

webMethods Integration Server Built- In Services Reference

Version 9.7

October 2014

This document applies to webMethods Integration Server Version 9.7 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 © 2007-2014 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://documentation.softwareag.com/legal/>.

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://documentation.softwareag.com/legal/> 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 and license terms, please refer to "License Texts, Copyright Notices and Disclaimers of Third Party Products". This document is part of the product documentation, located at <http://documentation.softwareag.com/legal/> and/or in the root installation directory of the licensed product(s).

Table of Contents

About this Guide	21
Document Conventions.....	25
Documentation Installation.....	25
Online Information.....	26
ART Folder	27
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

Cache Folder	53
About Checkpoint Restart.....	54
Summary of Elements in this Folder.....	55
pub.cache:containsKey.....	58
pub.cache:get.....	58
pub.cache:getKeys.....	60
pub.cache:put.....	61
pub.cache:remove.....	62
pub.cache:search.....	63
pub.cache.admin:clearAllCaches.....	66
pub.cache.admin:clearCache.....	67
pub.cache.admin:disableCache.....	68
pub.cache.admin:enableCache.....	69
pub.cache.admin:evictExpiredElements.....	69
pub.cache.admin:isCacheDisabled.....	70
pub.cache.atomic:putIfAbsent.....	71
pub.cache.atomic:remove.....	72
pub.cache.atomic:replace.....	73
pub.cache.atomic:replaceIfKeyExists.....	74
pub.cache.bulk:isClusterBulkLoadEnabled.....	75
pub.cache.bulk:isNodeBulkLoadEnabled.....	76
pub.cache.bulk:setNodeBulkLoadEnabled.....	77
pub.cache.bulk:waitUntilClusterBulkLoadComplete.....	78
pub.cache.lock:acquireLock.....	78
pub.cache.lock:isLockedByCurrentThread.....	80
pub.cache.lock:releaseLock.....	81
Client Folder	83
Summary of Elements in this Folder.....	84
pub.client:ftp.....	88
pub.client:ftp:append.....	92
pub.client:ftp:cd.....	93
pub.client:ftp:cdls.....	93
pub.client:ftp:delete.....	94
pub.client:ftp:dir.....	95
pub.client:ftp:get.....	95
pub.client:ftp:getCompletedNotification.....	98
pub.client:ftp:login.....	98
pub.client:ftp:logout.....	101
pub.client:ftp:ls.....	102
pub.client:ftp:mdelete.....	103
pub.client:ftp:mget.....	104
pub.client:ftp:mput.....	105
pub.client:ftp:put.....	106
pub.client:ftp:putCompletedNotification.....	107

pub.client.ftp:quote.....	108
pub.client.ftp:rename.....	108
pub.client.ftp:sessioninfo.....	109
pub.client:http.....	110
pub.client.ldap:add.....	119
pub.client.ldap:bind.....	120
pub.client.ldap:cancelNotification.....	121
pub.client.ldap:compare.....	122
pub.client.ldap:delete.....	124
pub.client.ldap:modify.....	125
pub.client.ldap:registerNotification.....	126
pub.client.ldap:rename.....	128
pub.client.ldap:search.....	129
pub.client.oauth:executeRequest.....	131
pub.client.sftp:cd.....	135
pub.client.sftp:chgrp.....	136
pub.client.sftp:chmod.....	136
pub.client.sftp:chown.....	137
pub.client.sftp:get.....	138
pub.client.sftp:login.....	139
pub.client.sftp:logout.....	140
pub.client.sftp:ls.....	140
pub.client.sftp:mkdir.....	141
pub.client.sftp:put.....	141
pub.client.sftp:pwd.....	143
pub.client.sftp:rename.....	143
pub.client.sftp:rm.....	144
pub.client.sftp:rmdir.....	144
pub.client.sftp:symlink.....	145
pub.client:smtp.....	145
pub.client:soapClient.....	149
pub.client:soapHTTP.....	167
pub.client:soapRPC.....	170
Date Folder.....	175
Pattern String Symbols.....	176
Time Zones.....	177
Examples.....	179
Notes on Invalid Dates.....	180
Summary of Elements in this Folder.....	180
pub.date:calculateDateDifference.....	181
pub.date:currentNanoTime.....	182
pub.date:dateBuild.....	183
pub.date:dateTimeBuild.....	184
pub.date:dateTimeFormat.....	185

pub.date:elapsedNanoTime.....	187
pub.date:formatDate.....	187
pub.date:getCurrentDate.....	188
pub.date:getCurrentDateString.....	188
pub.date:getWorkingDays.....	189
Db Folder.....	191
Summary of Elements in this Folder.....	192
pub.db:call.....	194
pub.db:clearTransaction.....	196
pub.db:close.....	197
pub.db:closeAll.....	198
pub.db:commit.....	198
pub.db:connect.....	199
pub.db:delete.....	201
pub.db:execSQL.....	203
pub.db:getProclInfo.....	207
pub.db:getProcs.....	208
pub.db:getTableInfo.....	210
pub.db:getTables.....	211
pub.db:insert.....	213
pub.db:query.....	215
pub.db:rollback.....	216
pub.db:startTransaction.....	217
pub.db:update.....	218
Document Folder.....	221
Summary of Elements in this Folder.....	222
pub.document:bytesToDocument.....	223
pub.document:deleteDocuments.....	224
pub.document:documentListToDocument.....	224
pub.document:documentToBytes.....	226
pub.document:documentToDocumentList.....	228
pub.document:documentToXMLValues.....	230
pub.document:groupDocuments.....	230
pub.document:insertDocument.....	231
pub.document:searchDocuments.....	232
pub.document:sortDocuments.....	233
pub.document:XMLValuesToDocument.....	234
Event Folder.....	237
Summary of Elements in this Folder.....	238
pub.event:addSubscriber.....	241
pub.event:alarm.....	245
pub.event:alarmInfo.....	246
pub.event:audit.....	246

pub.event:auditError.....	248
pub.event:auditErrorInfo.....	249
pub.event:auditInfo.....	250
pub.event:callstackItem.....	251
pub.event:deleteSubscriber.....	252
pub.event.eda:event.....	253
pub.event.eda:eventToDocument.....	255
pub.event.eda:schema_event.....	257
pub.event:error.....	257
pub.event:errorInfo.....	257
pub.event:exception.....	258
pub.event:exceptionInfo.....	260
pub.event:gdEnd.....	261
pub.event:gdEndInfo.....	262
pub.event:gdStart.....	262
pub.event:gdStartInfo.....	263
pub.event:getEventTypes.....	264
pub.event:getSubscribers.....	264
pub.event:jmsReceiveErrorEvent.....	266
pub.event:jmsSendErrorEvent.....	268
pub.event:journal.....	269
pub.event:journalInfo.....	270
pub.event:modifySubscriber.....	272
pub.event.nerv:eventToDocument.....	276
pub.event.nerv:send.....	277
pub.event:portStatus.....	280
pub.event:portStatusInfo.....	281
pub.event:reloadEventManagerSettings.....	281
pub.event:replication.....	282
pub.event:replicationInfo.....	282
pub.event:saveEventManagerSettings.....	283
pub.event:security.....	284
pub.event:securityInfo.....	285
pub.event:sessionEnd.....	287
pub.event:sessionEndInfo.....	287
pub.event:sessionExpire.....	288
pub.event:sessionExpireInfo.....	289
pub.event:sessionStart.....	289
pub.event:sessionStartInfo.....	290
pub.event:stat.....	291
pub.event:statInfo.....	292
pub.event:txEnd.....	294
pub.event:txEndInfo.....	295
pub.event:txStart.....	295
pub.event:txStartInfo.....	296

File Folder	297
File Access Control Configuration for the pub.file Services.....	298
Parameter Settings.....	298
Summary of Elements in this Folder.....	300
pub.file:bytesToFile.....	300
pub.file:checkFileExistence.....	301
pub.file:copyFile.....	302
pub.file:deleteFile.....	303
pub.file:getFile.....	303
pub.file:listFiles.....	305
pub.file:moveFile.....	305
pub.file:readerToFile.....	307
pub.file:streamToFile.....	308
pub.file:stringToFile.....	308
Flat File Folder	311
Summary of Elements in the Flat File Folder.....	312
pub.flatFile:convertToString.....	314
pub.flatFile:convertToValues.....	319
pub.flatFile:FormatService.....	325
pub.flatFile:getSupportedEncodings.....	328
pub.flatFile.generate:createDocumentType.....	329
pub.flatFile.generate:createFFDictionary.....	329
pub.flatFile.generate:deleteFFDictionary.....	330
pub.flatFile.generate:deleteFFDictionaryEntry.....	331
pub.flatFile.generate:deleteFFSchema.....	332
pub.flatFile.generate:FFDictionary.....	332
pub.flatFile.generate:FFSchema.....	335
pub.flatFile.generate:findDependants.....	338
pub.flatFile.generate:findReferences.....	338
pub.flatFile.generate:getFFDictionaryAsXML.....	339
pub.flatFile.generate:getFFDictionaryEntryAsXML.....	340
pub.flatFile.generate:getFFSchemaAsXML.....	340
pub.flatFile.generate:listFFDictionaryEntries.....	341
pub.flatFile.generate:saveXMLAsFFDictionary.....	342
pub.flatFile.generate:saveXMLAsFFSchema.....	343
pub.flatFile.generate:updateFFDictionaryEntryFromXML.....	345
Flow Folder	347
Summary of Elements in this Folder.....	348
pub.flow:clearPipeline.....	350
pub.flow:debugLog.....	350
pub.flow:getLastError.....	351
pub.flow:getRetryCount.....	352
pub.flow:getSession.....	353

pub.flow:getTransportInfo.....	353
pub.flow:invokeService.....	354
pub.flow:restorePipeline.....	355
pub.flow:restorePipelineFromFile.....	356
pub.flow:savePipeline.....	357
pub.flow:savePipelineToFile.....	358
pub.flow:setCustomContextID.....	360
pub.flow:setResponse.....	360
pub.flow:setResponse2.....	362
pub.flow:setResponseCode.....	364
pub.flow:setResponseHeader.....	365
pub.flow:setResponseHeaders.....	368
pub.flow:throwExceptionForRetry.....	368
pub.flow:tracePipeline.....	369
pub.flow:transportInfo.....	370
Hashtable Folder.....	379
Summary of Elements in this Folder.....	380
pub.hashtable:containsKey.....	380
pub.hashtable:createHashtable.....	381
pub.hashtable:get.....	381
pub.hashtable:listKeys.....	381
pub.hashtable:put.....	382
pub.hashtable:remove.....	382
pub.hashtable:size.....	383
IO Folder.....	385
Summary of Elements in this Folder.....	386
pub.io:bytesToStream.....	387
pub.io:close.....	388
pub.io:createByteArray.....	388
pub.io:mark.....	389
pub.io:markSupported.....	390
pub.io:read.....	391
pub.io:readAsString.....	391
pub.io:readerToString.....	392
pub.io:reset.....	393
pub.io:skip.....	393
pub.io:streamToBytes.....	394
pub.io:streamToReader.....	395
pub.io:streamToString.....	395
pub.io:stringToReader.....	396
pub.io:stringToStream.....	396
JDBC Folder.....	397
Summary of Elements in this Folder.....	398

pub.jdbc:getPoolInfo.....	398
JMS Folder.....	401
Summary of Elements in This Folder.....	402
pub.jms:acknowledge.....	403
pub.jms:batchTriggerSpec.....	404
pub.jms:createConsumer.....	404
pub.jms:documentResolverSpec.....	408
pub.jms:JMSMessage.....	409
pub.jms:receive.....	412
pub.jms:reply.....	417
pub.jms:send.....	425
pub.jms:sendAndWait.....	435
pub.jms:sendBatch.....	454
pub.jms:triggerSpec.....	463
pub.jms:waitForReply.....	463
pub.jms.wmjms:receiveStream.....	468
pub.jms.wmjms:sendStream.....	470
JSON Folder.....	473
Data Type Mapping.....	474
Summary of Elements in This Folder.....	475
pub.json:documentToJSONString.....	475
pub.json:jsonStreamToDocument.....	476
pub.json:jsonStringToDocument.....	477
List Folder.....	479
Summary of Elements in this Folder.....	480
pub.list:addItemToVector.....	480
pub.list:appendToDocumentList.....	481
pub.list:appendToStringList.....	482
pub.list:sizeofList.....	483
pub.list:stringListToDocumentList.....	483
pub.list:vectorToArray.....	484
Math Folder.....	487
Summary of Elements in this Folder.....	488
pub.math:absoluteValue.....	490
pub.math:addFloatList.....	490
pub.math:addFloats.....	491
pub.math:addIntList.....	492
pub.math:addInts.....	493
pub.math:addObjects.....	493
pub.math:divideFloats.....	494
pub.math:divideInts.....	495
pub.math:divideObjects.....	496

pub.math:max.....	496
pub.math:min.....	497
pub.math:multiplyFloatList.....	497
pub.math:multiplyFloats.....	498
pub.math:multiplyIntList.....	499
pub.math:multiplyInts.....	500
pub.math:multiplyObjects.....	501
pub.math:randomDouble.....	501
pub.math:roundNumber.....	502
pub.math:subtractFloats.....	502
pub.math:subtractInts.....	504
pub.math:subtractObjects.....	504
pub.math:toNumber.....	505
Metadata Folder.....	507
Summary of Elements in this Folder.....	508
pub.metadata.assets:publishPackages.....	508
MIME Folder.....	511
Summary of Elements in this Folder.....	512
pub.mime:addBodyPart.....	513
pub.mime:addMimeHeader.....	517
pub.mime:createMimeData.....	519
pub.mime:getBodyPartContent.....	522
pub.mime:getBodyPartHeader.....	523
pub.mime:getContentType.....	525
pub.mime:getEnvelopeStream.....	525
pub.mime:getMimeHeader.....	527
pub.mime:getNumParts.....	528
pub.mime:getPrimaryContentType.....	529
pub.mime:getSubContentType.....	530
pub.mime:mergeHeaderAndBody.....	531
OAuth Folder.....	533
Summary of Elements in this Folder.....	534
pub.oauth:authorize.....	534
pub.oauth:getAccessToken.....	536
pub.oauth:refreshAccessToken.....	538
Packages Folder.....	541
Summary of Elements in this Folder.....	542
pub.packages:activatePackage.....	542
pub.packages:backupPackage.....	543
pub.packages:disablePackage.....	544
pub.packages:enablePackage.....	545
pub.packages:installPackage.....	545

pub.packages:recoverPackage.....	547
pub.packages:reloadPackage.....	548
PKI Folder.....	549
Summary of Elements in this Folder.....	550
pub.pki.pkcs7:sign.....	551
pub.pki.pkcs7:verify.....	552
pub.pki.smime.createSignedAndEncryptedData.....	554
pub.pki.smime.createSignedData.....	555
pub.pki.smime.processEncryptedData.....	556
pub.pki.smime.processSignedData.....	557
Publish Folder.....	561
Summary of Elements in this Folder.....	562
pub.publish:deliver.....	563
pub.publish:deliverAndWait.....	564
pub.publish:documentResolverSpec.....	567
pub.publish:envelope.....	569
pub.publish:getRedeliveryCount.....	574
pub.publish:publish.....	575
pub.publish:publishAndWait.....	578
pub.publish:reply.....	582
pub.publish:syncToBroker.....	585
pub.publish:syncToProvider.....	586
pub.publish:waitForReply.....	587
pub.publish.notification:error.....	588
Remote Folder.....	591
Summary of Elements in this Folder.....	592
pub.remote:invoke.....	593
pub.remote.gd:end.....	595
pub.remote.gd:getStatus.....	595
pub.remote.gd:invoke.....	596
pub.remote.gd:restart.....	597
pub.remote.gd:retrieve.....	597
pub.remote.gd:send.....	598
pub.remote.gd:start.....	598
pub.remote.gd:submit.....	599
Replicator Folder.....	601
Summary of Elements in this Folder.....	602
pub.replicator:addReleaseRegistryEntry.....	603
pub.replicator:deleteReleaseRegistryEntry.....	604
pub.replicator:distributeViaFTP.....	605
pub.replicator:distributeViaSvcPull.....	606
pub.replicator:distributeViaSvcPush.....	607

pub.replicator:generateReplicationEvent.....	607
pub.replicator:getLocalReleasedList.....	607
pub.replicator:getRemoteReleasedList.....	608
pub.replicator:notifyPackageRelease.....	609
pub.replicator:packageCreation.....	610
Report Folder.....	613
Summary of Elements in this Folder.....	614
pub.report:runFileTemplate.....	614
pub.report:runFileTemplateOnPipe.....	615
pub.report:runStringTemplate.....	615
pub.report:runStringTemplateOnPipe.....	616
pub.report:runTemplate.....	616
pub.report:runTemplateOnPipe.....	617
Scheduler Folder.....	619
Summary of Elements in this Folder.....	620
pub.scheduler:addComplexTask.....	621
pub.scheduler:addOneTimeTask.....	624
pub.scheduler:addRepeatingTask.....	626
pub.scheduler:cancelTask.....	629
pub.scheduler:getTaskIDs.....	629
pub.scheduler:getTaskInfo.....	630
pub.scheduler:getUserTaskList.....	634
pub.scheduler:migrateTasksToJDBC.....	634
pub.scheduler:resumeTask.....	635
pub.scheduler:suspendTask.....	636
pub.scheduler:updateComplexTask.....	637
pub.scheduler:updateOneTimeTask.....	640
pub.scheduler:updateRepeatingTask.....	642
Schema Folder.....	647
Summary of Elements in this Folder.....	648
pub.schema:createXSD.....	649
pub.schema:validate.....	651
pub.schema:validatePipeline.....	654
pub.schema.w3c.....	655
pub.schema.w3c:datatypes.....	655
pub.schema.w3c:structures.....	655
pub.schema.w3c:xml.....	656
pub.schema.w3c:xsi.....	656
Security Folder.....	657
About the Security Elements.....	658
Summary of Elements in this Folder.....	659
pub.security:clearKeyAndChain.....	661

pub.security:setKeyAndChain.....	662
pub.security:setKeyAndChainFromBytes.....	663
pub.security.enterpriseGateway:alertSpec.....	664
pub.security.keystore:getCertificate.....	666
pub.security.keystore:getKeyAndChain.....	667
pub.security.keystore:getTrustedCertificates.....	667
pub.security.keystore:setKeyAndChain.....	668
pub.security.keystore.pkcs7:sign.....	668
pub.security.outboundPasswords:setPassword.....	669
pub.security.outboundPasswords:getPassword.....	671
pub.security.outboundPasswords:listKeys.....	672
pub.security.outboundPasswords:removePassword.....	672
pub.security.outboundPasswords:updatePassword.....	673
pub.security.pkcs7:sign.....	673
pub.security.pkcs7:verify.....	675
pub.security.util:createMessageDigest.....	676
pub.security.util:getCertificateInfo.....	677
pub.security.util:loadPKCS7CertChain.....	678
pub.security.util:createSecureString.....	678
pub.security.util:convertSecureString.....	679
pub.security.util:destroySecureString.....	680
pub.security.xml:decryptXML.....	680
pub.security.xml:encryptXML.....	682
pub.security.xml:signXML.....	684
pub.security.xml:verifyXML.....	690
SMIME Folder.....	693
Summary of Elements in this Folder.....	694
pub.smime:createCertsOnlyData.....	695
pub.smime:createEncryptedData.....	695
pub.smime:createSignedAndEncryptedData.....	696
pub.smime:createSignedData.....	698
pub.smime:processCertsOnlyData.....	700
pub.smime:processEncryptedData.....	700
pub.smime:processSignedData.....	701
pub.smime.keystore:createSignedAndEncryptedData.....	704
pub.smime.keystore:createSignedData.....	706
pub.smime.keystore:processEncryptedData.....	706
SOAP Folder.....	709
Summary of Elements in this Folder.....	710
pub.soap.handler:addBodyBlock.....	720
pub.soap.handler:addFaultBlock.....	722
pub.soap.handler:addHeaderBlock.....	724
pub.soap.handler:addHeaderElement.....	728
pub.soap.handler:generateDocumentTypesFromWSDL.....	731

pub.soap.handler:getBodyBlock.....	734
pub.soap.handler:getBodyBlockQNames.....	735
pub.soap.handler:getFaultBlock.....	736
pub.soap.handler:getHeaderBlock.....	738
pub.soap.handler:getHeaderBlockQNames.....	740
pub.soap.handler:getHeaderElement.....	741
pub.soap.handler:getInitialSOAPRequest.....	744
pub.soap.handler:getMessageAddressingProperties.....	745
pub.soap.handler:getProperty.....	747
pub.soap.handler:getServicePipeline.....	748
pub.soap.handler:getSOAPMessage.....	750
pub.soap.handler:getWebServiceInvocationProperties.....	751
pub.soap.handler:handlerSpec.....	753
pub.soap.handler:hasFaultMessage.....	754
pub.soap.handler:listConsumer.....	755
pub.soap.handler:listProvider.....	756
pub.soap.handler:registerConsumer.....	757
pub.soap.handler:registerProvider.....	758
pub.soap.handler:registerWmConsumer.....	758
pub.soap.handler:registerWmProvider.....	760
pub.soap.handler:removeBodyBlock.....	761
pub.soap.handler:removeHeaderBlock.....	762
pub.soap.handler:removeHeaderElement.....	764
pub.soap.handler:removeProperty.....	765
pub.soap.handler:setProperty.....	766
pub.soap.handler:setSOAPMessage.....	767
pub.soap.handler:unregisterConsumer.....	767
pub.soap.handler:unregisterProvider.....	768
pub.soap.handler:updateFaultBlock.....	768
pub.soap.processor:list.....	770
pub.soap.processor:processMessage.....	772
pub.soap.processor:processRPCMessage.....	772
pub.soap.processor:registerProcessor.....	773
pub.soap.processor:unregisterProcessor.....	774
pub.soap.schema:encoding.....	775
pub.soap.schema:encoding_1_2.....	775
pub.soap.schema:envelope.....	775
pub.soap.schema:envelope_1_2.....	775
pub.soap.utils:addBodyEntry.....	776
pub.soap.utils:addHeaderEntry.....	777
pub.soap.utils:addTrailer.....	779
pub.soap.utils:callbackServiceSpec.....	780
pub.soap.utils:convertToVersionSpecificSOAPFault.....	781
pub.soap.utils:createSoapData.....	785
pub.soap.utils:createXOPObject.....	786

pub.soap.utils:exitUnableToUnderstand.....	788
pub.soap.utils:getActor.....	788
pub.soap.utils:getBody.....	789
pub.soap.utils:getBodyEntries.....	790
pub.soap.utils:getDocument.....	790
pub.soap.utils:getEncoding.....	791
pub.soap.utils:getHeader.....	792
pub.soap.utils:getHeaderEntries.....	792
pub.soap.utils:getMustUnderstand.....	793
pub.soap.utils:getQName.....	794
pub.soap.utils:getTrailers.....	795
pub.soap.utils:getXOPObjectContent.....	796
pub.soap.utils:QName.....	798
pub.soap.utils:removeBodyEntry.....	798
pub.soap.utils:removeHeaderEntry.....	799
pub.soap.utils:removeTrailer.....	800
pub.soap.utils:requestResponseSpec.....	801
pub.soap.utils:resetWSDEffectivePolicy.....	801
pub.soap.utils:setWSDEffectivePolicy.....	802
pub.soap.utils:soapDataToBytes.....	803
pub.soap.utils:soapDataToString.....	803
pub.soap.utils:soapFault.....	804
pub.soap.utils:streamToSoapData.....	805
pub.soap.utils:stringToSoapData.....	806
pub.soap.utils:validateSoapData.....	807
pub.soap.wsa:action.....	808
pub.soap.wsa:faultTo.....	808
pub.soap.wsa:from.....	810
pub.soap.wsa:messageID.....	811
pub.soap.wsa:problemAction.....	811
pub.soap.wsa:problemHeaderQName.....	812
pub.soap.wsa:problemIRI.....	812
pub.soap.wsa:relatesTo.....	813
pub.soap.wsa:replyTo.....	814
pub.soap.wsa:retryAfter.....	815
pub.soap.wsa:to.....	815
pub.soap.wsa:schema_wsa.....	816
pub.soap.wsa.submission:action.....	816
pub.soap.wsa.submission:faultTo.....	817
pub.soap.wsa.submission:from.....	819
pub.soap.wsa.submission:messageID.....	820
pub.soap.wsa.submission:relatesTo.....	821
pub.soap.wsa.submission:replyTo.....	822
pub.soap.wsa.submission:retryAfter.....	824
pub.soap.wsa.submission:to.....	824

pub.soap.wsa.submission:schema_wsa_submission.....	825
pub.soap.wsrn:closeSequence.....	825
pub.soap.wsrn:createSequence.....	826
pub.soap.wsrn:sendAcknowledgementRequest.....	831
pub.soap.wsrn:terminateSequence.....	832
pub.soap.wsrn:waitUntilSequenceCompleted.....	833
Storage Folder.....	837
About the Storage Elements.....	838
Locking Considerations.....	839
Entry Locking.....	839
Data Store Locking.....	839
Automatic Promotion to Exclusive Lock.....	840
Wait Time and Duration.....	840
Sample Flow Service for Checkpoint Restart.....	841
Summary of Elements in this Folder.....	842
pub.storage:add.....	843
pub.storage:closeStore.....	844
pub.storage:deleteStore.....	844
pub.storage:get.....	845
pub.storage:keys.....	846
pub.storage:listLocks.....	847
pub.storage:lock.....	848
pub.storage:put.....	850
pub.storage:registerStore.....	851
pub.storage:releaseLocks.....	851
pub.storage:remove.....	852
pub.storage:shutdown.....	853
pub.storage:startup.....	853
pub.storage:unlock.....	853
String Folder.....	855
Summary of Elements in this Folder.....	856
pub.string:base64Decode.....	858
pub.string:base64Encode.....	858
pub.string:bytesToString.....	859
pub.string:concat.....	859
pub.string:HTMLDecode.....	860
pub.string:HTMLEncode.....	860
pub.string:indexOf.....	861
pub.string:length.....	862
pub.string:lookupDictionary.....	862
pub.string:lookupTable.....	863
pub.string:makeString.....	864
pub.string:messageFormat.....	864
pub.string:numericFormat.....	865

pub.string:objectToString.....	866
pub.string:padLeft.....	867
pub.string:padRight.....	868
pub.string:replace.....	869
pub.string:stringToBytes.....	869
pub.string:substring.....	870
pub.string:tokenize.....	870
pub.string:toLowerCase.....	871
pub.string:toUpperCase.....	871
pub.string:trim.....	872
pub.string:URLDecode.....	872
pub.string:URLEncode.....	873
Sync Folder.....	875
Summary of Elements in this Folder.....	876
pub.sync:notify.....	876
pub.sync:shutdown.....	877
pub.sync:wait.....	877
Synchronization Folder.....	879
Summary of Elements in this Folder.....	880
pub.synchronization.latch:closeLatch.....	880
pub.synchronization.latch:isLatchClosed.....	881
pub.synchronization.latch:openLatch.....	882
pub.synchronization.xref:createXReference.....	883
pub.synchronization.xref:deleteByObjectId.....	884
pub.synchronization.xref:deleteXReference.....	884
pub.synchronization.xref:getCanonicalKey.....	885
pub.synchronization.xref:getNativeId.....	886
pub.synchronization.xref:insertXReference.....	887
Trigger Folder.....	889
Summary of Elements in this Folder.....	890
pub.trigger:createJMSTrigger.....	891
pub.trigger:createTrigger.....	904
pub.trigger:deleteJMSTrigger.....	917
pub.trigger:deleteTrigger.....	917
pub.trigger:disableJMSTriggers.....	918
pub.trigger:enableJMSTriggers.....	920
pub.trigger:resourceMonitoringSpec.....	921
pub.trigger:resumeProcessing.....	922
pub.trigger:resumeRetrieval.....	925
pub.trigger:suspendJMSTriggers.....	927
pub.trigger:suspendProcessing.....	929
pub.trigger:suspendRetrieval.....	931

TX Folder	935
Summary of Elements in this Folder.....	936
pub.tx:init.....	936
pub.tx:shutdown.....	937
pub.tx:resetOutbound.....	937
UniversalName Folder	939
Summary of Elements in this Folder.....	940
pub.universalName:find.....	940
pub.universalName:findDocumentType.....	941
pub.universalName:list.....	941
pub.universalName:listAll.....	942
Utils Folder	945
Summary of Elements in this Folder.....	946
pub.utils:deepClone.....	946
pub.utils:executeOSCommand.....	947
pub.utils:generateUUID.....	949
pub.utils:getServerProperty.....	949
pub.utils.ws:setCompatibilityModeFalse.....	949
VCS Folder	955
Summary of Elements in this Folder.....	956
pub.vcs.admin:getUsers.....	956
pub.vcs.admin:removeCurrentUser.....	957
pub.vcs.admin:removeMultipleUsers.....	958
pub.vcs.admin:setCurrentUser.....	958
pub.vcs.admin:setMultipleUsers.....	959
XML Folder	961
Summary of Elements in this Folder.....	962
pub.xml:documentToXMLString.....	963
pub.xml:freeXMLNode.....	970
pub.xml:getNextXMLNode.....	971
pub.xml:getXMLNodeIterator.....	972
pub.xml:getXMLNodeType.....	975
pub.xml:loadEnhancedXMLNode.....	975
pub.xml:loadXMLNode.....	985
pub.xml:queryXMLNode.....	993
pub.xml:xmlNodeToDocument.....	996
pub.xml:xmlStringToEnhancedXMLNode.....	1006
pub.xml:xmlStringToXMLNode.....	1010
XSLT Folder	1013
Summary of Elements in this Folder.....	1014
pub.xslt.Transformations:transformSerialXML.....	1014

pub.xslt.Cache:removeAllTemplates.....	1017
pub.xslt.Cache:removeTemplate.....	1018
Index.....	1019

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...
ART Folder	(WmART package) Manage adapter components, including connections, adapter services, listeners, and notifications.
Cache Folder	Perform tasks on caches.
Client Folder	Formulate and submit requests to HTTP, FTP, e-mail, and LDAP servers.
Date Folder	Generate and format date values.
Db Folder	(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.
Document Folder	Perform operations on documents in the pipeline.
Event Folder	Build audit and event handler services.
File Folder	Perform operations on the local file system.

Folder	Contains services you use to...
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.
PKI Folder	(WmPKI package) Create and verify PKCS#7 signatures with PKI profiles and create and process S/MIME messages using PKI profiles.

Folder	Contains services you use to...
<p>Note: The WmPKI package is deprecated as of webMethods Integration Server version 9.7.</p>	
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.

Folder	Contains services you use to...
Trigger Folder	Create and delete triggers and manage document retrieval and document processing for individual 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	Manage user associations for the VCS Integration feature.
XML Folder	Perform operations on XML documents.
XSLT Folder	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...	Built-in services are assigned to this ACL...	
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	
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
Bold	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 (...).

Documentation Installation

You can download the product documentation using the Software AG Installer. The documentation is downloaded to a central directory named `_documentation` in the main installation directory (SoftwareAG by default).

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

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

Element	Package and Description
pub.art.notification:queryPollingNotificationState	WmART. Returns the current state for a polling notification.
pub.art.notification:resumePollingNotification	WmART. Resumes a specified polling notification node.
pub.art.notification:setListenerNotificationNodeListener	WmART. Changes the listener used by a specified listener notification.
pub.art.notification:setPollingNotificationNodeConnection	WmART. Changes the connection node used by a specified polling notification.
pub.art.notification:suspendPollingNotification	WmART. Suspends a specified polling notification.
pub.art.service:listAdapterServices	WmART. Lists adapter services associated with a specified adapter.
pub.art.service:setAdapterServiceNodeConnection	WmART. Changes the connection node used by a specified adapter service.
pub.art.transaction:commitTransaction	WmART. Commits an explicit transaction.
pub.art.transaction:rollbackTransaction	WmART. Rolls back an explicit transaction.
pub.art.transaction:setTransactionTimeout	WmART. Manually sets a transaction timeout interval for implicit and explicit transactions.
pub.art.transaction:startTransaction	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>	String The localized name that the Integration Server Administrator displays.
<i>adapterTypeName</i>	String 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.

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

<i>TotalHits</i>	Integer Number of times this connection node successfully provided connections since the last reset.
<i>TotalMisses</i>	Integer 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>	String The name of the adapter as registered with the WmART package.
------------------------	---

Output Parameters

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

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

- `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

<i>listenerName</i>	String Name of the listener you want to disable. The listener should have a state of <code>enabled</code> or <code>suspended</code> .
<i>forceDisable</i>	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: <ul style="list-style-type: none">■ <code>true</code> to disable the listener.■ <code>false</code> 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

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

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>	String Name of the listener for which you want to change the connection node.
<i>connectionAlias</i>	String 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>	String The name of the listener notification.
<i>packageName</i>	String The name of the package in which the listener notification resides.
<i>notificationEnabled</i>	<p>String The current state of the listener notification. The state will have one of these values:</p> <ul style="list-style-type: none"> ■ <code>no</code> if the listener notification is disabled. ■ <code>yes</code> if the listener notification is enabled.

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

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:

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

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

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>	String 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>	String Name of the listener notification for which you want to change the listener.
<i>listenerNode</i>	String 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>	String Name of the polling notification for which you want to change the connection node.
<i>connectionAlias</i>	String 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>	String 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>	String The name of the adapter service.
<i>packageName</i>	String 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>	String The name of an explicit transaction that you want to commit. The <i>transactionName</i> must have been previously used in a call to pub.art.transaction:startTransaction . This value must be mapped from the most recent pub.art.transaction:startTransaction 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>String 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 pub.art.transaction:startTransaction.</p> <p>This value must be mapped from the most recent pub.art.transaction:startTransaction 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>	String 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>	String 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 Cache Folder

■ About Checkpoint Restart	54
■ Summary of Elements in this Folder	55

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.

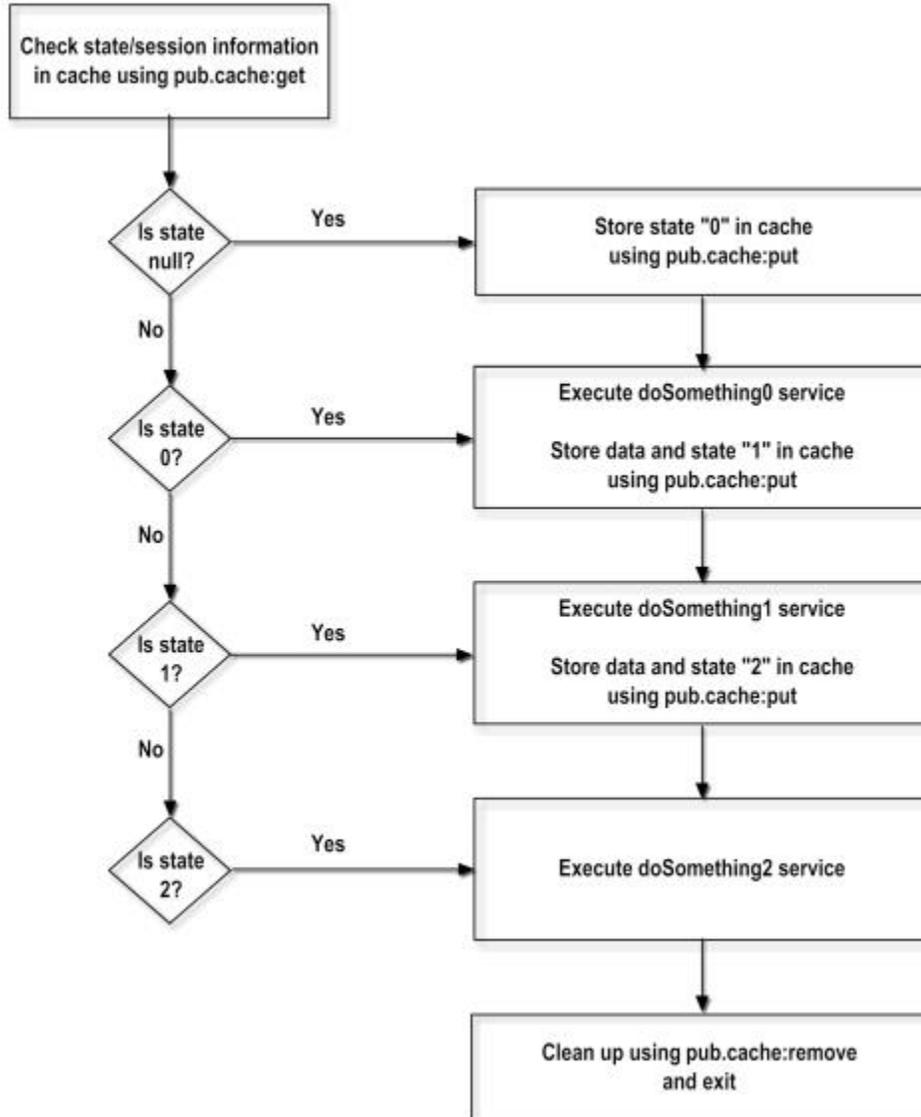
Note: You cannot use an IS document object as a key for storing objects in the cache because IS document objects do not implement the equals() method.

