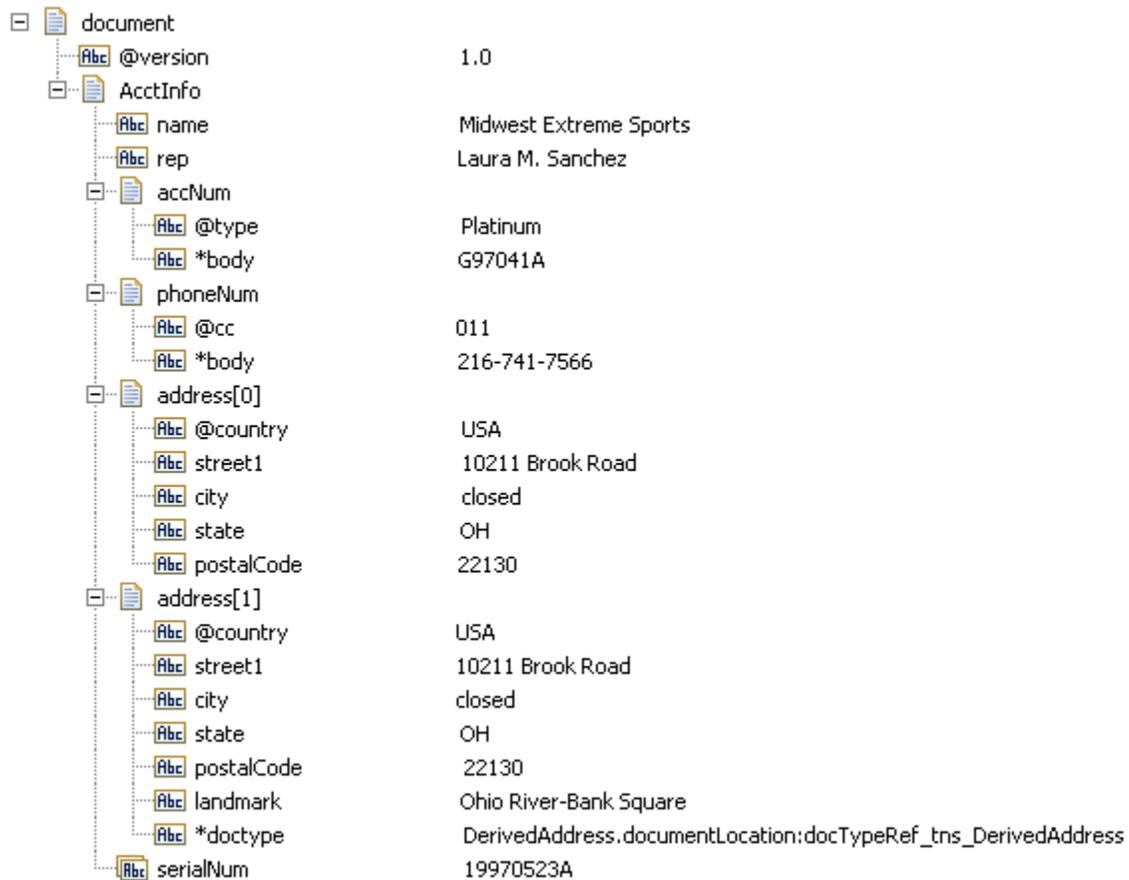
If you want to convert an XML node to an `XMLData` document, use `pub.xmldata.domeNodetoXMLData`.

This service transforms each element and attribute in the XML node to an element in an `IData` object. For example:

### This service would convert this XML document...

```
<?xml version="1.0" ?>
  <tns:AcctInfo>
    xmlns:tns="http://localhost/DerivedAddress/schema.xsd"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" >
    <name>Midwest Extreme Sports</name>
    <rep>Laura M. Sanchez</rep>
    <acctNum type=platinum>G97041A</acctNum>
    <phoneNum cc=011>216-741-7566</phoneNum>
    <address country=USA>
      <street1>10211 Brook Road</street1>
      <city>Cleveland</city>
      <state>OH</state>
      <postalCode>22130</postalCode>
    </address>
    <address country=USA xsi:type="tns:DerivedAddress">
      <street1>10211 Brook Road</street1>
      <city>Cleveland</city>
      <state>OH</state>
      <postalCode>22130</postalCode>
      <landMark>Besides Ohio River-Bank Square</landMark>
      <telNo>001222555</telNo>
    </address>
    <serialNum>19970523A</serialNum>
    <serialNum>20001106G</serialNum>
    <serialNum>20010404K</serialNum>
  </tns:AcctInfo>
```

### To an IData that looks like this...



Note that:

- The XML version attribute is converted to an element named `@version`.
- The resulting document is given the same name as the XML document's root element (`AcctInfo` in the example above) and is a child of the `document` variable that this service returns.
- Simple elements (such as `<name>` and `<rep>` in the example above) are converted to String elements.
- Complex elements (that is, elements with children, such as `<address>` in the example above) and simple elements that have attributes (such as `<acctNum>` and `<phoneNum>`) are converted to documents (IData objects). Note that keys derived from attributes are prefixed with a "@" character to distinguish them from keys derived from elements. Also note that when a simple element has an attribute, its value is placed in an element named `*body`.
- Repeated elements (such as `<serialNum>`) can be collected into arrays using the `makeArrays` and/or `arrays` parameters. See `makeArrays` and `arrays` below for additional information about producing arrays.

- While creating a document, the `pub.xml:xmlNodeToDocument` service assigns a value of `emptyString` to the fields that are empty in the document.
- The `*doctype` field is used to specify the IS document type to which the `IData` object should conform. The `*doctype` field in the `IData` object contains the full namespace name of the IS document type corresponding to the document type referred by the `xsi:type` attribute in the XML string.

The document type referred by the `*doctype` field should either be created by importing an XSD file (XML schema) or generated while consuming WSDL for a provider or consumer web service descriptor. You should also select the **Register document types with schema type** option in the New Document Type wizard when generating a document type from an XML schema.

## Input Parameters

*node*

XML node that is to be converted to a document (`IData` object).

This parameter supports the following types of input:

- `com.wm.lang.xml.Node`
- `org.w3c.dom.Node`

*attrPrefix*

**String** Optional. Prefix that is to be used to designate keys containing attribute values. The default is "@". For example, if you set *attrPrefix* to `ATT_` and *node* contained the following element:

```
<tx currency=dollars>
<acct>cash</acct>
<amt>120.00</amt>
</tx>
```

`xmlNodeToDocument` would convert the `currency` attribute as follows:



*arrays*

**String List** Optional. Names of elements that are to be generated as arrays, regardless of whether they appear multiple times in *node*. For example, if *arrays* contained the following values for the XML document shown in the example in the description for this service:

```
rep
address
```

`xmlNodeToDocument` would generate element `rep` as a String List and element `address` as a Document List.

**Important:** If you include namespace prefixes in the element names that you specify in *arrays*, you must define the namespaces associated with those prefixes in *nsDecls*.

### *makeArrays*

**String** Optional. Flag indicating whether you want `xmlNodeToDocument` to automatically create an array for every element that appears in *node* more than once. Set to:

- `true` to automatically create arrays for every element that appears more than once in *node*. This is the default.
- `false` to create arrays for only those elements specified in *arrays* or defined as arrays in the document type specified in *documentTypeName*.

**Important:** You must set *makeArrays* to `false` when using *documentTypeName* to define the structure of an element. Otherwise, an exception will be thrown at run time.

### *collect*

**Document** Optional. Elements that are to be placed into a new, named array (that is, a "collection"). Within *collect*, use key names to specify the names of the elements that are to be included in the collection. Then set the value of each key to specify the name of the collection in which you want that element placed. For example, if you wanted to place the `<name>` and `<rep>` elements in an array called *originator*, you would set *collect* as follows:

Key	Value
<i>name</i>	<i>originator</i>
<i>rep</i>	<i>originator</i>

If the set of elements in a collection are all simple elements, a String List is produced. However, if the set is made up of complex elements, or a combination of simple and complex elements, a Document List is produced. When this is the case, each member of the array will include a child element called *\*name* that contains the name of the element from which that member was derived.

You may optionally include namespace prefixes in the element names that you specify in *collect*; however, if you

do, you must define the namespaces associated with those prefixes in *nsDecls*.

**Important:** You cannot include an element in more than one collection.

*nsDecls*

**Document** Optional. Namespace prefixes to use for the conversion. This parameter specifies the prefixes that will be used when namespace-qualified elements are converted to key names in the resulting IData object. For example, if you want elements belonging to a particular namespace to have the prefix GSX in the resulting IData (for example, `GSX:acctNum`), you would associate the prefix GSX with that namespace in *nsDecls*. (This is important because incoming XML documents can use any prefix for a given namespace, but the key names expected by a target service or MAP step on the Integration Server will have a fixed prefix.)

Namespace prefixes in *nsDecls* also define the prefixes used by the *arrays*, *documents*, *documentTypeName*, and *collect* parameters.

Each entry in *nsDecls* represents a namespace prefix/URI pair, where a key name represents a prefix and the value of the key specifies the namespace URI.

For example, to define the URIs associated with two prefixes called GSX and TxMon, you would set *nsDecls* as follows:

```

nsDecls
├── abc GSX http://www.gsx.com
└── abc TxMon http://www.acutrak/txmonitor

```

*documents*

**String List** Optional. Names of any simple elements that are to be generated as documents (IData objects) instead of Strings. The document produced for each element specified in *documents* will have the same name as the source element from which it is derived. It will contain a String element named *\*body* that holds the element's value.

For example, if *documents* contained the Strings *name* and *rep* and the source document contained the following:

```

.
.
.
<name>Midwest Extreme Sports</name>
<rep>Laura M. Sanchez</rep>
.
.
.

```

`xmlNodeToDocument` would produce the following:



**Note:** If you include namespace prefixes in the element names that you specify, you must define the namespaces associated with those prefixes in *nsDecls* .

*documentTypeName*

**String** Optional. Fully qualified name of the document type that specifies the structure that is to be imposed on the resulting document. You can use this parameter to explicitly specify the order and dimensionality of elements. It is an alternative to using *makeArrays* and *arrays* to specify which elements are to be generated as arrays.

For example, if you had the XML document shown in the example in this service's description, and you wanted the `<name>` and `<rep>` elements to be generated as String lists, you would define them as String Lists fields in a document type and then specify that document type in *documentTypeName* .

**Note:** The document type specified in *documentTypeName* does not need to specify every element that will appear in the resulting document. It only needs to specify the elements whose structure you want to explicitly set. However, if you include namespace prefixes in the element names that you specify, you must define the namespaces associated with those prefixes in *nsDecls* .

This service always converts XML nodes to String or Document object fields. It does not generate constrained objects (for example, Floats or Integers), even if the fields in the specified document are defined as constrained objects.

**Important** When you use *documentTypeName* , set *makeArrays* to false and do not set *arrays* and *documents* . Otherwise, `xmlNodeToDocument` will throw an exception at run time.

*mixedModel*

**String** Optional. Flag specifying how mixed-content elements (elements containing both text values and child elements) are to be converted. The following is an example of a mixed-content element:

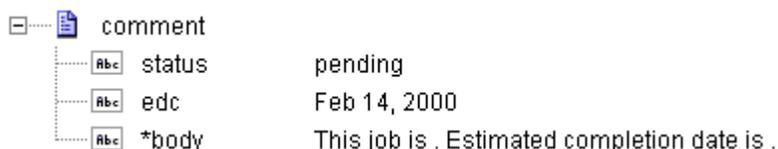
```

<comment>
  This job is <status>pending</status>. Estimated
  completion date is <edc>Feb 14, 2000</edc>.
</comment>

```

Set to:

- `true` to place top-level text in an element named `*body`. This setting would produce the following IData for the `<comment>` element shown above:



**Important** When you set `mixedModel` to `true`, you must also use `documentTypeName` to specify a document type that describes the structure of the IData that you want `xmlNodeToDocument` to produce. Within the document type, mixed-content elements must be defined as documents that include a String field named `*body`.

- `false` to omit top-level text and include only the child elements from mixed-content elements. This setting would produce the following IData for the `<comment>` element shown above:



*preserveUndeclaredNS*

**String** Optional. Flag indicating whether or not Integration Server keeps undeclared namespaces in the resulting document (IData). An undeclared namespace is one that is not specified as part of the `nsDecls` input parameter.

Set to:

- `True` to preserve undeclared namespaces in the resulting document. For each namespace declaration in the XML document that is not specified in the `nsDecls` parameter, Integration Server adds the `xmlns` attribute as a String variable to the document (IData). Integration Server gives the variable a name that begins with "@xmlns" and assigns the variable the namespace value specified in the XML document. Integration Server preserves the position of the undeclared namespace in the resulting document.
- `False` to ignore namespace declarations in the XML document that are not specified in the `nsDecls` parameter. This is the default.

*preserveNSPositions* **String** Optional. Flag indicating whether or not Integration Server maintains the position of namespaces declared in the *nsDecls* parameter in the resulting document.

Set to:

- `True` to preserve the position of namespaces declared in *nsDecls* in the resulting document. For each namespace specified in the *nsDecls* parameter, Integration Server adds the `xmlns` attribute to the document (IData) as a String variable named "`@xmlns:NSprefix`" where "`NSprefix`" is the prefix name specified in *nsDecls*. Integration Server assigns the variable the namespace value specified in the XML document. This variable maintains the position of the `xmlns` attribute declaration within the XML document.
- `False` to not maintain the position of the namespace declarations specified in *nsDecls* in the resulting document. This is the default.

*useNamespacesOfDocumentType* **String** Optional. Flag indicating whether or not Integration Server uses the namespaces defined in the document type specified for the *documentTypeName* input parameter when creating a document from an XML string.

Set to:

- `True` to use the namespaces defined in the document type specified in *documentTypeName* and those in *nsDecls* when creating elements in the output document.
- `False` to use the namespaces defined in the *nsDecls* parameter only. This is the default.

## Output Parameters

---

*document* **Document** Document (IData object) representation of the nodes and attributes in node.