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

Element	Package and Description
pub.cache:get	WmPublic. Retrieves the value of a cached element for the specified key.
pub.cache:getKeys	WmPublic. Retrieves the keys of all elements available in the cache.
pub.cache:put	WmPublic. Populates a cached element with a specified key-value pair.
pub.cache:remove	WmPublic. Removes the cached element associated with the specified key.
pub.cache:search	WmPublic. Searches through an indexed cache and returns the results.
pub.cache.admin:clearAllCaches	WmPublic. Deletes all of the elements from all caches contained in the specified cache manager.
pub.cache.admin:clearCache	WmPublic. Deletes all elements from the specified cache.
pub.cache.admin:disableCache	WmPublic. Disables a cache.
pub.cache.admin:enableCache	WmPublic. Enables a cache.
pub.cache.admin:evictExpiredElements	WmPublic. Deletes all of the expired elements from a cache.
pub.cache.admin:isCacheDisabled	WmPublic. Checks whether the cache is disabled.
pub.cache.atomic:putIfAbsent	WmPublic. Adds an element to the cache if the cache does not contain an element with the specified key.
pub.cache.atomic:remove	WmPublic. Removes the cached element associated with the specified key and value.

Element	Package and Description
pub.cache.atomic:replace	WmPublic. Replaces the cached element value with the supplied value.
pub.cache.atomic:replaceIfKeyExists	WmPublic. Replaces the cached element if an element for the specified key exists in a cache.
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.
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.
pub.cache.bulk:setNodeBulkLoadEnabled	WmPublic. Enables or disables bulk loading mode in the current Integration Server node for the cache.
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.
pub.cache.lock:acquireLock	WmPublic. Acquires a lock on the cached element that contains the specified key.
pub.cache.lock:isLockedByCurrentThread	WmPublic. Checks whether the current thread is holding a lock on the cached element for the specified key.
pub.cache.lock:releaseLock	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>	String Name of the cache manager that manages the cache. This parameter is case sensitive.
<i>cacheName</i>	String Name of the cache that contains the key. This parameter is case sensitive.
<i>key</i>	Object Key of the element for whose existence you are checking.

Output Parameters

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

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

<i>cacheManagerName</i>	String Name of the cache manager that manages the cache. This parameter is case sensitive.
<i>cacheName</i>	String Name of the cache from which to retrieve the element. This parameter is case sensitive.
<i>key</i>	Object Key associated with the cached value.
<i>useLoader</i>	String Optional. Indicates whether to use a cache loader to retrieve the element from the cache. Set to: <ul style="list-style-type: none">■ <code>true</code> to use a cache loader to retrieve the element if it is not already in the cache. <div data-bbox="568 840 1372 924" style="background-color: #f0f0f0; padding: 5px;">Note: If <i>useLoader</i> is <code>true</code> and no loaders are configured for the cache, Integration Server issues a <code>ServiceException</code>.</div> <ul style="list-style-type: none">■ <code>false</code> to retrieve the element from the cache without a cache loader. This is the default setting.

Output Parameters

<i>value</i>	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:getKeys

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

Input Parameters

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

Output Parameters

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

Usage Notes

Caution: 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>	String Name of the cache manager that manages the cache. This parameter is case sensitive.
<i>cacheName</i>	String Name of the cache in which to put the element. This parameter is case sensitive.
<i>key</i>	Object Key of the cached element.
<i>value</i>	Object Value of the cached element.
<i>useWriter</i>	String 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">■ <code>true</code> to use a cache writer to add the <i>value</i> to the cached element <i>and</i> the SOR.■ <code>false</code> to add the <i>value</i> to the cached element without writing to the SOR. This is the default setting.

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:remove

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

Input Parameters

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

Output Parameters

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

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

<i>criteria</i>	Document A document (IData) that specifies the conditions based on which the data in the cache is to be queried.								
	<table> <thead> <tr> <th><u>Value</u></th> <th><u>Description</u></th> </tr> </thead> <tbody> <tr> <td><i>OP1</i></td> <td>String or Document A string operand or a nested document type that can be evaluated by a logical operator.</td> </tr> <tr> <td><i>OPR</i></td> <td>String A logical (and, or, and not) or comparison operator (between, gt (greater than), eq (equal to), and so on).</td> </tr> <tr> <td><i>OP2</i></td> <td>String or Document A string operand or a nested document type that can be evaluated by a logical operator.</td> </tr> </tbody> </table>	<u>Value</u>	<u>Description</u>	<i>OP1</i>	String or Document A string operand or a nested document type that can be evaluated by a logical operator.	<i>OPR</i>	String A logical (and, or, and not) or comparison operator (between, gt (greater than), eq (equal to), and so on).	<i>OP2</i>	String or Document A string operand or a nested document type that can be evaluated by a logical operator.
<u>Value</u>	<u>Description</u>								
<i>OP1</i>	String or Document A string operand or a nested document type that can be evaluated by a logical operator.								
<i>OPR</i>	String A logical (and, or, and not) or comparison operator (between, gt (greater than), eq (equal to), and so on).								
<i>OP2</i>	String or Document A string operand or a nested document type that can be evaluated by a logical operator.								
<i>maxResults</i>	String Optional. The maximum number of results to return. If <i>maxResults</i> is not specified, Integration Server returns all the results.								
<i>groupBy</i>	String 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>	String Optional. The order in which the service results are to be returned. Values for this parameter should be specified as a two-dimensional string array (for example, "name", "ascending" or "age", "descending").								
<i>aggregator</i>	String 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 (for example, "age", "max" or "income", "average").								

Output Parameters

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

<u>Value</u>	<u>Description</u>
--------------	--------------------

<i>Key</i>	Object Conditional. The element key. Returned only if <i>includeKey</i> is set to <code>true</code> .
<i>Value</i>	Object Conditional. The element value. Returned only if <i>includeValue</i> is set to <code>true</code> .
<i>Result</i>	Object The search results. Each element in the cache that matches the specified criteria will be returned as a result object. The <i>Result</i> object contains aggregator results and attribute values. The aggregator results contain the values of the aggregator functions and attribute values contain the values of attributes included in the <i>includeAttribute</i> parameter.

Usage Notes

- 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.
- 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.
- 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);

```

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

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

Output Parameters

<i>status</i>	String The status indicating either <code>Error</code> or <code>Success</code> .
<i>message</i>	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>	String Name of the cache manager that manages the cache. This parameter is case sensitive.
<i>cacheName</i>	String Name of the cache to enable. This parameter is case sensitive.

Output Parameters

<i>status</i>	String The status indicating either <code>Error</code> or <code>Success</code> .
<i>message</i>	String 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>	String Name of the cache manager that manages the cache. This parameter is case sensitive.
<i>cacheName</i>	String Name of the cache from which you want to delete the expired elements. This parameter is case sensitive.

Output Parameters

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

<i>cacheManagerName</i>	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.

<i>key</i>	Object Key of the element.
<i>value</i>	Object Value to add to the element.

Output Parameters

<i>keyExists</i>	<p>String Indicates whether an element with the specified key exists in the cache.</p> <ul style="list-style-type: none"> ■ <code>true</code> indicates that an element with the key already exists in the cache. ■ <code>false</code> indicates that an element with the key does not exist in the cache.
<i>oldValue</i>	<p>Object 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>	String Name of the cache manager that manages the cache. This parameter is case sensitive.
<i>cacheName</i>	String Name of the cache from which to remove the element. This parameter is case sensitive.
<i>key</i>	Object Key of the element to remove.

value **Object** Value of the element to remove.

Output Parameters

removed **String** Indicates whether the service removed the element.

- `true` indicates that the element was removed from the cache.
- `false` indicates that the element was not removed from the cache. The service returns `false` if the specified *key* and *value* do not exist in the cache.

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

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 value you want to replace. This parameter is case sensitive.

key **Object** Key of the cache element whose value you want to replace.

oldValue **Object** Value of the cached element.

newValue **Object** Value with which to replace the value of the cached element.

Output Parameters

<i>replaced</i>	String Indicates whether the service replaced <i>oldValue</i> with <i>newValue</i> in the cache. <ul style="list-style-type: none">■ <code>true</code> indicates that the value was replaced in the cache.■ <code>false</code> 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>	String Name of the cache manager that manages the cache. This parameter is case sensitive.
<i>cacheName</i>	String Name of the cache that contains the key. This parameter is case sensitive.
<i>key</i>	Object Key of the element whose value you want to replace.
<i>value</i>	Object Value with which to replace the value of the cached element.

Output Parameters

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

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

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

- cacheManagerName* **String** Name of the cache manager that manages the cache. This parameter is case sensitive.
- cacheName* **String** Name of the distributed cache to check for bulk loading. This parameter is case sensitive.

Output Parameters

- enabled* **String** Indicates whether the cache is enabled for bulk loading.
- `true` indicates that the Integration Server node calling the service is the same one that enabled bulk-loading.
 - `false` indicates one of the following:

- The Integration Server node calling the service was not the same node that enabled bulk loading.
- The specified cache is a local cache.

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

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 enable or disable bulk loading. This parameter is case sensitive.

bulkMode **String** Indicates whether to enable the bulk-load mode for the cache. Set to:

- `true` to enable the bulk-load mode. This is the default.
- `false` to disable the bulk-load mode.

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.

<u>Key</u>	<u>Description</u>
<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.

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

<i>cacheManagerName</i>	String Name of the cache manager that manages the cache. This parameter is case sensitive.
<i>cacheName</i>	String Name of the cache that contains the key. This parameter is case sensitive.
<i>key</i>	Object Key for which you want to check locks.
<i>lockType</i>	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

<i>cacheManagerName</i>	String Name of the cache manager that manages the cache. This parameter is case sensitive.
<i>cacheName</i>	String Name of the cache that contains the key. This parameter is case sensitive.
<i>key</i>	Object Key of the cached element to unlock.
<i>lockType</i>	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.

3 Client Folder

- Summary of Elements in this Folder 84

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

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

Element	Package and Description
pub.client.ldap:cancelNotification	WmPublic. Cancels a previously created notification request.
pub.client.ldap:compare	WmPublic. Compares the value of an attribute in the LDAP directory with a value specified by the service.
pub.client.ldap:delete	WmPublic. Removes an entry from the directory.
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.
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.
pub.client.ldap:rename	WmPublic. Performs an LDAP rename (move) operation allowing you to rename an entry.
pub.client.ldap:search	WmPublic. Performs an LDAP search operation with the specified parameters and returns the results of the search.
pub.client.oauth:executeRequest	WmPublic. Allow clients to access protected resources using an OAuth token.
pub.client.sftp:cd	WmPublic. Changes the working directory on the remote SFTP server.
pub.client.sftp:chgrp	WmPublic. Changes the group ownership of one or more remote files.
pub.client.sftp:chmod	WmPublic. Changes permissions of one or more remote files.
pub.client.sftp:chown	WmPublic. Changes the user of one or more remote files.
pub.client.sftp:get	WmPublic. Retrieves a file from a remote SFTP server and saves it on the local machine.

Element	Package and Description
pub.client.sftp:login	WmPublic. Connects to a remote SFTP server and logs in with the specified SFTP user alias.
pub.client.sftp:logout	WmPublic. Logs off the user from the SFTP server and ends the current SFTP session.
pub.client.sftp:ls	WmPublic. Retrieves the remote directory listing of the specified path or current remote directory if path is not specified.
pub.client.sftp:mkdir	WmPublic. Creates a new remote directory.
pub.client.sftp:put	WmPublic. Transfers a file to a remote SFTP server.
pub.client.sftp:pwd	WmPublic. Displays the remote working directory on the SFTP server.
pub.client.sftp:rename	WmPublic. Renames a file or directory on a remote SFTP server.
pub.client.sftp:rm	WmPublic. Deletes one or more remote files on the SFTP server.
pub.client.sftp:rmdir	WmPublic. Deletes one or more remote directories on the SFTP server.
pub.client.sftp:symlink	WmPublic. Creates a symbolic link between the old path and the new path of a file.
pub.client:smtp	WmPublic. Sends a MIME-type e-mail message.
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.
pub.client:soapHTTP	WmPublic. <i>Deprecated</i> - There is no replacement service. Submits a SOAP message to a server via HTTP or HTTPS.

Element	Package and Description
pub.client:soapRPC	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>	String Name or IP address of the FTP server (for example, <code>ftp.netscape.com</code>).
<i>serverport</i>	String Port number of the FTP server (for example, <code>4566</code>).
<i>username</i>	String Valid FTP user of the remote FTP server (for example, <code>anonymous</code>).
<i>password</i>	String Optional. Valid password of the FTP user.
<i>command</i>	String One of the following FTP commands: <code>ls</code> , <code>put</code> , or <code>get</code> .
<i>dirpath</i>	String 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>	String One of two FTP file transfer modes: <code>ascii</code> or <code>binary</code> . The default is <code>ascii</code> .
<i>transfertype</i>	String One of two FTP data transfer types: <code>passive</code> or <code>active</code> . The default is <code>active</code> .

<i>localfile</i>	<p>String 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.)</p> <p>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.</p>
<i>remotefile</i>	<p>String 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.</p> <p>When <i>command</i> is set to <code>get</code>, this parameter specifies the name of the remote file that you want to retrieve.</p>
<i>content</i>	<p>java.io.InputStream, byte[], or String Data to be transferred when <i>command</i> is set to <code>put</code>.</p>
<i>encoding</i>	<p>String Optional. Character set in which the document is encoded. Specify an IANA-registered character set (for example, ISO-8859-1).</p> <p>This information is required to correctly convert the String object to bytes when performing a <code>get</code>. If parameter is null, the default JVM encoding is used.</p>
<i>serverencoding</i>	<p>String 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.</p>
<i>timeout</i>	<p>String 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.</p>
<i>putunique</i>	<p>String Optional. Indicates whether to send a STOR or a STOU (Store as Unique File) command to the remote FTP server. Set to:</p> <ul style="list-style-type: none"> ■ <code>true</code> to send a STOU (Store as Unique File) command. ■ <code>false</code> to send a STOR command. This is the default.
<i>secure</i>	<p>Document Indicates whether the FTP session is with a secure FTP server.</p>

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

<i>auth</i>	<p>String 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>String 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> <div style="background-color: #f0f0f0; padding: 5px; margin-top: 10px;"> <p>Note: If you do not set a value, the default is <code>false</code>.</p> </div>
<i>cleanlinefeeds</i>	<p>String 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"> ■ <code>true</code> to remove carriage returns. This is the default. ■ <code>false</code> to retain carriage returns.
<i>newSession</i>	<p>String Optional. Flag indicating whether a new FTP session will be created for this FTP operation. Set to:</p> <ul style="list-style-type: none"> ■ <code>yes</code> to create a new session for this FTP operation. ■ <code>no</code> to use the current session, if one is available, for this FTP operation. This is the default.
<i>clientTimeout</i>	<p>String 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>String 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</p>

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 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 `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*.

Output Parameters

<i>command</i>	String FTP command that was executed (<code>ls</code> , <code>get</code> , or <code>put</code>).
<i>dirlist</i>	String List File names returned by the <code>ls</code> command.
<i>localfile</i>	String Name of the local file used for a <code>get</code> or <code>put</code> operation.
<i>remotefile</i>	String Name of the remote file used for a <code>get</code> or <code>put</code> operation.
<i>content</i>	byte[] If <i>localfile</i> was not specified, this parameter contains the Content object sent to the remote server (if a <code>put</code> command was executed) or received from the remote server (if a <code>get</code> command was executed).
<i>returncode</i>	String Standard FTP protocol return code.
<i>returnmsg</i>	String Standard FTP protocol return message.
<i>logmsg</i>	String 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>	String Unique key for the current FTP session. The <i>sessionkey</i> is returned by the pub.client.ftp:login service.
<i>transfermode</i>	String FTP file transfer mode (<code>ascii</code> or <code>binary</code>). The default is <code>ascii</code> .
<i>content</i>	java.io.InputStream, byte[], or String Data to be transferred to the remote file.
<i>localfile</i>	String Optional. Name of the local file to append to the remote file. Used only when <i>content</i> is not specified.
<i>remotefile</i>	String 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>	String Standard FTP protocol return code.
<i>returnmsg</i>	String Standard FTP protocol return message.
<i>logmsg</i>	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

<i>sessionkey</i>	String Unique key for the current FTP session. The <i>sessionkey</i> is returned by the pub.client.ftp:login service.
<i>dirpath</i>	String Directory to which you want to switch on the FTP server. For example: <code>pub</code>

Output Parameters

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

<i>sessionkey</i>	String Unique key for the current FTP session. The <i>sessionkey</i> is returned by the pub.client.ftp:login service.
<i>dirpath</i>	String Directory to which you want to switch on the FTP server (for example, <code>pub</code>).
<i>filenamepattern</i>	String Optional. Pattern that specifies the file names to list (for example, <code>*.txt</code>).
<i>orderby</i>	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

<i>dirlist</i>	String List List of file names matching <i>filenamepattern</i> .
<i>returncode</i>	String Standard FTP protocol return code.
<i>returnmsg</i>	String Standard FTP protocol return message.
<i>logmsg</i>	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

<i>sessionkey</i>	String Unique key for the current FTP session. The <i>sessionkey</i> is returned by the pub.client.ftp:login service.
<i>remotefile</i>	String Name of the file to be deleted from the current working directory. For example: <code>text.txt</code> If you specify pattern-matching characters in <i>remotefile</i> , all files matching the pattern will be deleted.

Output Parameters

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

transfermode **String** FTP file transfer mode (`ascii` or `binary`). The default is `ascii`.

<i>localfile</i>	String Optional. Name of a local file where the retrieved file is to be saved.
<i>remotefile</i>	String Name of the remote file.
<i>encoding</i>	<p>String 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: <code>ISO-8859-1</code>).</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>	String 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"> ■ The output parameter <i>islargefile</i> will always be <code>true</code>. ■ The file content will be returned in the output parameter <i>contentstream</i> (as a <code>java.io.InputStream</code> object). ■ The output parameter <i>content</i> will be null.
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"> ■ The output parameter <i>islargefile</i> will be <code>true</code>. ■ The file content will be returned in the output parameter <i>contentstream</i> (as a <code>java.io.InputStream</code> object). ■ The output parameter <i>content</i> will be null.
Leave blank	<p>No file is considered large. This means:</p> <ul style="list-style-type: none"> ■ The output parameter <i>islargefile</i> will always be <code>false</code>. ■ The file content will be returned in the output parameter <i>content</i> ().

- 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 retrieve 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. 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 synchronizing the publishable document type with the messaging provider.

pub.client.ftp:login

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

You must use this service to initiate an FTP session before using most other services in `pub.client.ftp`.

Input Parameters

<i>serverhost</i>	String Name or IP address of the FTP server (for example, <code>ftp.netscape.com</code>).
<i>serverport</i>	String Port number on which the FTP server listens for requests (for example, <code>4566</code>). The default is <code>21</code> .
<i>dataport</i>	String 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>	String Valid FTP user on the remote FTP server (for example, <code>anonymous</code>).
<i>password</i>	String Optional. Valid password for the FTP user specified in <i>username</i> (for example, <code>someone@somewhere</code>).
<i>account</i>	String 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>	String Type of the FTP data transfer mode (<code>passive</code> or <code>active</code>). The default is <code>active</code> .
<i>encoding</i>	String 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>	String 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.
<i>timeout</i>	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.

<i>secure</i>	Document Indicates whether the FTP session is with a secure FTP server.						
	<table border="0"> <thead> <tr> <th style="border-bottom: 1px solid black; text-align: left;">Key</th> <th style="border-bottom: 1px solid black; text-align: left;">Description</th> </tr> </thead> <tbody> <tr> <td style="vertical-align: top;"><i>auth</i></td> <td> <p>String 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> </td> </tr> <tr> <td style="vertical-align: top;"><i>securedata</i></td> <td> <p>String 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> <div style="background-color: #f0f0f0; padding: 5px; margin-top: 10px;"> <p>Note: If you do not set a value, the default is <code>false</code>.</p> </div> </td> </tr> </tbody> </table>	Key	Description	<i>auth</i>	<p>String 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>String 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> <div style="background-color: #f0f0f0; padding: 5px; margin-top: 10px;"> <p>Note: If you do not set a value, the default is <code>false</code>.</p> </div>
Key	Description						
<i>auth</i>	<p>String 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>String 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> <div style="background-color: #f0f0f0; padding: 5px; margin-top: 10px;"> <p>Note: If you do not set a value, the default is <code>false</code>.</p> </div>						
<i>newSession</i>	<p>String Optional. Flag indicating whether a new FTP session will be created for this FTP operation. Set to:</p> <ul style="list-style-type: none"> ■ <code>yes</code> to create a new session for this FTP operation. ■ <code>no</code> to use the current session, if one is available, for this FTP operation. This is the default. 						
<i>clientTimeout</i>	String 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>String 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</p>						

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 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 `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*.

Output Parameters

<code>sessionkey</code>	String Unique key for the current FTP session. This session key must be provided to execute most other services in <code>pub.client.ftp</code> .
<code>returncode</code>	String Standard FTP protocol return code.
<code>returnmsg</code>	String Standard FTP protocol return message.
<code>logmsg</code>	String 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 <code>-t</code> 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>	String List List of file names matching <i>filenamepattern</i> .
<i>returncode</i>	String Standard FTP protocol return code.
<i>returnmsg</i>	String Standard FTP protocol return message.
<i>logmsg</i>	String 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>	String Unique key for the current FTP session. The <i>sessionkey</i> is returned by the pub.client.ftp:login service.
<i>filenamepattern</i>	String Pattern that specifies the names of the files to be deleted from the current working directory (for example, <code>*.txt</code>).

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

Output Parameters

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

logmsg **String** 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>	String Unique key for the current FTP session. The <i>sessionkey</i> is returned by the pub.client.ftp:login service.
<i>transfermode</i>	String FTP file transfer mode (<code>ascii</code> or <code>binary</code>). The default is <code>ascii</code> .
<i>localdir</i>	String Directory in the local file system where the retrieved files are to be saved (for example, <code>c:\temp\ftpfiles</code>).
<i>filenamepattern</i>	String Pattern that specifies the names of the files to be retrieved (for example, <code>*.txt</code>).
<i>encoding</i>	String 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 <i>encoding</i> , 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>	String List List of files retrieved from the remote FTP server.
<i>returncode</i>	String Standard FTP protocol return code.
<i>returnmsg</i>	String Standard FTP protocol return message.
<i>logmsg</i>	String 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 `mput`.)

Input Parameters

<i>sessionkey</i>	String Unique key for the current FTP session. The <i>sessionkey</i> is returned by the pub.client.ftp:login service.
<i>transfermode</i>	String FTP file transfer mode (<code>ascii</code> or <code>binary</code>). The default is <code>ascii</code> .
<i>localdir</i>	String Local directory containing the files you want to transfer to the remote FTP server (for example, <code>c:\temp\ftpfiles</code>).
<i>filenamepattern</i>	String Pattern that specifies the names of the files to be transferred (for example, <code>*.txt</code>).
<i>putunique</i>	String 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">■ <code>true</code> to send a <code>STOU</code> (Store as Unique File) command.■ <code>false</code> to send a <code>STOR</code> command. This is the default.

Output Parameters

<i>filenames</i>	String List List of files transferred to the remote FTP server.
<i>returncode</i>	String Standard FTP protocol return code.
<i>returnmsg</i>	String Standard FTP protocol return message.
<i>logmsg</i>	String 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>	String Unique key for the current FTP session. The <i>sessionkey</i> is returned by the pub.client.ftp:login service.
<i>transfermode</i>	String FTP file transfer mode (<code>ascii</code> or <code>binary</code>). The default is <code>ascii</code> .
<i>content</i>	java.io.InputStream, byte[], or String Data to be transferred to the remote file.
<i>localfile</i>	String 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>	String The name of the remote file.
<i>putunique</i>	String Optional. Indicates whether to send a STOR or a STOU (Store as Unique File) command to the remote FTP server. Set to: <ul style="list-style-type: none"> ■ <code>true</code> to send a STOU (Store as Unique File) command. ■ <code>false</code> to send a STOR command. This is the default.

Output Parameters

<i>returncode</i>	String Standard FTP protocol return code.
<i>returnmsg</i>	String Standard FTP protocol return message.
<i>logmsg</i>	String 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>	String The login user name through the FTP Listener.
<i>filename</i>	String The absolute path name of the file.
<i>_env</i>	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: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. 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 synchronizing the publishable document type with a messaging provider.

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>	String Unique key for the current FTP session. The <i>sessionkey</i> is returned by the pub.client.ftp:login service.
<i>string</i>	String 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>	String Standard FTP protocol return code.
<i>returnmsg</i>	String Standard FTP protocol return message.
<i>logmsg</i>	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

<i>sessionkey</i>	String Unique key for the current FTP session. The <i>sessionkey</i> is returned by the pub.client.ftp:login service.
<i>oldname</i>	String Fully qualified name of the file you want to rename (for example, <code>temp/oldname.txt</code>).
<i>newname</i>	String New fully qualified name for the file (for example, <code>temp/newname.txt</code>).

Output Parameters

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

<i>name</i>	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:

<u>Key</u>	<u>Description</u>
<i>serverhost</i>	String Name or IP address of the FTP server.
<i>serverport</i>	String Port number on which the FTP server listens for requests.
<i>dataport</i>	String Listener port of the data transfer channel used by this session.
<i>username</i>	String User logged on to FTP server.
<i>password</i>	String Password for the FTP user specified in <i>username</i> .
<i>account</i>	String 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 *username* and *password* input variables.

<i>transfertype</i>	String Data transfer mode (<i>passive</i> or <i>active</i>) used by this session.
<i>encoding</i>	String 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>String URL of the resource that you want to access. For example:</p> <pre>http://www.rubicon.com/orders/orders.html</pre> <p>Important: This string <i>must</i> begin with <code>http:</code> or <code>https:</code>.</p>
<i>method</i>	<p>String Specifies the HTTP method you want to use. Valid values are:</p> <ul style="list-style-type: none"> delete get head options post put trace

loadAs

String Optional. Form in which you want the http service to store the returned document. Set to:

- `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`). This is the default.
- `stream` to return the body of the response as a `java.io.InputStream`. Use this option if the document will be used as input to a service that can process documents incrementally (for example, `pub.xml:getXMLNodeIterator`).

data

Document Data that you want the http service to submit with the HTTP request. Specify data using one or more of the following keys.

Important: When you use more than one key, *args* is appended first, *table* is appended second, and *string* is appended last.

Key	Description
<i>args</i>	<p>Document Optional. Name/value pairs that you want this service to submit to the resource in <i>url</i>. You can use <i>args</i> to submit data via the POST, PUT, GET, or HEAD method.</p> <p>To specify data using <i>args</i>, 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.</p> <p>When you use <i>args</i>, the http service will automatically:</p> <ul style="list-style-type: none"> ■ URL-encode name/value pair, so you do not need to URL-encode the values you specify in <i>args</i>. ■ Insert the "&" character between pairs, so you do not need to include it in <i>args</i>. ■ Prefix the entire query string with the "?" character if it submits the data in <i>args</i> via a GET or HEAD. You do not need to include this character in <i>args</i>. <p>When you submit data using <i>args</i>, Integration Server automatically sets the value of the</p>

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.

Key	Description
<i>type</i>	<p>String 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"> ■ <code>Basic</code> to submit a user name and password. This is the default. ■ <code>Bearer</code> to submit authorization information for an OAuth resource server. ■ <code>Digest</code> to submit a password digest for authentication. ■ <code>NTLM</code> to use NTLM authentication.
<i>user</i>	<p>String User name that this service will submit when requesting a protected resource.</p> <p>Note: If you have specified <code>NTLM</code> as <i>type</i>, you must specify <i>user</i> in the following format:</p>

	<code>domain_name \user_name</code>
<i>pass</i>	String Password associated with <i>user</i> .
<i>token</i>	String The access token to submit to the OAuth resource server. Required only when <i>type</i> is set to <code>Bearer</code> .
<i>headers</i>	<p>Document 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>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 defined by the <code>watt.net.timeout</code> server configuration parameter. The default value for the <code>watt.net.timeout</code> server configuration parameter is 300 seconds (30000 milliseconds).</p> <p>For information about the <code>watt.net.timeout</code> server configuration parameter, see <i>webMethods Integration Server Administrator's Guide</i>.</p>
<i>maxKeepAliveConnections</i>	<p>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.</p> <p>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.</p> <ul style="list-style-type: none"> ■ If a suitable pool already exists, Integration Server uses a connection from the pool for the HTTP request. If the specified <i>maxKeepAliveConnections</i> 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 <i>not</i> 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*.

connectTimeout

String Optional. Time (measured in milliseconds) the server will wait to connect to the remote server before timing out and terminating the request.

If a value for *connectTimeout* is not specified, Integration Server uses the value specified for the `watt.net.timeout` server configuration parameter. If no value is specified for the `watt.net.timeout` 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 `watt.net.timeout` server configuration parameter, see *webMethods Integration Server Administrator's Guide*.

useJSSE

String Optional. Whether to enable the use of the Java Secure Socket Extension (JSSE) socket factory for creating outbound HTTPS connections. Set to:

- `yes` if the connection requires the use of TLS 1.1 or TLS 1.2. When set to `yes`, Integration Server creates the connection using the Java Secure Socket Extension (JSSE) socket factory.

Note: When *useJSSE* is set to `yes`, Integration Server ignores the *connectTimeout* parameter.

- `no` if the connection does not require support for TLS 1.1 or TLS 1.2. When set to `no`, the connection supports only SSL 3.0 and TLS 1.0. This is the default.

Output Parameters

<i>encodedURL</i>	String 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> .								
<i>header</i>	Document Conditional. HTTP response headers.								
	<table> <thead> <tr> <th><u>Key</u></th> <th><u>Description</u></th> </tr> </thead> <tbody> <tr> <td><i>lines</i></td> <td>Document Fields in the response header, where key names represent field names and values represent field values.</td> </tr> <tr> <td><i>status</i></td> <td>String HTTP status code of the response.</td> </tr> <tr> <td><i>statusMessage</i></td> <td>String HTTP status message of the response.</td> </tr> </tbody> </table>	<u>Key</u>	<u>Description</u>	<i>lines</i>	Document Fields in the response header, where key names represent field names and values represent field values.	<i>status</i>	String HTTP status code of the response.	<i>statusMessage</i>	String HTTP status message of the response.
<u>Key</u>	<u>Description</u>								
<i>lines</i>	Document Fields in the response header, where key names represent field names and values represent field values.								
<i>status</i>	String HTTP status code of the response.								
<i>statusMessage</i>	String HTTP status message of the response.								
<i>body</i>	Document Body of the HTTP response.								
	<table> <thead> <tr> <th><u>Key</u></th> <th><u>Description</u></th> </tr> </thead> <tbody> <tr> <td><i>bytes</i></td> <td>byte[] 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>.</td> </tr> <tr> <td><i>stream</i></td> <td>java.io.InputStream 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>.</td> </tr> </tbody> </table>	<u>Key</u>	<u>Description</u>	<i>bytes</i>	byte[] 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>	java.io.InputStream 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> .		
<u>Key</u>	<u>Description</u>								
<i>bytes</i>	byte[] 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>	java.io.InputStream 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 server.

pub.client.ldap:add

WmPublic. Inserts a new entry into the directory.

Input Parameters

<i>url</i>	String Optional. URL of the directory server to connect to. For example <code>ldap://servername:389</code> .
<i>principal</i>	String Optional. The principal for the directory server.
<i>credentials</i>	String Optional. Credentials for the directory server.
<i>timeout</i>	String Optional. Connection timeout in milliseconds.
<i>ldapEnv</i>	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.
<i>close</i>	<p>String Flag that specifies whether to close the connection after the service finishes. Set to:</p> <ul style="list-style-type: none"> ■ <code>yes</code> to close the connection. This is the default. ■ <code>no</code> to leave the connection open and available.
<i>dn</i>	String The distinguished name of the new entry to add to the directory.
<i>attrs</i>	Document List 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.



attrsData

Document 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.



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.

<i>principal</i>	String Optional. The principal for the LDAP server.
<i>credentials</i>	String Optional. Credentials for the LDAP server.
<i>timeout</i>	String Optional. Connection timeout in milliseconds.
<i>ldapEnv</i>	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.
<i>close</i>	String Flag that specifies whether to close the connection after the service finishes. Set to: <ul style="list-style-type: none"> ■ <code>yes</code> to close the connection. This is the default. ■ <code>no</code> to leave the connection open and available.

Output Parameters

<i>connectionHandle</i>	Object 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:cancelNotification

WmPublic. Cancels a previously created notification request.

Input Parameters

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

<i>timeout</i>	String Optional. Connection timeout in milliseconds.
<i>ldapEnv</i>	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.
<i>close</i>	<p>String Flag that specifies whether to close the connection after the service finishes. Set to:</p> <ul style="list-style-type: none"> ■ <code>yes</code> to close the connection. This is the default. ■ <code>no</code> to leave the connection open and available.
<i>dn</i>	String The distinguished name of the entry.
<i>connectionHandle</i>	Object Optional. Connection object returned by a previously invoked LDAP service.
<i>scope</i>	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

<i>connectionHandle</i>	Object 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:compare

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

Input Parameters

<i>url</i>	String Optional. URL of the LDAP server to connect to.
<i>principal</i>	String Optional. The principal for the LDAP server.
<i>credentials</i>	String Optional. Credentials for the LDAP server.
<i>timeout</i>	String Optional. Connection timeout in milliseconds.
<i>ldapEnv</i>	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.
<i>close</i>	String Flag that specifies whether to close the connection after the service finishes. Set to: <ul style="list-style-type: none">■ <code>yes</code> to close the connection. This is the default.■ <code>no</code> to leave the connection open and available.
<i>dn</i>	String The distinguished name of the entry whose attribute value you want to compare to <i>attrValue</i> .
<i>connectionHandle</i>	Object Optional. Connection object returned by a previously invoked LDAP service.
<i>attrName</i>	String Name of the attribute whose value you want to compare to <i>attrValue</i> .
<i>attrValue</i>	String The string to compare against the value of the attribute identified by <i>attrName</i> .

Output Parameters

<i>connectionHandle</i>	Object Optional. The returned connection object. Returned only if the <i>close</i> parameter is set to "no".
<i>result</i>	String 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>	String Optional. URL of the LDAP server to connect to.
<i>principal</i>	String Optional. The principal for the LDAP server.
<i>credentials</i>	String Optional. Credentials for the LDAP server.
<i>timeout</i>	String Optional. Connection timeout in milliseconds.
<i>ldapEnv</i>	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.
<i>close</i>	<p>String Flag that specifies whether to close the connection after the service finishes. Set to:</p> <ul style="list-style-type: none"> ■ <code>yes</code> to close the connection. This is the default. ■ <code>no</code> to leave the connection open and available.
<i>dn</i>	String The distinguished name of the entry to delete.
<i>connectionHandle</i>	Object Optional. Connection object returned by a previously invoked LDAP service.

Output Parameters

<i>connectionHandle</i>	Object 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*.

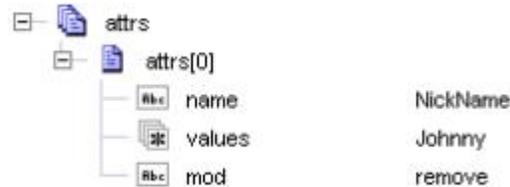
pub.client.ldap:modify

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>	String Optional. URL of the LDAP server to connect to.
<i>principal</i>	String Optional. The principal for the LDAP server.
<i>credentials</i>	String Optional. Credentials for the LDAP server.
<i>timeout</i>	String Optional. Connection timeout in milliseconds.
<i>ldapEnv</i>	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.
<i>close</i>	<p>String Flag that specifies whether to close the connection after the service finishes. Set to:</p> <ul style="list-style-type: none"> ■ <code>yes</code> to close the connection. This is the default. ■ <code>no</code> to leave the connection open and available.
<i>dn</i>	String The distinguished name of the entry to modify.
<i>connectionHandle</i>	Object Optional. Connection object returned by a previously invoked LDAP service.

attrs **Document List** 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.



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: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

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.

ldapEnv **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.

<i>close</i>	String Flag that specifies whether to close the connection after the service finishes. Set to: <ul style="list-style-type: none">■ <code>yes</code> to close the connection. This is the default.■ <code>no</code> to leave the connection open and available.
<i>dn</i>	String The distinguished name of the entry to be monitored.
<i>connectionHandle</i>	Object Optional. Connection object returned by a previously invoked LDAP service.
<i>scope</i>	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 entry, its children, and all of their children).
<i>service</i>	String The target service to be invoked when the LDAP event is retrieved.
<i>user</i>	String 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>	Object 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:

Pipeline Input	Description
<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 objectRenamed 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>	String Optional. URL of the LDAP server to connect to.
<i>principal</i>	String Optional. The principal for the LDAP server.
<i>credentials</i>	String Optional. Credentials for the LDAP server.
<i>timeout</i>	String Optional. Connection timeout in milliseconds.
<i>ldapErr</i>	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.

<i>close</i>	String Flag that specifies whether to close the connection after the service finishes. Set to: <ul style="list-style-type: none"> ■ <code>yes</code> to close the connection. This is the default. ■ <code>no</code> to leave the connection open and available.
<i>connectionHandle</i>	Object Optional. Connection object returned by a previously invoked LDAP service.
<i>newDn</i>	String The new name for the entry.

Output Parameters

<i>connectionHandle</i>	Object 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>	String Optional. URL of the LDAP server to connect to.
<i>principal</i>	String Optional. The principal for the LDAP server.
<i>credentials</i>	String Optional. Credentials for the LDAP server.
<i>timeout</i>	String Optional. Connection timeout in milliseconds.
<i>ldapErrv</i>	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.

<i>close</i>	String Flag that specifies whether to close the connection after the service finishes. Set to: <ul style="list-style-type: none">■ <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.■ <code>no</code> to leave the connection open and available.
<i>dn</i>	String The distinguished name indicating the root from to begin the search.
<i>connectionHandle</i>	Object 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>	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 entry, its children, and all their children).
<i>filter</i>	String The filter string that works with RFC 2254.
<i>countLimit</i>	String Optional. The maximum number of results to return (0, the default, indicates no limit).
<i>timeLimit</i>	String Optional. The number of milliseconds to wait for the search to complete (0, the default, indicates to wait forever).
<i>returnAttributes</i>	Record 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>	String 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>	String 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>	String Optional. Whether to return results as a list. Set to:

- `Yes` to return results as a Document List containing Documents (IData). The results are returned in the `resultsList` output parameter.
- `No` (the default) to return results as a Document containing Documents. The results are returned in the `results` output parameter.

Output Parameters

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

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

<i>provider</i>	<p>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.</p> <p>Possible values are <code>IntegrationServer</code> and <code>Other</code> (case insensitive).</p> <p>Note: If set to <code>Other</code>, you must specify a value for <i>providerClass</i>.</p>
<i>providerClass</i>	<p>String 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>Note: 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>String 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>String 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>String The access token assigned to the client application when it registered with the provider.</p>

Note: The process for obtaining the *accessToken* varies depending on the provider. For information about obtaining the *accessToken*, refer to your provider's documentation.

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.

accessTokenSecret **String** Optional. The access token secret assigned to the client application when it registered with the provider.

Note: The process for obtaining the *accessTokenSecret* varies depending on the provider. For information about obtaining the *accessTokenSecret*, refer to your provider's documentation.

Use this parameter to specify either the access token secret or the key to the access token secret in the outbound password store. 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.

resourceUri **String** The URI given to the client application when it registered with the provider.

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.

method **String** The HTTP method Integration Server will use to issue the request to the provider.

Possible values are `get`, `post`, `put`, and `delete`.

headers **Document List** Optional. One or more name/value pairs to add to the header of the request sent to the provider.

If the provider requires a request header, you must provide a value.

queryStringParameters **Document List** Optional. One or more name/value pairs to add to the URL of the `get` request sent to the provider.

<i>bodyParameters</i>	Document List Optional. One or more name/value pairs to add to the body of the request sent to the provider.
<i>requestDataType</i>	String Optional. If supplying data with the request to the provider, indicates the data type of the <i>requestData</i> parameter. If set to: <ul style="list-style-type: none"> ■ <i>bytes</i>, <i>requestData</i> must be a byte[]. This is the default. ■ <i>string</i>, <i>requestData</i> must be a java.lang.String.
<i>requestData</i>	Object Optional. Data to include in the body of the request sent to the provider. The value can be a string or a byte[]. <p>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>	String Optional. Indicates the data type of the <i>responseData</i> output parameter. Set to: <ul style="list-style-type: none"> ■ <i>stream</i> to return the <i>responseData</i> as a java.io.InputStream. This is the default. ■ <i>bytes</i> to return the <i>responseData</i> as a byte[]. ■ <i>string</i> to return the <i>responseData</i> 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

<i>responseData</i>	Object The response from the provider. If the <i>responseDataType</i> is set to: <ul style="list-style-type: none"> ■ <i>stream</i>, <i>responseData</i> is a java.io.InputStream. ■ <i>bytes</i>, <i>responseData</i> is a byte[]. ■ <i>string</i>, <i>responseData</i> 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>	String Unique key for the current SFTP session. The <i>sessionKey</i> is returned by the pub.client.sftp:login service.
<i>path</i>	String Absolute or relative path of the directory that you want as the working directory on the remote SFTP server.

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:chgrp

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

Input Parameters

sessionKey **String** Unique key for the current SFTP session. The *sessionKey* is returned by the [pub.client.sftp:login](#) service.

groupId **String** Numeric group identifier of the group to which you want to transfer ownership of the remote files.

path **String** Absolute or relative path of the remote files.

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:chmod

WmPublic. Changes permissions of one or more remote files.

Input Parameters

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

Output Parameters

<i>returnCode</i>	String Standard SFTP protocol return code.
<i>returnMsg</i>	String 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>	String Unique key for the current SFTP session. The <i>sessionKey</i> is returned by the pub.client.sftp:login service.
<i>uid</i>	String Numeric user ID of the new owning user of the file.
<i>path</i>	String Absolute or relative path of the remote files.

Output Parameters

<i>returnCode</i>	String Standard SFTP protocol return code.
<i>returnMsg</i>	String 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>	String Unique key for the current SFTP session. The <i>sessionKey</i> is returned by the pub.client.sftp:login service.
<i>remoteFile</i>	String Absolute or relative path of the remote file.
<i>localFile</i>	String Optional. Absolute or relative path of the local file. If <i>localFile</i> is not specified, the <code>pub.client.sftp:get</code> service returns the retrieved file in the output parameter <i>contentStream</i> (as a <code>java.io.InputStream</code> object).
<i>mode</i>	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 <i>localFile</i> . Set to: <ul style="list-style-type: none">■ <code>overwrite</code> to overwrite the contents of the local file with the contents of the remote file. This is the default.■ <code>append</code> to append the entire contents of the remote file to the local file.■ <code>resume</code> 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

<i>returnCode</i>	String Standard SFTP protocol return code.
<i>returnMsg</i>	String Text message describing the return code.
<i>contentStream</i>	Object Conditional. A <code>java.io.InputStream</code> 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>	String 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>	String 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">■ <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 <code>pub.client.sftp:login</code> service will create a new session and return the key for this new session.■ <code>false</code> to create a new session for the specified user alias. This is the default.

Output Parameters

<i>sessionKey</i>	String Unique key for the current SFTP session. Most other <code>pub.client.sftp</code> services require this session key to execute.
<i>returnCode</i>	String Standard SFTP protocol return code.
<i>returnMsg</i>	String 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

sessionKey **String** Unique key for the current SFTP session. The *sessionKey* parameter is returned by the [pub.client.sftp:login](#) 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

<i>returnCode</i>	String Standard SFTP protocol return code.
<i>returnMsg</i>	String Text message describing the return code.
<i>dirList</i>	String List List of directories matching the pattern specified in the <i>path</i> 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>	String Unique key for the current SFTP session. The <i>sessionKey</i> parameter is returned by the pub.client.sftp:login service.
<i>path</i>	String Absolute or relative path of the remote directory where you want to create a new directory.

Output Parameters

<i>returnCode</i>	String Standard SFTP protocol return code.
<i>returnMsg</i>	String 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>	String Unique key for the current SFTP session. The <i>sessionKey</i> is returned by the pub.client.sftp:login service.
<i>contentStream</i>	java.io.InputStream Optional. Data to be transferred to the remote file.
<i>localFile</i>	String 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>	String Optional. Absolute or relative path of the remote file to which the local file is to be appended.
<i>mode</i>	String Optional. Specifies how the local file is to be transferred to the remote SFTP server. Set to: <ul style="list-style-type: none">■ <code>overwrite</code> to overwrite the contents of the remote file with the contents of the local file. This is the default.■ <code>append</code> to append the entire contents of the local file to the remote file.■ <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.

Output Parameters

<i>returnCode</i>	String Standard SFTP protocol return code.
<i>returnMsg</i>	String 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

sessionKey **String** Unique key for the current SFTP session. The *sessionKey* is returned by the [pub.client.sftp:login](#) service.

Output Parameters

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

returnMsg **String** 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

sessionKey **String** Unique key for the current SFTP session. The *sessionKey* is returned by the [pub.client.sftp:login](#) service.

oldPath **String** Old path of the file for which you want to create a symbolic link.

newPath **String** New path of the file to which the symbolic link should point.

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: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>	String Optional. E-mail address of the receiver. If you specify multiple addresses, separate them with commas.
<i>cc</i>	String Optional. E-mail addresses of additional receivers. If you specify multiple addresses, separate them with commas.
<i>bcc</i>	String Optional. E-mail addresses of additional receivers. If you specify multiple addresses, separate them with commas.
<i>subject</i>	String Optional. Subject of the message.
<i>subjectCharset</i>	String 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 <code>watt.server.email.charset</code> is used. If that parameter is not set, the <code>utf-8</code> encoding is used.
<i>charset</i>	String Optional. The character encoding of the body text. If you do not specify the value of <i>charset</i> , 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.
<i>from</i>	String Optional. E-mail address of the sender. If you do not specify a <i>from</i> value, Integration Server uses the value specified for the <code>mail.smtp.from</code> JVM property. If no value is specified for that property, Integration Server uses the default value, <code>user@servername</code> , where <i>user</i> is the operating system user ID, and <i>servername</i> is the host name of the Integration Server.
<i>mailhost</i>	String SMTP host name for outbound messages. For example: <code>smtp.webMethods.com</code> If no value is provided for the <i>mailhost</i> parameter, Integration Server uses the value of the system property <code>mail.smtp.host</code> in the <code>startup.bat</code> (<code>startup.sh</code>) file as the <i>mailhost</i> value.
<i>mailhostPort</i>	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>	String User name that this service will submit when requesting a protected resource.
<i>pass</i>	String 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.

Key	Description
<i>transportLayerSecurity</i>	<p>String Type of security protocol Integration Server uses when communicating with the SMTP server port. Set to:</p> <ul style="list-style-type: none"> ■ <i>none</i> to use a non-secure mode when communicating with the port on the SMTP sever. This is the default. ■ <i>explicit</i> 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. ■ <i>implicit</i> 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.
<i>truststoreAlias</i>	String 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.

body **String** Optional. The content of the message.