## Usage Notes

If the IS document type in *documentTypeName* accurately represents the content model for the complex type from which it was created (the **Model type** property value is not "Unordered"), when Integration Server converts an XML node to a document (IData), Integration Server matches up the contents of an element in the XML node with the content model of the IS document type. If a mismatch occurs and Integration Server is unable to map the XML node contents to the IS document type, Integration Server appends the remaining data to the resulting document (IData). Integration Server stops attempting to map the XML node content to a field in the IS document type. This

mismatch does not result in an error at the time the document is created. However, the document would fail validation by the `pub.schema.validate` service.

The `watt.server.soap.decodeElementWithPrefix` server configuration parameter determines whether or not the `pub.xml.xmlNodeToDocument` service considers namespace and/or prefix declarations when decoding namespace qualified elements to corresponding fields in an IS document type.

- When `watt.server.soap.decodeElementWithPrefix` is set to `true`, the `pub.xml.xmlNodeToDocument` service does not consider namespace and/or prefix declarations. The service will map namespace qualified elements in the XML string to fields in an IS document type even if the field names do not include a prefix.
- When `watt.server.soap.decodeElementWithPrefix` is set to `false`, the `pub.xml.xmlNodeToDocument` service considers namespace and/or prefix declarations. The service does not map namespace qualified elements in the XML input to fields in an IS document type if the field names do not include a prefix. The `pub.xml.xmlNodeToDocument` service does not populate non-prefixed fields. Instead the service adds new fields that use the `prefix:name` structure and populates those fields with values from the XML instance.

Following are examples of XML documents and the documents (IData objects) that `xmlNodeToDocument` would produce.

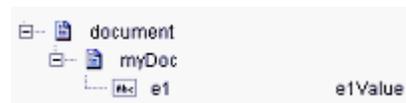
### XML Document

```
<myDoc>
<e1>e1Value</e1>
</myDoc>
```

```
<?xml version="1.0" encoding="UTF-8"
standalone="no"?>
<myDoc>
<e1>e1Value</e1>
</myDoc>
```

```
<?xml version="1.0" encoding="UTF-8"
standalone="no"?>
<myDoc>
<e1 e1Attr="attrValue">e1Value</e1>
</myDoc>
```

### Output from xmlNodeToDocument



## XML Document

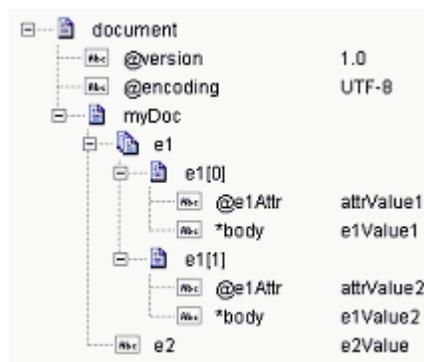
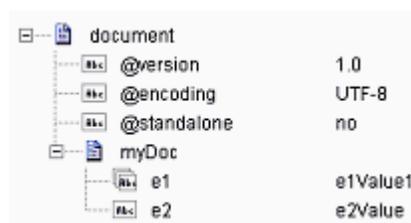
```
<?xml version="1.0" encoding="UTF-8"
standalone="no"?>
<myDoc>
<e1>e1Value</e1>
<e2>e2Value</e2>
</myDoc>
```

```
<?xml version="1.0" encoding="UTF-8"
standalone="no"?>
<myDoc>
<e1>e1Value1</e1>
<e2>e2Value</e2>
<e1>e1Value2</e1>
</myDoc>
```

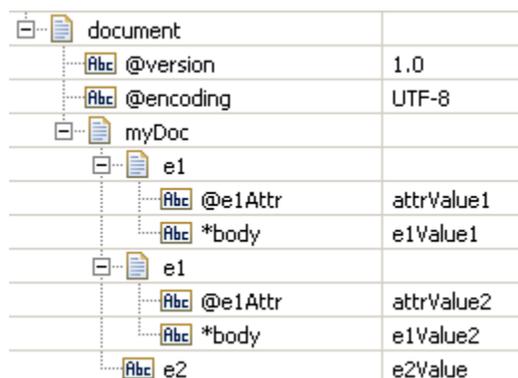
```
<?xml version="1.0"
encoding="UTF-8"?>
<myDoc>
<e1 e1Attr="attrValue1">e1Value1</e1>
<e2>e2Value</e2>
<e1 e1Attr="attrValue2">e1Value2</e1>
</myDoc>
```

```
<?xml version="1.0"
encoding="UTF-8"?>
<myDoc>
<e1 e1Attr="attrValue1">e1Value1</e1>
<e2>e2Value</e2>
<e1 e1Attr="attrValue2">e1Value2</e1>
</myDoc>
```

## Output from xmlNodeToDocument



**Note** This example assumes that *makeArrays* is set to true. Note that *e1* was created as a document list, which holds both *<e1>* elements from the XML document.

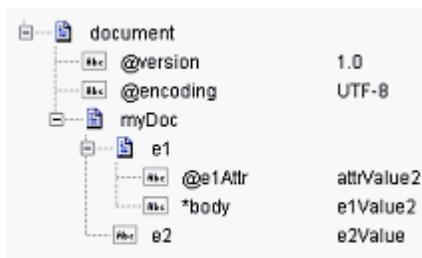


## XML Document

```
<?xml version="1.0"
encoding="UTF-8"?>
<myDoc>
<e1 e1Attr="attrValue1">e1Value1</e1>
<e2>e2Value</e2>
<e1 e1Attr="attrValue2">e1Value2</e1>
</myDoc>
```

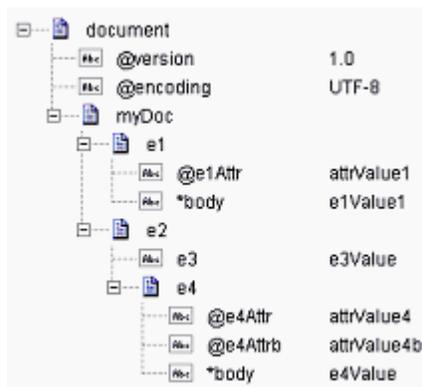
## Output from xmlNodeToDocument

**Note**This example assumes that *makeArrays* is set to *false* and that *watt.server.xml.xmlNodeToDocument*. *keepDuplicates* is set to *true* (the default). Note that both `<e1>` elements from the source XML are retained.



**Note**This example assumes that *makeArrays* is set to *false* and that *watt.server.xml.xmlNodeToDocument*. *keepDuplicates* is set to *false*. Note that only the last `<e1>` element in the source XML was retained in the resulting document.

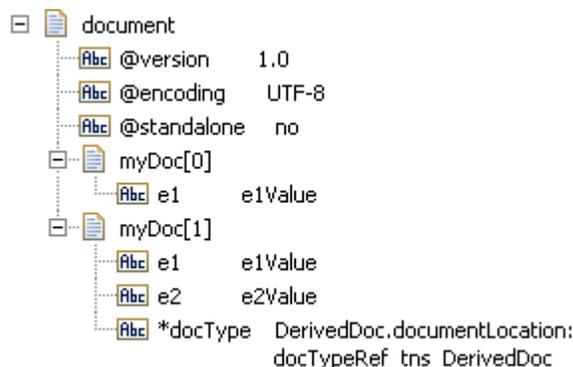
```
<?xml version="1.0"
encoding="UTF-8"?>
<myDoc>
<e1 e1Attr="attrValue1">e1Value1</e1>
<e2>
<e3>e3Value</e3>
<e4 e4Attr="attrValue4"
e4Attrb="attrValue4b">e4Value
</e4>
</e2>
</myDoc>
```



## XML Document

```
<?xml version="1.0" encoding="UTF-8"
standalone="no"?>
  <tns:AcctInfo>
    xmlns:tns="http://localhost/Derived
Address/schema.xsd"
    xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance" >
      <myDoc>    <e1>e1Value</e1>
    </myDoc>
      <myDoc xsi:type="tns:DerivedDoc">
        <e1>e1Value</e1>
        <e2>e1Value</e2>
      </myDoc>
    </tns:AcctInfo>
```

## Output from xmlNodeToDocument



## pub.xml:xmlStringToEnhancedXMLNode

WmPublic. Converts an XML document (represented as a String, byte[ ], or InputStream) to an org.w3c.dom.Node object using the enhanced XML parser.

An DOM node is a special representation of an XML document that can be consumed by any program that uses standard DOM APIs. The pub.xml:xmlNodeToDocument service accepts a DOM object as input.

### Input Parameters

*xmldata*

**String** Optional. String containing the XML document to convert to an XML node.

**Note:** If you specify *xmldata*, do not specify *\$filedata* or *\$filestream*.

*\$filedata*

**byte[ ]** Optional. byte[ ] containing the XML document to convert to an XML node.

**Note:** If you specify *\$filedata*, do not specify *xmldata* or *\$filestream*.

*\$filestream*

**java.io.InputStream** Optional. InputStream containing the XML document to convert to an XML node.

**Note:** If you specify *\$filestream*, do not specify *xmldata* or *\$filedata*.

*encoding*

**String** Optional. Character encoding in which text is represented. Specify UTF-8 for XML files and ISO-8859-1 for HTML files. To have the parser attempt to detect the type of

encoding, specify `autoDetect` (the default, if *encoding* is not specified).

### *inputProcessing*

**Document.** Optional. Contains a set of input parameters that instruct Integration Server how to read the XML document. The fields are comparable to options in the `javax.xml.stream.XMLInputFactory` class.

<b>Key</b>	<b>Description</b>
<i>isValidating</i>	<p><b>String</b> Optional. Determines whether Integration Server performs DTD validation. Set to:</p> <ul style="list-style-type: none"> <li>■ <code>true</code> to perform DTD validation.</li> <li>■ <code>false</code> to disable DTD validation. This is the default.</li> </ul>
<i>isNamespaceAware</i>	<p><b>String</b> Optional. Determines whether Integration Server provides namespace processing for XML 1.0 support while parsing the XML document. Set to:</p> <ul style="list-style-type: none"> <li>■ <code>true</code> to enable namespace processing. This is the default.</li> <li>■ <code>false</code> to disable namespace processing.</li> </ul>
<i>isCoalescing</i>	<p><b>String</b> Optional. Determines whether Integration Server coalesces adjacent character data while parsing the XML document. Set to:</p> <ul style="list-style-type: none"> <li>■ <code>true</code> to coalesce adjacent character data.</li> <li>■ <code>false</code> to indicate that Integration Server does not coalesce adjacent character data. This is the default.</li> </ul>
<i>isReplacingEntityReferences</i>	<p><b>String</b> Optional. Determines whether, while parsing the XML document, Integration Server replaces internal entity references with replacement text and treats them as characters. Set to:</p> <ul style="list-style-type: none"> <li>■ <code>true</code> to replace entity references. This is the default.</li> </ul>

- `false` to indicate entity references will not be replaced.

*isSupporting  
ExternalEntities*

**String** Optional. Determines whether Integration Server resolves external parsed entities while parsing the XML document. Set to:

- `true` to resolve external parsed entities.
- `false` to indicate Integration Server does not support external parsed entities.

The JVM in which Integration Server runs determines the default.

*supportDTD*

**String** Optional. Determines whether Integration Server supports DTDs while parsing the XML document. Set to:

- `true` to support DTDs while parsing the XML document. This is the default.
- `false` to disable support of DTDs while parsing the XML document.

*partitionSize*

**String** Optional. Specifies the size, measured in bytes, of the partitions on the heap where the enhanced XML parser stores parsed document information. Specify a suffix of “k” to indicate kilobytes or “m” to indicate megabytes. For example, 10k or 10m.

If you do not specify a value, Integration Server uses the default partition size value specified on the **Settings > Enhanced XML Parsing** screen in Integration Server Administrator.

## Output Parameters

---

*node*

**org.w3c.dom.Node** XML node representing the returned XML document. This object can be used as input to webMethods services that consume XML nodes in the form of a DOM object.

## Usage Notes

The input parameters *xmldata*, *\$filedata*, and *\$filestream* are mutually exclusive. Specify only one of the preceding parameters. Integration Server checks the parameters in

the following order, using the value of the first parameter that has a specified value: *\$filedata*, *\$filestream*, and *xmldata*.

Use the [pub.xml:xmlStringToXMLNode](#) service to convert an XML document to an XML node using the legacy XML parser. For more information about the legacy XML parser and the enhanced XML parser, see *webMethods Integration Server Administrator's Guide*.

Keep the following information in mind when specifying a *partitionSize*

- The *partitionSize* is a hint for the enhanced XML parser so that it can estimate the amount of heap space needed to parse the document. Often, it not possible to determine the size of an inbound XML document before parsing.
- As a general rule, Software AG recommends a *partitionSize* that is 1/2 the size of the unparsed XML document.
  - A *partitionSize* that is considerably larger than 1/2 the size of the unparsed XML document causes the enhanced XML parser to consume more heap space than necessary but might also improve throughput. However, this can impact the overall performance of Integration Server.
  - A *partitionSize* that is considerably smaller than 1/2 the size of the unparsed XML document causes the enhanced XML parser to create a large number of partitions to parse the document. While this might use less heap space, it may reduce the throughput of the parser.
  - A *partitionSize* that is three times smaller or three times larger than 1/2 the size of the unparsed XML document will likely have little impact on the performance.
- At run time, the enhanced XML parser overrides a *partitionSize* that consumes all of the available heap space.
- At run time, if the *partitionSize* results in an initial heap allocation that exceeds the single document limit set in the **Maximum heap allocation for any single document** field the limit for all documents set in the **Maximum heap allocation for all documents combined** field, the enhanced XML parser reduces the partition size automatically. For more information about heap allocation limits for the enhanced XML parser, see *webMethods Integration Server Administrator's Guide*
- If you do not specify *partitionSize*, the enhanced XML parser uses the default specified in the **Default partition size** field on the **Settings > Enhanced XML Parsing** page in Integration Server Administrator.

---

## pub.xml:xmlStringToXMLNode

WmPublic. Converts an XML document (represented as a String, byte[ ], or InputStream) to an XML node.