mimeStream **java.io.InputStream** Optional. MIME or S/MIME message that you want to send in the e-mail. A *mimeStream* is created by the [pub.mime:getEnvelopeStream](#), [pub.smime:createEncryptedData](#), [pub.smime:createSignedData](#), or [pub.smime.keystore:createSignedData](#) services. It contains both headers and content. If the *mimeStream* already contains the from, to, and subject headers, you do not need to pass them as individual inputs to this service.

attachments **Document List** Optional. Attachments to the message. Each attachment defines one message part in a multi-part message.

Key	Description
<i>contentType</i>	String MIME type of the message. For example: application/x-edl-message
<i>content</i>	byte[] , String , or java.io.InputStream Content of the message.
<i>filename</i>	String 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: <ul style="list-style-type: none"> ■ 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>. ■ 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>.
<i>encoding</i>	String 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>	String 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 `utf-8` 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.

Key	Description
<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`

<i>request</i>	Document The input parameters that are to be passed to the web service.
<i>wsdName</i>	String The name of the consumer web service descriptor that contains the operation you want to invoke.
<i>wsdBinderName</i>	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 <i>wsdName</i> . 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 <i>webMethods Integration Server Administrator's Guide</i> guide.
<i>wsdOperationName</i>	String The name of the operation that you want to invoke. This operation must be contained in the binder specified in <i>wsdBinderName</i> .
<i>targetInputSignature</i>	String Fully qualified name of the IS document type to use to validate and encode the contents of <i>request</i> .
<i>targetOutputSignature</i>	String Fully qualified name of the IS document type to use to validate and decode the output value returned by the web service.
<i>soapHeaders</i>	Document Optional. Header documents included in the SOAP request as SOAP headers.
<i>transportHeaders</i>	Document Optional. Transport header fields that you want to explicitly set in the request issued by the <code>pub.client:soapClient</code> service. Specify a key in <i>transportHeaders</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. The names and values supplied to <i>transportHeaders</i> 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 <i>transportHeaders</i> vary depending on the transport used to send the SOAP message. The supplied <i>wsdBinderName</i> 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:

Key	Value
<i>namespaceName</i>	String Namespace portion of the procedure's QName.
<i>localName</i>	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:

Key	Description				
<i>transport</i>	Document Optional. Transport level authorization parameters.				
	<table border="1"> <thead> <tr> <th>Key</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td><i>type</i></td> <td> String Optional. Type of authentication that the service will perform. Set to: <ul style="list-style-type: none"> ■ <code>Basic</code> to submit a user name and password. This is the default. ■ <code>Digest</code> to submit a password digest for authentication. </td> </tr> </tbody> </table>	Key	Description	<i>type</i>	String Optional. Type of authentication that the service will perform. Set to: <ul style="list-style-type: none"> ■ <code>Basic</code> to submit a user name and password. This is the default. ■ <code>Digest</code> to submit a password digest for authentication.
Key	Description				
<i>type</i>	String Optional. Type of authentication that the service will perform. Set to: <ul style="list-style-type: none"> ■ <code>Basic</code> to submit a user name and password. This is the default. ■ <code>Digest</code> to submit a password digest for authentication. 				

- NTLM to use NTLM authentication.

user **String** 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`

pass **String** Optional.
Password that this service will submit if one is requested.

serverCerts **Document** Optional.
The message signer's private key and certificate chain.

privateKey **Object**
The SOAP message signer's private key.

certChain **Object List**
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.

message **Document** Optional. Message level authorization parameters.

<u>Key</u>	<u>Description</u>
<i>user</i>	String Optional. User name that this service will use if one is requested.
<i>pass</i>	String Optional. Password that this service will submit if one is requested.
<i>serverCerts</i>	Document Optional. The message signer's private key and certificate chain. <i>privateKey</i> Object The SOAP message signer's private key. <i>certChain</i> Object List 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>partnerCert</i>	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`.

<i>soapProtocol</i>	String Optional. Indicates the SOAP protocol the service uses to send messages. Valid values are SOAP 1.1 Protocol or SOAP 1.2 Protocol.
<i>encoding</i>	String Optional. Specifies the encoding method. Default value is UTF-8. Integration Server ignores <i>encoding</i> if the Content-Type header is passed in to <i>transportHeaders</i> .

Output Parameters

<i>soapResponseData</i>	Object A SOAP object containing the SOAP response message returned by the server specified in <i>address</i> .
<i>response</i>	Document Output parameters returned from the web service.
<i>header</i>	Document Conditional. Headers from the response and request messages.

header contains the following keys:

<u>Key</u>	<u>Value</u>
<i>requestLines</i>	<p>Document 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 transport can add, remove, or alter specific headers while processing the request.</p> <p>Whether or not the web service connector returns the <i>requestLines</i> parameter depends on the success or failure of the <code>pub.client:soapClient</code> service. In the case of failure, the point at which the failure occurs determines the presence of the <i>requestLines</i> parameter. For more information, see the usage notes for this service.</p>

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.

Property Name	Description
<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.

Note: This property applies when `pub.client:soapClient` calls an operation with an In-Out message exchange pattern only

Value	Description
<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 true, 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.
-------------------------------	---

Value	Description
<code>PERSISTENT</code>	Indicates the request message is persistent.
<code>2</code>	Default. Indicates the request message is persistent.

Property Name	Description
	<p>NON_PERSISTENT Indicates the request message is not persistent.</p> <p>1 Indicates the request message is not persistent.</p> <p>Note: 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"> ■ BytesMessage ■ TextMessage <p>Note: 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>Note: 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> <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>

- 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

<i>soapRequestData</i>	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.
<i>address</i>	String URL to which you want the SOAP message sent. For example: <code>https://servername:5555/soap/default</code>
<i>auth</i>	Document Optional. Parameters specifying the credentials that are to be submitted to the server specified in <i>address</i> . Each element is defined as follows:

Key	Description
<i>type</i>	String 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>	String User name that this service will use if one is requested.
<i>pass</i>	String Password that this service will submit if one is requested.
<i>validateSOAP</i>	String Optional. Indicates whether or not the response message is to be validated against the SOAP schema. Set to: <ul style="list-style-type: none">■ <code>true</code> to validate the response message and throw an exception if the response does not conform to the SOAP schema.■ <code>false</code> to bypass the validation process. This is the default.
<i>SOAPAction</i>	String Optional. Value to which you want to set the SOAPAction HTTP header.
<i>contentType</i>	String Optional. Specifies the value of Content-Type in the HTTP header. Set to: <ul style="list-style-type: none">■ <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.■ <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.
<i>loadAs</i>	String 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: <ul style="list-style-type: none">■ <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.■ <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, pub.xml:queryXMLNode).■ <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. 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:

<u>Key</u>	<u>Description</u>
<i>lines</i>	Document 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>	String Status code from the HTTP response.
<i>statusMessage</i>	String Status message from the HTTP response.

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> 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

<i>address</i>	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>	<u>Value</u>
<i>acct</i>	<code>Cash</code>
<i>amt</i>	<code>150.00</code>
<i>org</i>	<code>Sales</code>

method

Document The QName of the requested procedure where:

<u>Key</u>	<u>Value</u>
<i>namespaceName</i>	String Namespace portion of the procedure's QName.
<i>localName</i>	String 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>	String 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>	String User name that this service will use if one is requested.
<i>pass</i>	String Password that this service will submit if one is requested.

<i>targetInputSignature</i>	String Optional. Fully qualified name of the IS document type to use to validate and encode the contents of <i>reqParms</i> .
<i>targetOutputSignature</i>	String 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>	String Optional. Value to which you want to set the SOAPAction HTTP header.
<i>contentType</i>	String Optional. Specifies the value of Content-Type in the HTTP header. Set to: <ul style="list-style-type: none">■ <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.■ <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.
<i>encoding</i>	String Optional. Specifies the encoding method. Default value is UTF-8.
<i>loadAs</i>	String Optional. Specifies the format of the <i>soapResponseData</i> . Default value is <code>stream</code> . Set to: <ul style="list-style-type: none">■ <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.■ <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.
<i>timeout</i>	String 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

<i>soapResponseData</i>	Object A SOAP object containing the SOAP response message returned by the server specified in <i>address</i> .
-------------------------	---

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>	Document 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>	String Status code from the HTTP response.
<i>statusMessage</i>	String 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`

4 Date Folder

■ Pattern String Symbols	176
■ Time Zones	177
■ Examples	179
■ Notes on Invalid Dates	180
■ Summary of Elements in this Folder	180

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

ID	Raw Offset	Name
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
pub.date.calculateDateDifference	WmPublic. Calculates the difference between two dates and returns the result as seconds, minutes, hours, and days.
pub.date.currentNanoTime	WmPublic. Returns the 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.

Element	Package and Description
pub.date:dateTimeBuild	WmPublic. Builds a date/time string using the specified pattern and the specified date elements.
pub.date:dateTimeFormat	WmPublic. Converts date/time (represented as a String) string from one format to another.
pub.date:elapsedNanoTime	WmPublic. Calculates the time elapsed between the current time and the given time, in nanoseconds.
pub.date:formatDate	WmPublic. Formats a Date object as a string.
pub.date:getCurrentDate	WmPublic. Returns the current date as a Date object.
pub.date:getCurrentDateString	WmPublic. Returns the current date as a String in a specified format.
pub.date:getWorkingDays	WmPublic. Returns the number of working days between two dates.

pub.date:calculateDateDifference

WmPublic. Calculates the difference between two dates and returns the result as seconds, minutes, hours, and days.

Input Parameters

<i>startDate</i>	String Starting date and time.
<i>endDate</i>	String Ending date and time.
<i>startDatePattern</i>	String Format in which the <i>startDate</i> parameter is to be specified (for example, yyyyMMdd HH:mm:ss.SSS). For pattern-string notation, see " Pattern String Symbols " on page 176.

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 176](#).

Output Parameters

dateDifferenceSeconds **String** The difference between the *startingDateTime* and *endingDateTime*, truncated to the nearest whole number of seconds.

dateDifferenceMinutes **String** The difference between the *startingDateTime* and *endingDateTime*, truncated to the nearest whole number of minutes.

dateDifferenceHours **String** The difference between the *startingDateTime* and *endingDateTime*, truncated to the nearest whole number of hours.

dateDifferenceDays **String** The difference between the *startingDateTime* and *endingDateTime*, 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: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

<i>pattern</i>	String Pattern representing the format in which you want the date returned. For pattern-string notation, see "Pattern String Symbols" on page 176 . If you do not specify <i>pattern</i> , <code>dateBuild</code> returns null. If <i>pattern</i> contains a time zone and <i>timezone</i> is not specified, the default time zone of webMethods Integration Server is used.
<i>year</i>	String Optional. The year expressed in <i>yyyy</i> or <i>yy</i> format (for example, 01 or 2001). If you do not specify <i>year</i> or you specify an invalid value, <code>dateBuild</code> uses the current year.
<i>month</i>	String Optional. The month expressed as a number (for example, 1 for January, 2 for February). If you do not specify <i>month</i> or you specify an invalid value, <code>dateBuild</code> uses the current month.
<i>dayofmonth</i>	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 <i>dayofmonth</i> or you specify an invalid value, <code>dateBuild</code> uses the current day.
<i>timezone</i>	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 177 (for example, EST 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>	String 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> .

Output Parameters

<i>value</i>	String The date specified by <i>year</i> , <i>month</i> , and <i>dayofmonth</i> , in the format of <i>pattern</i> .
--------------	--

pub.date:dateTimeBuild

WmPublic. Builds a date/time string using the specified pattern and the specified date elements.

Input Parameters

<i>pattern</i>	String Pattern representing the format in which you want the time returned. For pattern-string notation, see "Pattern String Symbols" on page 176 . If you do not specify <i>pattern</i> , <code>dateTimeBuild</code> returns null. If <i>pattern</i> contains a time zone and the <i>timezone</i> parameter is not set, the time zone of Integration Server is used.
<i>year</i>	String Optional. The year expressed in <i>yyyy</i> or <i>yy</i> format (for example, 01 or 2001). If you do not specify <i>year</i> or you specify an invalid value, <code>dateTimeBuild</code> uses the current year.
<i>month</i>	String Optional. The month expressed as a number (for example, 1 for January, 2 for February). If you do not specify <i>month</i> or you specify an invalid value, <code>dateTimeBuild</code> uses the current month.
<i>dayofmonth</i>	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 <i>dayofmonth</i> or you specify an invalid value, <code>dateTimeBuild</code> uses the current day.
<i>hour</i>	String 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>	String 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>	String 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>	String 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>	<p>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 177 (for example, <code>EST</code> 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>String 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>

Output Parameters

<i>value</i>	String Date and time in format of <i>pattern</i> .
--------------	---

pub.date:dateTimeFormat

WmPublic. Converts date/time (represented as a String) string from one format to another.

Input Parameters

<i>inString</i>	<p>String Date/time that you want to convert.</p> <p>Important: 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 "Notes on Invalid Dates" on page 180.</p>
<i>currentPattern</i>	String Pattern string that describes the format of <i>inString</i> . For pattern-string notation, see "Pattern String Symbols" on page 176 .
<i>newPattern</i>	String Pattern string that describes the format in which you want <i>inString</i> returned. For pattern-string syntax, see "Pattern String Symbols" on page 176 .
<i>locale</i>	<p>String 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>

<i>lenient</i>	<p>String 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"> ■ <code>true</code> to perform a lenient check. This is the default. <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 returns the date in the format specified in the <i>currentPattern</i> parameter. If the interpretation is incorrect, the service will return an invalid date.</p> <ul style="list-style-type: none"> ■ <code>false</code> to perform a strict check. <p>In a strict check, the Integration Server throws an exception 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.</p>
----------------	---

Output Parameters

<i>value</i>	<p>String The date/time given by <i>inString</i>, in the format of <i>newPattern</i>.</p>
--------------	--

Usage Notes

As described in ["Notes on Invalid Dates" on page 180](#), 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

elapsedNanoTime **java.lang.Long** The difference between the current time in nanoseconds and *nanoTime*. If *nanoTime* is greater than the current nano time, the service returns zero.

elapsedNanoTimeStr **String** The difference between the current time in nanoseconds and *nanoTime*. The difference is expressed as a String, in this format:

[years] [days] [hours] [minutes] [seconds] [millisec] [microsec] <nanosec>

If *nanoTime* is greater than the current nano time, the service returns zero.

pub.date:formatDate

WmPublic. Formats a Date object as a string.

Input Parameters

date **java.util.Date** Optional. Date/time that you want to convert.

pattern **String** Pattern string that describes the format in which you want the date returned. For pattern-string notation, see "[Pattern String Symbols](#)" on page 176.

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 177 (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`.

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 176.

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 177 (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`.

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 176](#).

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 176](#).

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.

5 Db Folder

■ Summary of Elements in this Folder	192
--	-----

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
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.
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.
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.
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.
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.
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.
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.

Element	Package and Description
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.
pub.db:getProclInfo	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.
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.
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.
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.
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.
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.
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.
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.
pub.db:update	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>	String Optional. Alias of the database on which you want to execute the stored procedure.
<i>\$dbURL</i>	String Optional. JDBC URL that identifies the database resource.
<i>\$dbUser</i>	String Optional. User name to use to log into the database.
<i>\$dbPass</i>	String Optional. Password for the user specified in <i>\$dbUser</i> .
<i>\$dbDriver</i>	String Optional. Name of the JDBC driver to use.
<i>\$dbConnection</i>	com.wm.app.b2b.server.DBConnection Optional. Connection object returned by pub.db:connect .
<i>\$dbCatalog</i>	<p>String 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>	String Parameter name defined in the stored procedure.																				
<i>sqlType</i>	<p>String Type of procedure parameter for <i>name</i> as defined in the database. Set to one of the following values:</p> <table border="0"> <tr> <td>BIT</td> <td>TINYINT</td> </tr> <tr> <td>SMALLINT</td> <td>INTEGER</td> </tr> <tr> <td>BIGINT</td> <td>FLOAT</td> </tr> <tr> <td>REAL</td> <td>DOUBLE</td> </tr> <tr> <td>NUMERIC</td> <td>DECIMAL</td> </tr> <tr> <td>CHAR</td> <td>VARCHAR</td> </tr> <tr> <td>LONGVARCHAR</td> <td>DATE</td> </tr> <tr> <td>TIME</td> <td>TIMESTAMP</td> </tr> <tr> <td>BINARY</td> <td>VARBINARY</td> </tr> <tr> <td>LONGVARBINARY</td> <td>NULL</td> </tr> </table>	BIT	TINYINT	SMALLINT	INTEGER	BIGINT	FLOAT	REAL	DOUBLE	NUMERIC	DECIMAL	CHAR	VARCHAR	LONGVARCHAR	DATE	TIME	TIMESTAMP	BINARY	VARBINARY	LONGVARBINARY	NULL
BIT	TINYINT																				
SMALLINT	INTEGER																				
BIGINT	FLOAT																				
REAL	DOUBLE																				
NUMERIC	DECIMAL																				
CHAR	VARCHAR																				
LONGVARCHAR	DATE																				
TIME	TIMESTAMP																				
BINARY	VARBINARY																				
LONGVARBINARY	NULL																				
<i>direction</i>	<p>String Way in which the parameter is used by the stored procedure. Set to one of the following values:</p> <p>in out inout return value</p>																				

\$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>	String Optional. User name to use to log into the database.
<i>\$dbPass</i>	String Optional. Password for the user.
<i>\$dbDriver</i>	String Optional. Name of the JDBC driver to use.
<i>\$dbConnection</i>	com.wm.app.b2b.server.DBConnection Optional. Connection object returned by pub.db:connect .

Output Parameters

<i>\$dbMessage</i>	String 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>	String Optional. Database alias.
<i>\$dbURL</i>	String Optional. JDBC URL that identifies the database resource.

<i>\$dbUser</i>	String Optional. User name to use to log into the database.
<i>\$dbPass</i>	String Optional. Password for the user.
<i>\$dbDriver</i>	String Optional. Name of the JDBC driver to use.
<i>\$dbConnection</i>	com.wm.app.b2b.server.DBConnection Optional. Connection object returned by pub.db:connect .
<i>\$closeDbConnection</i>	String Optional. Indicates whether to remove the database connection from the pool or return it to the pool for future use. Set to: <ul style="list-style-type: none">■ <code>true</code> to close the connection and remove it from the pool.■ <code>false</code> to close the connection and return it to the pool for future use. This is the default.

Output Parameters

<i>\$dbMessage</i>	String 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>	String 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>	String 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>	String Optional. JDBC URL that identifies the database resource.
<i>\$dbUser</i>	String Optional. User name to use to log into the database.
<i>\$dbPass</i>	String Optional. Password for the user.
<i>\$dbDriver</i>	String Optional. Name of the JDBC driver to use.
<i>\$dbConnection</i>	com.wm.app.b2b.server.DBConnection Optional. Connection object returned by pub.db:connect .

Output Parameters

<i>\$dbMessage</i>	String 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 217).

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>	String Optional. Database alias.
<i>\$dbURL</i>	String Optional. JDBC URL that identifies the database resource.
<i>\$dbUser</i>	String Optional. User name to use to log into the database.
<i>\$dbPass</i>	String Optional. Password for the user.
<i>\$dbDriver</i>	String Optional. Name of the JDBC driver to use.
<i>\$dbProperties</i>	<p>Document 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>

Key	Value
<i>user</i>	dbu
<i>password</i>	dbu
<i>weblogic.codeset</i>	GBK

Output Parameters

<i>\$dbConnection</i>	com.wm.app.b2b.server.DBConnection Connection object.
<i>\$dbMessage</i>	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*

<i>\$dbAlias</i>	String Optional. Database alias.
<i>\$dbURL</i>	String Optional. JDBC URL that identifies the database resource.
<i>\$dbUser</i>	String Optional. User name to use to log into the database.

<i>\$dbPass</i>	String Optional. Password for the user.
<i>\$dbDriver</i>	String Optional. Name of the JDBC driver to use.
<i>\$dbConnection</i>	com.wm.app.b2b.server.DBConnection Optional. Connection object returned by pub.db.connect .
<i>\$dbCatalog</i>	<p>String 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>String 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>	String Name of the table to remove rows from.
<i>\$data</i>	<p>Document Optional. Criteria that the rows to delete must meet.</p> <p>Important: If no criteria are provided, <i>all rows</i> are deleted from the table.</p>

Output Parameters

<i>\$updateCount</i>	String Number of rows deleted.
<i>\$dbMessage</i>	String 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>	String Optional. Database alias.
<i>\$dbURL</i>	String Optional. JDBC URL that identifies the database resource.
<i>\$dbUser</i>	String Optional. User name to use to log into the database.
<i>\$dbPass</i>	String Optional. Password for the user.
<i>\$dbDriver</i>	String Optional. Name of the JDBC driver to use.
<i>\$dbConnection</i>	com.wm.app.b2b.server.DBConnection Optional. Connection object returned by pub.db:connect .
<i>\$dbCatalog</i>	<p>String 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.</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>String 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>\$dbSQL</i>	String SQL statement to execute.
<i>\$dbProcessEsc</i>	<p>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.</p> <p>Set to:</p> <ul style="list-style-type: none"> ■ <code>true</code> to process JDBC SQL escapes. This is the default. ■ <code>false</code> to skip processing JDBC SQL escapes.
<i>\$dbProcessReporterTokens</i>	<p>String Optional. Flag that indicates whether reporter tags (for example, <code>%value xxx%</code>) 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.</p> <p>Set to:</p> <ul style="list-style-type: none"> ■ <code>true</code> to process tags. ■ <code>false</code> to ignore tags. This is the default.
<i>\$dbParamValues</i>	<p>Object List Optional. If the "?" parameters in the SQL statement are not supplied indirectly (with the <i>\$dbParamNames</i> parameter), they can be supplied directly via this parameter. See "Usage Notes" on page 205 below. Objects in <i>\$dbParamValues</i> can be of any type.</p>
<i>\$dbParamNames</i>	<p>String List Optional. Names of any "?" parameters in the SQL. See "Usage Notes" on page 205 below.</p>
<i>\$dbParamTypes</i>	<p>String List Optional. SQL type names for each parameter. Use type names from the JDBC 1.2 specification ("INTEGER", "VARCHAR", etc.).</p>

Output Parameters

<i>sql</i>	<p>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).</p>
------------	---

<i>paramsAsStrings</i>	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.
<i>\$rowCount</i>	String Conditional. Number of rows in <i>results</i> .
<i>results</i>	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.
<i>\$updateCount</i>	String Conditional. Number of rows updated.
<i>\$dbMessage</i>	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.

Key	Value	Description
<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.
<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

Key	Value	Description
		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*

<i>\$dbAlias</i>	String Optional. Database alias.
<i>\$dbURL</i>	String Optional. JDBC URL that identifies the database resource.
<i>\$dbUser</i>	String Optional. User name to use to log into the database.
<i>\$dbPass</i>	String Optional. Password for the user.
<i>\$dbDriver</i>	String Optional. Name of the JDBC driver to use.
<i>\$dbConnection</i>	com.wm.app.b2b.server.DBConnection Optional. Connection object returned by pub.db:connect .
<i>\$dbCatalog</i>	<p>String 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>String 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>	<p>String Name of the procedure about which you want information.</p>

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>	String Optional. Database alias.
<i>\$dbURL</i>	String Optional. JDBC URL that identifies the database resource.
<i>\$dbUser</i>	String Optional. User name to use to log into the database.

<i>\$dbPass</i>	String Optional. Password for the user.
<i>\$dbDriver</i>	String Optional. Name of the JDBC driver to use.
<i>\$dbConnection</i>	com.wm.app.b2b.server.DBConnection Optional. Connection object returned by pub.db.connect .
<i>\$dbCatalog</i>	String 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. 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>	String Optional. Name of the schema to which the stored procedures belong. 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> . If you are running against DB2, you use this parameter to specify the database's AuthID.
<i>\$dbProcNamePattern</i>	String Optional. Pattern-matching string that specifies the procedures that you want included in the returned list, where <code>_</code> represents a single character and <code>%</code> represents any string of characters. For example, the value <code>DATE%</code> would represent any procedure beginning with the characters <code>DATE</code> .

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*

<i>\$dbAlias</i>	String Optional. Database alias.
<i>\$dbURL</i>	String Optional. JDBC URL that identifies the database resource.
<i>\$dbUser</i>	String Optional. User name to use to log into the database.
<i>\$dbPass</i>	String Optional. Password for the user.
<i>\$dbDriver</i>	String Optional. Name of the JDBC driver to use.
<i>\$dbConnection</i>	com.wm.app.b2b.server.DBConnection Optional. Connection object returned by pub.db:connect .
<i>\$dbCatalog</i>	String 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>	String 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 `_` 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 whose column names you want to retrieve.

\$dbColumnNamePattern

String Optional. Pattern-matching string that specifies the column names that you want to retrieve, where `_` represents a single character and `%` represents any string of characters. For example, the value `ADDR%` would represent any column name beginning with the characters `ADDR`.

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*

\$dbAlias

String Optional. Database alias.

<i>\$dbURL</i>	String Optional. JDBC URL that identifies the database resource.
<i>\$dbUser</i>	String Optional. User name to use to log into the database.
<i>\$dbPass</i>	String Optional. Password for the user.
<i>\$dbDriver</i>	String Optional. Name of the JDBC driver to use.
<i>\$dbConnection</i>	com.wm.app.b2b.server.DBConnection Optional. Connection object returned by pub.db.connect .
<i>\$dbCatalog</i>	<p>String 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>String 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 <code>HR</code>.</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>String 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 <code>HR</code>.</p>

If you want all table names, set *\$dbTableNamePattern* to null.

\$dbTableTypeList

String List Optional. Set of parameters specifying the types of tables whose names you want to retrieve. Common JDBC table types include: TABLE, VIEW, SYSTEM TABLE, ALIAS, and SYNONYM. Check your driver documentation for others.

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*

\$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.

\$dbDriver **String** Optional. Name of the JDBC driver to use.

\$dbConnection **com.wm.app.b2b.server.DBConnection** Optional. Connection object returned by [pub.db:connect](#).

<i>\$dbCatalog</i>	<p>String 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>String 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>	<p>String Name of table in which you want to insert rows.</p>
<i>\$dbRollbackOnFail</i>	<p>String Optional. Flag that determine whether changes are committed if a failure occurs while processing multiple inserts. Set to:</p> <ul style="list-style-type: none"> ■ <code>true</code> to undo changes on failure. ■ <code>false</code> to commit changes on failure. This is the default.
<i>\$data</i>	<p>Document or Document List Optional. Data to insert.</p>

Output Parameters

<i>\$updateCount</i>	<p>String Number of rows the service inserted.</p>
<i>\$failCount</i>	<p>String Number of rows the service failed to insert.</p>
<i>\$errors</i>	<p>Document Conditional. Error messages generated during service execution.</p>
<i>\$dbMessage</i>	<p>String Conditional. Message indicating the success or failure of the operation.</p>

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>	String Optional. Database alias.
<i>\$dbURL</i>	String Optional. JDBC URL that identifies the database resource.
<i>\$dbUser</i>	String Optional. User name to use to log into the database.
<i>\$dbPass</i>	String Optional. Password for the user.
<i>\$dbDriver</i>	String Optional. Name of the JDBC driver to use.
<i>\$dbConnection</i>	com.wm.app.b2b.server.DBConnection Optional. Connection object returned by pub.db:connect .
<i>\$dbCatalog</i>	<p>String 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>String 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>	<p>String Name of table to query.</p>
<i>\$data</i>	<p>Document Optional. Criteria that the rows to retrieve must meet.</p>

Output Parameters

<i>results</i>	<p>com.wm.util.Table Conditional. Results of the query. The Integration Server recognizes and treats this parameter as a Document List at run time.</p>
<i>\$dbMessage</i>	<p>String Conditional. Message indicating the success or failure of an operation.</p>
<i>\$rowCount</i>	<p>String Conditional. Number of rows for the table that meet the criteria specified in <i>\$data</i>.</p>

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>	String Optional. Alias of the database for which you want to discard changes. This information is passed automatically.
<i>\$dbURL</i>	String Optional. JDBC URL that identifies the database resource.
<i>\$dbUser</i>	String Optional. User name to use to log into the database.
<i>\$dbPass</i>	String Optional. Password for the user.
<i>\$dbDriver</i>	String Optional. Name of the JDBC driver to use.
<i>\$dbConnection</i>	com.wm.app.b2b.server.DBConnection Optional. Connection object returned by pub.db:connect .

Output Parameters

<i>\$dbMessage</i>	String 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>	String Optional. Alias of the database for which you want to start the transaction. This information is passed automatically.
<i>\$dbURL</i>	String Optional. JDBC URL that identifies the database resource.
<i>\$dbUser</i>	String Optional. User name to use to log into the database.
<i>\$dbPass</i>	String Optional. Password for the user.
<i>\$dbDriver</i>	String Optional. Name of the JDBC driver to use.
<i>\$dbConnection</i>	com.wm.app.b2b.server.DBConnection Optional. Connection object returned by pub.db:connect .

Output Parameters

<i>\$dbMessage</i>	String 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>	String Optional. Database alias.
<i>\$dbURL</i>	String Optional. JDBC URL that identifies the database resource.
<i>\$dbUser</i>	String Optional. User name to use to log into the database.
<i>\$dbPass</i>	String Optional. Password for the user.
<i>\$dbDriver</i>	String Optional. Name of the JDBC driver to use.
<i>\$dbConnection</i>	com.wm.app.b2b.server.DBConnection Optional. Connection object returned by pub.db.connect .
<i>\$dbCatalog</i>	<p>String 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.</p> <p>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.</p> <p>If you are running against DB2, you use this parameter to specify the database location.</p>
<i>\$dbSchemaPattern</i>	<p>String 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>	String Name of table to update.
<i>\$criteria</i>	Document Criteria that the rows to update must meet. Important: If no criteria are provided, <i>all rows</i> are updated.
<i>\$set</i>	Document New data with which to update rows.

Output Parameters

<i>\$updateCount</i>	String Number of rows updated.
<i>\$dbMessage</i>	String Conditional. Message indicating the operation failed.

6 Document Folder

■ Summary of Elements in this Folder	222
--	-----

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
pub.document:bytesToDocument	WmPublic. Converts an array of bytes to a document.
pub.document:deleteDocuments	WmPublic. Deletes the specified documents from a set of documents.
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.
pub.document:documentToBytes	WmPublic. Converts a document to an array of bytes.
pub.document:documentToDocumentList	WmPublic. Expands the contents of a document into a list of documents.
pub.document:documentToXMLValues	WmPublic. Converts a document (IData object) to a String by encoding it in the webMethods XMLValues format.
pub.document:groupDocuments	WmPublic. Groups a set of documents based on specified criteria.
pub.document:insertDocument	WmPublic. Inserts a new document in a set of documents at a specified position.
pub.document:searchDocuments	WmPublic. Searches a set of documents for entries matching a set of Criteria.
pub.document:sortDocuments	WmPublic. Sorts a set of input documents based on the specified sortCriteria.

Element	Package and Description
pub.document:XMLValuesToDocument	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>Object An array of bytes (byte[]) to convert to a document.</p> <ul style="list-style-type: none"> ■ If <i>documentBytes</i> is null, the service does not return a document or an error message. ■ If <i>documentBytes</i> is not a byte array, the service throws a service exception. ■ If <i>documentBytes</i> is zero-length, the service produces an empty document.
----------------------	---

Output Parameters

<i>document</i>	Document 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

documents **Document List** Set of documents that contain the documents you want to delete.

indices **String List** Index values of documents to be deleted from the *documents* parameter document list.

Output Parameters

documents **Document List** List of documents whose indices do *not* match the values in *indices* parameter.

deletedDocuments **Document List** 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

documentList **Document List** 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.

name **String** Name of the element in the *documentList* 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>	<i>cx_timeout</i>

<i>pValue</i>	1000
---------------	------

Key	Value
<i>pName</i>	<i>cx_max</i>

<i>pValue</i>	2500
---------------	------

Key	Value
<i>pName</i>	<i>cx_min</i>

<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>	<p>Document Document (IData object) to convert to bytes.</p> <ul style="list-style-type: none"> ■ If <i>document</i> is null, the service does not return an output or an error message. ■ If <i>document</i> is not a document (IData), the service throws a service exception. ■ If <i>document</i> contains no elements, the service produces a zero-length byte array.
-----------------	--

Output Parameters

<i>documentBytes</i>	<p>Object A serialized representation of the document as an array of bytes (byte[]).</p>
----------------------	---

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).

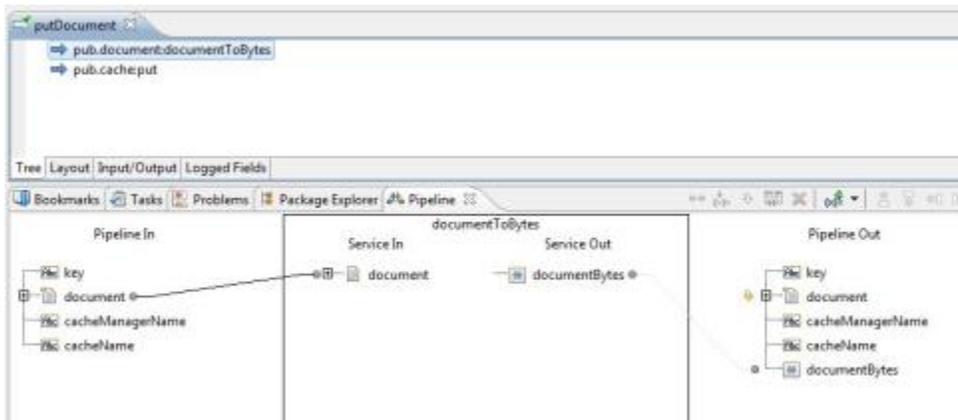
In order for the document-to-bytes-to-document conversion to work, the entire content of the document must be serializable. Every object in the document must be of a data type known to Integration Server, or it must support the `java.io.Serializable` interface. If Integration Server encounters an unknown 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.

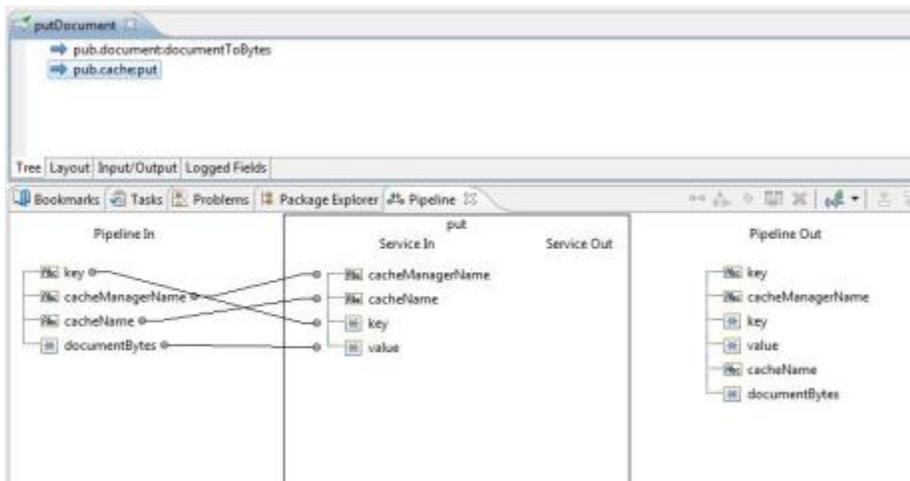
Example

This example describes how to use the [pub.document:documentToBytes](#) and [pub.document:bytesToDocument](#) services to cache a document in the pipeline.

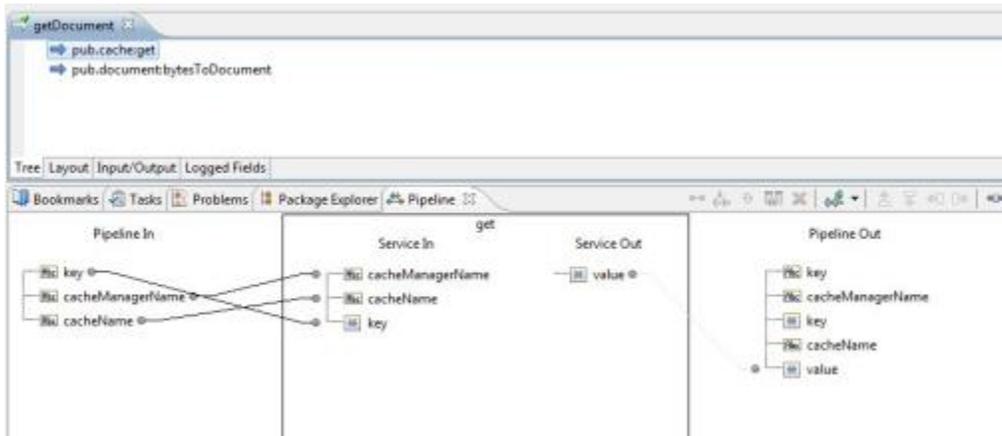
1. A document is not directly serializable, so you must first use the [pub.document:documentToBytes](#) service. Invoke the service and map the document to the `document` input parameter. This will add the output `documentBytes` to the pipeline.



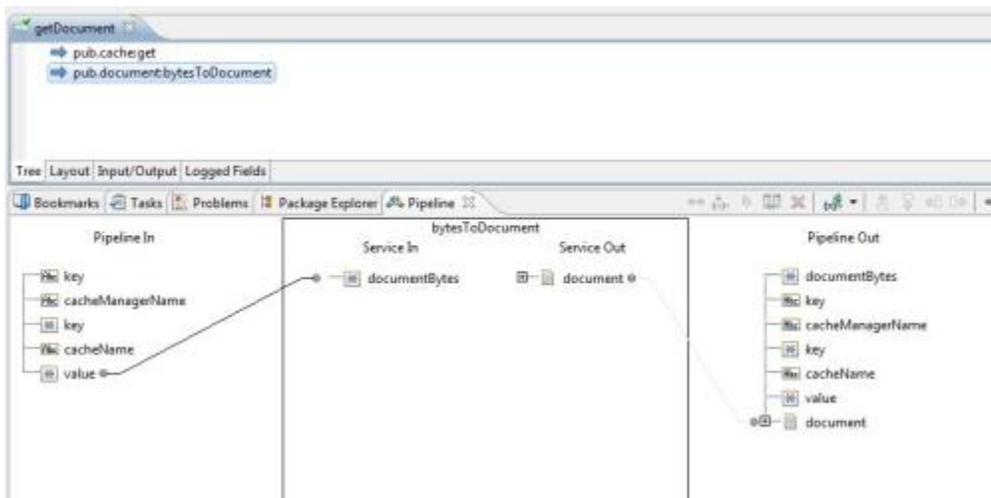
2. Invoke [pub.cache:put](#) and map `documentBytes` to the `value` input parameter for that service. Set the `key` input parameter to a value that is meaningful to your application.



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

document **Document** Document (IData object) to transform.

<i>name</i>	String 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 <code>pName</code> .)
<i>value</i>	String 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 <code>pValue</code> .)

Output Parameters

<i>documentList</i>	Document List 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:

Key	Value
<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:

Key	Value
<i>pName</i>	<i>cx_timeout</i>
<i>pValue</i>	1000

Key	Value
<i>pName</i>	<i>cx_max</i>

Key	Value
<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

documents **Document List** Set of documents to be grouped based on the specified criteria.

- groupCriteria* **Document List** The criteria on which the input documents are to be grouped. Valid values for the *groupCriteria* parameter are:
- *key* . Key in the pipeline. 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.
 - *compareStringsAs* . Optional. Valid values for *compareStringsAs* are *string*, *numeric*, and *datetime*. The default value is *string*.
 - *pattern* . Optional. *pattern* will be considered only if the *compareStringsAs* parameter is of type *datetime*. For information about using patterns, see ["Time Zones" on page 177](#).

Note: If *key* is not found in all the input documents, the documents that do not match the *groupCriteria* are grouped together as a single group.

Output Parameters

- documentGroups* **Document List** 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:

<i>key</i>	<i>compareStringsAs</i>	<i>pattern</i>
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>	Document List Set of documents in which a new document is to be inserted.
<i>insertDocument</i>	Document The new document to be inserted to the set of documents specified in the <i>documents</i> parameter.
<i>index</i>	<p>String Optional. The position in the set which the document is to be inserted.</p> <p>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.</p>

Output Parameters

<i>documents</i>	Document List 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>	Document List Set of documents from which the documents meeting the search criteria are to be returned.
<i>searchCriteria</i>	<p>Document Criteria on which the documents in the <i>documents</i> parameter are to be searched.</p> <p>Valid values for <i>searchCriteria</i> parameters are:</p> <ul style="list-style-type: none"> ■ <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. ■ <i>value</i>. Optional. Any search text. If no value is specified, the service searches for null in the document list. ■ <i>compareStringsAs</i>. Optional. Allowed values are <code>string</code>, <code>numeric</code>, and <code>datetime</code>. The default value is <code>string</code>.

- *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 177](#).

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>	Document List List of documents which are matching the search criteria.
<i>documentListIndices</i>	String List Positions of search documents in the document list.
<i>documents</i>	Document List 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:

key	value	compareStringsAs	pattern
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>	Document List Set of documents that are to be sorted.
<i>sortCriteria</i>	<p>Document List Criteria based on which the documents in the <i>documents</i> parameter are to be sorted.</p> <p>Valid values for <i>sortCriteria</i> parameters are:</p> <ul style="list-style-type: none"> ■ <i>key</i>. Name of the element in <code>documentList</code> 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 177](#).

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](#).

7 Event Folder

- Summary of Elements in this Folder 238

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
pub.event:addSubscriber	WmPublic. Creates a subscription for a specified event.
pub.event:alarm	WmPublic. Specification for alarm event handlers.
pub.event:alarmInfo	WmPublic. Document type for alarm event information.
pub.event:audit	WmPublic. Specification for audit event handlers.
pub.event:auditError	WmPublic. Specification for audit event errors.
pub.event:auditErrorInfo	WmPublic. Document type for audit error event information.
pub.event:auditInfo	WmPublic. Document type for audit event information.
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.
pub.event:deleteSubscriber	WmPublic. Removes an event handler from the subscription list for a specified event.
pub.event.eda:event	WmPublic. Document type that defines the structure of an event document.
pub.event.eda:eventToDocument	WmPublic. <i>Deprecated</i> - Replaced by pub.event.nerv:eventToDocument Converts an

Element	Package and Description
	EDA event in the form of an XML string to a document (IData object).
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.
pub.event:errorInfo	WmPublic. Document type for error event information.
pub.event:exception	WmPublic. Specification for exception event handlers.
pub.event:exceptionInfo	WmPublic. Document type for exception information.
pub.event:gdEnd	WmPublic. Specification for gdEnd event handlers.
pub.event:gdEndInfo	WmPublic. Document type for gdEnd event information.
pub.event:gdStart	WmPublic. Specification for gdStart event handlers.
pub.event:gdStartInfo	WmPublic. Document type for gdStart event information.
pub.event:getEventTypes	WmPublic. Returns the list of supported event types on Integration Server.
pub.event:getSubscribers	WmPublic. Returns the list of subscribers for a specified event type.
pub.event:jmsReceiveErrorEvent	WmPublic. Specification for a JMS retrieval failure event handler.
pub.event:jmsSendErrorEvent	WmPublic. Specification for the JMS delivery failure event handler.

Element	Package and Description
pub.event:journal	WmPublic. Specification for journal event handlers.
pub.event:journalInfo	WmPublic. Document type for journal event information
pub.event:modifySubscriber	WmPublic. Modifies the information about a subscription.
pub.event.nerv:eventToDocument	WmPublic. Converts an incoming JMS message into an IS document (pub.event.eda:event).
pub.event.nerv:send	WmPublic. Sends an EDA event to the Network for Event Routing and Variation (NERV).
pub.event:portStatus	WmPublic. Specification for a port status event.
pub.event:portStatusInfo	WmPublic. Document type for port event information.
pub.event:reloadEventManagerSettings	WmPublic. Reloads the settings from the event manager's configuration file (eventcfg.bin) on the server.
pub.event:replication	WmPublic. Specification for replication event handlers.
pub.event:replicationInfo	WmPublic. Document type for replication event information.
pub.event:saveEventManagerSettings	WmPublic. Saves the current subscriber information to the event manager's configuration file (eventcfg.bin) on the server.
pub.event:security	WmPublic. Specification for security event handlers.
pub.event:securityInfo	WmPublic. Document type for security event information.

Element	Package and Description
pub.event:sessionEnd	WmPublic. Specification for sessionEnd event handlers.
pub.event:sessionEndInfo	WmPublic. Document type for sessionEnd event information.
pub.event:sessionExpire	WmPublic. Specification for sessionExpire event handlers.
pub.event:sessionExpireInfo	WmPublic. Document type for sessionExpire event information.
pub.event:sessionStart	WmPublic. Specification for sessionStart event handlers.
pub.event:sessionStartInfo	WmPublic. Document type for sessionStart event information.
pub.event:stat	WmPublic. Specification for stat event handlers.
pub.event:statInfo	WmPublic. Document type for stat event information.
pub.event:txEnd	WmPublic. Specification for txEnd event handlers.
pub.event:txEndInfo	WmPublic. Document type for txEnd event information.
pub.event:txStart	WmPublic. Specification for txStart event handlers.
pub.event:txStartInfo	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:

This pattern string...	Would match...
*	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.

EventType	Filtered attribute
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> <p><code>*Administrators*</code></p> <p>To select session starts for the user ID "LRMalley" in the Administrators group, the filter would be:</p> <p><code>*Administrators*LRMalley</code></p> <p>To select session starts for the user ID "LRMalley" in any group, the filter would be:</p> <p><code>*LRMalley</code></p>
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>	String 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>	String Descriptive comment for this subscription. This comment is displayed when subscriptions are viewed with Designer.

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 when it is added.

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

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.

sessionID **String** Session ID of the service firing the alarm.

msg **String** Text describing the alarm.

pub.event:audit

WmPublic. Specification for audit event handlers.

Input 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.

<i>TID</i>	String Server thread that generated the audit event.
<i>service</i>	String Fully qualified name of the service that generated the event.
<i>sessionID</i>	String Session ID of the service that generated the event.
<i>result</i>	String Description of the audit point. A value of: <ul style="list-style-type: none">■ <code>Started</code> indicates that this event marks the beginning of a service.■ <code>Ended</code> indicates that this event marks the end of a service that executed successfully.■ <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.■ <code>Retried</code> indicates that this event is created each time a service is retried.
<i>pipeline</i>	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.
<i>userName</i>	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

time **String** Date and time at which the audit error occurred. This parameter is in the format *yyyy-MM-dd HH:mm:ss.SSS* (for example, "2004-10-28 14:46:39.505").

destination **String** 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>
ServiceDBDest	Service audit logger
ErrorDBDest	Error audit logger
SessionDBDest	Session audit logger
ProcessAuditDBDest	Process audit logger
CoreAuditDBDest	All other audit loggers

message **String** Error message of the exception.

stackTrace **String** Stack trace for the exception.

errorCode **String** 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>	String 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>	String 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>
	ServiceDBDest	Service audit logger
	ErrorDBDest	Error audit logger
	SessionDBDest	Session audit logger
	ProcessAuditDBDest	Process audit logger
	CoreAuditDBDest	All other audit loggers
<i>message</i>	String	Error message of the exception.
<i>stackTrace</i>	String	Stack trace for the exception.
<i>errorCode</i>	String	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.

<i>service</i>	String Fully qualified name of the service that generated the event.
<i>sessionID</i>	String Session ID of the service that generated the event.
<i>result</i>	String Description of the audit point. <ul style="list-style-type: none"> ■ <code>Started</code> indicates that this event marks the beginning of a service. ■ <code>Ended</code> indicates that this event marks the end of a service that executed successfully. ■ <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. ■ <code>Retried</code> indicates that this event is created each time a service is retried.
<i>pipeline</i>	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.
<i>userName</i>	String User ID that invoked the service that generated the event.

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>	String Fully qualified name of the last service (that is, most recently called) on the call stack.
<i>flowStep</i>	String 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 *flowStep* value that would be assigned to them if the exception occurred at that step:

Step	<i>flowStep</i> 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

EventType **String** 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>	String Start date and time of the event, in the format <code>yyyy-MM-dd'T'HH:mm:ss.SSSZ</code> .
<i>evt:End</i>	String 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>	String 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>	String 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, {http://namespaces.softwareag.com/EDA/WebM/Process/1.0}ProcessInstanceChange is the event type identifier that reports changes to a process instance.
<i>evt:Version</i>	String 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>	String Optional. Unique identifier used to associate an event with other events.
<i>evt:EventID</i>	String 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>	String Optional. The priority of the event to the producer. The value can be <code>Normal</code> or <code>High</code> .
<i>evt:ProducerID</i>	String 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>	String Optional. The identifier of the user who emitted the event.
<i>evt:CustomHeaders</i>	Document 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* - Replaced by [pub.event.nerv:eventToDocument](#). Converts an EDA event in the form of an XML string to a document instance in the form of [pub.event.eda:event](#).

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>	String Stack trace for the error.
<i>errorMessage</i>	String The error message, if any, generated by the error that causes the error event.
<i>serviceName</i>	String 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>	String Stack trace for the error.
<i>errorMessage</i>	String The error message, if any, generated by the error that causes the error event.
<i>serviceName</i>	String 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>	String 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>	String Optional. Error message of the exception.
<i>localizedError</i>	String Optional. Error message in the language that corresponds to the locale of your webMethods installation.
<i>errorType</i>	String Exception type that was thrown.
<i>errorDump</i>	String More detailed information about the exception.

<i>service</i>	String Optional. Fully qualified name of the service that generated the event.
<i>user</i>	String User that requested the service that generated the event.
<i>callStack</i>	Document List 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>	Document Optional. State of the pipeline at the time the exception occurred.
<i>threadID</i>	String Thread ID identifying the thread that invoked the service.
<i>ssnid</i>	String Session ID during which the exception occurred.
<i>errorMsgID</i>	String Optional. The identification number for the error message.
<i>errorDetails</i>	Document 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>	Document Optional. Nested errors and exceptions, if any. The structure of this document is defined by pub.event:exceptionInfo .

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>	String Date and time that the event occurred, in the format <code>yyyy/MM/dd HH:mm:ss.SS</code> .
<i>error</i>	String Optional. Error message of the exception.
<i>localizedError</i>	String Optional. Error message in the language that corresponds to the locale of your webMethods installation.
<i>errorType</i>	String Exception type that was thrown.
<i>errorDump</i>	String More detailed information about the exception.
<i>service</i>	String Optional. Fully qualified name of the service that generated the event.
<i>user</i>	String User that requested the service that generated the event.
<i>callStack</i>	Document List 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>	Document Optional. State of the pipeline at the time the exception occurred.

<i>threadID</i>	String Thread ID identifying the thread that invoked the service.
<i>ssnid</i>	String Session ID during which the exception occurred.
<i>errorMsgID</i>	String Optional. The identification number for the error message.
<i>errorDetails</i>	Document 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>	Document Optional. Nested errors and exceptions, if any. The structure of this document is defined by pub.event.exceptionInfo .

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>	String Date and time that the event occurred, in the format <code>yyyy/MM/dd HH:mm:ss.SS</code> .
<i>TID</i>	String Transaction ID of the service that generated the event.
<i>result</i>	String 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>	String Date and time that the event occurred, in the format <i>yyyy/MM/dd HH:mm:ss.SS</i> .
<i>TID</i>	String Transaction ID of the service that generated the event.
<i>result</i>	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:gdStart

WmPublic. Specification for `gdStart` event handlers.

Input Parameters

<i>time</i>	String Date and time that the event occurred, in the format <i>yyyy/MM/dd HH:mm:ss.SS</i> .
-------------	--

<i>TID</i>	String Transaction ID of the service that generated the event.
<i>svcname</i>	String Fully qualified name of the service that generated the event.
<i>result</i>	String 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>	String Date and time that the event occurred, in the format <code>yyyy/MM/dd HH:mm:ss.SS</code> .
<i>TID</i>	String Transaction ID of the service that generated the event.
<i>svcname</i>	String Fully qualified name of the service that generated the event.
<i>result</i>	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:

Key	Description
<i>Service</i>	String Fully qualified name of the event-handler service (that is, the service that subscribes to the event in <i>EventType</i>).
<i>Filter</i>	String 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:

- * would math any string.
- M* would math any string that starts with an uppercase "M."
- M*X would math any string that starts with an uppercase "M" and ends with an uppercase "X."

For a list of attributes to which the filter is applied, see [pub.event:addSubscriber](#).

<i>Comment</i>	String 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>	String Subscriber ID.
<i>Enabled</i>	String Flag indicating the status of the subscription. Will be one of the following values. A value of: <ul style="list-style-type: none"> ■ <code>true</code> indicates that the subscription is active. ■ <code>false</code> indicates that the subscription is inactive.

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>	String Specifies the name of the JMS trigger that executed the trigger service for which the JMS retrieval failure event occurred.
<i>triggerDestinationIndex</i>	java.lang.Integer 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>	java.lang.Integer Number of times the JMS provider delivered the message to the JMS trigger at the time the event occurred.
<i>time</i>	String Date and time that the event occurred, in the format <i>yyyy/MM/dd HH:mm:ss.SS</i>
<i>exceptionClass</i>	String Name of the class that caused the failure. This may be useful to determine programmatically why the error occurred.
<i>exceptionMessage</i>	String Message contained in the exception.
<i>data</i>	Document 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 pub.jms:JMSMessage 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>	String Name of the JMS connection alias used to send the message to the JMS provider.
<i>time</i>	String Date and time that the event occurred, in the format <i>yyyy/MM/dd HH:mm:ss.SS</i> .
<i>data</i>	Document 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>	String Date and time that the event occurred, in the format yyyy/mm/dd hh:mm:ss.ss.
<i>productID</i>	String Name of the product that generated the journal log message and the event.
<i>majorCode</i>	String Major code of the message.
<i>minorCode</i>	String Minor code of the message.
<i>severity</i>	String Number indicating the severity of the message. <ul style="list-style-type: none"> ■ 1 Fatal ■ 2 Error ■ 3 Warning ■ 4 Info ■ 5, 6 Debut ■ 7 - 10 Trace

defaultMessage **String** 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 WmIExtDC 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 WmIExtDC 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

time **String** Date and time that the event occurred, in the format yyyy/mm/dd hh:mm:ss.ss.

<i>productID</i>	String Name of the product that generated the journal log message and the event.
<i>majorCode</i>	String Major code of the message.
<i>minorCode</i>	String Minor code of the message.
<i>severity</i>	String Number indicating the severity of the message. <ul style="list-style-type: none">■ 1 Fatal■ 2 Error■ 3 Warning■ 4 Info■ 5, 6 Debut■ 7 - 10 Trace
<i>defaultMessage</i>	String 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> <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>	String	Fully qualified name of the event-handler service that you want the subscription to specify.
<i>Comment</i>	String	Descriptive comment that you want to assign to the subscription.
<i>Enabled</i>	String	Flag specifying the status of the subscription. Must be one of the following values. Set to: <ul style="list-style-type: none"> ■ true to make the subscription active. ■ false to make the subscription inactive. This is the default. <p>Note: Although the default value is false, you will generally want to set <i>Enabled</i> to <code>true</code> to activate the subscription immediately after it is modified.</p>

Output Parameters

<i>Result</i>	String	Flag indicating whether the subscriber was successfully modified. A value of: <ul style="list-style-type: none"> ■ 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 <i>gID</i>).
---------------	---------------	--

See Also

[pub.event:addSubscriber](#)
[pub.event:deleteSubscriber](#)
[pub.event:getSubscribers](#)
[pub.event:saveEventManagerSettings](#)

pub.event.nerv:eventToDocument

WmPublic. Converts an incoming JMS message into an IS document (pub.event.eda:event).

Input Parameters

<i>JMSMessage</i>	Document A document reference to the IS document of type pub.jms:JMSMessage .
<i>documentTypeName</i>	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: <ul style="list-style-type: none">■ The order and dimensionality of elements.■ The prefix associated with each namespace in the instance document. <p>The document type specified in <i>documentTypeName</i> 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.</p> <p>If you do not specify <i>documentTypeName</i>, the structure of the event body is determined solely by the event document.</p>

Output Parameters

<i>evt:Event</i>	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 replaces pub.event.eda:eventToDocument which has been deprecated.

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. 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.

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

<i>header</i>	Document A document containing the information that you want to set in the event header.
---------------	---

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

<i>Start</i>	java.util.date 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>	java.util.date Optional. End date and time of the event.
<i>Kind</i>	String 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>	String The name of the event type for the event.
<i>Version</i>	String Optional. Version of the event type with which the event is compatible. An event should contain a version only if the event type supports versioning. Uses this regular expression for validation: <code>[0-9]+(\.[0-9]+)*</code>
<i>CorrelationID</i>	String Optional. Unique identifier used to associate this event with other events.
<i>EventID</i>	String Optional. The unique identifier of the event. This element can be supplied by the event producer or can be system generated.
<i>Priority</i>	String Optional. The priority of the event to the producer. The value can be <code>Normal</code> or <code>High</code> .

<i>ProducerID</i>	String 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>	String Optional. The identifier of the user who emitted the event.
<i>CustomHeaders</i>	Document List Optional. Any other name/value parameters that should be included in the header field of the message.
<i>body</i>	Document 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>endpointUris</i>	<p>String Optional. Specifies the endpoint URIs to which the event will be sent.</p> <p>If this parameter is specified, NERV sends the event to the specified endpoint URIs.</p> <p>If this parameter is not specified, NERV sends the event to the default endpoint configured for NERV.</p>
<i>encode</i>	<p>String Optional. Flag indicating whether to encode the data. Set this parameter to true if your XML data contains special characters, including the following: < > & " ' .</p> <p>Set to:</p> <ul style="list-style-type: none"> ■ true to encode the data. <p>For example, the string expression 5 < 6 would be converted to <expr>5 &lt; 6</expr>, which is valid.</p> ■ false to not encode the data. This is the default. <p>For example, the string expression 5 < 6 would be converted to <expr>5 < 6</expr>, which is invalid.</p>

Output Parameters

None.

Usage Notes

The `pub.event.nerv:send` service replaces `pub.event.eda:send` which has been removed from Integration Server.

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.

This service replaces `pub.event.eda:send`, which is deprecated.

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.

Key	Description
<i>time</i>	String Date and time that the event occurred, in the format <i>yyyy/MM/dd HH:mm:ss.SS</i> .
<i>port</i>	String Number for the port.
<i>status</i>	String Status of the port.
<i>protocol</i>	String Type of port (for example, <i>http</i> , <i>https</i> , <i>ftp</i> , or <i>email</i>).
<i>primary</i>	String Primary port. By default, the webMethods Integration Server designates an HTTP port at port 5555 as the primary port.
<i>enabled</i>	String Flag indicating whether or not the port is enabled. Set to:

- true to indicate that the port is enabled.
- false to indicate that the port is disabled.

Output Parameters

None.

pub.event:portStatusInfo

WmPublic. Document type for port event information.

Parameters

<i>time</i>	String Date and time that the event occurred, in the format <i>yyyy/MM/dd HH:mm:ss.SS</i> .
<i>port</i>	String Number for the port.
<i>status</i>	String Status of the port.
<i>protocol</i>	String Type of port (for example, <i>http</i> , <i>https</i> , <i>ftp</i> , or <i>email</i>).
<i>primary</i>	String The primary port. By default, the webMethods Integration Server designates an HTTP port at port 5555 as the primary port.
<i>enabled</i>	String A flag indicating whether or not the port is enabled. A value of: <ul style="list-style-type: none"> ■ <code>true</code> indicates that the port is enabled. ■ <code>false</code> indicates that the port is disabled.

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>	String Date and time that the event occurred, in the format <i>yyyy/MM/dd HH:mm:ss.SS</i> .
<i>action</i>	String 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>	String Name of package being replicated.
<i>service</i>	String 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>	String Date and time that the event occurred, in the format <i>yyyy/MM/dd HH:mm:ss.SS</i> .
<i>action</i>	String 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>	String Name of package being replicated.
<i>service</i>	String 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: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>	String Date and time that the event occurred, in the format <i>yyyy/MM/dd HH:mm:ss.SS</i> .
<i>clientID</i>	String IP address of the host from which the request originated.
<i>serverID</i>	String IP address of the host on which Integration Server is running.
<i>userName</i>	String 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>	String Type of security event. Some examples are: <ul style="list-style-type: none">AuthenticationAuthorizationCertificatesConfigurationJDBCPoolsPackagesPasswordsPortsRemote ServersServicesSSL Web Services
<i>clientApplication</i>	User ID of the OAuth client application that initiated or performed the security event. 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.
<i>result</i>	String Flag indicating whether the security action completed successfully. Set to:

- `True` to indicate that the security event completed successfully.
- `False` to indicate that the security event ended because of failure.

message

String 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.

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>	String Date and time that the event occurred, in the format <code>yyyy/MM/dd HH:mm:ss.SS</code> .
<i>clientID</i>	String IP address of the host from which the request originated.
<i>serverID</i>	String IP address of the host on which Integration Server is running.
<i>userName</i>	String 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>	String Type of security event. Some examples are: <ul style="list-style-type: none">AuthenticationAuthorizationCertificatesConfigurationJDBC PoolsPackagesPasswordsPortsRemote ServersServicesSSL Web Services
<i>clientApplication</i>	String User ID of the OAuth client application that initiated or performed the security event. <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>	String Flag indicating whether the security action completed successfully. <ul style="list-style-type: none">■ <code>True</code> indicates that the security event completed successfully.■ <code>False</code> indicates that the security event ended because of failure.
<i>message</i>	String 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.

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>	String Date and time that the event occurred, in the format <i>yyyy/MM/dd HH:mm:ss.SS</i> .
<i>sessionID</i>	String Session ID of the service firing the alarm.
<i>rpcs</i>	String Number of service calls the session has performed.
<i>age</i>	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

<i>time</i>	String Date and time that the event occurred. Given in the format <i>yyyy/MM/dd HH:mm:ss.SS</i> .
<i>sessionID</i>	String Session ID of the service firing the alarm.

<i>rpcs</i>	String Number of service calls the session has performed.
<i>age</i>	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>	String Date and time that the event occurred. Given in the format <code>yyyy/MM/dd HH:mm:ss.SS</code> .
<i>sessionID</i>	String Session ID of the service firing the alarm.
<i>rpcs</i>	String Number of service calls the session has performed.
<i>age</i>	String 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>	String Date and time that the event occurred. Given in the format <i>yyyy/MM/dd HH:mm:ss.SS</i> .
<i>sessionID</i>	String Session ID of the service firing the alarm.
<i>rpcs</i>	String Number of service calls the session has performed.
<i>age</i>	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

<i>time</i>	String Date and time that the event occurred. Given in the format <i>yyyy/MM/dd HH:mm:ss.SS</i> .
<i>sessionID</i>	String ID of the new session.
<i>userid</i>	String User ID that the IS client or developer used to log on to the webMethods Integration Server.
<i>sessionName</i>	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>	String Date and time that the event occurred. Given in the format <i>yyyy/MM/dd HH:mm:ss.SS</i> .
<i>sessionID</i>	String ID of the new session.
<i>userid</i>	String User ID that the IS client or developer used to log on to the webMethods Integration Server.
<i>sessionName</i>	String 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>	String Date and time that the event occurred. Given in the format <i>yyyy/MM/dd HH:mm:ss.SS</i> .
<i>uptime</i>	String Amount of time the server has been up. Given in the format <i>yyyy/MM/dd HH:mm:ss.SS</i> .
<i>totalMem</i>	String 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>	String 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>	String Amount of storage used by the Integration Server, in kilobytes. For example, a value of 65535 represents 64 megabytes of storage.
<i>freeMemPer</i>	String Percent of total memory unused.
<i>usedMemPer</i>	String Percent of total memory used.
<i>svrT</i>	String Number of services currently running.
<i>svrTMax</i>	String Peak number of servers ever running concurrently.
<i>sysT</i>	String Number of JVM threads running.
<i>sysTMax</i>	String Peak number of threads ever running.
<i>conn</i>	String Number of current sessions.
<i>connMax</i>	String Peak number of concurrent sessions.
<i>reqTotal</i>	String Cumulative total number of services processed.

<i>reqAvg</i>	String Average duration of service.
<i>newReqPM</i>	String New requests per minute.
<i>endReqPM</i>	String End requests per minute.
<i>errSvc</i>	String Number of services completed in error state.
<i>svcRate</i>	String Number of end/start(s) per second.
<i>ssnUsed</i>	String Number of licensed sessions currently active.
<i>ssnPeak</i>	String Number of licensed sessions that have ever run concurrently on the server.
<i>ssnMax</i>	String Maximum number of sessions for which the server is licensed.
<i>errSys</i>	String 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>	String Date and time that the event occurred. Given in the format <code>yyyy/MM/dd HH:mm:ss.SS</code> .
------------------	--

<i>uptime</i>	String Amount of time the server has been up. Given in the format <i>yyyy/MM/dd HH:mm:ss.SS</i> .
<i>totalMem</i>	String 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>	String 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>	String Amount of storage used by the Integration Server, in kilobytes. For example, a value of 65535 represents 64 megabytes of storage.
<i>freeMemPer</i>	String Percent of total memory unused.
<i>usedMemPer</i>	String Percent of total memory used.
<i>svrT</i>	String Number of services currently running.
<i>svrTMax</i>	String Peak number of servers ever running concurrently.
<i>sysT</i>	String Number of JVM threads running.
<i>sysTMax</i>	String Peak number of threads ever running.
<i>conn</i>	String Number of current sessions.
<i>connMax</i>	String Peak number of concurrent sessions.
<i>reqTotal</i>	String Cumulative total number of services processed.
<i>reqAvg</i>	String Average duration of service.
<i>newReqPM</i>	String New requests per minute.
<i>endReqPM</i>	String End requests per minute.
<i>errSvc</i>	String Number of services completed in error state.
<i>svcRate</i>	String Number of end/start(s) per second.

<i>ssnUsed</i>	String Number of licensed sessions currently active.
<i>ssnPeak</i>	String Number of licensed sessions that have ever run concurrently on the server.
<i>ssnMax</i>	String Maximum number of sessions for which the server is licensed.
<i>errSys</i>	String 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>	String Date and time that the event occurred. Given in the format <code>yyyy/MM/dd HH:mm:ss.SS</code> .
<i>TID</i>	String Transaction ID of the service that generated the event.
<i>result</i>	String 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>	String Date and time that the event occurred. Given in the format <i>yyyy/MM/dd HH:mm:ss.SS</i> .
<i>TID</i>	String Transaction ID of the service that generated the event.
<i>result</i>	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).

pub.event:txStart

WmPublic. Specification for `txStart` event handlers.

Input Parameters

<i>time</i>	String Date and time that the event occurred. Given in the format <i>yyyy/MM/dd HH:mm:ss.SS</i> .
<i>TID</i>	String Transaction ID of the service that generated the event.
<i>result</i>	String 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

<i>time</i>	String Date and time that the event occurred. Given in the format <i>yyyy/MM/dd HH:mm:ss.SS</i> .
<i>TID</i>	String Transaction ID of the service that generated the event.
<i>result</i>	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).

8 File Folder

■ File Access Control Configuration for the pub.file Services	298
■ Parameter Settings	298
■ Summary of Elements in this Folder	300

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 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
pub.file:bytesToFile	WmPublic. Writes the specified byte array to a file.
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.
pub.file:copyFile	WmPublic. Copies a file from one directory to another.
pub.file:deleteFile	WmPublic. Deletes the specified file.
pub.file:getFile	WmPublic. Retrieves a specified file from the local file system.
pub.file:listFiles	WmPublic. List all the files in a specified directory.
pub.file:moveFile	WmPublic. Moves a file from one directory to another.
pub.file:readerToFile	WmPublic. Reads data from a java.io.Reader object and writes it to a file.
pub.file:streamToFile	WmPublic. Writes the data in the InputStream to a file.
pub.file:stringToFile	WmPublic. Writes text to a file.

pub.file:bytesToFile

WmPublic. Writes the specified byte array to a file.

Input Parameters

fileName **String** The absolute path name of the file to which to write the byte array.

<i>bytes</i>	Byte[] The byte array to write to the file specified in the <i>filename</i> parameter.
<i>append</i>	String 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">■ <code>true</code> to append to the file if the file already exists.■ <code>false</code> to overwrite the file if the file already exists.

Output Parameters

<i>length</i>	String 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` configuration file. If the input *fileName* is not on the allowed list, Integration Server throws an exception. For information about configuring the `FileAccessControl` configuration file, refer to ["File Access Control Configuration for the pub.file Services" on page 298](#).

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>	String The absolute path name of the file to be checked.
-----------------	---

Output Parameters

<i>exists</i>	String Indicates whether the specified <i>fileName</i> exists or not. A value of: <ul style="list-style-type: none">■ <code>true</code> indicates that the specified <i>fileName</i> exists.■ <code>false</code> indicates that the specified <i>fileName</i> does not exist.
---------------	---

isDirectory **String** Indicates whether the specified *fileName* is a file or a directory.

- `true` indicates that the specified *fileName* is a directory.
- `false` indicates that the specified *fileName* is a file.

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` configuration file. If the input *fileName* is not on the allowed list, Integration Server throws an exception. For information about configuring the `FileAccessControl` configuration file, refer to "[File Access Control Configuration for the pub.file Services](#)" on page 298.

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

fileName **String** The absolute path name of the file to be copied.

targetDirectory **String** Directory to which the file specified in the *fileName* parameter will be copied.

appendTimestamp **String** Optional. Specifies whether the current timestamp will be appended to the target filename. Set to:

- `true` to append a timestamp to the target *fileName* in the `yyyyMMddHHmmss` format.
- `false` to omit a timestamp from the target *fileName*.

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 configuration 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 configuration file, refer to "[File Access Control Configuration for the pub.file Services](#)" on page 298.

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 configuration file. If the input *fileName* is not on the allowed list, Integration Server throws an exception. For information about configuring the FileAccessControl configuration file, refer to "[File Access Control Configuration for the pub.file Services](#)" on page 298.

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

filename **String** The absolute path name of the file in the local file system (for example `c:\rubicon\document.xml`).

loadAs **String** Optional. Form in which you want the `getFile` service to make the contents of the file available to subsequent services. Set to:

- `bytes` 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, `pub.xml:queryXMLNode`). This is the default.
- `stream` 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, `pub.xml:getXMLNodeIterator`).
- `string` to return the file as a string.
- `reader` to return the file as a reader.

encoding **String** 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 get. If no value is specified or if the value is set to `autoDetect`, the service uses the default operating system encoding. If you specify an unsupported encoding, the system throws an exception.

bufferSize **String** Optional. Buffer size (in bytes) to use if you are loading an `InputStream` (that is, `loadAs=stream`). The default is 4096 bytes.

Output Parameters

body **Document** Document (IData object) containing the file as a `byte[]`, `InputStream`, `string`, or `reader`. The *body* parameter will contain one of the following keys, depending on how the *loadAs* parameter was set:

Key	Description
<i>bytes</i>	byte[] Conditional. Returns file contents in a byte array if the <i>loadAs</i> parameter is set to <code>bytes</code> .
<i>stream</i>	java.io.InputStream Conditional. Returns file contents as an <code>InputStream</code> if the <i>loadAs</i> parameter is set to <code>stream</code> .
<i>reader</i>	java.io.Reader Conditional. Returns file contents as a reader if the loadAs parameter is set to <code>reader</code> .
<i>string</i>	String Conditional. Returns file contents as a string if the <i>loadAs</i> parameter is set to <code>string</code> .

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.

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>	String[] Name of the directory that is to be queried.
<i>filter</i>	String Optional. Filter string for the file list (for example, <code>*.java</code> or <code>EDI*.tx*</code>).

Output Parameters

<i>filelist</i>	String 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>numFiles</i>	String 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` configuration file. If the specified directory is not on the allowed list, the service throws an exception. For information about configuring the `FileAccessControl` configuration file, refer to "[File Access Control Configuration for the pub.file Services](#)" on page 298.

pub.file:moveFile

WmPublic. Moves a file from one directory to another.

If a file with the same name as the *fileName* parameter exists in the target directory, the `pub.file:moveFile` service throws an error.

Input Parameters

<i>fileName</i>	String The absolute path name of the file to be moved.
<i>targetDirectory</i>	String Name of the directory to which the file specified in the <i>fileName</i> parameter is to be moved.
<i>appendTimeStamp</i>	String 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">■ <code>false</code> if you do not want to append a timestamp. This is the default.■ <code>true</code> if you want to append a timestamp in the following format: <code>yyyyMMddHHmmss</code>.

Output Parameters

<i>status</i>	String Status of the move. A value of: <ul style="list-style-type: none">■ <code>true</code> indicates that the move was successful.■ <code>false</code> indicates that the move failed.
<i>targetFileName</i>	String 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` configuration 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` configuration file, see ["File Access Control Configuration for the pub.file Services" on page 298](#).

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.

pub.file:readerToFile

WmPublic. Reads data from a `java.io.Reader` object and writes it to a file.

Input Parameters

<i>fileName</i>	String Fully qualified name of the file to which the data is to be written.
<i>reader</i>	java.io.Reader The reader object from which the data is read.
<i>append</i>	String 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: <ul style="list-style-type: none">■ <code>true</code> to append if the file already exists.■ <code>false</code> to overwrite if the file already exists.
<i>encoding</i>	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 <code>autoDetect</code> , 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` configuration file. If the input `fileName` is not on the allowed list, the service throws an exception. For information about configuring the `FileAccessControl` configuration file, refer to "[File Access Control Configuration for the pub.file Services](#)" on page 298.

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

<i>fileName</i>	String[] Fully qualified name of the file to be written to.
<i>stream</i>	java.io.InputStream The stream from which data is read.
<i>append</i>	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: <ul style="list-style-type: none">■ <code>true</code> to append if the file already exists.■ <code>false</code> 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` configuration file. If the input *fileName* is not on the allowed list, an exception is thrown. For information about configuring the `FileAccessControl` configuration file, refer to "[File Access Control Configuration for the pub.file Services](#)" on page 298.

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>	String[] Fully qualified name of the file to be written to.
-----------------	--

<i>data</i>	String Text to be written.
<i>append</i>	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: <ul style="list-style-type: none">■ <code>true</code> to append if the file already exists.■ <code>false</code> to overwrite if the file already exists.
<i>encoding</i>	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 <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` configuration file. If the input *fileName* is not on the allowed list, an exception is thrown. For information about configuring the `FileAccessControl` configuration file, refer to "[File Access Control Configuration for the pub.file Services](#)" on page 298.

9 Flat File Folder

■ Summary of Elements in the Flat File Folder	312
---	-----

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
pub.flatFile:convertToString	WmFlatFile. Converts an IS document (IData object) to a flat file document based on the flat file schema that you specify.
pub.flatFile:convertToValues	WmFlatFile. Converts a flat file document to an IS document (IData object) based on the input flat file schema.
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.
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.
pub.flatFile.generate:createDocumentType	WmFlatFile. Creates an IS document type that defines the XML representation of a flat file schema.
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

Element	Package and Description
	already exists when the service is invoked.
pub.flatFile.generate:deleteFFDictionary	WmFlatFile. Deletes a flat file dictionary.
pub.flatFile.generate:deleteFFDictionaryEntry	WmFlatFile. Deletes a single entry from a flat file dictionary.
pub.flatFile.generate:deleteFFSchema	WmFlatFile. Deletes a flat file schema.
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 <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.
pub.flatFile.generate:FFSchema	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.
pub.flatFile.generate:findDependants	WmFlatFile. Returns the names of all flat file schemas and dictionaries that are dependent on a given flat file dictionary.
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.
pub.flatFile.generate:getFFDictionaryAsXML	WmFlatFile. Returns a dictionary as an XML string.

Element	Package and Description
pub.flatFile.generate:getFFDictionaryEntryAsXML	WmFlatFile. Returns a single dictionary entry as an XML string.
pub.flatFile.generate:getFFSchemaAsXML	WmFlatFile. Returns the specified flat file schema as an XML string.
pub.flatFile.generate:listFFDictionaryEntries	WmFlatFile. Lists all entries in a specified flat file dictionary that are of a specified type.
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.
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.
pub.flatFile.generate:updateFFDictionaryEntryFromXML	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.

originalError **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.

Value	Description
<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)

Value	Description
<i>record</i>	String 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>	String Character to use to separate fields.
<i>subfield</i>	String Character to use to separate subfields.

<i>release</i>	String 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>	String 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>	Document 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>	String 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>	String The type of encoding used to write data to the output file. The default encoding is UTF-8.
<i>sortInput</i>	String 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>	<p>You want the service to sort the input records to match the flat file schema.</p> <p>If you select to sort the input records, note that:</p> <ul style="list-style-type: none"> ■ The service will run slower. ■ All undefined records will be sorted <i>after</i> 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.

	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.
<i>returnAsBytes</i>	Value	Description
	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>	String Data that represents the flat file document.
<i>bytes</i>	Object 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>	Object 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>	Object The flat file input with type of String, InputStream, or ByteArray.
<i>ffSchema</i>	String The full name of the flat file schema object used to parse the <i>ffData</i> object.
<i>ffIterator</i>	Object 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>	String Optional. The encoding of the InputStream passed in to <i>ffData</i> . The default encoding is UTF-8.
<i>delimiters</i>	<p>Document 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"> ■ One character or character representation (for example, <code>*</code>, <code>\n</code> for line terminator, <code>\t</code> for tab) ■ Hexidecimal value with prefix "0X" (for example, <code>0X09</code>, <code>0X13</code>) ■ Octal value with prefix "0" or decimal value (for example, <code>011</code>, <code>023</code>) ■ Unicode characters (for example, <code>\uXXXX</code> where <code>XXXX</code> 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>	String 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>	String Character used to separate fields.
<i>subfield</i>	String Character used to separate subfields.
<i>release</i>	String 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>	<p>String 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.</p> <p>For example, if * is a delimiter, the field value is a*b, and the quoted release character is ", the string appears as "a*b".</p>
<i>FormatInfo</i>	Document 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>	String Optional. Whether you want to process the input all at one time.

<u>Value</u>	<u>Description</u>
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.

Value	Description
true	No IS document (IData object) will be created if all the fields are null. This is the default.
false	Always create IS document even though all the fields are null.

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.

Value	Description
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.

Value	Description
true	The parsed <i>ffData</i> will be returned in the output <i>ffValues</i> . This is the default.
false	<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.

validate **String** Optional. Whether to return error messages that describe how *ffData* differs from the flat file schema.

Value	Description
-------	-------------

	true	Do not return error messages describing how <i>ffData</i> differs from the specified flat file schema. This is the default.
	false	Return errors describing how the given <i>ffData</i> violates the constraints described in the flat file schema.
<i>returnErrors</i>	String Optional. Whether to return the validation errors. Validation errors are returned only if <i>validate</i> is set to true.	

<u>Value</u>	<u>Description</u>
asArray	Return any validation errors with the <i>ffData</i> in an array called <i>errors</i> . This is the default.
inResults	Return validation errors in the <i>ffValues</i> object.
both	Return validation errors in both <i>errors</i> and <i>ffValues</i> .

maxErrors **String** 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 *validate* is set to true.

flags **String** Optional. Flags that you can set to govern *convertToValues* options.

<u>Variable</u>	<u>Description</u>
<i>addRecordCount</i>	String Whether you want the service to add an additional field (<i>@record-count</i>) to each parsed record in the resulting <i>IData</i> object (<i>ffValues</i>). The <i>@record-count</i> field is used to identify the record number of each parsed record.
<u>Value</u>	<u>Description</u>
true	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 `@record-count` field for a record is 2 and that record contains 5 undefined records, the `@record-count` field for the next defined record will be 8.

`false` The `@record-count` 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`.

Value	Description
<code>true</code>	When a conditional validation error occurs, the output <code>errors</code> 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.

Value	Description
<code>true</code>	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>
<code>none</code>	The service will not delete any blank spaces from fields. This is the default.
<code>left</code>	The service will delete all blank spaces at the beginning of all fields.
<code>right</code>	The service will delete all blank spaces at the end of all fields.
<code>both</code>	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>
<code>false</code>	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.
<code>true</code>	The service returns the <i>ffValues</i> output parameter as an <code>IData</code> object and not as an <code>IData[]</code> .

Output Variables

<i>ffValues</i>	Document The IData object that represents the input flat file data.						
<i>ffIterator</i>	Object 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>	String Whether flat file contains validation errors.						
	<table> <thead> <tr> <th><u>Value</u></th> <th><u>Description</u></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>	<u>Value</u>	<u>Description</u>	<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> .
<u>Value</u>	<u>Description</u>						
<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>	String 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 `recWithNoIDLlike46` 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:

<u>Value</u>	<u>Description</u>
<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 <code>false</code> when the value of the <i>direction</i> parameter is <code>convertToString</code> .

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.

Value	Description
true	Indicates that the value could be properly formatted.
false	Indicates that the value could not be properly formatted.

<i>errorMessage</i>	String If <i>meetsFormat</i> is <code>false</code> , this parameter provides a text message describing the formatting error.
<i>valueToValidate</i>	String 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 <code>true</code> .
<i>maxLength</i>	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 (<code>-1</code>) or there is no Length Validator, this parameter will not be present in the pipeline.
<i>FormatInfo</i>	<p>Document Information that can be used by individual formatting services. This information can be obtained from one of 3 locations:</p> <ul style="list-style-type: none"> ■ <code>convertToString</code> – You can specify <i>FormatInfo</i> in addition to the delimiter information for a call to this service. ■ <code>convertToValues</code> – If delimiter information is explicitly passed into the <code>convertToValues</code> service, <i>FormatInfo</i> can be specified. ■ From the UNEDIFACT UNA segment – The EDI document type automatically extracts the decimal separator from the UNA segment. <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 <code>'</code>, the number would be formatted as 100,10 (European format) instead of 100.10, as is common in the US.</p> <p>Note: Changes to the data in this object will be reflected in all other format services that are invoked during execution of <code>convertToString</code> and <code>convertToValues</code>.</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.

`false` The flat file dictionary was *not* 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`.

Value	Description
<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:

RecordDictionaryEntry **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.

Variable	Description
<i>EntryName</i>	String The name of the record.
<i>RecordDefinition</i>	Document 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> .

CompositeDictionaryEntry **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.

Variable	Description
<i>EntryName</i>	String The name of the composite.
<i>CompositeDefinition</i>	Document 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 *webMethods Service Development Help*.

*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>	String The name of the field.
<i>FieldDefinition</i>	Document 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>	Document 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 Flat File Definition tab in the Flat File Schema Editor. For a description of the fields, see <i>webMethods Service Development Help</i> .
<i>Document Structure</i>	Document 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 Flat File Structure 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>
<i>Ordered</i>	String Whether the child records appear in the flat file in the order they are defined in the flat file schema.
<i>RecordStructure</i>	Document List Definitions of the records within the flat file.
<u>Variable</u>	<u>Description</u>

<i>Ordered</i>	String Whether the child records appear in the flat file in the order they are defined in the flat file schema.
<i>RecordUsage</i>	Document Information about how the record is used, including either the dictionary reference for this record or the definition of the record.
<i>RecordStructure</i>	Document List Child records of this record. This is a recursive reference to the <i>RecordStructure</i> defined in <i>FFSchema/DocumentStructure</i> .
<i>RecordParser</i>	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 <i>FixedLengthParser</i> , <i>DelimitedParser</i> , <i>VariableLengthParser</i> , or <i>EDIParser</i> . For <i>DelimitedParser</i> , <i>VariableLengthParser</i> , and <i>EDIParser</i> , you do not need to specify a value; just have the variable in the pipeline.
<i>DefaultRecord Reference</i>	Document 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>Document 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"> ■ Use <i>NthFieldIdentifier</i> to identify the field in the record (counting from zero) that contains the identifier. ■ Use <i>FixedPositionIdentifier</i> to identify the character position in the record (counting from zero) where the record identifier is located.
<i>UndefinedData Allowed</i>	<p>String Whether you want the <code>pub.flatFile:convertToValues</code> 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"> ■ Specify <code>true</code> if you want to allow undefined data and do not want the <code>pub.flatFile:convertToValues</code> service to flag undefined data errors. ■ Specify <code>false</code> if you do not want to allow undefined data and you do want the <code>pub.flatFile:convertToValues</code> service to flag undefined data errors.
<i>Document Areas</i>	<p>String List Areas for this flat file schema. An area is a way to associate an arbitrary string with a given record.</p>
<i>FloatingRecord</i>	<p>String Optional. The name of the record that is defined in the schema as a floating record.</p> <p>Note: If the floating record has an alternate name, specify the alternate name.</p>
<i>Description</i>	<p>String Description of the flat file schema.</p>

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>	String The name of the package that contains the dependent object.
<i>name</i>	String 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.

<u>Variable</u>	<u>Description</u>
-----------------	--------------------

<i>packageName</i>	String The name of the package that contains the referenced object.
--------------------	--

<i>name</i>	String 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

<i>saved</i>	String Whether the flat file dictionary was saved successfully. It will have one of the following values.						
	<table> <thead> <tr> <th><u>Value</u></th> <th><u>Description</u></th> </tr> </thead> <tbody> <tr> <td>true</td> <td>The flat file dictionary was successfully saved.</td> </tr> <tr> <td>false</td> <td>The flat file dictionary was <i>not</i> successfully saved.</td> </tr> </tbody> </table>	<u>Value</u>	<u>Description</u>	true	The flat file dictionary was successfully saved.	false	The flat file dictionary was <i>not</i> successfully saved.
<u>Value</u>	<u>Description</u>						
true	The flat file dictionary was successfully saved.						
false	The flat file dictionary was <i>not</i> successfully saved.						
<i>Errors</i>	String List Optional. Errors that occurred while attempting to save the flat file dictionary to the Integration Server namespace.						
<i>Warnings</i>	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

- FFSchemaName* **String** The fully-qualified name of the flat file schema 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 schema.
- FFXML* **String** 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 [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 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.

Value	Description
true	The flat file schema was successfully saved.
false	The flat file schema was <i>not</i> successfully saved.

<i>Errors</i>	String List Optional. Errors that occurred while attempting to save the flat file schema to the Integration Server namespace.
<i>Warnings</i>	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

10 Flow Folder

- Summary of Elements in this Folder 348

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
pub.flow:clearPipeline	WmPublic. Removes all fields from the pipeline. You may optionally specify fields that should not be cleared by this service.
pub.flow:debugLog	WmPublic. Writes a message to the server log.
pub.flow:getLastError	WmPublic. Obtains detailed information about the last exception that was trapped within a flow.
pub.flow:getRetryCount	WmPublic. Retrieves the retry count and the maximum retry count for a service.
pub.flow:getSession	WmPublic. Inserts the Session object into the pipeline as a document named <i>\$session</i> .
pub.flow:getTransportInfo	WmPublic. Retrieves information about the protocol from which the current service was invoked.
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 .
pub.flow:restorePipeline	WmPublic. Restores a pipeline previously saved by pub.flow:savePipeline .
pub.flow:restorePipelineFromFile	WmPublic. Restores a pipeline that was previously saved to a file.
pub.flow:savePipeline	WmPublic. Saves a pipeline into memory, for later retrieval with pub.flow:restorePipeline .

Element	Package and Description
pub.flow:savePipelineToFile	WmPublic. Saves the current pipeline to a file on the machine running webMethods Integration Server.
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.
pub.flow:setResponse	WmPublic. <i>Deprecated</i> - Replaced by pub.flow:setResponse2 .
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).
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).
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.
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.
pub.flow:throwExceptionForRetry	WmPublic. Throws an <code>ISRuntimeException</code> and instructs the Integration Server to re-execute a service using the original service input.
pub.flow:tracePipeline	WmPublic. Writes the names and values of all fields in the pipeline to the server log.
pub.flow:transportInfo	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: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

retryCount **String** The number of times the Integration Server has re-executed the service.

maxRetryCount **String** 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

ifcname **String** The flow interface name of the service to be invoked. For example, `pub.math`.

svcname **String** The name of the service to invoke. For example, `addInts`.

pipeline **Document** Optional. The name of the pipeline for the `pub.flow:invokeService` 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:restorePipeline

WmPublic. Restores a pipeline previously saved by [pub.flow:savePipeline](#).

Input Parameters

<i>\$name</i>	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.
<i>\$merge</i>	<p>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:</p> <ul style="list-style-type: none"> ■ <code>false</code> to clear the existing pipeline before restoring the saved pipeline. This is the default. ■ <code>true</code> 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.