An XML node is a special representation of an XML document that can be consumed by the Integration Server. Most webMethods services that operate on XML documents require an XML node as input.

---

## Input Parameters

---

<i>xmldata</i>	<p><b>String</b> Optional. String containing the XML document to convert to an XML node.</p> <p><b>Note:</b> If you specify <i>xmldata</i>, do not specify <i>\$filedata</i> or <i>\$filestream</i>.</p>
<i>\$filedata</i>	<p><b>byte[ ]</b> Optional. <code>byte[ ]</code> containing the XML document to convert to an XML node.</p> <p><b>Note:</b> If you specify <i>\$filedata</i>, do not specify <i>xmldata</i> or <i>\$filestream</i>.</p>
<i>\$filestream</i>	<p><b>java.io.InputStream</b> Optional. <code>InputStream</code> containing the XML document to convert to an XML node.</p> <p><b>Note:</b> If you specify <i>\$filestream</i>, do not specify <i>xmldata</i> or <i>\$filedata</i>.</p>
<i>encoding</i>	<p><b>String</b> Optional. Character encoding in which text is represented. Specify UTF-8 for XML files and ISO-8859-1 for HTML files. To have the parser attempt to detect the type of encoding, specify <code>autoDetect</code> (the default, if <i>encoding</i> is not specified).</p>
<i>expandDTD</i>	<p><b>String</b> Optional. Flag indicating whether references to parameter entities in the XML document's DTD are to be processed. Set to:</p> <ul style="list-style-type: none"><li>■ <code>true</code> to expand references to parameter entities to their full definition.</li><li>■ <code>false</code> to ignore references to parameter entities. This is the default.</li></ul>
<i>isXML</i>	<p><b>String</b> Optional. Flag specifying whether the input document is XML or HTML. (<code>xmlStringToXMLNode</code> must know this so that it can parse the document correctly.) Set to:</p> <ul style="list-style-type: none"><li>■ <code>autoDetect</code> to parse the document based on its type. When you use this option, <code>xmlStringToXMLNode</code> detects the document's type based on its <code>&lt;!DOCTYPE... \&gt;</code> or <code>&lt;?XML... \&gt;</code> tag. If it cannot determine a document's type, it parses it as HTML. This is the default.</li><li>■ <code>true</code> to parse the document as XML.</li></ul>

- `false` to parse the document as HTML.

*expandGeneralEntities* **String** Optional. Flag indicating whether `pub.xml:xmlStringToXMLNode` should expand references to general entities in the XML document's DTD. Set to:

- `true` to expand references to general entities to their full definition. This is the default.
- `false` to ignore references to general entities.

## Output Parameters

---

*node* **com.wm.lang.xml.Node** XML node representation of the XML document in *xmlData*. This object can be used as input to `webMethods` services that consume XML nodes.

## Usage Notes

The input parameters *xmlData*, *\$filedata*, and *\$filestream* are mutually exclusive. Specify only one of the preceding parameters. Integration Server checks the parameters in the following order, using the value of the first parameter with a specified value: *\$filedata*, *\$filestream*, and *xmlData*.

If *expandGeneralEntities* is not specified, Integration Server uses the value in `watt.core.xml.expandGeneralEntities`. If `watt.core.xml.expandGeneralEntities` is not set, the references to general entities are always expanded.

Use the `pub.xml:xmlStringToEnhancedXMLNode` service to convert an XML document to an XML node using the enhanced XML parser. For more information about the legacy XML parser and the enhanced XML parser, see *webMethods Integration Server Administrator's Guide*.

# 42 XMLData Folder

---

- Summary of Elements in this Folder ..... 1076

You can use the services in the `xmldata` folder to convert a DOM node to an XMLData formatted document, convert XMLData to an XML String, and retrieve and set instance data from XMLData.

## Summary of Elements in this Folder

The following elements are available in this folder

Element	Package and Description
<a href="#">pub.xmldata.domNodeToXMLData</a>	WmPublic. Converts a DOM node and its child nodes to a document in the XMLData format and optionally validates each node.
<a href="#">pub.xmldata:getAttributes</a>	WmPublic. Retrieves the attributes from an XMLData document.
<a href="#">pub.xmldata:getInstanceTag</a>	WmPublic. Retrieves the value of a specified instance tag for a particular element in an XMLData document.
<a href="#">pub.xmldata:getNamespaceTags</a>	WmPublic. Retrieves the namespace declarations for a particular element in an XMLData document.
<a href="#">pub.xmldata:setAttribute</a>	WmPublic. Sets the value for a specific attribute in an XMLData document.
<a href="#">pub.xmldata:setInstanceTag</a>	WmPublic. Sets the value of an instance tag for a particular element in an XMLData document
<a href="#">pub.xmldata:setNamepaceTag</a>	WmPublic. Sets the namespace declaration for a particular element in an XMLData document.
<a href="#">pub.xmldata:xmlDataToXMLString</a>	WmPublic. Converts an XMLData document to an XML string.

### pub.xmldata.domNodeToXMLData

WmPublic. Converts a DOM node and its child nodes to a document in the XMLData format and optionally validates each node.

This service converts each element and attribute in the DOM node to the XMLData format. Note that XMLData is an IData document arranged in a particular format that is described by an XML document type.

A DOM node tree is a special representation of an XML document that can be consumed by any program that uses standard DOM APIs. For example, the `pub.xml.xmlStringtoXMLNode` and `pub.xmlStringToEnhancedXMLNode` services produce a DOM tree. A DOM node tree contains parsed XML content.

## Input Parameters

---

<i>node</i>	<b>org.w3c.dom.Node</b> DOM node that is to be converted to an XMLData (Document). Note that <code>com.wm.lang.xml.Node</code> is a DOM node.
<i>conformsTo</i>	<b>String</b> Optional. Fully-qualified name of the XML document type on Integration Server against which to validate the DOM node. Specify a value for <i>conformsTo</i> only if you want to do the following: <ul style="list-style-type: none"><li>■ Validate the DOM node as part of converting it to XMLData. Integration Server validates a DOM node in the tree as part of converting the tree node to and XMLData-formatted Document.</li><li>■ Use the XML document type to direct the conversion of the DOM node. That is, the structure of the XML document type determines the structure of the resulting XMLData.</li></ul>

**Note:** You must specify an XML document type for *conformsTo* if you intend to modify the generated XMLData using the mapping tools available in the pipeline.

**Note:** One or more XML document types are created from a single XML schema definition. To create an XMLData-formatted Document using the XML document types created from an XML Schema definition, you need specify only one of the XML document types or XML fields created from the schema. Integration Server locates the related XML document types, using the complete collection of XML document types and XML fields created from the schema during conversion and validation.

If you do not specify an XML document type for *conformsTo*, Integration Server does not validate the input nodes as part of the conversion. Additionally, the service formats the resulting XMLData in a generic way instead of in a manner that matches an XML document type.

- maxErrors* **String** Optional. Number of errors to be collected. Default value is 1. When the number of errors found is equal to *maxErrors*, the validation processor stops validation and returns the result. If *maxErrors* is set to -1, the validation processor returns all errors.
- The service uses the *maxErrors* input parameter only when a *conformsTo* value is specified.
- ignoreContent* **String** Optional. Flag that specifies whether the validation processor will validate simple content where simple content is content keys of the type String, String List, or String Table.
- Set to:
- `true` to ignore content (that is, do not validate the content of keys of the type String, String List, or String Table). Integration Server validates the structure and key names of the XML document only.
  - `false` to validate content. This is the default.
- The service uses the *ignoreContent* input parameter only when a *conformsTo* value is specified.
- failIfInvalid* **String** Optional. Flag that indicates whether the service should fail and throw an exception if the object is invalid. Set to:
- `true` to indicate that the service should fail if the object is invalid.
  - `false` to indicate that service should signal success and return errors to the pipeline if object is invalid. This is the default.
- The service uses the *ignoreContent* input parameter only when a *conformsTo* value is specified.

## Output Parameters

---

- isValid* **String** Conditional. Flag that indicates whether or not validation of the DOM node was successful. A value of:
- `true` indicates that the validation was successful.
  - `false` indicates that the validation was unsuccessful.
- The service returns the *isValid* parameter only if *conformsTo* specifies an XML document type.
- errors* **Document List** Conditional. Errors encountered during validation. Each document will contain the following information:

<u>Key</u>	<u>Description</u>
<i>pathName</i>	<b>String</b> Location of the error in XQL.
<i>errorCode</i>	<b>String</b> Error code (for example, VV-001).
<i>errorMessage</i>	<b>String</b> Error message (for example, Missing Object).

The service returns the *errors* parameter only if *conformsTo* specifies an XML document type.

*xmlDataDocument* **Document** An XMLData-formatted document that represents the elements and attributes in the DOM node.

### Usage Notes

If the *conformsTo* input parameter specifies an XML document type or an XML field, the `pub.xmldata:domNodetoXMLData` service does the following:

- Uses the XML document type (and any other XML document types and XML fields created from the same XML schema definition) to direct the structure of the XMLData.
- Validates each DOM node in the tree before converting it to XMLData.

You can use the `pub.schema:validate` service to validate XMLData.

By default, when validating XML, Integration Server uses the Perl5 regular expression compiler. As a result, Integration Server uses Perl5 pattern matching to evaluate element content. The Perl5 pattern matching is not identical to the pattern matching defined by the World Wide Web Consortium (W3C) for the XML Schema standard. When validating XML against an IS schema or IS document type created from an XML Schema definition, an element defined to be of simple type with a pattern constraining facet might be invalid according to the Perl5 pattern matching rules but valid according to the pattern matching rules associated with the XML Schema standard.

You can change how Integration Server performs pattern matching during validation by switching from the Perl5 regular expression compiler to the Java regular expression compiler. To do this, set the server configuration parameter `watt.core.datatype.usejavaregex` to `true`. When this parameter is set to `true`, Integration Server performs pattern matching as described by `java.util.regex.pattern`.

When validating XML, Integration Server uses the W3C recommendation *XML Schema Part 2: Datatypes*. If you want to validate XML for illegal values, set *ignoreContent* to `false` and the `watt.core.validation.w3cConformant` configuration parameter to `true`. For information about setting this configuration parameter, see *webMethods Integration Server Administrator's Guide*.

---

## pub.xmldata:getAttributes

WmPublic. Retrieves the attributes from a an XMLData document.

If an XML Schema definition contains a complex type definition with an anyAttribute declaration, the XML document type created from the schema contains a corresponding *\*anyAttribute* field. As the purpose of the anyAttribute declaration to act as a placeholder or wildcard, the XML document type cannot have a more specific representation. Use the pub.xmldata:getAttributes service to retrieve the attribute and attribute value included in an XML document at run time. That is, use the service to find the unknown attributes that are not declared in the XML document type. To set the attribute value in the XMLData, you must use the pub.xml:setAttribute service.

### Input Parameters

---

<i>xmlDataAttributes</i>	<p><b>Document</b> The <i>*attributes</i> from which you want to retrieve a list of attributes. The <i>*attributes</i> field is an XMLData document.</p> <p>If an element of complex type defines or carries attributes, Integration Server places the attributes in an <i>*attributess</i> document that is of type XMLData.</p>
<i>ncName</i>	<p><b>String</b> Optional. The NCName (non-colonized name), of the attribute. The NCName is also called the local name. If you do not specify a value for <i>ncName</i>, the service returns all attributes belonging to the namespace specified in the <i>namespace</i> input parameter.</p>
<i>namespace</i>	<p><b>String</b> Optional. The namespace URI for the attribute. If you do not specify a value, the service looks for attributes that do not belong to a namespace.</p>

### Output Parameters

---

<i>results</i>	<p><b>Document List</b> A list of XMLData documents that contain the attributes that match the supplied <i>ncName</i> and <i>namespace</i> values.</p> <p>If no attributes match the supplied <i>ncName</i> and <i>namespace</i> values, the service returns an empty (zero length) list.</p>
----------------	---

<u>Key</u>	<u>Description</u>
<i>ncName</i>	<b>String</b> The NCName of the attribute.

---

<i>namespaceName</i>	<b>String</b> The namespace URI for the attribute.
<i>value</i>	<b>String</b> The value of the attribute.

---

## pub.xmldata:getInstanceTag

WmPublic. Retrieves the value of a specified instance tag from a field in XMLData document.

An element in an XML document may contain information that is unique to that particular instance document, including instance (xsi) assertions such as type, nil, or schemaLocation. Because instance information cannot be asserted in an XML schema definition, an XML document type cannot contain fields for the instance information. It is impossible for the XML Schema definition and the XML document types generated from that schema to know where the instance assertions might be specified in an instance document. When an XML instance document is parsed and converted to XMLData, specifically an XMLData-formatted document, Integration Server places the instance information in \*instance field under the parent of the field that contains the instance information. That is, Integration Server stores the instance information in an \*instance field that is a child of the field that corresponds to the element that contains the instance information in the XML document. Use the pub.xmldata:getInstanceTag service to retrieve the instance information from the XMLData created from the XML document. After you retrieve the instance information, you can manipulate or use the instance information in the pipeline. To set the value of the instance information in the XMLData, use the pub.xml:setInstanceTag service.

### Input Parameters

---

<i>xmlDataParent</i>	<b>Document</b> The XMLData document that is a parent of the field for which to retrieve the instance tag value.
<i>ncName</i>	<b>String</b> The NCName (non-colonized name), of the field for which you want to retrieve an instance tag value. The NCName is also called the local name.
<i>namespace</i>	<b>String</b> Optional. The namespace URI for the field for which you want to retrieve an instance tag value. If you do not specify a value, the service looks for tags that do not belong to a namespace.
<i>particleId</i>	<b>String</b> The sequential number for the particle in a model group for which you want to retrieve an instance tag value. A value of 1 refers to the first instance of the field with <i>ncName</i> # <i>namespace</i> name in a model group.

*index* **String** Optional. If the field for which you want to retrieve a tag value is a repeating field (`minOccurs > 0`), specify the index for the occurrence for which you want to retrieve a tag value. A value of 0 refers to the first instance of a repeating field.