<i>\$remove</i>	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 XOPObjct 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. XOPObjct fields are Objects that use the com.wm.util.XOPObjct 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 XOPObjct 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 webMethods Integration Server.

Input Parameters

fileName **String** Relative path to a file on webMethods Integration Server in which to save the contents of the pipeline. If the file does not exist, the service creates it. If the file already exists, the service overwrites it.

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... [pub.flow:savePipelineToFile](#) supports...

Java data types	<ul style="list-style-type: none"> ■ <code>byte[]</code> ■ <code>Date</code> ■ <code>GregorianCalendar</code>
-----------------	--

For...	pub.flow:savePipelineToFile supports...
	<ul style="list-style-type: none"> ■ IData ■ IData[] (IData list) ■ String ■ String[] (String list) ■ String[][] (String table) ■ Vector
Java wrapper classes	<ul style="list-style-type: none"> ■ Boolean ■ Byte ■ Character ■ Double ■ Float ■ Integer ■ Long ■ Short ■ Single dimension arrays of any of the above
webMethods classes	<ul style="list-style-type: none"> ■ MBoolean ■ MByte ■ MDouble ■ MFloat ■ MInteger ■ MLong ■ MShort ■ Single dimension arrays of any of the above
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>	String Optional. Response to be returned to the caller, specified as a string.
<i>responseBytes</i>	byte[] Optional. Response to be returned to the caller, specified as a byte array.
<i>string</i>	<i>Deprecated</i> - Replaced by <i>responseString</i> . String Optional. Response to be returned to the caller, specified as a string.
<i>bytes</i>	<i>Deprecated</i> - Replaced by <i>responseBytes</i> . byte[] Optional. Response to be returned to the caller, specified as a byte array.
<i>response</i>	<i>Deprecated</i> - Replaced by <i>responseString</i> . String Optional. Response to be returned to the caller, specified as a string.
<i>contentType</i>	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: <code>text/XML</code> and <code>text/HTML</code> . You are not limited to these two values. You may either select one of these or type a new value.
<i>encoding</i>	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:setReponse` 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.setReponse.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 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

responseString **String** Optional. Response to be returned to the caller, specified as a string.

responseBytes **byte[]** Optional. Response to be returned to the caller, specified as a byte array.

responseStream **java.io.InputStream** 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 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

<i>responseCode</i>	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 <i>responseCode</i> that is not listed in RFC 2616, Section 10, you must also provide a value for <i>reasonPhrase</i> .
<i>reasonPhrase</i>	String Optional. HTTP reason phrase to be returned to the caller. If no reason is provided, the default reason phrase associated with <i>responseCode</i> will be used. You must provide a <i>reasonPhrase</i> for any <i>responseCode</i> 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 JMSDeliveryMode header.

Value	Description
PERSISTENT	Indicates the request message is persistent.
2	Default. Indicates the request message is persistent.
NON_PERSISTENT	Indicates the request message is not persistent.
1	Indicates the request message is not persistent.

Note: If the `jms.deliveryMode` is not one of the above values, Integration Server ignores the name/value pair and uses the default value of 2.

<code>jms.timeToLive</code>	Length of time, in milliseconds, that the JMS provider retains the message. A value of 0 means that the message does not expire.
-----------------------------	--

The JMS provider uses this value to set the JMSExpiration header in the sent JMS message.

Note: If the `jms.timeToLive` value is not a valid Long, Integration Server ignores the property and uses the default value of 0.

Property Name	Description
<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> <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 an 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 *messaging* parameter.

subprotocol **String** Conditional. Subprotocol used to invoke the service.

<u>A value of...</u>	<u>Indicates that...</u>
broker	<p>The service was invoked by a webMethods messaging trigger that received a document from the Broker or a locally published document.</p> <p>This subprotocol can only occur if <i>protocol</i> is <i>messaging</i>.</p>
HTTP	<p>The service was invoked through HTTP.</p> <p>This subprotocol can only occur if <i>protocol</i> is <i>http</i>.</p>
HTTPS	<p>The service was invoked through HTTPS.</p> <p>This subprotocol can only occur if <i>protocol</i> is <i>http</i>.</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 <i>jndi</i> 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 <i>jms</i>.</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 <i>native</i> also indicates that the <i>jms/destinationName</i> value is the name of the actual destination.</p>

This subprotocol can only occur if *protocol* is *jms*.

um

The service was invoked by a webMethods messaging trigger that received a document from the Universal Messaging.

This subprotocol can only occur if *protocol* is *messaging*.

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.

Key	Description
<i>to</i>	String List E-mail addresses for the recipients of the e-mail.
<i>from</i>	String List E-mail addresses for the senders of the e-mail.
<i>cc</i>	String List Conditional. E-mail addresses receiving a copy of the e-mail.
<i>bcc</i>	String List Conditional. E-mail addresses receiving a blind copy of the e-mail.
<i>replyto</i>	String List Conditional. E-mail address to which replies of this e-mail should be sent
<i>subject</i>	String Subject of the e-mail.
<i>filename</i>	String Conditional. Name of the attached file.
<i>contenttype</i>	String Conditional. Content-Type of the attached file.
<i>content</i>	java.io.InputStream Conditional. Contents of the attached file.
<i>recvdate</i>	String Conditional. Date the e-mail was received in String format. <i>recvdate</i> may

be passed as parameter for the `java.util.Date` constructor.

sentdate **String** Conditional. Date the e-mail was sent in String format. *sentdate* may be passed as parameter for the `java.util.Date` 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>	String URL used by the client to invoke the service.
<i>query</i>	String Conditional. Query portion of request URL.
<i>method</i>	String HTTP method used by the client to request the top-level service. Possible values are GET, PUT, POST, and DELETE.
<i>requestHdrs</i>	Document Fields in the request header, where key names represent header field names and values represent the header field values.
<i>ipInfo</i>	Document Information about the http socket connection. Contains the following information:

<u>Key</u>	<u>Description</u>
<i>clientIp</i>	String Optional. IP address of the client connecting to this socket. If null, <i>clientIP</i> is not included in the output.
<i>clientPort</i>	String 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>	String Local IP address for this socket connection to client.
	<i>localPort</i>	String Local port number for this socket connection to client.
	<i>remoteIp</i>	String Remote IP address for this socket connection to client.
	<i>remotePort</i>	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>	String Name of file that was put into the service directory.
<i>mimetype</i>	String 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>ipInfo</i>	Document Information about the FTP socket connection. Contains the following information:
<u>Key</u>	<u>Description</u>
<i>localIp</i>	String Local IP address for this socket connection to client.

localPort **String** Local port number for this socket connection to client.

remoteIp **String** Remote IP address for this socket connection to client.

remotePort **String** 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>	String Fully qualified name of the file submitted to the file polling listener.
<i>originalFilename</i>	String Name of the file when it was submitted to the file polling listener.
<i>contentType</i>	String 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>	String The original file length in bytes.
<i>lastModified</i>	java.util.Date Java date object indicating when the original file was last modified.

jms **Document** 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>	String Name of the JMS connection alias used by the JMS trigger to retrieve the message.

<i>triggerName</i>	String Fully qualified name of the JMS trigger that retrieved the JMS message that resulted in invocation of the service.
<i>destinationName</i>	String Name or lookup name of the destination from which the message that invoke the service was received. 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.
<i>destinationType</i>	String 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. <i>destinationType</i> will be one of the following: <code>QUEUE</code> indicates the destination is a queue. <code>TOPIC</code> indicates the destination is a topic.
<i>requestHdrs</i>	Document 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. 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. For a standard JMS trigger, Integration Server passes the entire JMS message, including request headers, on to the trigger service for processing.

<i>messaging</i>	Document Conditional. Information about the messaging transport. This parameter is returned only if the messaging transport invoked the service.				
<i>connectionAliasName</i>	String Name of the webMethods messaging connection alias used by the webMethods messaging trigger to retrieve the document. If the webMethods messaging 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.				
<i>triggerName</i>	String Fully qualified name of the webMethods messaging trigger that retrieved the document that resulted in invocation of the service.				
<i>documentTypeName</i>	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 <i>documentTypeName</i> 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 <i>documentTypeName</i> parameter contains folder.subfolder:documentA.				
<i>redeliveryCount</i>	String Number of times the messaging provider has redelivered the document to the trigger.				
	<table> <thead> <tr> <th><u>A value of...</u></th> <th><u>Indicates...</u></th> </tr> </thead> <tbody> <tr> <td>-1</td> <td>The transport used to send the document does not maintain a document redelivery count. For example,</td> </tr> </tbody> </table>	<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,
<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

A document with this structure is produced by the [pub.flow:getTransportInfo](#) service.

11 Hashtable Folder

■ Summary of Elements in this Folder	380
--	-----

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
pub.hashtable:containsKey	WmPublic. Checks for the existence of a hashtable element.
pub.hashtable:createHashtable	WmPublic. Creates a hashtable object.
pub.hashtable:get	WmPublic. Gets the value for a specified key in the hashtable.
pub.hashtable:listKeys	WmPublic. Lists all the keys stored in the hashtable.
pub.hashtable:put	WmPublic. Adds a key/value pair in the hashtable.
pub.hashtable:remove	WmPublic. Removes a key/value pair from the hashtable.
pub.hashtable:size	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.

12 IO Folder

■ Summary of Elements in this Folder	386
--	-----

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
pub.io:bytesToStream	WmPublic. Converts a <code>byte[]</code> to <code>java.io.ByteArrayInputStream</code> .
pub.io:close	WmPublic. Closes an <code>InputStream</code> or a reader object and releases the resources.
pub.io:createByteArray	WmPublic. Creates a byte array of the specified length.
pub.io:mark	WmPublic. Marks the current position in the <code>InputStream</code> or reader object.
pub.io:markSupported	WmPublic. Enables you to test whether your <code>InputStream</code> or reader object supports the mark and reset operations.
pub.io:read	WmPublic. Reads a specified number of bytes from the <code>InputStream</code> and stores them into a buffer.
pub.io:readAsString	WmPublic. Reads the data from a reader object and returns the contents as a string.
pub.io:readerToString	WmPublic. Reads the data from a reader object and converts it to a string.

Element	Package and Description
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.
pub.io:skip	WmPublic. Skips over and discards the specified number of bytes or characters from the input stream or a reader object.
pub.io:streamToBytes	WmPublic. Creates a byte[] from data that is read from an InputStream.
pub.io:streamToReader	WmPublic. Converts a java.io.InputStream to a java.io.Reader object.
pub.io:streamToString	WmPublic. Creates a string from data that is read from an InputStream.
pub.io:stringToReader	WmPublic. Converts a string object to a java.io.StringReader object.
pub.io:stringToStream	WmPublic. Converts a string to a byte stream.

pub.io:bytesToStream

WmPublic. Converts a byte[] to java.io.ByteArrayInputStream.

Input Parameters

<i>bytes</i>	byte[] The byte array to convert.
<i>length</i>	String 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>	String 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>	Object The InputStream. Object from which the service is to read bytes.
<i>offset</i>	String 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>	String 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>	Object 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>	Object The InputStream. If any bytes were read from the stream, the stream is repositioned after the last byte read.
<i>buffer</i>	Object The buffer into which data was written.
<i>bytesRead</i>	String 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>	java.io.Reader The reader object.
<i>length</i>	String The maximum number of characters to read from the input reader object.

Output Parameters

<i>reader</i>	java.io.Reader The reader object.
<i>lengthRead</i>	String 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>	String 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>	java.io.Reader The reader object.
---------------	--

Output Parameters

<i>string</i>	String 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>	java.io.Reader Optional. The reader object.
<i>length</i>	String The number of bytes or characters to skip.

Output Parameters

<i>stream</i>	java.io.InputStream Conditional. The InputStream. Returned only if the input parameter is <i>stream</i> .
<i>reader</i>	java.io.Reader Conditional. The reader object. Returned only if the input parameter is <i>reader</i> .
<i>bytesSkipped</i>	String Conditional. The actual number of bytes that were skipped. Returned only if the input parameter is <i>stream</i> .
<i>charactersSkipped</i>	String 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>	java.io.InputStream The <code>InputStream</code> that you want to convert.
---------------	---

Output Parameters

<i>bytes</i>	byte[] The bytes read from <i>stream</i> .
--------------	---

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

<i>inputStream</i>	java.io.InputStream The <code>InputStream</code> to convert to a reader object.
<i>encoding</i>	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 <code>autoDetect</code> , the default operating system encoding is used.

Output Parameters

<i>reader</i>	java.io.Reader The reader object read from <i>inputStream</i> .
---------------	--

pub.io:streamToString

WmPublic. Creates a string from data that is read from an `InputStream`.

Input Parameters

<i>inputStream</i>	java.io.InputStream The <code>InputStream</code> to convert to a string.
<i>encoding</i>	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 <code>autoDetect</code> , 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*.

13 JDBC Folder

■ Summary of Elements in this Folder	398
--	-----

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
pub.jdbc:getPoolInfo	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>	String Name of the Integration Server functional alias for which you want to obtain JDBC information.
<i>getConnectionWait</i>	String 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>	String The minimum number of connections the pool can have.
<i>maxConnections</i>	String The maximum number of connections the pool can have.
<i>totalConnections</i>	String 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>	String The number of connections that are available for use.

<i>status</i>	<p>String A status indicating whether the service was able to connect to the pool. A value of:</p> <ul style="list-style-type: none">■ <code>Active</code> indicates that the service was able to connect to the JDBC pool.■ <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.
<i>message</i>	<p>String 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>

14 JMS Folder

■ Summary of Elements in This Folder	402
--	-----

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
pub.jms:acknowledge	WmPublic. Sends an acknowledgment for a message to the JMS provider.
pub.jms:batchTriggerSpec	WmPublic. Specification for the signature of a JMS trigger that processes a batch of messages at one time.
pub.jms:createConsumer	WmPublic. Creates a message consumer to receive messages from destinations on the JMS provider.
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.
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 .
pub.jms:receive	WmPublic. Receives a message from a queue or topic on the JMS provider.
pub.jms:reply	WmPublic. Sends a reply message to a requesting client.
pub.jms:send	WmPublic. Sends a JMS message to the JMS provider.
pub.jms:sendAndWait	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
pub.jms:sendBatch	WmPublic. Sends multiple JMS messages to the same destination on the JMS provider.
pub.jms:triggerSpec	WmPublic. Specification for the input signature of a JMS trigger that processes one message at a time.
pub.jms:waitForReply	WmPublic. Retrieves the reply message for an asynchronous request.
pub.jms.wmjms:receiveStream	WmPublic. Receives a large message stream from a queue or topic on the webMethods Broker.
pub.jms.wmjms:sendStream	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>	String 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>	String The name of the JMS trigger that received the message whose status needs to be resolved.
<i>JMSMessage</i>	Document The message whose status needs to be resolved. This is a document reference (IData) to the pub.jms:JMSMessage document type, which defines the structure of a JMS message.

Output Parameters

<i>status</i>	String 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">■ NEW 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.■ DUPLICATE 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.■ IN_DOUBT 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.
<i>message</i>	String Conditional. A user-specified string that indicates why the message status is DUPLICATE or IN_DOUBT . 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>	String 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>	java.lang.Integer 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"> ■ <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.

	<ul style="list-style-type: none"> ■ <code>NON-PERSISTENT</code> 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.
	<p>Note: When sending a message, this value is obtained from the <i>JMSMessage/header/deliveryMode</i> input parameter.</p>
<i>JMSDestination</i>	Object Optional. Destination (queue or topic) to which the message was sent.
<i>JMSExpiration</i>	java.lang.Long Optional. 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>Note: When sending a message, this value is obtained from the <i>JMSMessage/header/timeToLive</i> input parameter.</p>
<i>JMSMessageID</i>	String . Optional. Unique identifier assigned to this message by the JMS provider.
<i>JMSPriority</i>	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.
	<p>Note: When sending a message, this value is obtained from the <i>JMSMessage/header/priority</i> input parameter.</p>
<i>JMSRedelivered</i>	<p>java.lang.Boolean Optional. Flag indicating the JMS provider delivered this message to the JMS client previously. A value of:</p> <ul style="list-style-type: none"> ■ <code>True</code> indicates that the message may have been delivered in the past. ■ <code>False</code> indicates that the JMS provider has not delivered this message previously.
<i>JMSReplyTo</i>	Object Optional. Destination to which a reply to this message should be sent.

	<i>JMSTimestamp</i>	java.lang.Long Optional. Time at which the message was given to the JMS provider.
	<i>JMSType</i>	String Optional. Message type identifier specified by the client when sending the message.
<i>properties</i>		Document. 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.
	Key	Description
	<i>JMSXDeliveryCount</i>	java.lang.Integer Optional. Specifies the number of times the JMS provider delivered the message. Most JMS providers set this value.
	<i>JMS_WMClusterNodes</i>	String 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>	String 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>	String 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>		Document Optional A Document (IData) contenting the JMS message body. Integration Server supports the following formats for the JMS message body:
	Key	Description
	<i>string</i>	String Optional. Message body in the form of a String.

<i>bytes</i>	primitive type Optional Message body in the form of a one-dimensional byte array.
<i>object</i>	Object. Optional. Message body in the form of a Serializable Java object.
<i>data</i>	<p>Document Optional. Message body in the form of a document (IData object).</p> <p>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.</p>
<i>message</i>	<p>Object Optional. Message body in the form of an actual javax.jms.Message.</p> <p>Note: When a 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.</p> <p>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.</p>

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>	Object A message consumer object that the session uses to receive messages sent to the specified destination.
<i>timeout</i>	java.lang.Long 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

<i>JMSMessage</i>	Document A document (IData) containing the JMS message received by the consumer.
-------------------	---

<u>Key</u>	<u>Description</u>
<i>header</i>	Document Conditional. A Document containing the header fields for the received message.
<u>Key</u>	<u>Description</u>
<i>JMSCorrelationID</i>	String 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>	java.lang.Integer Conditional. Delivery mode specified at the time the message was sent. PERSISTENT 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>	Object Conditional. Destination (queue or topic) to which the message was sent.
<i>JMSExpiration</i>	java.lang.Long 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>	String Conditional. Unique identifier assigned to this message by the JMS provider.
<i>JMSPriority</i>	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.
<i>JMSRedelivered</i>	java.lang.Boolean 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>	Object Conditional. Destination to which a reply to this message should be sent.
<i>JMSTimestamp</i>	java.lang.Long Conditional. Time at which the message was given to the JMS provider.
<i>JMSType</i>	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 may add the following properties to JMS messages it receives.

Key	Description
<i>JMSXDeliveryCount</i>	java.lang.Integer Conditional. Specifies the number of times the JMS provider delivered the message. Most JMS providers set this value.
<i>JMS_WMClusterNodes</i>	<p>String 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"> ■ 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.
<i>activation</i>	String 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>	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 (IData) containing the JMS message body. Integration Server supports the following formats for the JMS message body:

Key	Description
<i>string</i>	String Conditional. Message body in the form of a String.
<i>bytes</i>	primitive type Conditional. Message body in the form of a one-dimensional byte array.
<i>object</i>	Object. Conditional. Message body in the form of a Serializable Java object.
<i>data</i>	Document 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>
------------	--------------------

header **Document** Optional. A document containing the header of the replying JMS message.

<u>Key</u>	<u>Description</u>
<i>deliveryMode</i>	<p>String Optional. Specifies the message delivery mode for the reply message. Specify one of the following:</p> <p>PERSISTENT Default. Provide once-and-only-once delivery for the message. The message will not be lost if a JMS provider failure occurs.</p> <p>NON_PERSISTENT 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>java.lang.Integer 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>java.lang.Long Optional. Length of time, in milliseconds, that the JMS provider system retains the reply message. The default is 0, meaning that the message does not expire.</p>
<i>JMSType</i>	<p>String Optional. Message type identifier for the message.</p>

properties **Document** Optional. A Document containing optional fields added to the message header.

<u>Key</u>	<u>Description</u>
<i>activation</i>	<p>String Optional. A unique identifier that you want to assign to the message. JMS triggers use the</p>

activation value to determine if a message satisfies a join.

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>	String Optional. Message body in the form of a String.
<i>bytes</i>	primitive type Optional Message body in the form of a one-dimensional byte array.
<i>object</i>	Object. Optional. Message body in the form of a Serializable Java object.
<i>data</i>	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 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
<i>header</i>	Document Conditional. A Document containing the header fields for the reply message.
<i>JMSCorrelation ID</i>	String Conditional. A unique identifier used to link the reply message with the initial request message. The replying Integration Server automatically sets this value when it executes the pub.jms:reply service.
<i>JMSDelivery Mode</i>	java.lang.Integer 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. <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.

Note: When sending a reply 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. The *JMSReplyTo* value of the request message determines the destination of the reply message.

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 *JMSReplyMessage/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 reply 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 may add the following properties to JMS messages it receives.

<u>Key</u>	<u>Description</u>
<i>activation</i>	String Conditional. A unique identifier assigned by the sending service. A JMS trigger can join together messages with the same <i>activation</i> .
<i>uuid</i>	String 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>	String Conditional. Message body in the form of a String.

<i>bytes</i>	primitive type Conditional Message body in the form of a one-dimensional byte array.
<i>object</i>	Object. Conditional. Message body in the form of a Serializable Java object.
<i>data</i>	Document 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>	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:reply](#) service, Integration Server automatically sets the value of the *JMSCorrelationID* field in the *JMSReplyMessage*. Integration Server uses the value of either the *uuid* or *JMSMessageID* fields in the requesting JMS message to correlate the request and the response. If you specify the *uuid* when sending the request, the replying Integration Server will use the *uuid* as the *JMSCorrelationID* of the reply message. If 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

<i>connectionAliasName</i>	<p>String 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>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.</p>
<i>destinationType</i>	<p>String Optional. Type of destination to which you want to send the message. Specify one of the following:</p> <ul style="list-style-type: none"> ■ <code>QUEUE</code> to send the message to a particular queue. This is the default. ■ <code>TOPIC</code> to send the message to a topic. <p>Note: You need to specify <i>destinationType</i> only if you specified a <i>connectionAliasName</i> that uses the native webMethods API.</p>
<i>JMSMessage</i>	<p>Document A document representing the JMS message you want to send.</p>

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

<i>header</i>	Document Optional. A document containing the header of the JMS message.
---------------	--

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

<i>deliveryMode</i>	<p>String Optional. Specifies the message delivery mode for the message. Specify one of the following:</p> <p>PERSISTENT Default. Provide once-and-only-once delivery for the message. The message will not be lost if a JMS provider failure occurs.</p> <p>NON_PERSISTENT 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>java.lang.Integer 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>java.lang.Long 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>String Optional. Message type identifier for the message.</p>
<i>properties</i>	<p>Document Optional. A Document containing optional fields added to the message header. Integration Server adds the following properties to JMS messages it sends.</p>

<u>Key</u>	<u>Description</u>
<i>JMS_WMClusterNodes</i>	String 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 *JMS_WMClusterNodes* value should contain multiple Brokers.

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>	String Optional. Message body in the form of a String.
<i>bytes</i>	primitive type Optional Message body in the form of a one-dimensional byte array.
<i>object</i>	Object . Optional. Message body in the form of a Serializable Java object.
<i>data</i>	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`.

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.

<u>Key</u>	<u>Description</u>						
<i>header</i>	Document 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>String Conditional. A unique identifier used to link messages together.</td> </tr> <tr> <td><i>JMSDeliveryMode</i></td> <td>java.lang.Integer Delivery mode used to send the message. <code>PERSISTENT</code> indicates that the JMS provider provides once-and-only-once delivery for the</td> </tr> </tbody> </table>	<u>Key</u>	<u>Description</u>	<i>JMSCorrelationID</i>	String Conditional. A unique identifier used to link messages together.	<i>JMSDeliveryMode</i>	java.lang.Integer Delivery mode used to send the message. <code>PERSISTENT</code> indicates that the JMS provider provides once-and-only-once delivery for the
<u>Key</u>	<u>Description</u>						
<i>JMSCorrelationID</i>	String Conditional. A unique identifier used to link messages together.						
<i>JMSDeliveryMode</i>	java.lang.Integer 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 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.

Key	Description
<i>JMS_WMClusterNodes</i>	<p>String 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"> ■ 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.

<i>activation</i>	String Conditional. A unique identifier assigned by the sender. A JMS trigger can join together messages with the same <i>activation</i> .
<i>uuid</i>	String 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>	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>	String Conditional. Message body in the form of a String.
<i>bytes</i>	primitive type Conditional. Message body in the form of a one-dimensional byte array.
<i>object</i>	Object. Conditional. Message body in the form of a Serializable Java object.
<i>data</i>	Document 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: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*.

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:sendAndWait

`WmPublic`. Sends a request in the form of a JMS message to the JMS provider and optionally, waits for a reply.

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.

<i>destinationType</i>	<p>String Optional. Type of destination to which you want to send the message. Specify one of the following:</p> <ul style="list-style-type: none">■ <code>QUEUE</code> to send the message to a particular queue. This is the default.■ <code>TOPIC</code> to send the message to a topic. <p>Note: You need to specify a <i>destinationType</i> only if you specified a <i>connectionAliasName</i> that uses the native <code>webMethods</code> API.</p>
<i>destinationName</i> <i>ReplyTo</i>	<p>String 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 <code>webMethods</code> API to connect directly to the <code>webMethods</code> Broker.</p> <p>If you do not specify a destination for reply messages, Integration Server uses a <code>temporaryQueue</code> to receive the reply. A <i>temporaryQueue</i> 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.</p> <p>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 <i>destinationNameReplyTo</i> value.</p>
<i>destinationType</i> <i>ReplyTo</i>	<p>String Optional. Type of destination to which you want the reply to be sent. Specify one of the following:</p> <ul style="list-style-type: none">■ <code>QUEUE</code> to send the reply message to a particular queue. This is the default.■ <code>TOPIC</code> to send the reply message to a specific topic.
<i>timeout</i>	<p>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.</p> <p>The <i>timeout</i> value only applies for a synchronous request/reply. If <i>async</i> is set to <code>true</code>, Integration Server ignores the <i>timeout</i> value.</p>
<i>JMSMessage</i>	<p>Document A document representing the JMS message you want to send.</p>

<u>Key</u>	<u>Description</u>
<i>header</i>	Document Optional. A document containing the header of the JMS message.
<u>Key</u>	<u>Description</u>
<i>deliveryMode</i>	<p>String Optional. Specifies the message delivery mode for the message. Specify one of the following:</p> <p>PERSISTENT Default. Provide once-and-only-once delivery for the message. The message will not be lost if a JMS provider failure occurs.</p> <p>NON_PERSISTENT 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>java.lang.Integer 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>java.lang.Long 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>String Optional. Message type identifier for the message. Integration Server expects the reply message to be of this type.</p>

properties **Document** Optional. 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>	String 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.
---------------------------	---

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>	String Optional. Message body in the form of a String.
<i>bytes</i>	primitive type Optional. Message body in the form of a one-dimensional byte array.
<i>object</i>	Object . Optional. 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.

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>	Document 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.
<u>Key</u>	<u>Description</u>
<i>JMSCorrelationID</i>	String Conditional. A unique identifier used to link messages together.

<i>JMSDeliveryMode</i>	<p>java.lang.Integer Delivery mode used to send the message.</p> <p>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> <p>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.</p> <p>Note: When sending a message, this value is obtained from the <i>JMSMessage/header/deliveryMode</i> input parameter.</p>
<i>JMSDestination</i>	<p>Object Conditional. Destination (queue or topic) to which the message was sent.</p>
<i>JMSExpiration</i>	<p>java.lang.Long Optional. 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>Note: When sending a message, this value is obtained from the <i>JMSMessage/header/timeToLive</i> input parameter.</p>
<i>JMSMessageID</i>	<p>String Conditional. Unique identifier assigned to</p>

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.

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>	String Conditional. Message body in the form of a String.

bytes **primitive type** Conditional Message body in the form of a one-dimensional byte array.

object **Object.** Conditional. Message body in the form of a Serializable Java object.

data **Document** 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.

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, the *JMSReplyMessage* is null.

If this is an asynchronous reply, the *JMSReplyMessage* is null.

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

<i>header</i>	Document Conditional. A Document containing the header fields for the reply message.
---------------	---

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

<i>JMSCorrelationID</i>	<p>String 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 pub.jms:reply service.</p>
<i>JMSDeliveryMode</i>	<p>java.lang.Integer 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>Object Conditional. Destination (queue or topic) to which the message was sent.</p>
<i>JMSExpiration</i>	<p>java.lang.Long 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>String Conditional. Unique identifier assigned to this message by the JMS provider.</p>

delivered the message. Most JMS providers set this value.

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 uses the *activation* value to determine whether a message satisfies 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>	String Conditional. Message body in the form of a String.
<i>bytes</i>	primitive type Conditional Message body in the form of a one-dimensional byte array.
<i>object</i>	Object. Conditional. Message body in the form of a Serializable Java object.
<i>data</i>	Document 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>	Object 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.

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 `uuid` or `JMSMessageID` in the requesting JMS message to correlate the request and the response. If you specify the `uuid` when sending the request, the replying Integration Server will use the `uuid` as the `JMSCorrelationID` of the reply message. 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.

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 multiseed 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.

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.

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>	Document Optional. A document containing the header of the JMS message.				
	<table border="1"> <thead> <tr> <th><u>Key</u></th> <th><u>Description</u></th> </tr> </thead> <tbody> <tr> <td><i>JMSType</i></td> <td>String Optional. Message type identifier for the message.</td> </tr> </tbody> </table>	<u>Key</u>	<u>Description</u>	<i>JMSType</i>	String Optional. Message type identifier for the message.
<u>Key</u>	<u>Description</u>				
<i>JMSType</i>	String Optional. Message type identifier for the message.				
<i>properties</i>	Document Optional. 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>activation</i>	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 <i>activation</i> .
<i>uuid</i>	String 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>	String Optional. Message body in the form of a String.
<i>bytes</i>	primitive type Optional Message body in the form of a one-dimensional byte array.
<i>object</i>	Object. Optional. Message body in the form of a Serializable Java object.
<i>data</i>	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.

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:

Key	Description
<i>header</i>	Document 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.

Key	Description
<i>JMSCorrelationID</i>	<p>String Conditional. A unique identifier used to link messages together.</p>
<i>JMSDeliveryMode</i>	<p>java.lang.Integer Delivery mode used to send the message.</p> <p>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> <p>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.</p> <p>Note: When sending a message, this value is obtained from the <i>JMSMessage/header/deliveryMode</i> input parameter.</p>
<i>JMSDestination</i>	<p>Object Conditional. Destination (queue or topic) to which the message was sent.</p>
<i>JMSExpiration</i>	<p>java.lang.Long 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>Note: When sending a message, this value is obtained from the <i>JMSMessage/header/timeToLive</i> input parameter.</p>

<i>JMSMessageID</i>	String Conditional. Unique identifier assigned to this message by the JMS provider.
<i>JMSPriority</i>	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 <i>JMSMessage/header/priority</i> input parameter.
<i>JMSReplyTo</i>	Object Conditional. Specifies the destination to which a response to this message should be sent.
<i>JMSTimestamp</i>	java.lang.Long Time at which the message was given to the JMS provider.
<i>JMSType</i>	String Conditional. Message type identifier specified by the client when sending the message.
<i>properties</i>	Document 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>activation</i>	String Conditional. A unique identifier assigned by the sender. A JMS trigger can join together messages with the same <i>activation</i> .
<i>uuid</i>	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>	String Conditional. Message body in the form of a String.
<i>bytes</i>	primitive type Conditional. Message body in the form of a one-dimensional byte array.
<i>object</i>	Object. Conditional. Message body in the form of a Serializable Java object.
<i>data</i>	Document 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>	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 no value is specified, the service does not wait for a reply.

Output Parameters

JMSReplyMessage **Document** Conditional. Document containing the JMS message received as a reply.

If this is an asynchronous request/reply and Integration Server does not receive a reply before the specified *timeout* value elapses, the *JMSReplyMessage* is null.

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

<i>header</i>	Document Conditional. A Document containing the header fields for the reply message.
---------------	---

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

<i>JMSCorrelationID</i>	String Conditional. A unique identifier used to link the reply message with the initial request message.
-------------------------	---

<i>JMSDeliveryMode</i>	<p>java.lang.Integer Conditional. Delivery mode used to send the message.</p> <p>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> <p>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.</p>
------------------------	---

<i>JMSDestination</i>	Object Conditional. Destination (queue or topic)
-----------------------	---

	to which the message was sent.
<i>JMSExpiration</i>	java.lang.Long 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>	String Conditional. Unique identifier assigned to this message by the JMS provider.
<i>JMSPriority</i>	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.
<i>JMSRedelivered</i>	java.lang.Boolean 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>	Object Conditional. Specifies the destination to which a response to this message should be sent.
<i>JMSTimestamp</i>	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 may add the following properties to JMS messages it sends or receives.

<u>Key</u>	<u>Description</u>
<i>JMSXDeliveryCount</i>	java.lang.Integer 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>String 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"> ■ 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.

<i>activation</i>	String Conditional. A unique identifier assigned by the sending service. A JMS trigger uses the <i>activation</i> to determine whether a message is part of a join.
<i>uuid</i>	String 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>	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>	String Conditional. Message body in the form of a String.
<i>bytes</i>	primitive type Conditional Message body in the form of a one-dimensional byte array.
<i>object</i>	Object. Conditional. Message body in the form of a Serializable Java object.
<i>data</i>	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 Integration Server does not receive a reply by the time the *timeout* interval elapses, the `pub.jms:waitForReply` service returns a null document. This indicates that the *timeout* interval expired.

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.

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>	Object A message consumer object that the service uses to receive the large message stream. Create the message consumer object using the pub.jms:createConsumer service.
<i>timeout</i>	java.lang.Long 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>	Object 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 `InputStream.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, see `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 `InputStream.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* `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

<i>connectionAliasName</i>	String Name of the JMS connection alias that you want to use to send the message.
<i>destinationName</i>	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.
<i>destinationType</i>	String Optional. Type of destination to which you want to send the message. Specify one of the following: <ul style="list-style-type: none">■ <code>QUEUE</code> to send the message to a particular receiver/queue. This is the default.■ <code>TOPIC</code> to send the message to a topic. <div style="border: 1px solid #ccc; background-color: #f0f0f0; padding: 5px; margin-top: 10px;">Note: You need to specify a <i>destinationType</i> only if you specified a <i>connectionAliasName</i> that uses the native webMethods API.</div>
<i>stream</i>	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](#)

15 JSON Folder

■ Data Type Mapping	474
■ Summary of Elements in This Folder	475

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 " pub.json:jsonStreamToDocument " on page 476 and " pub.json:jsonStringToDocument " on page 477.
number (real)	Float or Double Java wrapper. For more information about converting real numbers, see the <i>decodeRealAsDouble</i> input parameter in " pub.json:jsonStreamToDocument " on page 476 and " pub.json:jsonStringToDocument " on page 477.
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`".

Summary of Elements in This Folder

The following elements are available in this folder:

Element	Package and Description
pub.json:documentToJSONString	WmPublic. Converts a document (IData object) to a JSON string.
pub.json:jsonStreamToDocument	WmPublic. Converts content from the JSON content stream to a document (an IData object).
pub.json:jsonStringToDocument	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>	Document The document (IData object) to be converted to a JSON string.
<i>prettyPrint</i>	<p>String 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"> ■ <code>true</code> to format <i>jsonString</i> output variable for human readability. ■ <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. ■ <code>null</code> to use the <code>prettyPrint</code> setting already in effect for the HTTP client making the request, as follows: <ul style="list-style-type: none"> ■ If the HTTP client request includes <code>jsonPrettyPrint=true</code> in the URI, JSON pretty printing is in effect. ■ If the HTTP client request includes <code>jsonPrettyPrint=false</code> in the URI, JSON pretty printing is not in effect. ■ If the HTTP client request does not include the <code>jsonPrettyPrint</code> 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`.

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

The JSON content handler in Integration Server creates a *jsonStream* object in the pipeline when a client sends a request to Integration Server using the `application/json` Content-Type. The [pub.json:jsonStreamToDocument](#) service converts the *jsonStream* into a document (IData object) so you can use the elements from *jsonStream* in a flow service.

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. For more information about `watt.server.http.jsonFormat`, see *webMethods Integration Server Administrator's Guide*.

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*.

16 List Folder

■ Summary of Elements in this Folder	480
--	-----

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
pub.list:addItemToVector	WmPublic. Adds an item or a list of items to a java.util.Vector object.
pub.list:appendToDocumentList	WmPublic. Adds documents to a document list.
pub.list:appendToStringList	WmPublic. Adds Strings to a String list.
pub.list:sizeofList	WmPublic. Returns the number of elements in a list.
pub.list:stringListToDocumentList	WmPublic. Converts a String list to a document list.
pub.list:vectorToArray	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>	Object[] Optional. List of items to be added to the vector object.
<i>addNulls</i>	<p>String Optional. Specifies whether a null item can be added to the vector object. Set to:</p> <ul style="list-style-type: none"> ■ <code>false</code> to prevent null values from being added to the vector object. This is the default. ■ <code>true</code> to allow null values to be added to the vector object.

Output Parameters

<i>vector</i>	java.util.Vector 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>	Document List 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>	Document List Optional. Documents you want to append to the end of <i>toList</i> .
<i>fromItem</i>	Document 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:

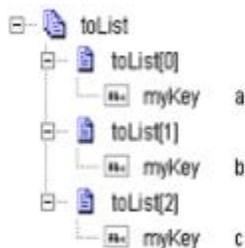
fromList[0] = "a"

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.

17 Math Folder

■ Summary of Elements in this Folder	488
--	-----

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
pub.math:absoluteValue	WmPublic. Returns the 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.
pub.math:addFloats	WmPublic. Adds one floating point number (represented as a String) to another and returns the sum.
pub.math:addIntList	WmPublic. Adds a list of integers (represented in a String list) and returns the sum.
pub.math:addInts	WmPublic. Adds one integer (represented as a String) to another and returns the sum.
pub.math:addObjects	WmPublic. Adds one java.lang.Number object to another and returns the sum.
pub.math:divideFloats	WmPublic. Divides one floating point number (represented as a String) by another (<i>num1/num2</i>) and returns the quotient.
pub.math:divideInts	WmPublic. Divides one integer (represented as a String) by another (<i>num1/num2</i>) and returns the quotient.
pub.math:divideObjects	WmPublic. Divides one java.lang.Number object by another (<i>num1/num2</i>) and returns the quotient.

Element	Package and Description
pub.math:max	WmPublic. Returns the largest number from a list of numbers.
pub.math:min	WmPublic. Returns smallest number from a list of numbers.
pub.math:multiplyFloatList	WmPublic. Multiplies a list of floating point numbers (represented in a String list) and returns the product.
pub.math:multiplyFloats	WmPublic. Multiplies one floating point number (represented as String) by another and returns the product.
pub.math:multiplyIntList	WmPublic. Multiplies a list of integers (represented in a String list) and returns the product.
pub.math:multiplyInts	WmPublic. Multiplies one integer (represented as a String) by another and returns the product.
pub.math:multiplyObjects	WmPublic. Multiplies one java.lang.Number object by another and returns the product.
pub.math:randomDouble	WmPublic. Returns the next pseudorandom, uniformly distributed double between 0.0 and 1.0.
pub.math:roundNumber	WmPublic. Returns a rounded number.
pub.math:subtractFloats	WmPublic. Subtracts one floating point number (represented as a String) from another and returns the difference.
pub.math:subtractInts	WmPublic. Subtracts one integer (represented as a String) from another and returns the difference.
pub.math:subtractObjects	WmPublic. Subtracts one java.lang.Number object from another and returns the difference.
pub.math:toNumber	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:

Value	Description
<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>	String Number (floating point number represented as a String) to multiply.
<i>num2</i>	String Number (floating point number represented as a String) to multiply.
<i>precision</i>	String 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:

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 *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.

18 Metadata Folder

- Summary of Elements in this Folder 508

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
pub.metadata.assets:publishPackages	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>	String A comma-separated list of Integration Server packages about which you want to publish metadata.
<i>includeServerAssets</i>	<p>String 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"> ■ <code>True</code> to indicate that information about administrative assets will be published to the CentraSite shared registry. This is the default. ■ <code>False</code> to indicate that information about administrative assets will not be published to the CentraSite shared registry.

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.

19 MIME Folder

■ Summary of Elements in this Folder	512
--	-----

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
pub.mime:addBodyPart	WmPublic. Adds a body part (header fields and content) to a specified MIME object.
pub.mime:addMimeHeader	WmPublic. Adds one or more header fields to a specified MIME object.
pub.mime:createMimeData	WmPublic. Creates a MIME object.
pub.mime:getBodyPartContent	WmPublic. Retrieves the content (payload) from the specified MIME object.
pub.mime:getBodyPartHeader	WmPublic. Returns the list of header fields for the specified body part.
pub.mime:getContentType	WmPublic. Returns the value of the Content-Type message header from the specified MIME object.
pub.mime:getEnvelopeStream	WmPublic. Generates an InputStream representation of a MIME message from a specified MIME object.
pub.mime:getMimeHeader	WmPublic. Returns the list of message headers from a specified MIME object.
pub.mime:getNumParts	WmPublic. Returns the number of body parts in the specified MIME object.
pub.mime:getPrimaryContentType	WmPublic. Returns the top-level portion of a MIME object's Content-Type value.
pub.mime:getSubContentType	WmPublic. Returns the sub-type portion of a MIME object's Content-Type value.

Service	Function
pub.mime:mergeHeaderAndBody	WmPublic. Concatenates the contents of the header and body returned by the pub.client:http service.

pub.mime:addBodyPart

WmPublic. Adds a body part (header fields and content) to a specified MIME object.

Input Parameters

<i>mimeData</i>	Document MIME object to which you want to add a body part. (This IData object is produced by pub.mime:createMimeData .)
<i>content</i>	java.io.InputStream or Object 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>isErrStream</i>	String Flag that specifies whether <i>content</i> is to be treated as a MIME entity. <p>Important: 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"> ■ <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. <p>Note: <code>addBodyPart</code> assumes that all data up to the first blank line represents the entity's header fields.</p> ■ <i>no</i> to treat <i>content</i> as an ordinary payload.
<i>mimeHeader</i>	Document 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. <p>For example, if you wanted to add the following header fields:</p> <pre>X-Doctype: RFQ X-Severity: 10</pre>

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

- mimeData* **Document** MIME object to which you want the header fields added. (This IData object is produced by [pub.mime:createMimeData](#).)
- mimeHeader* **Document** 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:

Key	Description
<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.

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` property `watt.server.mime.decodeHeaders`. 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>	Document MIME object whose content you want to retrieve. (This IData object is produced by pub.mime:createMimeData .)
<i>index</i>	String 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. Note: If <i>contentID</i> is specified, <i>index</i> is ignored.
<i>contentID</i>	String Optional. Value of the <code>Content-ID</code> 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>	IData The payload of the specified body part.
<i>encrypted</i>	String Flag whose value indicates whether <i>content</i> is an encrypted MIME message. A value of: <ul style="list-style-type: none">■ <code>true</code> indicates that <i>content</i> is an encrypted message.■ <code>false</code> indicates that <i>content</i> is not an encrypted message.
<i>signed</i>	String Flag indicating whether <i>content</i> is a signed MIME message. A value of: <ul style="list-style-type: none">■ <code>true</code> indicates that <i>content</i> is a signed MIME message.■ <code>false</code> indicates that <i>content</i> 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.

<i>contentID</i>	String Optional. Value of the <code>Content-ID</code> header field of the body part whose header fields you want to retrieve.
<i>decodeHeaders</i>	<p>String Conditional. Flag whose value indicates whether to decode encoded headers in the MIME object. Set to:</p> <ul style="list-style-type: none"> ■ <code>true</code> to indicate that the headers should be decoded. ■ <code>false</code> 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:

<u>Key</u>	<u>Value</u>
<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>	Document MIME object from which you want to generate the MIME message. (This IData object is produced by pub.mime:createMimeData .)
<i>index</i>	String 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>	String 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). Note: If <i>index</i> is specified, <i>contentID</i> is ignored.
<i>suppressHeaders</i>	String List 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>	String 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">■ <code>yes</code> to create a multipart message (Content-Type message header is set to "multipart/mixed").■ <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">■ 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>.■ If the message contains multiple body parts, Content-Type is automatically set to "multipart/mixed."

Output Parameters

<i>envStream</i>	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

<i>mimeData</i>	Document MIME object whose sub-type you want to discover. (This IData object is produced by pub.mime:createMimeData .)
-----------------	---

Output Parameters

<i>subContentType</i>	String Message's sub-type. For example, if the message's Content-Type header were: <pre>Content-Type: multipart/mixed</pre> <i>subContentType</i> 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

<i>headerLines</i>	Document <code>IData</code> object containing the message headers returned by pub.client:http . (The message headers are returned in the <i>lines</i> document inside the <i>header</i> output parameter that is produced by pub.client:http .)
<i>body</i>	Document <code>IData</code> 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>	byte[] Optional. Body of the message (if pub.client:http returned the body as a <code>byte[]</code>).
<i>stream</i>	java.io.InputStream Optional. The body of the message (if pub.client:http returned the body as an <code>InputStream</code>).

Output Parameters

<i>stream</i>	java.io.InputStream <code>InputStream</code> 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](#)

20 OAuth Folder

■ Summary of Elements in this Folder	534
--	-----

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>String 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"> ■ <code>code</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.
----------------------------	---

- `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 535](#).

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>String 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>String 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>String 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 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>	String Specify the type of grant flow required by the client. For refresh tokens, you must specify <code>refresh_token</code> .
<i>refresh_token</i>	String Refresh token issued to the client by the authentication server.
<i>scope</i>	String 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>	String The access token issued by the authorization server.
<i>token_type</i>	String The type of access token issued by the authorization server. The value is <code>Bearer</code> .
<i>expires_in</i>	String The number of seconds for which the access token is valid.
<i>refresh_token</i>	String 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.

21 Packages Folder

■ Summary of Elements in this Folder	542
--	-----

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
pub.packages:activatePackage	WmPublic. Activates (makes available to clients) an inactive package.
pub.packages:backupPackage	WmPublic. Creates a backup copy of a specified package.
pub.packages:disablePackage	WmPublic. Disables a package, thus prohibiting access to the services in the package.
pub.packages:enablePackage	WmPublic. Enables a package that has been disabled.
pub.packages:installPackage	WmPublic. Installs a package that has been published to this server.
pub.packages:recoverPackage	WmPublic. Recovers a package that exists in the server's salvage directory.
pub.packages:reloadPackage	WmPublic. Loads a new copy of the package into memory from disk.

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

package **String** Name of the package that you want to disable. Package names are case sensitive.

Output Parameters

message **String** 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>	String Name of the package that you want to recover. Package names are case sensitive.
<i>activateOnRecover</i>	String 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">■ <i>yes</i> to activate the package after it is recovered and make it immediately available to clients.■ <i>no</i> 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 pub.packages:activatePackage service.

Output Parameters

<i>message</i>	String 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.

22 PKI Folder

- Summary of Elements in this Folder 550

You use the elements in the pki folder to create and verify PKCS#7 signatures with PKI profiles. You also use elements in this folder to create and process S/MIME messages using PKI profiles.

Note: The WmPKI package is deprecated as of webMethods Integration Server version 9.7.

Summary of Elements in this Folder

The following elements are available in this folder:

Element	Package and Description
pub.pki.pkcs7:sign	WmPKI. <i>Deprecated</i> - There is no replacement service. Creates a PKCS7 SignedData object using a PKI profile.
pub.pki.pkcs7:verify	WmPKI. <i>Deprecated</i> - There is no replacement service. Processes a digital signature to make sure that the provided data has not been modified.
pub.pki.smime.createSignedAndEncryptedData	WmPKI. <i>Deprecated</i> - There is no replacement service. Digitally signs a MIME message and then encrypts it.
pub.pki.smime.createSignedData	WmPKI. <i>Deprecated</i> - There is no replacement service. Digitally signs a MIME message using a specified PKI profile.
pub.pki.smime:processEncryptedData	WmPKI. <i>Deprecated</i> - There is no replacement service. Decrypts an encrypted S/MIME message using a specified PKI profile.
pub.pki.smime:processSignedData	WmPKI. <i>Deprecated</i> - There is no replacement service. Verifies the signature from a signed S/MIME entity using a specified PKI profile, and then extracts the message from the S/MIME entity.

pub.pki.pkcs7:sign

WmPKI. *Deprecated* - There is no replacement service.

Creates a PKCS7 SignedData object using a PKI profile.

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.

Note: This service is similar to [pub.security.pkcs7:sign](#) except that it uses a PKI profile to create the PKCS7 SignedData object.

Input Parameters

signerInfo

Document List Information about a single signer of the signed data object.

Note: This service accepts only one *signerInfo* .

Key	Description
<i>profileAlias</i>	String PKI profile alias used to sign the data. This service retrieves the key from the profile to perform the signing operation and includes the associated public and CA certificates in the signature that it generates.
<i>hashAlgorithm</i>	String Optional. The algorithm to use when computing the digest of the provided data. Specify: <ul style="list-style-type: none"> ■ MD5 ■ SHA-1 (the default) ■ SHA-256 ■ SHA-384 ■ SHA-512

data

byte[] Data to be digitally signed.

<i>detachedSignature</i>	<p>String 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"> ■ <code>true</code> to generate a detached signature. ■ <code>false</code> to generate an implicit signature (one that includes the signed data). This is the default.
--------------------------	--

Output Parameters

<i>signature</i>	<p>byte[] Signature generated from the supplied data. This is a DER-encoded representation of the SignedData object as specified in PKCS#7.</p>
------------------	---

pub.pki.pkcs7:verify

WmPKI. *Deprecated* - There is no replacement service.

Processes a digital signature to make sure that the provided data has not been modified.

Note: This service is similar to [pub.security.pkcs7:verify](#) except that it uses a PKI profile to obtain the certificate against which to verify the signer's signature.

Input Parameters

<i>profileAlias</i>	<p>String Name of the PKI profile to be used for certificate verification.</p>
<i>signature</i>	<p>byte[] Signature to use to determine whether the signed data is intact (a DER-encoded representation of the SignedData object as specified in PKCS#7). 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 entity in <i>signature</i> .</p>
<i>data</i>	<p>byte[] Optional. The data that was signed. If you are processing an implicitly signed message, you do not need to supply <i>data</i> because both the data and the signature reside in <i>signature</i> .</p>
<i>detachedSignature</i>	<p>String Flag indicating whether the message has a detached signature. Set to:</p> <ul style="list-style-type: none"> ■ <code>true</code> to indicate that the message has a detached signature.

- `false` to indicate that the message has an implicit signature. This is the default.

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 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.

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>	java.security.cert.X509Certificate[] 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>	java.util.Date Time at which the signer signed the data.
<i>trusted</i>	String Flag indicating whether the certificate chain presented by the signer is trusted. A value of: <ul style="list-style-type: none"> ■ <code>true</code> indicates that the chain is trusted. ■ <code>false</code> indicates that the chain is not trusted.
<i>status</i>	String Code indicating whether the signatures were successfully verified. If successful, <i>status</i> contains <code>verified</code> . If the signatures were not successfully verified, <i>status</i> contains an error message.

errorMessage **String** Conditional. If the signatures were not successfully verified, this parameter contains the text "Invalid signer certificate file information."

pub.pki.smime.createSignedAndEncryptedData

WmPKI. *Deprecated* - There is no replacement service.

Digitally signs a MIME message and then encrypts it.

Note: This service is similar to [pub.smime:createSignedAndEncryptedData](#) and [pub.smime.keystore:createSignedAndEncryptedData](#) except that a PKI profile is used to provide signing key and certificate information.

Input Parameters

<i>envStream</i>	java.io.InputStream The MIME message that you want to sign and encrypt (for example, the output produced by pub.mime:getEnvelopeStream).
<i>profileAlias</i>	String PKI profile alias to use to sign the data. This service retrieves the key from the profile to perform the signing operation and includes the associated public and CA certificates in the signature that it generates.
<i>explicit</i>	<p>String Optional. Flag indicating whether an implicit or explicit signature is to be generated. Set to:</p> <ul style="list-style-type: none"> ■ <code>true</code> to generate an explicit (detached) signature. This is the default. ■ <code>false</code> to generate an implicit signature.
<i>recipientCerts</i>	<p>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.</p> <p>Note: For multiple recipients, this service creates a single message that is encrypted for all recipients. It <i>does not</i> create a separate message for each recipient.</p>
<i>encryptionAlg</i>	<p>String Optional. Code specifying the encryption algorithm to use. Must be one of the following values:</p> <pre>TripleDES DES RC2</pre>

Default is `TripleDES`

keyLength

String Optional. Length of the encryption key for RC2 encryption. Must be one of the following values:

40
64
128

Default is 128. This parameter is ignored if *encryptionAlg* is not RC2.

Output Parameters

SMimeEnvStream **java.io.InputStream** Signed and encrypted MIME message.

pub.pki.smime.createSignedData

WmPKI. *Deprecated* - There is no replacement service.

Digitally signs a MIME message using a specified PKI profile.

Note: This service is similar to [pub.smime:createSignedData](#) and [pub.smime.keystore:createSignedData](#) except that a PKI profile supplies signing key and certificate information.

Input Parameters

envStream **java.io.InputStream** MIME message that you want to sign (for example, the output produced by [pub.mime:getEnvelopeStream](#)).

profileAlias **String** PKI profile alias to use to sign the message. This service retrieves the key from the profile to perform the signing operation and includes the associated public and CA certificates in the signature that it generates.

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.

Output Parameters

SMimeEnvStream **java.io.InputStream** Signed MIME message.

pub.pki.smime:processEncryptedData

WmPKI. *Deprecated* - There is no replacement service.

Decrypts an encrypted S/MIME message using a specified PKI profile.

Note: This service is similar to [pub.smime:processEncryptedData](#) and [pub.smime.keystore:processEncryptedData](#) except that a PKI profile supplies signing key and certificate information.

Input Parameters

SMimeEnvStream **java.io.InputStream** The encrypted S/MIME entity (for example, the output produced by [pub.smime:createEncryptedData](#)).

profileAlias **String** PKI profile to use to decrypt the message. This service retrieves the decryption key and public encryption certificate from the profile to perform the decryption operation.

Output Parameters

mimeData **Document** MIME object containing 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** Conditional. Flag indicating whether the decrypted MIME entity is encrypted. A value of:

- `true` indicates that the MIME entity is encrypted.
- `false` indicates that the MIME entity is not encrypted.

signed **String** Conditional. Flag indicating whether the decrypted MIME entity is signed. A value of:

- `true` indicates that the MIME entity is signed.

	<ul style="list-style-type: none"> ■ <code>false</code> indicates that the MIME entity is not signed.
<i>certsOnly</i>	<p>String Conditional. Flag indicating whether the decrypted MIME entity is a certs-only entity. A value of</p> <ul style="list-style-type: none"> ■ <code>true</code> indicates that the MIME entity is a certs-only entity. ■ <code>false</code> indicates that the MIME entity is not a certs-only entity.
<i>stream</i>	<p>java.io.InputStream Conditional. The decrypted MIME entity. This parameter is present only when the decrypted entity is an S/MIME message.</p>

pub.pki.smime:processSignedData

WmPKI. *Deprecated* - There is no replacement service.

Verifies the signature from a signed S/MIME entity using a specified PKI profile, and then extracts the message from the S/MIME entity.

Note: This service is like [pub.smime:processSignedData](#) except that a PKI profile supplies the certificates against which the signature is verified.

Input Parameters

<i>SMimeEnvStream</i>	java.io.InputStream Signed MIME entity (for example, the output produced by " pub.pki.smime.createSignedData " on page 555).
<i>profileAlias</i>	String PKI profile to use for certificate validation.
<i>signerCertChain</i>	<p>byte[][] Optional. Certificate chain of the party that signed the message. Certificates must appear in hierarchical order, starting with the signer's certificate in element 0.</p> <p>The following shows how the elements of a complete chain would appear for a certificate that was issued through two intermediate CAs.</p>

<u>Element</u>	<u>Contents</u>
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

<i>mimeData</i>	Document MIME object containing the extracted MIME entity.
<i>contentDigest</i>	String Message digest (base64-encoded) that processSignedData recalculated.
<i>signerCert</i>	java.security.cert.X509Certificate Signer's X509 certificate.
<i>encrypted</i>	<p>String Conditional. Flag indicating whether the extracted MIME entity is encrypted. A value of:</p> <ul style="list-style-type: none"> ■ <code>true</code> indicates that the MIME entity is encrypted. ■ <code>false</code> indicates that the MIME entity is not encrypted.
<i>signed</i>	<p>String Conditional. Flag indicating whether the extracted MIME entity is signed.</p> <ul style="list-style-type: none"> ■ <code>true</code> indicates that the MIME entity is signed. ■ <code>false</code> indicates that the MIME entity is not signed.
<i>certsOnly</i>	<p>String Conditional. Flag indicating whether the extracted MIME entity is a certs-only entity.</p> <ul style="list-style-type: none"> ■ <code>true</code> indicates that the MIME entity is a certs-only entity. ■ <code>false</code> indicates that the MIME entity is not a certs-only entity.
<i>stream</i>	java.io.InputStream Conditional. Extracted MIME entity. This parameter is present only when the decrypted entity is an S/MIME message.
<i>verify</i>	<p>String Flag indicating whether the signature was successfully processed (that is, the signature was successfully verified with the public key supplied by the PKI profile).</p> <ul style="list-style-type: none"> ■ <code>true</code> indicates that signature processing was successful.

- `false` indicates that signature processing failed. The signature could not be verified because `errorCode` 1 or 4 occurred (see `errorCode` below).

trusted **String** The signer is a trusted entity. For the signer to be trusted, the signer's certificate or one of its root certificates should be present in the trusted CA directory.

- `true` indicates that the signer is a trusted entity.
- `false` indicates that the signer is not a trusted entity.

errorCode **String** Conditional. Number indicating what kind of error occurred, if any, while processing the signature. See *errorMessage* for possible values.

If no error occurred, *errorCode* is not returned.

errorMessage **String** Conditional. Textual error message indicating what kind of error occurred, if any, while processing the signature.

<u><i>errorCode</i></u>	<u><i>errorMessage</i></u>
1	Invalid signer certificate chain file information.
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-verification 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, then pass *stream* to the `processEncryptedData` service as necessary.

See Also

[pub.smime:processEncryptedData](#)

pub.smime:createSignedData
pub.smime.keystore:createSignedData
pub.smime.keystore:processEncryptedData

23 Publish Folder

■ Summary of Elements in this Folder	562
--	-----

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
pub.publish:deliver	WmPublic. Delivers a document to a specific destination.
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.
pub.publish:documentResolverSpec	WmPublic. Specification for the signature of a service that determines whether a document's status is New, Duplicate, or In Doubt.
pub.publish:envelope	WmPublic. Document type that defines the content and structure of the envelope that accompanies a published document.
pub.publish:getRedeliveryCount	WmPublic. Retrieves the redelivery count for a document.
pub.publish:publish	WmPublic. Publishes a document locally or to the messaging provider.
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.
pub.publish:reply	WmPublic. Delivers a reply document to the requesting client.
pub.publish:syncToBroker	WmPublic. <i>Deprecated</i> - Replaced by pub.publish:syncToProvider .

Element	Package and Description
pub.publish:syncToProvider	Synchronizes one or more publishable document types with their associated definition on the messaging provider.
pub.publish:waitForReply	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 pub.publish:deliverAndWait or pub.publish:publishAndWait service elapses.
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.

pub.publish:deliver

WmPublic. Delivers a document to a specific destination.

Note: The `pub.publish:deliver` service can be used to publish a document to webMethods Broker only.

Input Parameters

<i>documentTypeName</i>	<p>String Fully qualified name of the publishable document type being delivered.</p> <p>The publishable document type must be synchronized with the associated Broker document type. If the document types are not synchronized, publication fails.</p>
<i>document</i>	<p>Document Document (IData object) conforming to the publishable document type in <i>documentTypeName</i>.</p>
<i>destId</i>	<p>String The client ID to which the document will be delivered. You can specify the default client ID for an Integration Server, or you can specify the client ID for an individual trigger. If you specify an incorrect client ID, the Integration Server delivers the document to the Broker, but the Broker never delivers the document to the intended recipient and no error is produced.</p>

- 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 client on the Broker. You cannot use `pub.publish:deliver` with Universal Messaging.

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 more information about how the Integration Server and Broker deliver documents and for information about building a service that delivers a document, see the *Publish-Subscribe Developer's Guide*.

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.

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.

Note: The `pub.publish:deliverAndWait` can be used to publish a document to webMethods Broker only.

Input Parameters

<i>documentTypeName</i>	<p>String Fully qualified name of the publishable document type being delivered.</p> <p>Note: The publishable document type must be synchronized with the associated Broker document type. If the document types are not synchronized, publication fails.</p>
<i>document</i>	<p>Document Document (IData object) conforming to the publishable document type in <i>documentTypeName</i>.</p>
<i>receiveDocumentTypeName</i>	<p>String 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>
<i>destId</i>	<p>String The client ID to which the document will be delivered. You can specify the default client ID for an Integration Server, or you can specify the client ID for an individual trigger. If you specify an incorrect client ID, the Integration Server delivers the document to the webMethods Broker, but the webMethods Broker never delivers the document to the intended recipient and no error is produced.</p>
<i>waitTime</i>	<p>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.</p>
<i>async</i>	<p>String Optional. Flag specifying whether this is an asynchronous or synchronous request/reply. Set to:</p> <ul style="list-style-type: none"> ■ <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. <p>Note: To retrieve the reply to an asynchronous request, invoke the <code>pub.publish:waitForReply</code> service.</p> <ul style="list-style-type: none"> ■ <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.

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 publish a document to the Broker. You cannot use `pub.publish:deliverAndWait` with Universal Messaging.

You can use the `pub.publish:deliverAndWait` service to initiate and continue a private conversation between two webMethods Broker 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, the Integration Server ends the request, and the service returns a null document indicating that the request timed out. The Integration Server then executes the next step in the flow service. If a reply document arrives after the flow service resumes execution, the 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.

To view a list of client IDs on the webMethods Broker, use the webMethods Broker user interface or use Designer to test the publishable document type that you want to deliver.

Use `pub.publish:deliverAndWait` if you need to know that a specific client successfully received and processed the request document.

For more information about how to build a services that initiate synchronous or asynchronous request/reply scenarios, see the *Publish-Subscribe Developer's Guide*.

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.

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

<i>documentTypeName</i>	String Fully qualified name of the document whose status is In Doubt.
<i>redeliveryCount</i>	String Number of times the document has been redelivered to the trigger queue on the Integration Server.
<i>uuid</i>	String Universally unique identifier for the document. The publishing application assigns the <i>uuid</i> to a document.

<i>document</i>	Document 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>	String The transport (such as LOCAL, BROKER, or UM) used to send the document to the Integration Server.
<i>triggerName</i>	String The name of the webMethods messaging trigger that received the document whose status needs to be resolved.

Output Parameters

<i>status</i>	<p>String 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"> ■ NEW indicates that the document is new and has not been processed by the trigger. Integration Server instructs the trigger to process the document. ■ DUPLICATE 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. ■ IN_DOUBT 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.
<i>message</i>	<p>String Conditional. A user-specified string that indicates why the document status is DUPLICATE or IN_DOUBT. 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>	String 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>	String Optional. Used by the Integration Server to track business process context and audit context across multiple Integration Servers. Important: 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>	java.lang.Short[] Optional. This field is provided for backward compatibility.
<i>errorsTo</i>	String 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 pub.publish.notification:error .
<i>errorRequestsTo</i>	String Optional. This field is provided for backward compatibility.
<i>locale</i>	String 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>	java.lang.Integer Optional. This field is provided for backward compatibility.
<i>priority</i>	java.lang.Integer 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>	String 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>	java.lang.Integer Optional. This field is provided for backward compatibility.
<i>signature</i>	byte[] Optional. A byte sequence that holds a digital signature. Specify a digital signature if clients receiving this document requires one.
<i>signatureType</i>	String Optional. The type of digital signature being used.
<i>startResult</i>	java.lang.Integer Optional. This field is provided for backward compatibility.
<i>tag</i>	java.lang.Integer Optional. Used with pub.publish:publishAndWait and pub.publish:deliverAndWait to match a request document with its corresponding reply document. Important: 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>	String 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>	String Optional. This field is provided for backwards compatibility.
<i>transformState</i>	String 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 *pubID*, and *uuid* parameters only.

<i>age</i>	java.lang.Integer 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, *age* also includes the length of time the document spends on the other webMethods Brokers.

<i>connectionIntegrity</i>	<p>String 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:</p> <ul style="list-style-type: none">■ <i><empty></i> indicates that at some point, the document passed through a connection that was not encrypted.■ <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.■ <code>U.S. Domestic</code> indicates that the event traveled exclusively over connections with an encryption strength of <code>ENCRYPT_LEVEL_US_DOMESTIC</code>.
<i>destId</i>	<p>String Optional. The ID of the client to which the document is being delivered. The publishing client sets the <i>destID</i> when it publishes the document. For example, the Integration Server uses the <i>destID</i> value specified in the pub.publish:deliver service or the pub.publish:deliverAndWait service to populate the <i>destID</i> value in the document envelope.</p>
<i>enqueueTime</i>	<p>java.util.Date Optional. The date and time that the webMethods Broker placed the document into the client queue.</p>
<i>logBroker</i>	<p>String 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.</p>
<i>logHost</i>	<p>String 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.</p>
<i>pubDistinguishedName</i>	<p>String 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</p>

	webMethods Broker and clears this parameter when the publisher has a non-SSL connection.
<i>pubId</i>	String Optional. An identifier for the document publisher assigned by the messaging provider.
<i>pubNetAddr</i>	byte[] Optional. The IP address and port number of the document's publisher.
<i>pubSeqn</i>	java.lang.Long Optional. This field is provided for backwards compatibility.
<i>pubLabel</i>	java.lang.Short[] Optional. This field is provided for backwards compatibility.
<i>recvTime</i>	java.util.Date Optional. The date and time the document was received by the messaging provider.
<i>route</i>	Document List 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.

Key	Description
<i>broker</i>	String Optional. The name of the webMethods Broker.
<i>recvTime</i>	java.util.Date Optional. The time the webMethods Broker received the document from the publishing client or another webMethods Broker.
<i>enqueueTime</i>	java.util.Date 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	<p>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.)</p> <p>Integration Server may or may not have received the document before.</p>
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>String 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</p>
-------------------------	---

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.

document

Document Document (IData object) conforming to the document type in *documentTypeName* .

local

String Optional. Flag specifying whether the document is to be published locally or to the messaging provider. Set to:

- `true` to publish locally (to this Integration Server only).
- `false` 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.

If the publishable document type does not specify a messaging provider and there is no default messaging connection alias configured for Integration Server, all publishes become local (that is, the *local* flag is set implicitly to `true`).

A publishable document type that specifies Universal Messaging as the messaging provider cannot be published locally. If the document type in *documentTypeName* is associated with a Universal Messaging connection alias, Integration Server ignores the value of *local* .

delayUntilServiceSuccess

String Optional. Flag indicating whether the publish should happen when the `pub.publish:publish` 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.

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 there is no default messaging connection alias, Integration Server publishes the document locally.

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` service 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` service 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

documentTypeName **String** Fully qualified name of the publishable document type being published.

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.

<i>document</i>	Document Document (IData object) conforming to the document type in <i>documentTypeName</i> .
<i>receiveDocumentTypeName</i>	String 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.
<i>local</i>	String Optional. Flag specifying whether the document is to be published locally or to the webMethods Broker. Set to: <ul style="list-style-type: none">■ <code>true</code> to publish locally (to this Integration Server only).■ <code>false</code> to publish to the webMethods Broker. This is the default. <p>If the publishable document type does not specify a messaging provider and there is no default messaging connection alias configured for Integration Server, all publishes become local (that is, the <i>local</i> flag is set implicitly to <code>true</code>).</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>	String 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>	String Optional. Flag specifying whether this is an asynchronous or synchronous publish. Set to: <ul style="list-style-type: none">■ <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. <div data-bbox="581 1682 1367 1768" style="background-color: #f0f0f0; padding: 5px;">Note: To retrieve the reply to an asynchronous publish, invoke the <code>pub.publish:waitForReply</code> service.</div> <ul style="list-style-type: none">■ <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.

Output Parameters

<i>receivedDocument</i>	<p>Document Document (IData object) received as response. If no matching document is received within the wait time, this will be null.</p>
	<p>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.</p>
<i>tag</i>	<p>String 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 pub.publish:waitForReply service to retrieve the reply.</p>
	<p>Note: 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>String Conditional. Status indicating whether the service was successful. Integration Server reports status only for locally published documents. A value of:</p>
	<ul style="list-style-type: none"> ■ <code>success</code> indicates that the service executed successfully.
	<p>Note: If at least one subscribing trigger has room in its queue, the status is set to <code>success</code>.</p>
	<ul style="list-style-type: none"> ■ <code>requestTimedOut</code> indicates that the service timed out (that is, the <i>waitTime</i> specified in the service elapsed before Integration Server received a reply).
	<ul style="list-style-type: none"> ■ <code>noSubscriber</code> indicates that Integration Server does not contain any triggers that subscribe to the document.
	<ul style="list-style-type: none"> ■ <code>capacityExceeded</code> 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.
	<p>Note: Integration Server reports the <code>capacityExceeded</code> status only when the <code>watt.server.publish.local.rejectOOS</code> property is set to <code>true</code>.</p>

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 there is no default messaging connection alias, Integration Server publishes the document locally.

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>	Document 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>	String 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>	Document The reply IData object. This document must conform to the publishable document type specified in <i>documentTypeName</i> .
<i>delayUntilServiceSuccess</i>	String 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: <ul style="list-style-type: none">■ <code>true</code> to delay publishing until after the top-level service executes successfully.■ <code>false</code> 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.

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

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: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

tag **String** A unique identifier for the publish request for which you are retrieving a reply. Integration Server uses the *tag* 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>	String Optional. The resource producing the error. Integration Server sets the value of this field to Integration Server.
<i>errorCategory</i>	String Optional. Type of exception. Integration Server sets the value of this field to <code>Application</code> .
<i>errorText</i>	String Optional. Exception text message. At Dispatcher debug level 9, a stack trace of the exception will also be returned.
<i>eventID</i>	java.lang.Long 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>	Document Optional. A document reference to the pub.publish:envelope 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	592
--	-----

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.

\$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.

- **DONE** indicates that the transaction is completed.
- **FAILED** indicates that the transaction expired because the time-to-live or the retry limit has been exceeded.
- **UNKNOWN** indicates that the transaction identification number in *tid* is not recognized.

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>	String Name of the service to be run on the remote webMethods Integration Server.
<i>tid</i>	String Transaction identification number for the service.
<i>inputs</i>	Document Optional. Document (IData object) containing the inputs for the service.

Output Parameters

<i>results</i>	Document Conditional. Document (IData object) containing the pipeline as it exists after the service is invoked.
----------------	---

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

<i>service</i>	String Service to be run on the remote Integration Server.
<i>tid</i>	String Transaction identification number for the service.
<i>inputs</i>	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>	String 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 Remote Servers tab of the Integration Server Administrator.
--------------	---

<i>ttl</i>	String Optional. Transaction time-to-live measured in minutes. The transaction expires when <i>ttl</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>	String Optional. Maximum number of times to retry the transaction. Default is 0 (no retry limit).
<i>followtid</i>	String 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>	String 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>	String Service to be run on the remote webMethods Integration Server.
<i>tid</i>	String Transaction identification number for the service.
<i>inputs</i>	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	602
--	-----

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
pub.replicator:addReleaseRegistryEntry	WmPublic. Adds an entry to the webMethods Integration Server's Package Release Registry.
pub.replicator:deleteReleaseRegistryEntry	WmPublic. Deletes an entry from the webMethods Integration Server's Package Release Registry.
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.
pub.replicator:distributeViaSvcPull	WmPublic. Pulls a specified package release from a publisher's server.
pub.replicator:distributeViaSvcPush	WmPublic. Pushes a package from your server to a list of subscribers (other webMethods Integration Servers).
pub.replicator:generateReplicationEvent	WmPublic. Generates a replication event.
pub.replicator:getLocalReleasedList	WmPublic. Returns all entries in your webMethods Integration Server's Package Release Registry.
pub.replicator:getRemoteReleasedList	WmPublic. Queries the publisher for released packages.
pub.replicator:notifyPackageRelease	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.
pub.replicator:packageCreation	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>	String 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>	String Name of the release. This name could be different from the name of the package.
<i>version</i>	String 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>	String Build number of the release (for example, <i>12, 530</i>).
<i>patchNums</i>	String One or more comma-separated patch numbers included in this release.
<i>JVMVersion</i>	String Minimum JVM version number that this release requires.
<i>description</i>	String 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>	String Name of the release.
<i>version</i>	String 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>	String Conditional. Build number of the release (for example, <code>12, 530</code>).
<i>patch_nums</i>	String Conditional. Comma-separated list of patch numbers included in this release.
<i>time</i>	String Time when the package was released.
<i>jvm_version</i>	String Minimum JVM version number that the release requires.
<i>description</i>	String Conditional. Brief description of this release.
<i>source_server_version</i>	String 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>	String 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>	String Name of the release.
<i>version</i>	String Version number of the release, in the format <i>##</i> or <i>###</i> (for example, 1.2 or 1.2.1).
<i>build</i>	String Conditional. Build number of the release (for example, 12, 530).
<i>patch_nums</i>	String Conditional. Comma-separated list of patch numbers included in this release.
<i>time</i>	String Time when the package was released.
<i>jvm_version</i>	String Minimum JVM version number that the release requires.
<i>description</i>	String Conditional. Brief description of the release.
<i>source_server_version</i>	String 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>	String 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">■ <code>get</code> to retrieve a package from the publisher's server. This is the default.■ <code>put</code> to send a package to a subscriber's server.
<i>serverhost</i>	String Host name or IP address of the remote Integration Server.
<i>serverport</i>	String Number of the FTP port on the remote Integration Server.
<i>username</i>	String User name that your server will use to log on to the remote Integration Server.
<i>password</i>	String 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>	String Name of the release.
<i>publisher</i>	String 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.

Key	Description
<i>name</i>	String Name of the release.
<i>version</i>	String 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>	String Conditional. Build number of the release (for example, <i>12, 530</i>).
<i>patch_nums</i>	String Conditional. Comma-separated list of the patch numbers included in this release.
<i>time</i>	String Time when the package was released.
<i>jvm_version</i>	String Minimum JVM version number that the release requires.
<i>description</i>	String Conditional. Brief description of the release.
<i>source_server_version</i>	String 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.

Key	Description
<i>name</i>	String Name of the release.
<i>version</i>	String 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>	String Conditional. Build number of the release (for example, <code>12, 530</code>).
<i>patch_nums</i>	String Conditional. Comma-separated list of the patch numbers included in this release.
<i>time</i>	String Time when the package was released.
<i>jvm_version</i>	String Minimum JVM version number that the release requires.
<i>description</i>	String Conditional. Brief description of the release.
<i>source_server_version</i>	String 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.

	<ul style="list-style-type: none"> ■ <code>partial</code> to indicate a patch or an update for the package.
<i>filter</i>	<p>String Flag that specifies whether all files are to be included in the distribution file or only selected files.</p> <p>If only selected files are to be included, use this parameter in conjunction with <i>fileList</i> to specify which files to include.</p> <p>Set to:</p> <ul style="list-style-type: none"> ■ <code>includeall</code> to include all the files in the distribution file. This is the default. ■ <code>include</code> to include selected files in the distribution file. ■ <code>exclude</code> to include all except selected files in the distribution file.
<i>fileList</i>	<p>String List Names of files to include or exclude from the distribution file, depending on the value of <i>filter</i>.</p>
<i>fileNamePattern</i>	<p>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, <code>*.java</code>, <code>*.dsp</code>).</p>
<i>filesToDeleteList</i>	<p>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.</p>

Output Parameters

<i>\$result</i>	<p>String Conditional. If the distribution file is created successfully, this parameter contains the value <code>OK</code>. If the distribution file was not created successfully, this parameter is not present in the output signature and the service throws an exception.</p>
-----------------	--

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	614
--	-----

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
pub.report:runFileTemplate	WmPublic. Applies a template file to a document (IData object).
pub.report:runFileTemplateOnPipe	WmPublic. Applies a template to the pipeline.
pub.report:runStringTemplate	WmPublic. Applies an output template to a specified document (IData object).
pub.report:runStringTemplateOnPipe	WmPublic. Applies a template to the pipeline.
pub.report:runTemplate	WmPublic. Applies a template in a file to a specified document (IData object).
pub.report:runTemplateOnPipe	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>	java.io.File Template file.
<i>\$values</i>	Document Document (IData object) to bind against <i>\$template</i> .
<i>fileEncoding</i>	String 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	620
--	-----

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
pub.scheduler:addComplexTask	WmPublic. Adds a complex task to the Scheduler.
pub.scheduler:addOneTimeTask	WmPublic. Adds a task that runs only once to the Scheduler.
pub.scheduler:addRepeatingTask	WmPublic. Adds a recurring task to the Scheduler.
pub.scheduler:cancelTask	WmPublic. Removes a task from the Scheduler.
pub.scheduler:getTaskIDs	WmPublic. Retrieves a list of identification numbers for tasks currently in the Scheduler.
pub.scheduler:getTaskInfo	WmPublic. Retrieves information about a task on the Scheduler.
pub.scheduler:getUserTaskList	WmPublic. Returns a list of scheduled user tasks.
pub.scheduler:migrateTasksToJDBC	WmPublic. Migrates scheduled user tasks from the Integration Server embedded database to an external database.
pub.scheduler:resumeTask	WmPublic. Resumes a suspended task.
pub.scheduler:suspendTask	WmPublic. Suspends a task on the Scheduler.
pub.scheduler:updateComplexTask	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>	String Name of the service you want to schedule for execution on the server.
<i>description</i>	String Text string describing this task.
<i>target</i>	<p>String Server or servers on which the task is to run. (Clustered environments only). Set to:</p> <ul style="list-style-type: none"> any to run the task on any server in the cluster. The task will run on only <i>one</i> of the servers. <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> <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.</p> <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"> <code>all</code> 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 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>	String 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>	String 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>	String 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>	String List 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>	String List 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>	String List 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>	String List 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>	String List 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>	String Identification number of the task added to the scheduler.
<i>type</i>	String Code indicating the type of task added. For this type of task, the value of <i>type</i> will be <code>complex</code> .
<i>taskAdded</i>	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

<i>service</i>	String Name of the service you want to schedule for execution.
<i>description</i>	String Text string describing this task.
<i>target</i>	String Server or servers on which the task is to run.(Clustered environments only). Set to: <ul style="list-style-type: none">■ <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.<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><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.</p>■ <code>all</code> to run the task on all servers in the cluster.<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><p>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.</p>

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>	String Identification number of the task added to the scheduler.
<i>type</i>	String 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>	String 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>	String Name of the service you want to schedule for execution on the server.
<i>description</i>	String Text string describing this task.
<i>target</i>	<p>String Server or servers on which the task is to run. (Clustered environments only). Set to:</p> <ul style="list-style-type: none"> ■ <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. <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>	String 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>	String 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>	String 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>	String 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>	String Time interval (measured in seconds) between executions of the task.
<i>doNotOverlap</i>	<p>String Optional. Flag that indicates whether you want executions of this task to overlap. Set to:</p> <ul style="list-style-type: none"> ■ <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. ■ <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.

Output Parameters

<i>taskID</i>	String Identification number of the task added to the Scheduler.