**Note:** Repeating elements are represented as arrays in XMLData.

*tagName* **String** Name of the instance tag for which you want to retrieve a value, such as `type`, `nil`, `schemaLocation`.

**Note:** Tag names are case-sensitive.

### Output Parameters

*tagValue* **String** Value of the tag.

### Usage Notes

Together, the *ncName*, *namespace*, *particleId*, and *index* identify a particular field in the XMLData document.

## pub.xmldata:getNamespaceTags

WmPublic. Retrieves the namespace declarations, specifically the prefix and namespace URI, associated with a particular field in an XMLData Document.

An element in an XML document may contain information that is unique to that particular instance document, including namespace declarations and instance assertions such as `type`, `nil`, or `schemaLocation`. A namespace declaration associates a prefix with a namespace URI. Because namespace declarations cannot be asserted for an element in an XML schema definition, an XML document type cannot contain fields for the namespace declarations. It is impossible for the XML Schema definition and the XML document types generated from that schema to know where the namespace declarations might be specified in an instance document. When an XML instance document is parsed and converted to XMLData, specifically an XMLData-formatted document, Integration Server places the namespace declarations in a `*namespace` field under the parent of the element that contains the namespace declaration. That is, Integration Server stores the namespace declarations in an `*namespace` field that appears as a child of the field that corresponds to the element containing the namespace declaration in the XML document. Use the `pub.xmldata:getNamespaceTag` service to retrieve the namespace declarations for a particular field from the XMLData created from the XML document. After you retrieve the namespace declaration, you can manipulate the namespace declarations in the pipeline. To set the value of a namespace declaration for a particular field in the XMLData, use the `pub.xml:setNamespaceTag` service.

---

## Input Parameters

---

<i>xmlDataParent</i>	<b>Document</b> The XMLData document that is a parent of the field for which to retrieve the namespace tag and value.
<i>ncName</i>	<b>String</b> The NCName (non-colonized name), of the field for which you want to retrieve an namespace declaration. The NCName is also called the local name.
<i>namespace</i>	<b>String</b> Optional. The namespace URI for the field for which you want to retrieve the namespace declaration. If you do not specify a namespace, the service looks for a field that does not belong to a namespace.
<i>particleId</i>	<b>String</b> The sequential number for the particle in a model group for which you want to retrieve a namespace declaration. A value of 1 refers to the first instance of the field with <i>ncName</i> # <i>namespace</i> name in a model group.
<i>index</i>	<b>String</b> Optional. If the field for which you want to retrieve a namespace declaration is a repeating field ( <code>minOccurs &gt; 0</code> ), the index for the occurrence for which you want to retrieve the namespace declaration. A value of 0 refers to the first instance of a repeating field.

**Note:** Repeating elements are represented as arrays in XMLData.

## Output Parameters

---

<i>results</i>	<b>Document List</b> A document (IData) containing namespace declarations for the specified field. A namespace declaration consists of a prefix and a namespace URI.						
	<table> <thead> <tr> <th><u>Key</u></th> <th><u>Description</u></th> </tr> </thead> <tbody> <tr> <td><i>tagName</i></td> <td><b>String</b> Prefix specified in the namespace declaration. The <i>tagName</i> parameter contains an empty string if this is the default namespace declaration.</td> </tr> <tr> <td><i>tagValue</i></td> <td><b>String</b> The namespace URI associated with the namespace declaration.</td> </tr> </tbody> </table>	<u>Key</u>	<u>Description</u>	<i>tagName</i>	<b>String</b> Prefix specified in the namespace declaration. The <i>tagName</i> parameter contains an empty string if this is the default namespace declaration.	<i>tagValue</i>	<b>String</b> The namespace URI associated with the namespace declaration.
<u>Key</u>	<u>Description</u>						
<i>tagName</i>	<b>String</b> Prefix specified in the namespace declaration. The <i>tagName</i> parameter contains an empty string if this is the default namespace declaration.						
<i>tagValue</i>	<b>String</b> The namespace URI associated with the namespace declaration.						

---

### Usage Notes

Together, the *ncName*, *namespace*, *particleId*, and *index* identify a particular field in the XMLData.

---

## pub.xmldata:setAttribute

WmPublic. Sets the value for a specific attribute in an XMLData document.

While the primary purpose of this service is to set the value of an attribute that is the runtime replacement for an anyAttribute declaration, you can use this service to set the value of any of the attributes in an XMLData document.

---

### Input Parameters

*xmlDataAttributes* **Document** The *\*attributes* document of type XMLData in which you want to set the value of an attribute.

*ncName* **String** The NCName (non-colonized name) of the attribute whose value you want to set. The NCName is also called a local name.

*namespace* **String** Optional. The namespace URI for the attribute whose value you want to set. Leave *namespace* blank if the attribute does not belong to a namespace.

*value* **String** The value that you want to assign to the attribute.

---

### Output Parameters

None.

---

## pub.xmldata:setInstanceTag

WmPublic. Sets the value of an instance tag for a specific field in an XMLData document.

---

### Input Parameters

*xmlDataParent* **Document** The XMLData document that is a parent of the field for which to set the instance tag value.

*ncName* **String** The NCName (non-colonized name), of the field for which you want to set an instance tag value. The NCName is also called the local name.

<i>namespace</i>	<b>String</b> Optional. The namespace URI for the field for which you want to set an instance tag value. If the field does not belong to a namespace, do not specify a value for the <i>namespace</i> input parameter.
<i>particleID</i>	<b>String</b> The sequential number for the particle in a model group for which you want to set an instance tag value. A value of 1 refers to the first instance of the field with <i>ncName #namespace</i> name.
<i>index</i>	<b>String</b> Optional. If the field for which you want to set a tag value is a repeating field ( <i>minOccurs</i> > 0), the index for the occurrence for which you want to set a tag value. A value of 0 refers to the first instance of a repeating field. If you specify the array index that is greater than the current size of the array, the array size increases and missing entries are null.  <b>Note:</b> Repeating elements are represented as arrays in XMLData.
<i>tagName</i>	<b>String</b> Name of the instance tag for which you want to set a value, such as <i>type</i> , <i>nil</i> , <i>schemaLocation</i> .  <b>Note:</b> Tag names are case-sensitive.
<i>tagValue</i>	<b>String</b> Value to assign to the instance tag. If you specify a null value, the service removes the tag and its value from the XMLData.

### Output Parameters

---

None.

## pub.xmldata:setNamespaceTag

---

WmPublic. Sets the namespace declaration for a particular field.

### Input Parameters

---

<i>xmlDataParent</i>	<b>Document</b> The XMLData document that is a parent of the field for which to set a namespace declaration.
<i>ncName</i>	<b>String</b> The NCName (non-colonized name), of the field for which you want to set a namespace declaration. The NCName is also called the local name.

<i>namespace</i>	<b>String</b> Optional. The namespace URI for the field for which you want to set a namespace declaration. If the field does not belong to a namespace, do not specify a value for the <i>namespace</i> input parameter.
<i>particleID</i>	<b>String</b> The sequential number for the particle in a model group for which you want to set a namespace declaration. A value of 1 refers to the first instance of the field with <i>ncName #namespace</i> name.
<i>index</i>	<b>String</b> Optional. If the field for which you want to set a namespace declaration is a repeating element ( <i>minOccurs</i> > 0), the index for the occurrence for which you want to set a tag value. A value of 0 refers to the first instance of a repeating field. If you specify the array index that is greater than the current size of the array, the array size increases and missing entries are null. If you specify an index that is beyond the end of the array, the service increases the size of the array and the intervening indexes have a null value.
	<b>Note:</b> Repeating elements are represented as arrays in XMLData.
<i>tagName</i>	<b>String</b> The prefix to associate with the namespace declaration. Specify an empty string to associate a value with the default namespace declaration, that is, specify: ""
<i>tagValue</i>	<b>String</b> The namespace URI to associate with the prefix in <i>tagName</i> . If you specify a null value, the service removes the namespace declaration from the XMLData.

### Output Parameters

---

None.

## pub.xmldata.xmlDataToXMLString

---

WmPublic. Converts an XMLData document to an XML string.

The `pub.xmldata.xmlDataToXMLString` service recurses through a given document formatted as XMLData, building an XML representation from the elements within it. Key names are turned into XML elements and attributes, and the key values are turned into the contents of those elements.

### Input Parameters

---

*xmlDataDocument* **Document** XMLData document that is to be converted to XML.

*encode*      **String** Optional. Flag indicating whether to HTML-encode the data. Set this parameter to true if your XML data contains special characters, including the following: < > & " ' .

Set to:

- `true` to HTML-encode the data.  
For example, the string expression `5 < 6` would be converted to `<expr>5 &lt; 6</expr>`, which is valid.
- `false` to not HTML-encode the data. This is the default.  
For example, the string expression `5 < 6` would be converted to `<expr>5 < 6</expr>`, which is invalid.

### Output Parameters

---

*xmlString*      **String** XML String produced from *xmlDataDocument* .



# 43 XSLT Folder

---

- Summary of Elements in this Folder ..... 1090

You use the elements in the XSLT folder to transform XML into a byte array, file, or XML node, and to maintain the XSLT stylesheet cache.

## Summary of Elements in this Folder

The following elements are available in this folder:

Element	Package and Description
<code>pub.xslt.Transformations:transformSerialXML</code>	WmXSLT. Uses an XSLT stylesheet to transform XML, then stores the transformed XML in a byte array, file, or XML node.
<code>pub.xslt.Cache:removeAllTemplates</code>	WmXSLT. Clears the XSLT stylesheet cache.
<code>pub.xslt.Cache:removeTemplate</code>	WmXSLT. Removes one stylesheet from the XSLT stylesheet cache.

The WmXSLT package also comes with sample services that show you how to use the public services.

## `pub.xslt.Transformations:transformSerialXML`

WmXSLT. Uses an XSLT stylesheet to transform XML, then stores the transformed XML in a byte array, file, or XML node.

To optimize performance, the service stores the XSLT stylesheet in a cache so the stylesheet will be instantly available to the service for later runs.

### Input Parameters

<i>stylesheetSystemId</i>	<b>String</b> URI (simple file path or URL) for the XSLT stylesheet to use.
<i>systemId</i>	<b>String</b> URL of the XML to transform. If you specify this parameter, do not specify the <i>filename</i> , <i>bytes</i> , or <i>xmlStream</i> parameter.
<i>filename</i>	<b>String</b> Fully qualified name of the file that contains the XML to transform. The file must be located on the local machine. If you

---

	specify this parameter, do not specify the <i>systemId</i> , <i>bytes</i> , or <i>xmlStream</i> parameter.
<i>bytes</i>	<b>byte[]</b> XML to transform. If you specify this parameter, do not specify the <i>systemId</i> , <i>filename</i> , or <i>xmlStream</i> parameter.
<i>xmlStream</i>	<b>Input stream</b> XML to transform. If you specify this parameter, do not specify the <i>systemId</i> , <i>filename</i> , or <i>bytes</i> parameter.
<i>xslParamInput</i>	<b>Document</b> Optional. Name/value pairs to pass to the stylesheet. See the <i>webMethods Service Development Help</i> for instructions on setting up a stylesheet to work with this parameter.
<i>resultType</i>	<b>String</b> Tells Designer what to transform the XML into. Must be one of these values: <ul style="list-style-type: none"><li>■ <i>bytes</i> to transforms the XML into a byte array.</li><li>■ <i>file</i> to transforms the XML into a file. If you specify <i>file</i>, you must also specify the <i>outFileName</i> parameter.</li><li>■ <i>xmlNode</i> to transforms the XML into an XML node.</li></ul>
<i>outFileName</i>	<b>String</b> Fully qualified name of the file in which to store the transformed XML. The file must be located on the local machine. Use this parameter only if you specified <i>file</i> on the <i>resultType</i> parameter.
<i>useCompilingProcessor</i>	<b>Boolean.</b> Optional. Specifies whether to use the Xalan compiling processor (XSLTC), which creates and uses compiled stylesheets or <i>translets</i> . Set to: <ul style="list-style-type: none"><li>■ <i>true</i> to use the <code>org.apache.xalan.xsltc.trax.TransformerFactoryImpl</code> class as a transformer factory.</li><li>■ <i>false</i> to use the transformer factory that is specified on the home page of the WmXSLT package. This is the default.</li></ul> If no translet currently exists for the stylesheet, the processor creates one. If a translet exists and the stylesheet has changed since the translet was created, the processor replaces the existing translet with a new one. If the stylesheet has not changed since the translet was created, the processor reuses the existing translet.  If the <i>stylesheetSystemId</i> input parameter specifies a simple file path, the service writes the translet to the same folder in which the stylesheet for your XSLT service resides.

If the *stylesheetSystemId* input parameter specifies a URL, the service writes the translet to memory. As a result, the translet will not survive Integration Server restart.

*loadExternalEntities* **String** Optional. Specifies whether or not to load external entities (file URIs, HTTP URLs, and so on) referenced in the XML that the service receives or in the XSLT stylesheet the service uses to transform the XML. Set to:

- `true` to load content from all external entities that are referenced in the XSLT stylesheet or in the XML. This is the default.
- `false` to not load content from external entities that are referenced in the XSLT stylesheet or in the XML. Use this setting to prevent attacks from external entities by blocking those entities.

**Important:** To help prevent an external entity attack in a production environment, set *loadExternalEntities* to `false` in each instance of the `transformSerialXML` service.

## Output Parameters

*bytes* **byte[]** Byte array that contains the transformed XML. The service places the byte array in the pipeline so that subsequent services can use it. This value is present only if you specified *bytes* in the *resultType* input parameter.

*node* **com.wm.lang.xml.Node** Node that contains the transformed XML. The service places the XML node in the pipeline so that subsequent services can use it. This value is present only if you specified `xmlNode` in the *resultType* input parameter.

*xslParamOutput* **Document** Document that contains name/value pairs that were returned by the stylesheet. The service places the document in the pipeline so that subsequent services can use it. This value is present only if you add name/value pairs to it within your stylesheet.

See the section on passing name/value pairs from the stylesheet to the pipeline in *webMethods Service Development Help* for instructions on setting up your stylesheet to work with this parameter.

## Example

You want to transform an XML document named `cdCatalog.xml` into an HTML document using an XSLT stylesheet named `cdCatalog.xsl`. You would pass the `transformSerialXML` service these values:

## Input Parameters

---

<i>stylesheetSystemId</i>	<code>http://localhost:5555/WmXSLT/samples/xdocs/ cdCatalog.xsl</code>
<i>systemId</i>	<code>http://localhost:5555/WmXSLT/samples/xdocs/ cdCatalog.xml</code>
<i>resultType</i>	<code>bytes</code>

The service transforms the XML stream into a byte array containing an HTML document and puts the byte array in the pipeline. You could convert the byte array into a `String` using the Integration Server built-in service `pub.string:bytesToString`, then display the `String` using a dynamic server page (DSP). For information about using DSPs, see *Dynamic Server Pages and Output Templates Developer's Guide*.

## Usage Notes

If `loadExternalEntities` is set to `false`, you can have the service load, read, and transform content from a trusted external entity by doing one of the following:

- Place the trusted external entity file in the Integration Server installation directory or subdirectories.
- Include the trusted external entity in the list of trusted entities identified in the server parameter `watt.core.xml.allowedExternalEntities`. For more information about this parameter, see *webMethods Integration Server Administrator's Guide*.

If `loadExternalEntities` is not specified in the service signature, Integration Server checks the value of the server parameter `watt.core.xml.expandGeneralEntities`. If this parameter is set to `false`, the `transformSerialXML` service blocks all external entities that are not included in the list of trusted entities specified in `watt.core.xml.allowedExternalEntities`. For more information about `watt.core.xml.expandGeneralEntities`, see *webMethods Integration Server Administrator's Guide*.

---

## pub.xslt.Cache:removeAllTemplates

WmXSLT. Clears the XSLT stylesheet cache.

**Input Parameters**

---

None.

**Output Parameters**

---

*message*      **String** Indicates whether the service was able to clear the cache.

---

**pub.xslt.Cache:removeTemplate**

WmXSLT. Removes one stylesheet from the XSLT stylesheet cache.

**Input Parameters**

---

*stylesheetSystemId*      **String** URI for the XSLT stylesheet to remove from the cache.

**Output Parameters**

---

*message*      **String** Indicates whether the service was able to remove the stylesheet from the cache.

# Index

## A

- absoluteValue 532
- access permissions assigned to built-in services 24
- ACLs assigned to built-in services 24
- action 850
- activatePackage 584
- addBodyBlock 762
- addBodyEntry 818
- addBodyPart 555
- addComplexTask 657
- addFaultBlock 764
- addFloatList 532
- addFloats 533
- addHeaderBlock 766
- addHeaderEntry 819
- adding
  - body parts to MIME messages 555
  - documents to list 523
  - entries into a data store 886
  - MIME message headers 559
  - numeric values 532, 533, 534, 535
  - strings to list 524
- addIntList 534
- addInts 535
- addItemToVector 522
- addMimeHeader 559
- addObjects 535
- addOnetimeTask 660
- addReleaseRegistryEntry 639
- addRepeatingTask 662
- addSubscriber 260
- addTrailer 821
- alarm specification 264
- alarmInfo document type 265
- appending data to a remote file 102
- appendToDocumentList 523
- appendToStringList 524
- arithmetic services 530
- ART folder 28
- ART services 28
- assets folder 54
- asynchronous call to a remote server 635
- asynchronous request/reply
  - description of 617
- asynchronous request/reply
  - delivering request 600

- description of 601
  - publishing request 615
  - publishing request to a specific client 600
  - retrieving reply 622
- attribute
  - setting in XMLData 1084
- audit specification 265
- auditError event 266
- auditErrorInfo document type 268
- auditInfo document type 269

## B

- backupPackage 585
- base64 Decode 900
- base64Encode 900
- binding
  - output template to Values object 652
- BPEL folder 549, 981
- byte 422
- byte array from string 915
- bytesToDocument 241
- bytesToFile 330
- bytesToStream 423
- bytesToString 901

## C

- Cache folder 57
- calculateDateDifference 195
- call 212
- calling stored procedures 212
- callStackItem document type 270
- cancelTask 665
- canonical document, definition 929
- canonical keys
  - creating 929
  - inserting 933
  - retrieving 931
- cd 103, 149
- cdgrp 149
- cdls 103, 117
- certificate chain 698
- certs-only S/MIME Message
  - creating 737
  - extracting certificates from 742
- changing the working directory 103, 149
- character sequence, indexing first occurrence 904
- checkFileExistence 331
- checksum 54

- chmod 150
  - chown 151
  - clear pipeline 382
  - clearing transactional state 214
  - clearKeyAndChain 698
  - clearTransaction 214
  - Client folder 94
  - close 424
  - close connection 215
  - closeAll 216
  - closeLatch 926
  - closeStore 886
  - closing database connection 215
  - committing changes to a database 216
  - commitTransaction 49
  - compareDates 196
  - comparing
    - strings 902
  - compatibility mode 1009
  - complex tasks
    - adding to scheduler 657
    - updating to scheduler 673
  - components
    - XML schema definitions 692
  - concatenating
    - array of strings 909
    - strings 902
  - connections
    - creating 217
  - containsKey 416
  - content type, getting from MIME message 567
  - converting
    - string list to document list 525
    - XML nodes to IData objects 1058
  - converting to
    - InputStream 422
  - convertToString service 344
  - convertToValues service 349
  - copyFile 332
  - createByteArray 424
  - createCertsOnlyData 737
  - createDocumentType service 359
  - createEncryptedData 737
  - createFFDictionary service 360
  - createHashtable 417
  - createJMSTrigger 937
  - createMessageDigest 718
  - createMimeData 561
  - createSignedAndEncryptedData 738
  - createSignedData 740
  - createSoapData 827
  - createTrigger 951
  - createXReference 929
  - createXSD 685
  - creating MIME messages 561
  - cross-references
    - creating 929
    - deleting by object ID 930
    - deleting individual records 930
    - inserting 933
  - current VCS user
    - removing 1018
    - setting 1019
  - currentNanoTime 197
- D**
- dateTimeFormat 201
  - data
    - converting 422
    - waiting for delivery 923
  - data stores
    - adding entries 886
    - closing 886
    - deleting 887
    - inserting entries 893
    - registering 894
    - removing entries 895
    - retrieving values 888
    - unlocking entries 893, 896
    - unregistering 886
    - updating entries 893
  - database connection, closing 215, 216
  - database services 210
  - databases
    - clearing transactional state 214
    - closing connections 215, 216
    - committing changes 216
    - creating connections 217
    - deleting rows 219
    - discarding changes 235
    - invoking stored procedures 212
    - querying 233
    - retrieving names of stored procedures 227
    - retrieving tables 230
    - starting a transaction 236
  - datatypes in XML schemas 691, 692
  - Date folder 189

- dateBuild 198
- dates
  - converting formats 201
  - formatting 203
  - invalid dates 194
  - returning current 204, 204
  - symbols used for 190
  - time zones 191
- dateTimeBuild 199
- debug log 382
- debugging services 380
- decoding URL-encoded strings 919
- decrypting
  - MIME messages 742
- deepClone 992
- default key, associating with invoked services 698
- deleteByObjectId 930
- deleteFFDictionary service 360
- deleteFFDictionaryEntry service 361
- deleteFFSchema service 362
- deleteFile 333
- deleteJMSTrigger 964
- deleteReleaseRegistryEntry 640
- deleteStore 887
- deleteSubscriber 271
- deleteXReference 930
- deleting
  - cross-reference by object ID 930
  - cross-references 930
  - data stores 887
  - files in working directory 104
  - rows from a database 219
  - subscribers from subscription list 271
- deliver service 595
- deliverAndWait service 598
- digital signatures
  - attaching to MIME messages 740
  - verifying 712, 744
- dir 105
- directory
  - changing 103
- disableConnection 31
- disableListener 34
- disableListenerNotification 39
- disablePackage 586
- disablePollingNotification 39
- disablePublishEvents 40
- distributeViaFTP 641
- distributeViaSvcPull 642
- distributeViaSvcPush 643
- divideFloats 536
- divideInts 537
- divideObjects 538
- document types
  - sessionStartInfo 320
- documentResolverSpec specification 602
- Document folder 239
- document processing
  - resuming 969
  - suspending 976
- document resolver spec, for JMS messages 444
- document retrieval
  - resuming 972
  - suspending 978
- document services 240
- document types
  - alarmInfo 265
  - audit error event information 268
  - audit event information 269
  - callStackItem 270
  - exception information 279
  - gdEndInfo 281
  - gdStartInfo 282
  - migrating to Universal Messaging 996
  - portStatusInfo 303
  - QName 840
  - replicationInfo 305
  - sessionEndInfo 317
  - sessionExpireInfo 318
  - statInfo 322
  - txEndInfo 324
  - txStartInfo 326
- documentation
  - using effectively 21
- documentListToDocument 242
- documents
  - adding to list 523
  - constructing from a list of documents 242
  - converting a byte array to a document 241
  - converting a document to an array of bytes 244
  - converting list from string list 525
  - converting to a String 248
  - converting to JSON strings 515
  - converting to XML strings 1025
  - creating from JSON streams 517
  - creating from JSON strings 518
  - delivering 595
  - delivering and waiting for reply 598

- expanding contents into a list of documents 246
  - publishing 610
  - publishing and waiting for reply 613
  - replying to 618
  - resolving status of 602
  - retrieving redelivery count 609
  - synchronizing 620, 621
  - waiting for 622
  - documentToBytes 244
  - documentToDocumentList 246
  - documentToJSONString 515
  - documentToXMLString 1025
  - documentToXMLValues 248
  - DOM node
    - converting to XMLData 1076
  - dynamic text generation 652
- E**
- eda
    - event 272
    - eventToDocument 274
    - schema\_event 276
  - elapsedNanoTime 202
  - elements, number of in a list 525
  - enableConnection 32
  - enableListener 35
  - enableListenerNotification 40
  - enablePackage 587
  - enablePollingNotification 41
  - enablePublishEvents 41
  - encoding schema, SOAP 817
  - encoding schema, SOAP, version 1.2 817
  - encoding URL strings 919
  - encrypting MIME messages 737
  - ending guaranteed delivery transactions 631
  - Enterprise Gateway specification 700
    - customFilter 702
  - entries
    - inserting into repository 886
  - envelope schema, SOAP 818
  - envelope schema, SOAP, version 1.2 818
  - envelope stream, generating from MIME message 568
  - envelope, for published documents 604
  - error event 276, 277
  - error notification document type 624
  - error service 624
  - event
    - creating subscription for 260
    - sending 296
    - subscribing 299
    - unsubscribing 302
  - event document type 272
  - Event folder 255
  - event handlers
    - get list of subscribers 284
  - event handlers 256
    - alarm specification 264
    - alarmInfo document type 265
    - audit specification 265
    - auditError specification 266
    - auditErrorInfo document type 268
    - auditInfo document type 269
    - callStackItem document type 270
    - exception specification 277
    - exceptionInfo document type 279
    - gdEnd specification 280
    - gdEndInfo document type 281
    - gdStart specification 281
    - gdStartInfo document type 282
    - get list of event types 283
    - modify subscriber info 291
    - portStatus specification 303
    - portStatusInfo document type 303
    - reload event manager settings 304
    - replication specification 304
    - replicationInfo document type 305
    - save event manager settings 313
    - sessionEnd specification 316
    - sessionEndInfo document type 317
    - sessionExpire specification 318
    - sessionExpireInfo document type 318
    - sessionStart specification 319
    - sessionStartInfo document type 320
    - stat specification 320
    - statInfo document type 322
    - subscriptions for 260
    - txEnd specification 324
    - txEndInfo document type 324
    - txStart specification 325
    - txStartInfo document type 326
    - unsubscribing from an event 271
  - event subscribe 309
  - event unsubscribe 311
  - eventrouting
    - EventAcknowledge 308
    - send 306

eventToDocument 274, 295  
 exception specification 277  
 exceptionInfo document type 279  
 exceptions for retry 403  
 exceptions, last trapped in a flow 384, 384  
 execSQL 221  
 executeOSCommand 993  
 exitUnableToUnderstand 830  
 extensions  
   files with no extension, using with FTP put  
   command 117  
 extracting MIME content 564  
 extracting MIME message headers 565, 569

## F

faultTo 851  
 FFDictionary IS document type 363  
 FFSchema IS document type 365  
 fields, removing from pipeline 382  
 File folder 327  
 file object, templates 650  
 file services 328, 434  
 files  
   appending data to 102  
   change owner 151  
   change permissions 150  
   deleting 104  
   listing 103, 105, 112, 117, 154  
   multiple delete 113  
   multiple get 114  
   multiple transfer 115  
   renaming 119  
   retrieving 105  
   saving pipeline contents 392  
   transferring 116  
   transferring SFTP 155  
 findDependants service 368  
 finding universal names 986  
 findReferences service 369  
 fire-and-forget call 634  
 floating point numbers  
   adding 532  
   dividing 536  
   multiplying 539  
   subtracting 544  
 Flow folder 379  
 flow services 380  
 forcing a response string 395

formatDate 203  
 FormatService service 356  
 formatting  
   array of strings 910  
   number to numeric pattern 910  
 freeXMLNode 1032  
 from 852  
 FTP  
   changing directory and listing files 103, 117  
   changing working directory 103, 149  
   client actions 98  
   delete file 104  
   executing non-standard FTP commands 118  
   files with no extension 117  
   filetransfer 116  
   ftp\_no\_extensionkey 117  
   listing files 105, 112, 154  
   login 109  
   logout 112, 153  
   multiple file delete 113  
   multiple file get 114  
   multiple file transfer 115  
   renaming files 119  
   retrieving files 105  
   session info 119  
   transferring files 116  
 FTP servers, submitting requests to 121

## G

gdEnd specification 280  
 gdEndInfo document type 281  
 gdStart specification 281  
 gdStartInfo document type 282  
 generateDocumentTypesFromWSDL 773  
 generateReplicationEvent 643  
 generateUUID 995  
 get 417  
 get (FTP) files 105  
 get (SFTP) files 151  
 getActor 830  
 getBody 831  
 getBodyBlock 776  
 getBodyBlockQNames 778  
 getBodyEntries 832  
 getBodyPartContent 564  
 getBodyPartHeader 565  
 getCanonicalKey 931  
 getCertificateInfo 718

getChecksums 54  
getConnectionStatistics 32  
getContentType 567  
getCurrentDate 204  
getCurrentDateString 204  
getDocument 832  
getEncoding 833  
getEnvelopeStream 568  
getEventTypes 283  
getFaultBlock 778  
getFFDictionaryAsXML service 369  
getFFDictionaryEntryAsXML service 370  
getFFSchemaAsXML service 371  
getFile 333  
getHeader 834  
getHeaderEntries 834  
getInitialSOAPRequest 787  
getLastError 384, 384  
getLocalReleasedList 643  
getMessageAddressingProperties 787  
getMimeHeader 569  
getMustUnderstand 835  
getNativeId 932  
getNextXMLNode 1033  
getNumParts 571  
getPoolInfo 434  
getPrimaryContentType 571  
getProclInfo 225  
getProcs 227  
getQName 836  
getRedeliveryCount service 609  
getRemoteReleasedList 644  
getSession 386  
getStatus 631  
getSubContentType 572  
getSubscribers 284  
getSupportedEncodings 359  
getTableInfo 228  
getTables 230  
getTaskIDs 665  
getTaskInfo 666  
getTrailer service 837  
getTransportInfo 386  
getUsers 1017  
getWebServiceInvocationProperties 794  
getWorkingDays 205  
getXMLNodeIterator 1034  
getXMLNodeType 1037  
guaranteed delivery services 628

guaranteed delivery transactions  
    ending 631  
    getting status of 631  
    invoking service 632  
    restarting 633  
    retrieving results of 633  
    starting 635  
guaranteed one-way call 634

## H

handling large flat files  
    iterator variable of convertToValues service 351  
Hashtable folder 415  
hashtable services 416  
header fields, adding to MIME object 559  
headers, changing HTTP 121  
HTMLDecode 903  
HTMLEncode 904  
HTTP  
    changing header variables 121  
    response code 399  
    response header 399  
    response headers 402  
    submitting requests to servers 121

## I

IData objects  
    converting to JSON strings 515  
    converting to XML strings 1025  
    creating from JSON streams 517  
    creating from JSON strings 518  
IDs, retrieving list of 665  
incrementDate 206  
indexOf 904  
InputStream, converting to byte 422  
inserting table rows 231  
insertXReference 933  
installPackage 587  
instance tag  
    getting from XMLData 1081  
    setting in XMLData 1084  
integers  
    adding 534  
    dividing 537  
    multiplying 541  
    subtracting 546  
invalid dates 194  
invoke, guaranteed delivery service 632

- invokeService 387
- invoking
  - remote services 628
  - report services 650
  - stored procedures 212
- invoking a service
  - guaranteed delivery 635
  - protocols used 404
  - remote webMethods Integration Server 629
- IO services 422
- IS Internal 880
- isAlphanumeric 905
- isDate 906
- isLatchClosed 927
- isNullorBlank 906
- isNumber 907
- iterating through XML nodes 1034
- iterator 388
  
- J**
- JDBC folder 433
- JDBC pools 434
- JDBC-related services 210
- JMS
  - sending an event 296
  - subscribing to an event 299
  - unsubscribing from an event 302
- JMS folder 437
- JMS messages
  - send batch 492
  - sending multiple 492
- JMS trigger
  - create 937
  - delete 964
- journal event 288, 290
- JSON streams
  - converting to IData objects 517
- JSON strings
  - converting to IData objects 518
  - creating from documents 515
- jsonStreamToDocument 517
- jsonStringToDocument 518
  
- K**
- key string lookup 908, 908
- keys, obtaining a list in a data store 889
  
- L**
- large flat file handling
  - iterator variable of convertToValues service 351
- latch operations
  - checking latch status 927
  - closing 926
  - opening 928
- length
  - of a string 907
- list
  - adding, retrieving, or replacing elements 522
  - number of elements 525
  - registered consumer handlers 797
  - registered processors 813
  - registered provider handlers 798
  - universal names 987
- list services 522, 522
- listAdapterConnections 33
- listAdapterListenerNotifications 42
- listAdapterListeners 36
- listAdapterPollingNotifications 42
- listAdapterServices 47
- listAll 988
- listFFDictionaryEntries service 371
- listFiles 335
- listKeys 417, 709
- listRegisteredAdapters 31
- loadEnhancedXMLNode 1037
- loadPKCS7CertChain 720
- loadXMLNode 1047
- locking a repository entry 889, 890, 894
- locking considerations for pub.storage services 881
- log off of FTP server 112, 153
- logging
  - pipeline fields 404
- login to FTP server 109
- login to SFTP server 152
- lookupDictionary 908
- lookupTable 908
- lowercase, converting to 917
- ls 112, 154
  
- M**
- makeString 909
- mark 425
- markSupported 426
- Math folder 529
- math services 530
- max 538

mdelete 113  
 mergeHeaderAndBody 573  
 messageFormat 910  
 messageId 853  
 mget 114  
 migrateDocTypesTriggersToUM 996  
 MIME messages
 

- adding body parts to 555
- adding headers to 559
- creating 561
- creating encrypted messages 737
- creating signed messages 740
- decrypting 742
- discovering content type 567
- discovering number of parts 571
- discovering subtype 572
- generating message stream 568
- getting content from 564
- getting headers from 569
- getting part headers from 565
- getting top-level portion 571
- merging HTTP response into InputStream 573
- sending through SMTP 159
- signing and encrypting 738
- verifying signed messages 744

 MIME services 554  
 min 539  
 mkdir 155  
 moveFile 336  
 modifySubscriber 291  
 mput 115  
 multiple VCS users
 

- removing 1019
- setting 1020

 multiplyFloatList 539  
 multiplyFloats 540  
 multiplyIntList 541  
 multiplyInts 542  
 multiplyObjects 543

**N**

namespace components 692, 692, 692  
 namespace tag
 

- getting from XMLData 1082
- setting in XMLData 1085

 native ID, retrieving 932  
 nerv
 

- eventToDocument 295

node iterator
 

- creating 1034
- freeXMLNode 1032
- getNextXMLNode 1033
- getXMLNodeIterator 1034

 notification error document type 624  
 notifying services 922  
 notifyPackageRelease 645  
 ns
 

- getting tags from XMLData 1082
- setting in XMLData 1085

 numeric values
 

- dividing 537

 numeric values
 

- adding 532, 533, 534, 535
- dividing 538
- multiplying 539, 540, 541, 542, 543
- subtracting 544, 546

 numericFormat 910

**O**

OAuth client 145  
 objects
 

- adding 535
- dividing 538
- multiplying 543
- subtracting 546

 objectToString 912  
 obtaining information about 434  
 one-time tasks
 

- adding to scheduler 660
- updating to scheduler 676

 one-way call, guaranteed 634  
 OpenID Connect, userInfoSpec 715  
 openLatch 928  
 outbound passwords 694
 

- creating a WmSecureString 720
- internal vs. public 708
- listing keys 709
- removing 710
- retrieving 708
- setting 707
- updating 710

 output templates 650

**P**

package management services 584  
 Package Release Registry

- adding entries 639
- deleting entries 640
- obtaining local server list 643
- package replication services 638
- packageCreation 646
- packages
  - activating 584
  - backing up 585
  - disabling 586
  - enabling 587
  - hotdeployment 590
  - installing 587
  - recovering 589
  - reloading 590
- padLeft 913
- padRight 914
- pattern strings, date patterns 190
- pipeline
  - applying templates to 652
  - inserting Session object 386
  - removing fields 382
  - restoring previously saved 387, 390
  - saving contents to a file 392
  - saving into memory 392
  - tracing 404
  - using previously saved keys and values 391
  - validating against a document type 690
- pipeline services 380
- PKCS7 SignedData objects
  - creating 711
- portStatus specification 303
- portStatusInfo document type 303
- Pre-8.2 compatibility mode property 1009
- problemAction 853
- problemHeaderQName 854
- problemIRI 855
- processCertsOnlyData 742
- processEncryptedData 742
- processing
  - resuming for triggers 969
  - suspending for triggers 976
- processMessage 814
- processRPCMessage 815
- processSignedData 744
- protocols, retrieving information about 404
- protocols, retrieving information about 386
- pseudorandom number generator 543
- pub.b.client.sftp
  - chmod 150
- pub.b.publish.notification
  - error 624
- pub.b.xmldata
  - getInstanceTag 1081
- pub
  - flow
    - setResponse2 397
- pub.event
  - portStatus 303
- pub.art
  - listRegisteredAdapters 31
- pub.art.connection
  - disableConnection 31
  - enableConnection 32
  - getConnectionStatistics 32
  - queryConnectionState 34
- pub.art.listener
  - disableListener 34
  - enableListener 35
  - queryListenerState 37
  - setListenerNodeConnection 38
- pub.art.notification
  - disableListenerNotification 39
  - disablePollingNotification 39
  - disablePublishEvents 40
  - enableListenerNotification 40
  - enablePollingNotification 41
  - enablePublishEvents 41
  - listAdapterPollingNotifications 42
  - queryListenerNotificationState 43
  - queryPollingNotificationState 44
  - resumePollingNotification 45
  - setListenerNotificationNodeListener 46
  - setPollingNotificationNodeConnection 46
  - suspendPollingNotification 47
- pub.art.service
  - listAdapterServices 47
  - setAdapterServiceNodeConnection 48
- pub.art.transaction
  - commitTransaction 49
  - rollbackTransaction 50
  - setTransactionTimeout 50
  - startTransaction 51
- pub.assets
  - getChecksums 54
- pub.client
  - ftp 98, 99
  - http 121
  - smtp 159

- soapClient 163
- soapHTTP 180
- soapRPC 183
- pub.client.ftp
  - append 102
  - cd 103
  - cdls 103
  - delete 104
  - dir 105
  - get 105
  - login 109
  - logout 112, 153
  - ls 112, 154
  - mdelete 113
  - mget 114
  - mput 115, 115
  - put 116, 155
  - putCompletedNotification 117
  - quote 118
  - rename 119
  - sessioninfo 119
- pub.client.oauth
  - executeRequest 145
- pub.client.sftp
  - cd 149
  - chgrp 149
  - chown 151
  - get 151
  - login 152
  - mkdir 155
  - pwd 156
  - rename 157
  - rm 157
  - rmdir 158
  - symlink 158
- pub.date
  - calculateDateDifference 195
  - compareDates 196
  - currentNanoTime 197
  - dateBuild 198
  - dateTimeBuild 199
  - dateTimeFormat 201
  - elapsedNanoTime 202
  - formatDate 203
  - getCurrentDate 204
  - getCurrentDateString 204
  - getWorkingDays 205
  - incrementDate 206
- pub.db
  - call 212
  - clearTransaction 214
  - close 215
  - closeAll 216
  - commit 216
  - connect 217
  - delete 219
  - execSQL 221
  - getProclInfo 225
  - getProcs 227
  - getTableInfo 228
  - getTables 230
  - insert 231
  - query 233
  - rollback 235
  - startTransaction 236
  - update 237
- pub.document
  - bytesToDocument 241
  - documentListToDocument 242
  - documentToBytes 244
  - documentToDocumentList 246
  - documentToXMLValues 248
  - XMLValuesToDocument 253
- pub.e vent.eda
  - event 272
- pub.event
  - addSubscriber 260
  - alarm 264
  - alarmInfo 265
  - audit 265
  - auditError 266
  - auditErrorInfo 268
  - auditInfo 269
  - callstack Item 270
  - deleteSubscriber 271
  - error 276
  - errorInfo 277
  - exception 277
  - exceptionInfo 279
  - gdEnd 280
  - gdEndInfo 281
  - gdStart 281
  - gdStartInfo 282
  - getEventTypes 283
  - getSubscribers 284
  - journal 288
  - journalInfo 290
  - modifySubscriber 291

- portStatusInfo 303
- reloadEventManagerSettings 304
- replication 304
- replicationInfo 305
- saveEventManagerSettings 313
- security 313
- securityInfo 315
- sessionExpireInfo 318
- sessionEnd 316
- sessionEndInfo 317
- sessionExpire 318
- sessionStart 319
- sessionStartInfo 320
- stat 320
- statInfo 322
- txEnd 324
- txEndInfo 324
- txStart 325
- txStartInfo 326
- pub.event.eda
  - eventToDocument 274
  - schema\_event 276
- pub.event.nerv
  - eventToDocument 295
  - send 296
  - subscribe 299
  - unsubscribe 302
- pub.event.routing
  - EventAcknowledge 308
  - send 306
  - subscribe 309
  - unsubscribe 311
- pub.file
  - bytesToFile 330
  - checkFileExistence 331
  - copyFile 332
  - deleteFile 333
  - getFile 333
  - listFiles 335
  - moveFile 336
  - readerToFile 337
  - streamToFile 338
  - stringToFile 339
- pub.flow
  - setResponseCode 399
- pub.flatFile
  - convertToString 344
  - convertToValues 349
  - FormatService 356
  - getSupportedEncodings 359
- pub.flatFile.generate
  - createDocumentType 359
  - createFFDictionary 360
  - deleteFFDictionary 360
  - deleteFFDictionaryEntry 361
  - deleteFFSchema 362
  - FFDictionary IS document type 363
  - FFSchema IS document type 365
  - findDependants 368
  - findReferences 369
  - getFFDictionaryAsXML 369
  - getFFDictionaryEntryAsXML 370
  - getFFSchemaAsXML 371
  - listFFDictionaryEntries 371
  - saveXMLAsFFDictionary 372
  - saveXMLASFFSchema 374
  - updateFFDictionaryEntryFromXML 375
- pub.flow
  - clearPipeline 382
  - debugLog 382
  - getLastError 384, 384
  - getRetryCount 385, 949
  - getSession 386
  - getTransportInfo 386
  - invokeService 387
  - iterator 388
  - restorePipeline 390
  - restorePipelineFromFile 391
  - savePipeline 392
  - savePipelineToFile 392
  - setCustomContextID 394
  - setResponse 395
  - setResponseHeader 399
  - setResponseHeaders 402
  - throwExceptionForRetry 403
  - tracePipeline 404
  - transportInfo 404
- pub.hashtable
  - containsKey 416
  - createHashtable 417
  - get 417
  - listKeys 417
  - put 418
  - remove 418
  - size 419
- pub.io
  - bytesToStream 423
  - close 424, 424

- createByteArray 424, 424
- mark 425
- markSupported 426
- read 427, 427
- readAsString 427
- readerToString 428
- reset 429, 429
- skip 429, 429
- streamToBytes 430, 430
- streamToReader 431
- streamToString 431
- stringToReader 432
- stringToStream 432
- pub.jms
  - documentResolverSpec 444
  - reply 454
  - send 461
- pub.jms.wmjms
  - sendStream 510
- pub.json
  - documentToJSONString 515
  - jsonStreamToDocument 517
  - jsonStringToDocument 518
- pub.list
  - addItemToVector 522
  - appendToDocumentList 523
  - appendToStringList 524
  - sizeOfList 525
  - stringListToDocumentList 525
  - vectorToArray 526
- pub.math
  - absoluteValue 532
  - addFloatList 532
  - addFloats 533
  - addIntList 534
  - addInts 535
  - addObjects 535
  - divideFloats 536
  - divideInts 537
  - divideObjects 538
  - max 538
  - min 539
  - multiplyFloatList 539
  - multiplyFloats 540
  - multiplyIntList 541
  - multiplyInts 542
  - multiplyObjects 543
  - randomDouble 543
  - roundNumber 544
  - subtractFloats 544
  - subtractInts 546
  - subtractObjects 546
  - toNumber 547
- pub.mime
  - addBodyPart 555
  - addMimeHeader 559
  - createMimeData 561
  - getBodyPartContent 564
  - getBodyPartHeader 565
  - getContentType 567
  - getEnvelopeStream 568
  - getMimeHeader 569
  - getNumParts 571
  - getPrimaryContentType 571
  - getSubContentType 572
  - mergeHeaderAndBody 573
- pub.packages
  - backupPackage 585
  - disablePackage 586
  - enablePackage 587
  - installPackage 587
  - recoverPackage 589
  - reloadPackage 590
- pub.packages.hotdeployment
  - cancel 590
- pub.publish
  - deliver 595
  - deliverAndWait 598
  - documentResolverSpec 602
  - envelope 604
  - getRedeliveryCount 609
  - publish 610
  - publishAndWait 613
  - reply 618
  - syncDocuments 620
  - syncToProvider 621
  - waitForReply 622, 622
- pub.remote
  - invoke 629
- pub.remote.gd
  - end 631
  - getStatus 631
  - invoke 632
  - restart 633
  - retrieve 633
  - send 634
  - start 635
  - submit 635

- pub.replicator
  - addReleaseRegistryEntry 639
  - deleteReleaseRegistryEntry 640
  - distributeViaFTP 641
  - distributeViaSvcPull 642
  - distributeViaSvcPush 643
  - generateReplicationEvent 643
  - getLocalReleasedList 643
  - getRemoteReleasedList 644
  - notifyPackageRelease 645
  - packageCreation 646
- pub.report
  - runFileTemplate 650
  - runFileTemplateOnPipe 651
  - runStringTemplate 651
  - runStringTemplateOnPipe 652
  - runTemplate 652
  - runTemplateOnPipe 653
- pub.security.util
  - convertSecureString 721
- pub.scheduler
  - addComplexTask 657
  - addOnetimeTask 660
  - addRepeatingTask 662
  - cancelTask 665
  - getTaskIDs 665
  - getTaskInfo 666
  - getUserTaskList 670
  - migrateTasksToJDBC 670
  - resumeTask 671
  - suspendTask 672
  - updateComplexTask 673
  - updateOneTimeTask 676
  - updateRepeatingTask 678
- pub.schema
  - createXSD 685
  - validate 687
  - validatePipeline 690
- pub.schema.w3c 691
  - datatypes 692
  - structures 692
  - xml 692
  - xsi 692
- pub.securty
  - userInfoSpec 715
- pub.security
  - clearKeyAndChain 698
  - setKeyAndChain 699
  - setKeyAndChainFromBytes 699
- pub.security.enterpriseGateway
  - alertSpec 700
  - customFilterSpec 702
- pub.security.keystore
  - getCertificate 703
  - getKeyAndChain 704, 704
  - getTrustedCertificates 704
  - setKeyAndChain 705
- pub.security.keystore.pkcs7
  - sign 705
- pub.security.outboundPasswords
  - listKeys 709
  - removePassword 710
  - setPassword 707
  - updatePassword 710
- pub.security.pkcs7
  - sign 711
  - verify 712
- pub.security.util
  - createMessageDigest 718
  - createSecureString 720
  - destroySecureString 721
  - getCertificateInfo 718
  - loadPKCS7CertChain 720
- pub.smime
  - createCertsOnlyData 737
  - createEncryptedData 737
  - createSignedAndEncryptedData 738
  - createSignedData 740
  - processCertsOnlyData 742
  - processEncryptedData 742
  - processSignedData 744
- pub.smime.keystore
  - createSignedAndEncryptedData 747
  - createSignedData 748
  - processEncryptedData 748
- pub.soa.p.wsa
  - to 858
- pub.soap.handler
  - registerConsumer 802
- pub.soap.handler
  - addBodyBlock 762
  - addFaultBlock 764
  - addHeaderBlock 766
  - generateDocumentTypesFromWSDL 773
  - getBodyBlock 776
  - getBodyBlockQNames 778
  - getFaultBlock 778
  - getInitialSOAPRequest 787

- getMessageAddressingProperties 787
- getServicePipeline 791
- getWebServiceInvocationProperties 794
- hasFaultMessage 796
- listConsumer 797
- listProvider 798
- registerConsumer 770, 783, 790, 793, 795, 800, 800, 806, 807, 808, 809
- registerProvider 800
- removeBodyBlock 804
- removeHeaderBlock 804
- unregisterConsumer 810
- unregisterProvider 810
- updateFaultBlock 810
- pub.soap.processor
  - list 813
  - processMessage 814
  - processRPCMessage 815
  - registerProcessor 815
  - unregisterProcessor 817
- pub.soap.schema
  - encoding 817
  - encoding\_1\_2 817
  - envelope 818
  - envelope\_1\_2 818
- pub.soap.utils
  - stringToSoapData 848
- pub.soap.util
  - addBodyEntry 818
  - addHeaderEntry 819
  - addTrailer 821
  - createSoapData 827
  - exitUnableToUnderstand 830
  - getActor 830
  - getBody 831
  - getBodyEntries 832
  - getDocument 832
  - getEncoding 833
  - getHeader 834
  - getHeaderEntries 834
  - getMustUnderstand 835
  - getQName 836
  - getTrailer 837
  - QName 840
  - removeBodyEntry 840
  - removeHeaderEntry 841
  - removeTrailer 842
  - requestResponseSpec 843
  - soapDataToBytes 845
  - soapDataToString 845
  - streamToSoapData 847
  - validateSoapData 849
- pub.soap.wsa
  - action 850
  - faultTo 851
  - from 852
  - messageID 853
  - problemAction 853
  - problemHeaderQName 854
  - problemIRI 855
  - relatesTo 855
  - replyTo 856
  - retryAfter 857
  - schema\_wsa 858
- pub.soap.wsa.submission
  - action 858
  - faultTo 859
  - from 861
  - messageID 863
  - relatesTo 863
  - replyTo 864
  - retryAfter 866
  - schema\_wsa\_submission 867
  - to 867
- pub.soap.wsrn
  - closeSequence 867
  - createSequence 869
  - sendAcknowledgementRequest 873
  - waitUntilSequenceCompleted 876
- pub.storage
  - add 886
  - checkpoint restart example 883
  - closeStore 886
  - data store locking 881
  - deleteStore 887
  - entry locking 881
  - get 888
  - keys 889
  - listLocks 889
  - lock 890
  - lock duration 882
  - lock wait time 882
  - put 893
  - registerStore 894
  - releaseLocks 894
  - remove 895
  - short-term store 880
  - unlock 896

- pub.string
  - base64Decode 900
  - base64Encode 900
  - bytesToString 901
  - compareStrings 902
  - concat 902
  - HTMLDecode 903
  - HTMLEncode 904
  - indexOf 904
  - isAlphanumeric 905
  - isDate 906
  - isNullorBlank 906
  - isNumber 907
  - length 907
  - lookupDictionary 908
  - lookupTable 908
  - makeString 909
  - messageFormat 910
  - numericFormat 910
  - objectToString 912
  - padLeft 913
  - padRight 914
  - replace 914
  - stringToBytes 915
  - substitutePipelineVariables 916
  - substring 916
  - tokenize 917
  - toLower 917
  - toUpper 918
  - trim 919
  - URLDecode 919
  - URLEncode 919
- pub.sync
  - notify 922
  - shutdown 923
  - wait 923
- pub.synchronization.latch
  - closeLatch 926
  - isLatchClosed 927
  - openLatch 928
- pub.synchronization.xref
  - createXReference 929
  - deleteByObjectId 930
  - deleteXReference 930
  - getCanonicalKey 931
  - getNativeId 932
  - insertXReference 933
- pub.trigger
  - createJMSTrigger 937
  - createTrigger 951
  - deleteJMSTrigger 964
  - resumeProcessing 969
  - resumeRetrieval 972
  - suspendProcessing 976
  - suspendRetrieval 978
- pub.universalName
  - find 986
  - list 987
  - listAll 988
- pub.utils
  - deepClone 992
  - executeOSCommand 993
  - generateUUID 995
  - getServerProperty
    - retrieves value of a server property 995
  - transcode 1006
- pub.utils.messaging
  - migrateDocTypesTriggersToUM 996
- pub.utils.ws
  - setCompatibilityModeFalse 1009
- pub.vcs.admin
  - getUsers 1017
  - removeCurrentUser 1018
  - removeMultipleUsers 1019
  - setCurrentUser 1019
  - setMultipleUsers 1020
- pub.xml
  - documentToXMLString 1025
  - freeXMLNode 1032
  - getNextXMLNode 1033
  - getXMLNodeIterator 1034
  - getXMLNodeType 1037
  - loadEnhancedXMLNode 1037
  - loadXMLNode 1047
  - queryXMLNode 1055
  - xmlNodeToDocument 1058
  - xmlStringToEnhancedXMLNode 1069
  - xmlStringToXMLNode 1072
- pub.xmldata
  - domNodeToXMLData 1076
  - getNamespaceTag 1082
  - setAttribute
    - XMLData
    - setting attribute 1084
  - setInstanceTag
    - XMLData
    - setting instance tag 1084
  - setNamespaceTag

- XMLData
    - setting namespace tag 1085
    - xmlDataToXMLString 1086
  - pub.xmlData
    - getAttributes
      - attributes
        - getting from XMLData 1080
  - pub.xslt.Transformations
    - transformSerialXML 1090
  - publish service 610
  - publish/subscribe, latching and cross-referencing operations 926
  - publishable document types
    - synchronizing 621
  - publishAndWait service 613
  - publishing packages 638
    - addReleaseRegistryEntry 639
    - deleteReleaseRegistryEntry 640
    - distributeViaFTP 641
    - distributeViaSvcPull 642
    - distributeViaSvcPush 643
    - generateReplicationEvent 643
    - getLocalReleasedList 643
    - getRemoteReleasedList 644
    - notifyPackageRelease 645
    - packageCreation 646
  - put 418
  - put (FTP) files 116
  - put (SFTP) files 155
- Q**
- QName document type 840
  - queryConnectionState 34
  - querying a database 233
  - queryListenerNotificationState 43
  - queryListenerState 37
  - queryPollingNotificationState 44
  - queryXMLNode 1055
  - quote 118
- R**
- random number generator 543
  - randomDouble 543
  - raw offset for time zones 191
  - read 427
  - readAsString 427
  - readerToFile 337
  - readerToString 428
  - records, deleting particular type 930
  - recoverPackage 589
  - recurring tasks, adding to scheduler 662
  - redelivery count, retrieving for documents 609
  - registering
    - consumer handlers 790
    - consumer handlers 770, 783, 793, 795, 800, 800, 802, 806, 807, 808, 809
    - providerhandlers 800
    - unregistering consumer handlers 810
    - unregistering provider handlers 810
  - registering data stores 894
  - registerProcessor 815
  - registerStore 894
  - relatesTo 855
  - release registry. 639
  - reloadEventManagerSettings 304
  - reloadPackage 590
  - remote file, appending data to 102
  - remote procedure call, submitting 184
  - remote servers, invoking services on 628, 629
  - remove 418
  - removeBodyBlock 804
  - removeBodyEntry 840
  - removeCurrentUser 1018
  - removeHeaderBlock 804
  - removeHeaderEntry 841
  - removeMultipleUsers 1019
  - removeTrailer 842
  - removing entries from data stores 895
  - renaming files on an FTP server 119
  - renaming files on an SFTP server 157
  - repeating tasks, updating to scheduler 678
  - replacing strings 914
  - replication specification 304
  - replicationInfo document type 305
  - replicator services 638
  - reply documents
    - for join conditions 619
    - retrieving for asynchronous request 622
    - sending 618
  - reply service 618
  - replyTo 856
  - report services 650
  - repository
    - closing data store 886
    - creating/opening 894
    - deleting data store 887
    - inserting entries 886

- inserting/updating entries 893
  - list of keys in 889
  - locking entries 889, 890, 894
  - registering 894
  - removing entries 895
  - retrieving keys from 889
  - retrieving values from 888
  - unlocking entries 896
  - request/ reply model
    - multiple reply documents 616
  - request/reply model
    - deliverAndWait service 601
    - delivering a request 598
    - publishAndWait service 616
    - publishing request 613
    - publishing request to a specific client 598
    - retrieving reply 622
    - sending replies 618
    - waitForReply service 622
    - waiting for reply 615
  - requestResponseSpec 843
  - reset 429
  - response string, forcing 395
  - restarting a guaranteed delivery transaction 633
  - restorePipeline 390
  - restorePipelineFromFile 391
  - resumeListener 37
  - resumePollingNotification 45
  - resumeTask 670, 670, 671
  - resuming
    - document processing for triggers 969
    - document retrieval for triggers 972
  - retrieval
    - resuming for triggers 972
    - suspending for triggers 978
  - retrieving
    - documents from a local file system 328, 416
    - files from FTP server 105
    - files from SFTP server 151
    - MIME content 564
    - MIME message headers 565, 569
    - results of guaranteed delivery transaction 633
  - retrieving reply 622
  - retry count, retrieving 385
  - retry exception 403
  - retry limit
    - setting to zero 949
  - retryAfter 857
  - rm 157
  - rmdir 158
  - rollback 235
  - rollbackTransaction 50
  - roundNumber 544
  - RPC, submitting 184
  - runFileTemplate 650
  - runFileTemplateOnPipe 651
  - runStringTemplate 651
  - runStringTemplateOnPipe 652
  - runTemplate 652
  - runTemplateOnPipe 653
- S**
- S/MIME messages
    - creating certs-only message 737
    - creating encrypted messages 737
    - creating signed messages 740
    - decrypting 742
    - extracting certificates from certs-only message 742
    - verifying signed messages 744
  - S/MIME services 736
  - saveEventManagerSettings 313
  - savePipeline 392
  - savePipelineToFile 392
  - saveXML AsFFDictionary service 372
  - saveXMLASFFSchema service 374
  - sc hemas
    - schema\_ws a\_submission 867
  - scheduler
    - adding complex tasks 657
    - adding one-time tasks 660
    - canceling tasks 665
    - recurring tasks 662
    - resuming tasks 671
    - retrieving list of task IDs 665
    - retrieving task info 666
    - suspending tasks 672
    - updating complex tasks 673
    - updating one-time tasks 676
    - updating repeating tasks 678
  - scheduler services 656
  - schema
    - for events 276
  - schema domain 689
  - schema services 684
  - schemas
    - components 692

- creating 685
- datatypes 692
- for validating 687
- namespace components 692, 692
- retrieving tables 230
- SOAP encoding schema, version 1.2 817
- SOAP encoding schema 817
- SOAP envelope schema 818
- SOAP envelope schema, version 1.2 818
- wsa 858
- XML datatypes 691
- sending a guaranteed call 634
- sending STOR or STOU commands to remove server 99, 115, 116
- sendStream 510
- server log
  - pipeline field names and values 404
  - writing to 382
- services
  - default access permissions 24
  - listing of 21
- Session object, inserting in pipeline 386
- sessionEnd specification 316
- sessionEndInfo document type 317
- sessionExchangeInfo document type 318
- sessionExpire specification 318
- sessioninfo 119
- sessionStart specification 319
- sessionStartInfo document type 320
- setAdapterServiceNodeConnection 48
- setCurrentUser 1019
- setCustomContextID 394
- setKeyAndChain 699, 699
- setListenerNodeConnection 38
- setListenerNotificationNodeListener 46
- setMultipleUsers 1020
- setPollingNotificationNodeConnection 46
- setResponseHeaders 402
- setResponse 395
- setResponse2 397
- setResponseCode 399
- setResponseHeader 399
- setTransactionTimeout 50
- SFTP
  - changing file owner 151
  - changing owner group 149
  - changing permissions 150
  - file transfer 155
  - login 152
  - re move files 157
  - remove directory 158
  - renaming files 157
  - retrieving files 151
  - symbolic link between files 158
  - transferring files 155
  - view remote working directory 156
- sftp working directory, changing 149
- short-term store 880
- shutdown service 923
- signed data object, creating 711
- signing MIME messages 740, 744
- size 419
- sizeOfList 525
- skip 429
- SMTP servers, submitting requests to 121
- SMTP, sending MIME messages 159
- SOAP
  - actor attribute 830
  - adding body entries 818
  - adding header entries 819
  - converting SOAP object to Byte Array 845
  - converting SOAP object to String 845
  - converting stream to SOAP object 847
  - converting string to SOAP object 848
  - creating SOAP objects 827
  - encoding schema, version 1.2 817
  - encoding schema 817
  - envelope schema 818
  - envelope schema, version 1.2 818
  - executing the default processor 814
  - executing the default RPC processor 815
  - getting body entries 831, 832, 837
  - getting encoding entries 833
  - getting entire message 832
  - getting header entries 834
  - getting QNames 836
  - listing handlers, consumer 797
  - listing handlers, provider 798
  - listing processors 813
  - mustUnderstand attribute 830, 835
  - registering a processor 815
  - registering handlers, consumer 770, 783, 790, 793, 795, 800, 800, 802, 806, 807, 808, 809
  - registering handlers, provider 800
  - removing body entries 840
  - removing header entries 841
  - removing trailers 842
  - sending a SOAP message 180

- sending a SOAP message over HTTP 163
- specification for custom processor 843
- specification for target service 843
- submitting an RPC request 183
- unregistering handlers, provider 810
- unregistering handlers, consumer 810
- unregistering processors 817
- updating Fault Code and Fault String 810
- validating SOAP objects 849
- WS-Addressing wsa
  - faultTo 851
- WS-Addressing wsa
  - from 852
- WS-Addressing wsa
  - action 850
  - message ID 853
  - problemAction 853
  - problemHeader QName 854
  - problemURI 855
  - relatesTo 855
  - replyTo 856
  - retryAfter 857
  - to 858
- WS-Addressing wsa.submission
  - action 858
  - faultTo 859
  - from 861
  - message ID 863
  - relatesTo 863
  - replyTo 864
  - retryAfter 866
  - to 867
- SOAP messages, inserting trailer 821
- SOAP services 752
- soapClient 163
- soapDataToBytes 845
- soapDataToString 845
- soapHTTP 180
- soapRPC 183
- special characters
  - handling in XML strings 1028
- specifications
  - sessionEnd event handler 316
- specification
  - audit error event 266
- specifications
  - alarm event handler 264
  - audit event handler 265
  - exception event handler 277
  - gdEnd 280
  - gdStart 281
  - portStatus event handler 303
  - replication event handler 304
  - sessionExpire event handler 318
  - sessionStart event handler 319
  - stat event handler 320
  - txEnd event handler 324
  - txStart event handler 325
- SQL statements, executing 221
- starting a guaranteed delivery transaction 635
- startTransaction 51, 236
- stat specification 320
- statInfo document type 322
- status of guaranteed delivery transactions 631
- STOR or STOU commands, sending to remote server
  - using pub.client
    - ftp service 99
  - using pub.client.ftp
    - mput service 115
    - put service 116
- storage services 880, 880
- stored procedures
  - invoking 212
  - retrieving information about 225
  - retrieving names of 227
- streamToBytes 430, 430
- streamToFile 338
- streamToReader 431
- streamToSoapData 847
- streamToString 431
- string
  - substituting a pipeline variable 916
- string array, formatting 910
- stringListToDocumentList 525
- strings
  - adding to list 524
  - base64 decoding 900
  - base64 encoding 900
  - byte array conversion 901
  - comparing 902
  - concatenating 902
  - concatenating an array of strings 909
  - converted from documents 248
  - converting list to document list 525
  - converting to byte array 915
  - converting to lower case 917
  - converting to uppercase 918

- decoding URL-encoded 919
  - HTML to native characters 903
  - length of 907
  - native characters to HTML 904
  - padding beginning 913
  - padding end 914
  - replacing all 914
  - returning substring of 916
  - tokenizing into an array 917
  - transformation services 898
  - trimming white space 919
  - URL-encoding 919
  - XML values to a document 253
  - stringToBytes 915
  - stringToFile 339
  - stringToReader 432
  - stringToSoapData 848
  - stringToStream 432
  - submission action 858
  - submission
    - faultTo 859
    - from 861
    - messageID 863
    - relatesTo 863
    - replyTo 864
    - retryAfter 866
    - to 867
  - submitting an asynchronous call to a remote server 635
  - subscribing to packages 638
  - subscriptions, creating 260
  - substituting a pipeline variable 916
  - substring 916
    - indexing first occurrence 904
  - subtractFloats 544
  - subtractInts 546
  - subtractObjects 546
  - subtype, getting from MIME message 572
  - sum of numeric values 532, 533, 534, 535
  - suspended tasks, resuming 671
  - suspending
    - document processing for triggers 976
    - document retrieval for triggers 978
  - suspendListener 38
  - suspendPollingNotification 47
  - suspendTask 672
  - symlink 158
  - Synchronization folder 925
  - synchronization services
    - coordinating the execution of services 922
    - performing latching and cross-referencing operations 926
  - synchronous call to a remote server 632
  - synchronous request / r eply
    - delivering a re quest 600
  - synchronous request/reply
    - description of 601, 617
    - publishing a request 615
  - syncToBroker 620
  - syncToProvider 621
- T**
- tables
    - inserting rows 231
    - removing rows 219
    - retrieving column information 228
    - retrieving names of 230
    - retrieving rows from 233
    - updating all rows 237
  - tasks
    - adding complex to scheduler 657
    - adding one-time to scheduler 660
    - adding recurring to scheduler 662
    - migrating to external database 670
    - obtaining information about 666
    - obtaining list of 670
    - removing from scheduler 665
    - resuming 671
    - retrieving IDs for 665
    - suspending 672
    - updating complex to scheduler 673
    - updating one-time to scheduler 676
    - updating repeating to scheduler 678
  - templates
    - applying to a document 650
    - applying to pipeline 651, 652
    - from string objects 651
  - time zones 191
  - time, converting formats 201
  - to 858
  - tokenizing a string 917
  - toLowerCase 917
  - toNumber 547
  - toUpperCase 918
  - tracePipeline 404
  - trailers, adding to SOAP messages 821
  - transactional state, clearing 214

transactions, ending guaranteed delivery 631  
 transcoding service 1006  
 transforming XML 1090  
 transient error, description of 403  
 transportInfo 404  
 triggers  
   suspending document retrieval 978  
 Trigger folder 935  
 trigger services  
   performance 949  
   retry count 385  
   retry count, retrieving 949  
   retrying 403  
 triggers  
   creating JMS 937  
   deleting JMS 964  
   management services 936  
   resuming document processing 969  
   resuming document retrieval 972  
   retry, setting to 0 949  
   suspending processing 976  
 trimming white space 919  
 txEnd specification 324  
 txEndInfo document type 324  
 txStart specification 325  
 txStartInfo document type 326

## U

universal name services 986  
 universal names  
   finding service names for 986  
   listing the registry 987  
 unlocking entries in data stores 896  
 unregisterProcessor 817  
 unsubscribing from an event 271  
 updateComplexTask 673  
 updateFFDictionaryEntryFromXML service 375  
 updateOneTimeTask 676  
 updateRepeatingTask 678  
 updating  
   rows in a data base table 237  
 uppercase, converting to 918  
 URLDecode 919  
 URLEncode 919  
 user management, VCS 1016  
   get user names 1017  
   remove current user 1018  
   remove multiple users 1019

  set current user 1019  
   set multiple users 1020  
 userInfoSpec 715  
 Utils services  
   retrieves value of server property 992

## V

validatePipeline 690  
 validateSoapData 849  
 validating an object 687  
 VCS folder 1015  
 VCS Integration feature 1016  
   getting users 1017  
   remove current user 1018  
   remove multiple users 1019  
   set current user 1019  
   set multiple users 1020  
 vectorToArray 526  
 verifying digital signatures 712

## W

waitForReply service 622  
 waiting for delivery from a notifying service 923  
 watt.security.ope.AllowInternalPasswordAccess 708  
 Web service descriptor  
   consumer  
     registering the handler 807  
 Web service descriptor  
   consumer  
     registering the handler 800  
   consumer  
     registering the handler 770, 783, 790, 793, 795, 800, 802, 806, 808, 809  
     unregistering the handler 810  
   Pre-8.2 compatibility mode property 1009  
   provider  
     registering the handler 800  
     unregistering the handler 810  
 webMethods messaging triggers  
   migrating to Universal Messaging 996  
 white space, trimming 919  
 WmSecureString 720  
   converting 721  
   creating 720  
   destroying 721  
 working directory, changing 103  
 wsrn

- closeSequence 867
- createSequence 869
- sendAcknowledgementRequest 873
- waitUntilSequenceCompleted 876

## X

### XML

- datatypes 691
- pub.schema.w3c 692, 692

### XML documents

- converting to IData objects 1058
- converting to XML nodes 1069, 1072

### XML nodes

- creating from XML string 1069, 1072
- extracting data from 1055
- freeing 1032
- getting type info 1037
- loading from URL 1037, 1047
- querying 1055

### XML schemas

- components 692
- datatypes 692

### XML services 1024

### XML String

- converting from XMLData 1086

### XML strings

- creating from documents 1025

### XML strings, creating from documents

- handling special characters 1028

### XML values

- converting from a document 248

### XMLData 1084, 1085

- converting DOM node to 1076
- converting to XML String 1086
- getting attributes from 1080
- getting instance tag 1081
- getting namespace tag 1082
- nsi tag 1082
- xsi tag 1081

### XMLData services 1076

### xmlNodeToDocument 1058

### xmlStringToEnhancedXMLNode 1069

### xmlStringToXMLNode 1072

### XMLValuesToDocument 253

### XQL queries 1055

### xsi

- getting tags from XMLData 1081
- setting in XMLData 1084

- xsi, setting 1084, 1085

- XSLT folder 1089

### XSLT services

- maintaining the stylesheet cache 1090

- transforming an XML stream 1090