<i>type</i>	String 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>	String 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>	String Conditional. Date on which to run the task, in <i>yyyy/MM/dd</i> format.

	<i>time</i>	String Conditional. Time at which to run the task, in <i>HH:mm:ss</i> format.														
<i>repeatingTaskInfo</i>		Document Conditional. Information about the task represented by <i>taskID</i> . This parameter is present only if <i>type</i> is <i>repeat</i> .														
		<table> <thead> <tr> <th><u>Key</u></th> <th><u>Description</u></th> </tr> </thead> <tbody> <tr> <td><i>interval</i></td> <td>String Conditional. Time interval (measured in seconds) between repetitions of the task.</td> </tr> <tr> <td><i>doNotOverlap</i></td> <td>String Conditional. Indicates whether recurrences of this task will overlap.</td> </tr> </tbody> </table>	<u>Key</u>	<u>Description</u>	<i>interval</i>	String Conditional. Time interval (measured in seconds) between repetitions of the task.	<i>doNotOverlap</i>	String Conditional. Indicates whether recurrences of this task will overlap.								
<u>Key</u>	<u>Description</u>															
<i>interval</i>	String Conditional. Time interval (measured in seconds) between repetitions of the task.															
<i>doNotOverlap</i>	String Conditional. Indicates whether recurrences of this task will overlap.															
<i>complexTaskInfo</i>		Document Conditional. Information about the task. This parameter is present only if <i>type</i> is <i>complex</i> .														
		<table> <thead> <tr> <th><u>Key</u></th> <th><u>Description</u></th> </tr> </thead> <tbody> <tr> <td><i>startDate</i></td> <td>String Conditional. Date on which the task is scheduled to start, in <i>yyyy/MM/dd</i> format.</td> </tr> <tr> <td><i>startTime</i></td> <td>String Conditional. Time at which the task is scheduled to start, in <i>HH:mm:ss</i> format.</td> </tr> <tr> <td><i>endDate</i></td> <td>String Conditional. Date on which the task expires, in <i>yyyy/MM/dd</i> format.</td> </tr> <tr> <td><i>endTime</i></td> <td>String Conditional. Time at which the task expires, in <i>HH:mm:ss</i> format.</td> </tr> <tr> <td><i>minutes</i></td> <td>String List Conditional. Minutes at which the task is scheduled to run. Minutes are represented by integers between 0 and 59.</td> </tr> <tr> <td><i>hours</i></td> <td>String List Conditional. Hours when the task is scheduled to run. Hours are represented by integers between 0 and 23.</td> </tr> </tbody> </table>	<u>Key</u>	<u>Description</u>	<i>startDate</i>	String Conditional. Date on which the task is scheduled to start, in <i>yyyy/MM/dd</i> format.	<i>startTime</i>	String Conditional. Time at which the task is scheduled to start, in <i>HH:mm:ss</i> format.	<i>endDate</i>	String Conditional. Date on which the task expires, in <i>yyyy/MM/dd</i> format.	<i>endTime</i>	String Conditional. Time at which the task expires, in <i>HH:mm:ss</i> format.	<i>minutes</i>	String List Conditional. Minutes at which the task is scheduled to run. Minutes are represented by integers between 0 and 59.	<i>hours</i>	String List Conditional. Hours when the task is scheduled to run. Hours are represented by integers between 0 and 23.
<u>Key</u>	<u>Description</u>															
<i>startDate</i>	String Conditional. Date on which the task is scheduled to start, in <i>yyyy/MM/dd</i> format.															
<i>startTime</i>	String Conditional. Time at which the task is scheduled to start, in <i>HH:mm:ss</i> format.															
<i>endDate</i>	String Conditional. Date on which the task expires, in <i>yyyy/MM/dd</i> format.															
<i>endTime</i>	String Conditional. Time at which the task expires, in <i>HH:mm:ss</i> format.															
<i>minutes</i>	String List Conditional. Minutes at which the task is scheduled to run. Minutes are represented by integers between 0 and 59.															
<i>hours</i>	String List Conditional. Hours when the task is scheduled to run. Hours are represented by integers between 0 and 23.															

<i>months</i>	String List Conditional. 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>	String List Conditional. 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>	String List Conditional. 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>	Document List List of one-time and simple repeating tasks.
<i>extTasks</i>	Document List 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 webMethods and Intelligent Business Operations 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>	String Identification number of the task to be updated.
<i>service</i>	String Optional. Name of the service you want to schedule for execution on the server.
<i>description</i>	String Optional. Text string describing this task.
<i>target</i>	<p>String Optional. Server or servers on which the task is to run. (Clustered environments only). Set to:</p> <ul style="list-style-type: none"> ■ <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. <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: 5px;"> <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.</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 in <i>webMethods Integration Server Administrator's Guide</i>.</p> <ul style="list-style-type: none"> ■ <code>all</code> to run the task on all servers in the cluster. For clustered environments only. <p>For example, suppose you run an application on each server in the cluster, and each server maintains its own database for</p>

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>	Document Optional. Document (IData object) containing input to the scheduled service.
<i>startTime</i>	String 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>	String 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>	String 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>	String 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>	String List 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>	String List 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>	String List 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>	String List 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>	String List 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>	String 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>	String 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>	String Identification number of the task to be updated.
<i>service</i>	String Optional. Name of the service to be scheduled.
<i>description</i>	String Optional. Text string describing this task.
<i>target</i>	<p>String Optional. Server or servers on which the task is to run. (Clustered environments only). Set to:</p> <ul style="list-style-type: none"> ■ <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. <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.

runAsUser

String Optional. User ID under which the service is to be executed.

<i>inputs</i>	Document Optional. Document (IData object) containing inputs to the scheduled service.
<i>date</i>	String Optional. Date on which to run the task, in <i>yyyy/MM/dd</i> format.
<i>time</i>	String Optional. Time at which to run the service, in <i>HH:mm:ss</i> format.

Output Parameters

<i>type</i>	String 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>	String 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>	String Identification number of the task to be updated.
<i>service</i>	String Optional. Name of the service run by the task.
<i>description</i>	String Optional. Text string describing this task.
<i>target</i>	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.

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

	<ul style="list-style-type: none">■ 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>	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>startTime</i>	String 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>	String 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>	String 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>	String 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>inputs</i>	Document Optional. Document (IData object) containing inputs to the scheduled service.
<i>interval</i>	String Optional. Time interval (measured in seconds) between repetitions of the task.
<i>doNotOverlap</i>	String Optional. Flag indicating whether or not you want the executions of this task to overlap. Set to: <ul style="list-style-type: none">■ <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.■ <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.

Output Parameters

type **String** Code indicating the type of task updated. For this type of task, the value of *type* will be `repeat`.

taskUpdated **String** Indicates whether the task was successfully updated. If the task was successfully updated, *taskUpdated* contains `true`. 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	648
--	-----

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
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.
pub.schema:validate	WmPublic. Validates an object using an IS document type or a schema.
pub.schema:validatePipeline	WmPublic. Validates the pipeline against a document type.
pub.schema.w3c	WmPublic. This folder contains definitions for XML Schemas as defined in the W3C specification <i>XML Schema Part 2: Datatypes</i> .
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 <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> .
pub.schema.w3c:xsi	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>	String Conditional. Relative url of the generated schema.
<i>source</i>	String 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>	String 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:

Key	Description
<i>warningMessage</i>	String 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 or an IS schema.

Input Parameters

<i>object</i>	Document or com.wm.lang.xml.Document or <code>com.wm.lang.xml.Node</code> Object to be validated.
<i>conformsTo</i>	<p>String Document type or schema to validate <i>object</i> against.</p> <ul style="list-style-type: none"> ■ If <i>object</i> is a document (IData object), <i>conformsTo</i> must specify the fully qualified name of a document type on the Integration Server. ■ If <i>object</i> is a <code>com.wm.lang.xml.Document</code> or <code>com.wm.lang.xml.Node</code> object, <i>conformsTo</i> 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.

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.

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 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.

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>	String Location of the error in XQL.
<i>errorCode</i>	String Error code (for example, vv-001).
<i>errorMessage</i>	String Error message (for example, Missing Object).

Usage Notes

When validating supplied XML against an IS document type or 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>	String Fully qualified name of the document type that you want to validate the pipeline against.
<i>maxErrors</i>	String 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.
<i>ignoreContent</i>	<p>String Optional. Flag that specifies whether the validation processor will validate content keys of the type String, String List, or String Table. Set to:</p> <ul style="list-style-type: none"> ■ <code>true</code> to ignore content (that is, do not validate keys of these types). ■ <code>false</code> to validate content. This is the default.
<i>failIfInvalid</i>	<p>String 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"> ■ <code>true</code> to indicate that service should fail if object is invalid. ■ <code>false</code> 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>	String Location of the error in XQL.
<i>errorCode</i>	String Error code (for example, vv-001).
<i>errorMessage</i>	String 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	658
■ Summary of Elements in this Folder	659

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
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.
pub.security.setKeyAndChain	WmPublic. <i>Deprecated</i> - Replaced by pub.security.keystore.setKeyAndChain .
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).
pub.security.enterpriseGateway.alertSpec	WmPublic. Specification for flow services used to send alerts about violations of webMethods Enterprise Gateway rules.
pub.security.keystore.getCertificate	WmPublic. Returns the trusted certificate, stored in a truststore, that corresponds to the certificate's alias.
pub.security.keystore.getKeyAndChain	WmPublic. Returns a private key and its associated certificate chain from a designated keystore.
pub.security.keystore.getTrustedCertificates	WmPublic. Returns the trusted certificates located in a specified truststore.

Element	Package and Description
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.
pub.security.keystore.pkcs7:sign	WmPublic. Creates a PKCS7 signed Data object.
pub.security.outboundPasswords:setPassword	WmPublic. Stores a key and password in the password store.
pub.security.outboundPasswords:getPassword	WmPublic. Retrieves a password from the password store for a given key.
pub.security.outboundPasswords:listKeys	WmPublic. Lists the keys in the password store.
pub.security.outboundPasswords:removePassword	WmPublic. Removes a password from the password store for a given key.
pub.security.outboundPasswords:updatePassword	WmPublic. Changes the password value for a key already in the password store.
pub.security.pkcs7:sign	WmPublic. Creates a PKCS7 SignedData object.
pub.security.pkcs7:verify	WmPublic. Processes a digital signature to guarantee that the data associated with the signature has not been modified.
pub.security.util:createMessageDigest	WmPublic. Generates a message digest for a given message.
pub.security.util:getCertificateInfo	WmPublic. Retrieves information such as serial number, issuer,

Element	Package and Description
	and expiration date from a digital certificate.
pub.security.util:loadPKCS7CertChain	WmPublic. Converts a certificate chain that is in PKCS #7 format to a list of byte arrays.
pub.security.util:createSecureString	WmPublic. Creates a WmSecureString object from either a Java String, byte array, or character array.
pub.security.util:convertSecureString	WmPublic. Returns a WmSecureString in Java String, byte array, or character array format.
pub.security.util:destroySecureString	WmPublic. Destroys a WmSecureString such that it no longer resides in memory and is removed from the pipeline.
pub.security.xml:decryptXML	WmPublic. Decrypts the encrypted XML, and returns the XML as either a string or stream object.
pub.security.xml:encryptXML	WmPublic. Encrypt an XML document or node in an XML document.
pub.security.xml:signXML	WmPublic. Digitally sign an outgoing XML node or document.
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.

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>	String 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\instances\instance_name\config\certs\cert1.der</i>).
<i>certFiles</i>	String List 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>provoker</i>	Object A byte array containing the client's private key.
<i>certs</i>	Object List 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>	String The Enterprise Gateway rule that the request violates.						
<i>alertInfo</i>	Document List 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> <p>String The action that the Integration Server functioning as the Enterprise Gateway Server takes when a request violates an Enterprise Gateway rule. The action will have one of these values:</p> <ul style="list-style-type: none"> ■ DENY if the server denies the request and issues alerts as configured. ■ ALERT if the server allows the request and issues alerts as configured. </td> </tr> <tr> <td><i>requestType</i></td> <td> <p>String The specific request type to which the server applies the Enterprise Gateway rule. The values are:</p> <ul style="list-style-type: none"> ■ ALL if the rule applies to all requests. </td> </tr> </tbody> </table>	<u>Key</u>	<u>Description</u>	<i>alertAction</i>	<p>String The action that the Integration Server functioning as the Enterprise Gateway Server takes when a request violates an Enterprise Gateway rule. The action will have one of these values:</p> <ul style="list-style-type: none"> ■ DENY if the server denies the request and issues alerts as configured. ■ ALERT if the server allows the request and issues alerts as configured. 	<i>requestType</i>	<p>String The specific request type to which the server applies the Enterprise Gateway rule. The values are:</p> <ul style="list-style-type: none"> ■ ALL if the rule applies to all requests.
<u>Key</u>	<u>Description</u>						
<i>alertAction</i>	<p>String The action that the Integration Server functioning as the Enterprise Gateway Server takes when a request violates an Enterprise Gateway rule. The action will have one of these values:</p> <ul style="list-style-type: none"> ■ DENY if the server denies the request and issues alerts as configured. ■ ALERT if the server allows the request and issues alerts as configured. 						
<i>requestType</i>	<p>String The specific request type to which the server applies the Enterprise Gateway rule. The values are:</p> <ul style="list-style-type: none"> ■ ALL if the rule applies to all requests. 						

	<ul style="list-style-type: none"> ■ SOAP if the rule applies to SOAP requests only. ■ REST if the rule applies to REST requests only. ■ INVOKE if the rule applies to INVOKE requests only.
<i>filterName</i>	<p>String The filter that the request satisfies. The values are:</p> <ul style="list-style-type: none"> ■ <code>DoSFilter</code> if the request satisfies the Denial of Service settings specified for Enterprise Gateway rules. ■ <code>MsgSizeLimitFilter</code> if the request satisfies the Message Size Limit filter in the rule. ■ <code>OAuthFilter</code> if the request satisfies the OAuth filter in the rule. ■ <code>mobileAppProtectionFilter</code> if the request satisfies the Mobile Application Protection filter in the rule. ■ <code>None</code> if the request violates a rule that has no filters.
<i>message</i>	<p>String The details about the filter condition that the request satisfies.</p>
<i>requestUser</i>	<p>String The user ID that sent the request that violates the Enterprise Gateway rule. This field is empty if authentication is disabled on Enterprise Gateway Server. For more information about authentication on Enterprise Gateway Server, see <i>webMethods Integration Server Administrator's Guide</i>.</p>
<i>requestHost</i>	<p>String The host name or IP address of the client that sent the request.</p>
<i>requestTime</i>	<p>String Date and time at which the request reached Enterprise Gateway Server. This parameter is in the format</p>

yyyy-MM-dd HH:mm:ss z, where z indicates the time zone.

<i>resourcePath</i>	String The path of the resource to be accessed by the request.
<i>serverHost</i>	String The host name or IP address of the Enterprise Gateway Server that received the client request.
<i>serverPort</i>	String The number of the Enterprise Gateway Server port that received the client request.

Output Parameters

None.

pub.security.keystore:getCertificate

WmPublic. Returns the trusted certificate, stored in a truststore, that corresponds to the certificate's alias.

Input Parameters

<i>trustStoreAlias</i>	String Alias for the truststore containing the certificate.
<i>certAlias</i>	String Alias identifying a particular trusted certificate within a truststore.

Output Parameters

<i>certificate</i>	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>	String Alias for the keystore that contains the private key of interest and its certificate.
<i>keyAlias</i>	String Alias for the private key stored in the specified keystore.

Output Parameters

<i>privateKey</i>	java.security.PrivateKey Object representing the private key.
<i>certChain</i>	byte[][] 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>	String Name of the truststore alias.
------------------------	---

Output Parameters

<i>certificates</i>	byte[][] 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>	String Name of the keystore alias.
<i>keyAlias</i>	String 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.

Key	Description
<i>keyStoreAlias</i>	String Name of the keystore alias.
<i>keyAlias</i>	String Alias of the private key.

<i>hashAlgorithm</i>	String The algorithm to use when computing the digest of the provided data. Specify: <ul style="list-style-type: none"> ■ MD5 (the default) ■ SHA-1 ■ SHA-256 ■ SHA-384 ■ SHA-512
<i>data</i>	byte[] Data to be digitally signed.
<i>detachedSignature</i>	String 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"> ■ <code>true</code> to generate a detached signature. ■ <code>false</code> to generate an implicit signature (one that includes the signed data). This is the default.

Output Parameters

<i>signature</i>	byte[] 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 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>	String Key to be associated with the password entry.
<i>value</i>	WmSecureString Password to be stored.

isInternal **String** "true" if this should be saved as an internal password; "false" if it should be saved as a public password. Default is "false". (See "[Internal and Public Passwords](#)" on page 670 for more information.)

Output Parameters

result **String** "true" if password was successfully stored; "false" otherwise.

message **String** "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>	String Key of the password entry to be retrieved.
<i>isInternal</i>	String "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 "Internal and Public Passwords" on page 670.)

Output Parameters

<i>value</i>	WmSecureString Value of the retrieved password.
<i>result</i>	String "true" if the password value was successfully retrieved; "false" otherwise.
<i>message</i>	String "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.

- 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

isInternal **String** "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 "[Internal and Public Passwords](#)" on page 670.)

Output Parameters

key **IData** List of keys in the password store.

result **String** "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

key **String** Key of the password to be removed.

isInternal **String** "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 "[Internal and Public Passwords](#)" on page 670.)

Output Parameters

result **String** "true" if the password was successfully removed; "false" otherwise.

message **String** "successful" or reason for failure.

pub.security.outboundPasswords:updatePassword

WmPublic. Changes the password value for a key already in the password store.

Input Parameters

key **String** Key of the password to be updated.

newPassword **WmSecureString** New password value for the key.

isInternal **String** "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 ["Internal and Public Passwords" on page 670.](#))

Output Parameters

result **String** "true" if the password value was successfully changed; "false" otherwise.

message **String** "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.

Key	Description
<i>certChain</i>	java.security.cert.X509Certificate[] or byte[][] 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>	java.security.PrivateKey or byte[] 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>	String Alias of the certificate chain and private key in the key store. This key is not currently used.
<i>hashAlgorithm</i>	String The algorithm to use when computing the digest of the provided data (SHA-1 or MD5). The default value is MD5.
<i>data</i>	byte[] Data to be digitally signed.
<i>detachedSignature</i>	<p>String 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"> ■ <code>true</code> to generate a detached signature. ■ <code>false</code> to generate an implicit signature (one that includes the signed data). This is the default.

Output Parameters

<i>signature</i>	byte[] 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>	byte[] Signature to use to determine whether the signed data is intact (a DER-encoded representation of the SignedData object as specified in PKCS#7). 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> .
<i>data</i>	byte[] Optional. The data that was signed. If you are processing a detached signature, you must supply <i>data</i> . If you are processing an implicitly signed message, you do not need to supply <i>data</i> because both the message and the signature reside in <i>signature</i> .
<i>detachedSignature</i>	String Optional. Flag indicating whether the message has a detached signature. Set to: <ul style="list-style-type: none">■ <code>true</code> when the message has a detached signature.■ <code>false</code> when the message has an implicit signature. This is the default.
<i>signerCertChain</i>	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 <i>signerCertChain</i> .

Output Parameters

<i>content</i>	byte[] Conditional. The data (for example, the document that was originally signed) extracted from an implicit signature. If you are verifying a detached signature, <i>content</i> is not returned.
----------------	--

Note: The extracted data is returned in *content* even if signature verification fails.

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>	java.security.cert.X509Certificate[] 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>	java.util.Date Time at which the signer signed the data.
<i>trusted</i>	String Flag indicating whether the certificate chain presented by the signer is trusted. A value of: <ul style="list-style-type: none"> ■ <code>true</code> indicates that the chain is trusted. ■ <code>false</code> indicates that the chain is not trusted.
<i>status</i>	String Flag indicating whether the signatures were successfully verified. If successful, <i>status</i> contains <code>verified</code> . If the signatures were not successfully verified, <i>status</i> contains an error message.

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[]** Message for which you want the digest generated.

Output Parameters

output **byte[]** Computed digest.

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.

Key	Description
<i>version</i>	java.lang.Number X509 certificate version number.
<i>serialNumber</i>	String Serial number of the certificate.
<i>signature</i>	String Signature algorithm used by the issuer to sign this certificate.
<i>issuer</i>	Document Detailed information about the CA that signed the certificate, such as name, location, and e-mail address.
<i>validity</i>	Document The time period over which the certificate is valid.
Key	Description
<i>notBefore</i>	String First date on which this

		certificate is valid (for example, 3/15/00 3:36PM).
	<i>notAfter</i>	String Last date on which this certificate is valid (for example, 3/15/00 3:36PM).
<i>subject</i>	Document	Detailed information about the owner of the certificate, such as name, location, and mail address.
<i>subjectPublicKeyAlgorithm</i>	String	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

certificateChain **byte[]** The certificate chain in PKCS #7 format.

Output Parameters

certificates **byte[][]** List of byte arrays in which each `byte[]` in the list contains a certificate from *certificateChain*.

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>	String Java String to be made into a WmSecureString.
<i>bytes</i>	byte[] Byte array to be made into a WmSecureString.
<i>chars</i>	char[] Character array to be made into a WmSecureString.
<i>encoding</i>	String 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

<i>secureString</i>	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

<i>secureString</i>	WmSecureString WmSecureString to be converted.
<i>returnAs</i>	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

<i>string</i>	String The WmSecureString converted to a Java String.
<i>bytes</i>	byte[] The WmSecureString converted to a native Java byte array.
<i>chars</i>	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

<i>secureString</i>	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

<i>xmlData</i>	String Optional. Encrypted XML that needs to be decrypted as plain text.
<i>xmlStream</i>	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* .

<i>keyStoreAlias</i>	String Optional. Alias of the keystore that contains the private key used for decryption.
<i>keyAlias</i>	String Optional. Alias of the private key, contained in the keystore specified by the <i>keyStoreAlias</i> parameter, that is used for decryption.
<i>encoding</i>	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 <i>encoding</i> parameter, Integration Server uses UTF-8. The <i>encoding</i> value must be a valid IANA encoding.

Output Parameters

<i>decryptedXMLData</i>	String Conditional. Decrypted XML data. <i>decryptedXMLData</i> is returned when the input parameter <i>xmlData</i> is provided.
<i>decryptedXMLStream</i>	Object 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>	String Optional. The XML to be encrypted.
<i>xmlStream</i>	InputStream Optional. Input stream to the XML that needs to be encrypted.
	Note: If both <i>xmldata</i> and <i>xmlStream</i> are provided, <i>xmlStream</i> takes precedence.
<i>nodeSelectors</i>	String List XPath to the node to be encrypted. If the value for this parameter is left empty, no XML will be encrypted.
<i>nsDecls</i>	Document 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>	String 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>	Byte[] Optional. The certificate containing the public key that will be used to encrypt the XML. If the input parameters

recipientCert and *recipientID* are both provided, *recipientCert* is used.

contentOnly

Boolean Optional. Indicates whether the XML tags surrounding the content will be encrypted along with the content. Set to:

- `true` to encrypt only the content.
- `false` to encrypt both the tags and the content. This is the default.

algorithm

String Optional. The symmetric key algorithm to use for encryption. Set to:

- `tripledes-cbc` for the algorithm at <http://www.w3.org/2001/04/xmlenc#tripledes-cbc>

This is the default.

- `aes256-cbc` for the algorithm at <http://www.w3.org/2001/04/xmlenc#aes256-cbc>

- `aes192-cbc` for the algorithm at <http://www.w3.org/2001/04/xmlenc#aes192-cbc>

- `aes128-cbc` for the algorithm at <http://www.w3.org/2001/04/xmlenc#aes128-cbc>

Note: If you are using `aes256-cbc` or `aes192-cbc` with JVM 1.6, make sure the unlimited policy jar files have been installed.

encryptedKeyAlgorithm

String 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:

- `rsa-1_5` for the algorithm at http://www.w3.org/2001/04/xmlenc#rsa-1_5

This is the default.

- `rsa-oaep-mgf1p` for the algorithm at <http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p>

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

- encryptedXMLData* **String** Conditional. Encrypted XML data. *encryptedXMLData* is returned when the input parameter *xmlData* is provided.
- encryptedXMLStream* **OutputStream** Conditional. Encrypted XML in the form of an *OutputStream*. *encryptedXMLStream* is returned when the input parameter *xmlStream* is provided.

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>	String Optional. XML that needs to be signed.
<i>xmlStream</i>	InputStream Optional. Input stream containing the XML that needs to be signed. Note: If both <i>xmldata</i> and <i>xmlStream</i> are provided, <i>xmlStream</i> takes precedence.
<i>uri</i>	String Optional. URI to the element to be signed. In combination with the <i>nodeSelectors</i> parameter, the <i>uri</i> identifies the nodes to be signed.
<i>noNamespace SchemaLocation</i>	String Optional. A URI that identifies the location of the XML schema definition that contains the ID attribute specified in <i>uri</i> . 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.
<i>schemaLocations</i>	Document 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. 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. For example, <ul style="list-style-type: none">■ XML namespace = http://www.w3schools.com■ XML schema definition location = file:C:/note.xsd
<i>nodeSelectors</i>	String List 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). Important: Do <i>not</i> use absolute location XPaths here.
<i>nsDecls</i>	Document 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>isEnveloped</i>	<p>String Optional. Indicates whether the signature should be enveloped or enveloping. Set to:</p> <ul style="list-style-type: none">■ <code>True</code> to indicate the signature is enveloped. This is the default.■ <code>False</code> to indicate the signature is enveloping.
<i>signatureNodeSelector</i>	<p>String 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.</p>
<i>signatureAlgorithm</i>	<p>String Optional. Signature algorithm to use when signing the XML node or document. Specify one of the following or use the default value (first algorithm):</p> <ul style="list-style-type: none">■ SHA1 (default)■ SHA256■ SHA384■ SHA512
<i>digestAlgorithm</i>	<p>String Optional. Digest algorithm to use when signing the XML node or document. Specify one of the following or use the default value (first algorithm):</p> <ul style="list-style-type: none">■ SHA1 (default)■ SHA256■ SHA384■ SHA512
<i>canonicalizationAlgorithm</i>	<p>String Optional. Canonical algorithm used with the XML. Specify one of the following or use the default value (first algorithm):</p> <ul style="list-style-type: none">■ http://www.w3.org/TR/2001/rec-xml-c14n-20010315 (default)■ http://www.w3.org/TR/2001/rec-xml-c14n-20010315#WithComments■ http://www.w3.org/2001/10/xml-exc-c14n#

	<ul style="list-style-type: none">■ http://www.w3.org/2001/10/xml-exc-c14n#WithComments
<i>signatureId</i>	String Optional. ID attribute for the signature node.
<i>keyStoreAlias</i>	String Optional. Name (alias) of the keystore that contains the private key/certificate.
<i>keyAlias</i>	String 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>	String Optional. Indicates whether the certificate chain should be included in the signature. Set to: <ul style="list-style-type: none">■ <code>True</code> to include the certificate chain in the signature.■ <code>False</code> to leave the certificate chain out of the signature. This is the default.
<i>certData</i>	String 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">■ <code>X509_CERTIFICATE</code> (default)■ <code>X509_SUBJECT_NAME</code>■ <code>X509_ISSUER_SERIAL</code>■ <code>X509_SKI</code>
<i>idXmlObject</i>	String Optional. Specifies the ID for the node that holds the original XML that is signed. Applicable only for enveloping signatures.
<i>encoding</i>	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 <i>encoding</i> parameter, Integration Server uses UTF-8. The <i>encoding</i> value must be a valid IANA encoding.
<i>addSignatureAsLastElement</i>	Boolean 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:

- `True` to add the signature element as the last child of the root.
- `False` to add the signature element as the first child of the root. This is the default.

Output Parameters

<i>signedXMLData</i>	String Conditional. Signed XML data. <i>signedXMLData</i> is returned when <i>xmlData</i> is provided.
<i>signedXMLStream</i>	OutputStream 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>	String Optional. Signed XML that needs to be verified.
<i>xmlStream</i>	InputStream Optional. Signed XML as an input stream that needs to be verified.
	Note: If both <i>xmldata</i> and <i>xmlStream</i> are provided, <i>xmlStream</i> takes precedence.
<i>signatureSelectors</i>	String Array 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>	Document 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>noNamespaceSchemaLocation</i>	String 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>	Document Optional. Holds the schema locations against the namespaces.
<i>encoding</i>	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 <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>	Boolean Indicates whether the signed XML is authentic (true) or cannot be verified or shows signs of tampering (false).
---------------------------	--

<i>failedSignatureSelector</i>	String Conditional. In case of a verification failure (the digests do not equate), indicates which signature selector failed.
<i>failureReason</i>	String 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">■ SIGNATURE FAILED■ CERTIFICATE NOT TRUSTED
<i>certMap</i>	Document List 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	694
--	-----

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
pub.smime:createCertsOnlyData	WmPublic. Generates a PKCS #7 certs-only S/MIME entity from an array of certificates.
pub.smime:createEncryptedData	WmPublic. Encrypts a MIME message.
pub.smime:createSignedAndEncryptedData	WmPublic. <i>Deprecated</i> - Replaced by pub.smime.keystore:createSignedAndEncryptedData . Digitally signs a MIME message and then encrypts it.
pub.smime:createSignedData	WmPublic. <i>Deprecated</i> - Replaced by pub.smime.keystore:createSignedData . Digitally signs a MIME message.
pub.smime:processCertsOnlyData	WmPublic. Extracts the certificates from a PKCS #7 certs-only S/MIME entity.
pub.smime:processEncryptedData	WmPublic. <i>Deprecated</i> - Replaced by pub.smime.keystore:processEncryptedData .
pub.smime:processSignedData	WmPublic. Verifies the signature from a signed S/MIME entity and extracts the message from it.
pub.smime.keystore:createSignedAndEncryptedData	WmPublic. Digitally signs and encrypts a MIME message.
pub.smime.keystore:createSignedData	WmPublic. Creates signed S/MIME data.

Element	Package and Description
pub.smime.keystore:processEncryptedData	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>	String Optional. Code specifying the encryption algorithm to use. Must be <code>TripleDES</code> (default), <code>DES</code> , or <code>RC2</code> .
<i>keyLength</i>	String 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

<i>SMimeEnvStream</i>	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

<i>envStream</i>	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.

encryptionAlg **String** Optional. Code specifying the encryption algorithm to use. Must be `TripleDES` (default), `DES`, or `RC2`.

keyLength **String** 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

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

SMimeEnvoStream

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

SMimeEncStream **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

SMimeEncStream **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.

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.)

<i>encrypted</i>	<p>String Conditional. Flag indicating whether the decrypted MIME entity is encrypted. A value of:</p> <ul style="list-style-type: none"> ■ <code>true</code> indicates that the MIME entity is encrypted. ■ <code>false</code> indicates that the MIME entity is not encrypted.
<i>signed</i>	<p>String Conditional. Flag indicating whether the decrypted MIME entity is signed. A value of:</p> <ul style="list-style-type: none"> ■ <code>true</code> indicates that the MIME entity is signed. ■ <code>false</code> indicates that the MIME entity is not signed.
<i>certsOnly</i>	<p>String Conditional. Flag indicating whether the decrypted MIME entity is a certs-only entity. A value of:</p> <ul style="list-style-type: none"> ■ <code>true</code> indicates that the MIME entity is a certs-only entity. ■ <code>false</code> indicates that the MIME entity is not a certs-only entity.
<i>stream</i>	<p>java.io.InputStream Conditional. The decrypted MIME entity.</p>

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

SMimeEnvoStream **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:

- `true` indicates that the MIME entity is encrypted.
- `false` indicates that the MIME entity is not encrypted.

signed **String** Conditional. Flag indicating whether the extracted MIME entity is signed. A value of:

	<ul style="list-style-type: none"> ■ <code>true</code> indicates that the MIME entity is signed. ■ <code>false</code> indicates that the MIME entity is not signed. 						
<i>certsOnly</i>	<p>String Conditional. Flag indicating whether the extracted MIME entity is a certs-only entity. A value of:</p> <ul style="list-style-type: none"> ■ <code>true</code> indicates that the MIME entity is a certs-only entity. ■ <code>false</code> indicates that the MIME entity is not a certs-only entity. 						
<i>stream</i>	<p>java.io.InputStream Conditional. Extracted MIME entity.</p>						
<i>verify</i>	<p>String 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"> ■ <code>true</code> indicates that signature processing was successful. ■ <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. 						
<i>trusted</i>	<p>String Flag indicating whether the signer certificate is trusted or not. A value of:</p> <ul style="list-style-type: none"> ■ <code>true</code> indicates that the signer certificate is trusted. ■ <code>false</code> indicates that the signer certificate is not trusted. 						
<i>errorCode</i>	<p>String 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>String Conditional. Textual error message indicating what kind of error occurred while processing the signature. Error codes and messages are as follows:</p> <table> <thead> <tr> <th><u><i>errorCode</i></u></th> <th><u><i>errorMessage</i></u></th> </tr> </thead> <tbody> <tr> <td>1</td> <td>Invalid signer certificate file information.</td> </tr> <tr> <td>2</td> <td>Certificate at index 'i' is not in recognizable format.</td> </tr> </tbody> </table>	<u><i>errorCode</i></u>	<u><i>errorMessage</i></u>	1	Invalid signer certificate file information.	2	Certificate at index 'i' is not in recognizable format.
<u><i>errorCode</i></u>	<u><i>errorMessage</i></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>	java.io.InputStream The MIME message that you want to sign and encrypt (for example, the output produced by pub.mime:getEnvelopeStream).
<i>keyStoreAlias</i>	String Alias of the keystore containing the signing key.
<i>keyAlias</i>	String Alias of the private key to be used for signing.
<i>explicit</i>	<p>String Optional. Flag indicating whether an implicit or explicit signature is to be generated. Set to:</p> <ul style="list-style-type: none"> ■ <code>True</code> to generate an explicit (detached) signature. This is the default. ■ <code>False</code> to generate an implicit signature.
<i>recipientCerts</i>	Object List A list of <code>byte[]</code> for the partner certificates for whom this message will be encrypted.
<i>encryptionAlg</i>	String Optional. Code specifying the encryption algorithm to use. Must be TripleDES (default), DES, or RC2.
<i>keyLength</i>	<p>String Optional. Length of the encryption key for RC2 encryption. Must be 40, 64, or 128 (default).</p> <p>This parameter is ignored if <code>encryptionAlg</code> is not RC2.</p>

Output Parameters

<i>SMimeEnvStream</i>	java.io.InputStream 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>	java.io.InputStream MIME message that you want to sign (for example, the output produced by pub.mime:getEnvelopeStream).
<i>keyStoreAlias</i>	String Alias of the keystore.
<i>keyAlias</i>	String Alias of the private key of interest in the keystore.
<i>explicit</i>	String Optional. Flag indicating whether an implicit or explicit signature is generated. Set to: <ul style="list-style-type: none">■ <code>True</code> to generate an explicit (detached) signature. This is the default.■ <code>False</code> to generate an implicit signature.

Output Parameters

<i>SMimeEnvStream</i>	java.io.InputStream 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>	java.io.InputStream The encrypted S/MIME stream.
<i>keyStoreAlias</i>	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	710
--	-----

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
pub.soap.handler:addBodyBlock	WmPublic. Inserts a document into a SOAP message as a new body block.
pub.soap.handler:addFaultBlock	WmPublic. Inserts a document into a SOAP message as a new fault block.
pub.soap.handler:addHeaderBlock	WmPublic. Inserts a document into a SOAP message as a new header block.
pub.soap.handler:addHeaderElement	WmPublic. <i>Deprecated</i> - Replaced by pub.soap.handler:addHeaderBlock .
pub.soap.handler:generateDocumentTypesFromWSDL	WmPublic. Generates IS document types from a WSDL.
pub.soap.handler:getBodyBlock	WmPublic. Retrieves a body block from a SOAP message.
pub.soap.handler:getBodyBlockQNames	WmPublic. Returns the QName for each body block in a SOAP message.
pub.soap.handler:getFaultBlock	WmPublic. Retrieves a fault block from a SOAP message.
pub.soap.handler:getHeaderBlock	WmPublic. Retrieves a header block from a SOAP message.
pub.soap.handler:getHeaderBlockQNames	WmPublic. Returns the QName for each header block in a SOAP message.

Element	Package and Description
pub.soap.handler:getHeaderElement	WmPublic. <i>Deprecated</i> - Replaced by pub.soap.handler:getHeaderBlock .
pub.soap.handler:getInitialSOAPRequest	WmPublic. Gets the initial SOAP request message from a given message context.
pub.soap.handler:getMessageAddressingProperties	WmPublic. Gets the value of the specified message addressing properties in a SOAP message.
pub.soap.handler:getProperty	WmPublic. Gets the value of a specified property from a message context.
pub.soap.handler:getServicePipeline	WmPublic. Gets the service pipeline from a given message context.
pub.soap.handler:getSOAPMessage	WmPublic. Gets the message addressing properties of the SOAP message in the provided message context.
pub.soap.handler:getWebServiceInvocationProperties	WmPublic. Fetches the web service invocation properties.
pub.soap.handler:handlerSpec	WmPublic. Specification to use as the signature for a service that acts as a header handler.
pub.soap.handler:hasFaultMessage	WmPublic. Determines whether the SOAP message in a given message context contains a SOAP fault message.
pub.soap.handler:listConsumer	WmPublic. Returns a list of the consumer handlers currently registered on Integration Server.

Element	Package and Description
pub.soap.handler:listProvider	WmPublic. Returns a list of the provider handlers currently registered on Integration Server.
pub.soap.handler:registerConsumer	WmPublic. <i>Deprecated</i> - Replaced by pub.soap.handler:registerWmConsumer .
pub.soap.handler:registerProvider	WmPublic. <i>Deprecated</i> - Replaced by pub.soap.handler:registerWmProvider .
pub.soap.handler:registerWmConsumer	WmPublic. Registers a handler for use with consumer.
pub.soap.handler:registerWmProvider	WmPublic. Registers a header handler for use with provider.
pub.soap.handler:removeBodyBlock	WmPublic. Removes a body block from a SOAP message, leaving an empty SOAP body, <SOAP-ENV:Body></SOAP-ENV:Body>.
pub.soap.handler:removeHeaderBlock	WmPublic. Removes a header block from a SOAP message.
pub.soap.handler:removeHeaderElement	WmPublic. <i>Deprecated</i> - Replaced by pub.soap.handler:removeHeaderBlock .
pub.soap.handler:removeProperty	WmPublic. Removes a property from a message context.
pub.soap.handler:setProperty	WmPublic. Sets the value of a specific property in a message context.
pub.soap.handler:setSOAPMessage	WmPublic. Sets the SOAP message in a message context.
pub.soap.handler:unregisterConsumer	WmPublic. Unregisters a consumer web service descriptor handler.

Element	Package and Description
pub.soap.handler:unregisterProvider	WmPublic. Unregisters a provider web service descriptor handler.
pub.soap.handler:updateFaultBlock	WmPublic. Updates the fault code (<i>code</i>) and the fault string (<i>reasons</i>) in the existing fault block.
pub.soap.processor:list	WmPublic. <i>Deprecated</i> - There is no replacement service. Returns a list of the SOAP processors that are currently registered on the Integration Server.
pub.soap.processor:processMessage	WmPublic. <i>Deprecated</i> - There is no replacement service. Executes the Integration Server's default SOAP processor.
pub.soap.processor:processRPCMessage	WmPublic. <i>Deprecated</i> - There is no replacement service. Executes the Integration Server's SOAP RPC processor.
pub.soap.processor:registerProcessor	WmPublic. <i>Deprecated</i> - There is no replacement service. Registers a service as a SOAP processor on the Integration Server.
pub.soap.processor:unregisterProcessor	WmPublic. <i>Deprecated</i> - There is no replacement service. Unregisters a SOAP processor by removing it from the registry.
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.

Element	Package and Description
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.
pub.soap.utils:addHeaderEntry	WmPublic. Inserts an entry into the header element of a SOAP message.
pub.soap.utils:addTrailer	WmPublic. Inserts a trailer in a SOAP message.
pub.soap.utils:callbackServiceSpec	WmPublic. Defines the input signature for an outbound callback service.
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.
pub.soap.utils:createSoapData	WmPublic. Creates a SOAP object consisting of SOAP envelope, body, and header entries.
pub.soap.utils:createXOPObject	WmPublic. Generates a <code>com.wm.util.XOPObject</code> instance from a <code>base64Binary</code> string, a byte array, or an input stream.
pub.soap.utils:exitUnableToUnderstand	WmPublic. Terminates processing and returns a <code>mustUnderstand</code> fault to the client.
pub.soap.utils:getActor	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
pub.soap.utils:getBody	WmPublic. Retrieves the body from a SOAP message as a single node object.
pub.soap.utils:getBodyEntries	WmPublic. Retrieves the body entries from a SOAP message as an array of node objects.
pub.soap.utils:getDocument	WmPublic. Retrieves an entire SOAP message as a node object.
pub.soap.utils:getEncoding	WmPublic. Retrieves the encoding from a SOAP message as a single string.
pub.soap.utils:getHeader	WmPublic. Retrieves the header from a SOAP message as a single node object.
pub.soap.utils:getHeaderEntries	WmPublic. Retrieves the header entries from a SOAP message as an array of node objects.
pub.soap.utils:getMustUnderstand	WmPublic. Returns the mustUnderstand status for a given header entry.
pub.soap.utils:getQName	WmPublic. Returns the qualified name for a given node.
pub.soap.utils:getTrailers	WmPublic. Retrieves the trailers from a SOAP message.
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.
pub.soap.utils:QName	WmPublic. Document type that defines the structure of a qualified name.

Element	Package and Description
pub.soap.utils:removeBodyEntry	WmPublic. Deletes a body entry from a SOAP message.
pub.soap.utils:removeHeaderEntry	WmPublic. Deletes a header entry from a SOAP message.
pub.soap.utils:removeTrailer	WmPublic. Deletes a trailer from a SOAP message.
pub.soap.utils:requestResponseSpec	WmPublic. Defines the input/output signature for a custom processor and a target service for the default processor.
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.
pub.soap.utils:setWSDEffectivePolicy	WmPublic. Sets the effective policy for a handler in a web service descriptor.
pub.soap.utils:soapDataToBytes	WmPublic. Converts a SOAP object to a Byte Array.
pub.soap.utils:soapDataToString	WmPublic. Converts a SOAP object to a String.
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.
pub.soap.utils:streamToSoapData	WmPublic. Converts an InputStream containing a SOAP message to a SOAP object.

Element	Package and Description
pub.soap.utils:stringToSoapData	WmPublic. Converts a String containing a SOAP message to a SOAP object.
pub.soap.utils:validateSoapData	WmPublic. Verifies that a SOAP object represents a valid SOAP message.
pub.soap.wsa:action	WmPublic. Document type that defines the contents of the <code>wsa:Action</code> WS-Addressing header.
pub.soap.wsa:faultTo	WmPublic. Document type that defines the contents of the <code>wsa:FaultTo</code> WS-Addressing header.
pub.soap.wsa:from	WmPublic. Document type that contains the details of the source of the message.
pub.soap.wsa:messageID	WmPublic. Document type that defines the contents of the <code>wsa:MessageID</code> WS-Addressing header.
pub.soap.wsa:problemAction	WmPublic. Document type that captures additional information about faults.
pub.soap.wsa:problemHeaderQName	WmPublic. Document type that captures additional information about faults.
pub.soap.wsa:problemIRI	WmPublic. Document type that captures the IRI that caused the problem.
pub.soap.wsa:relatesTo	WmPublic. Document type that defines the contents of the <code>wsa:RelatesTo</code> WS-Addressing header.

Element	Package and Description
pub.soap.wsa:replyTo	WmPublic. Document type that defines the contents of the <code>wsa:ReplyTo</code> WS-Addressing header.
pub.soap.wsa:retryAfter	WmPublic. Document type that specifies the minimum duration in milliseconds that Integration Server waits before retransmitting a message.
pub.soap.wsa:to	WmPublic. Document type that defines the contents of the <code>wsa:To</code> WS-Addressing header.
pub.soap.wsa:schema_wsa	WmPublic. A schema containing the elements from <code>http://www.w3.org/2005/08/addressing</code> namespace.
pub.soap.wsa.submission:action	WmPublic. Document type that defines the contents of the <code>wsa:Action</code> WS-Addressing header.
pub.soap.wsa.submission:faultTo	WmPublic. Document type that defines the contents of the <code>wsa:FaultTo</code> WS-Addressing header.
pub.soap.wsa.submission:from	WmPublic. Document type that contains the details about the source of the message.
pub.soap.wsa.submission:messageID	WmPublic. Document type that defines the contents of the <code>wsa:MessageID</code> WS-Addressing header.
pub.soap.wsa.submission:relatesTo	WmPublic. Document type that defines the contents of the <code>wsa:RelatesTo</code> WS-Addressing header.

Element	Package and Description
pub.soap.wsa.submission:replyTo	WmPublic. Document type that specifies the destination to which the response message is to be sent.
pub.soap.wsa.submission:retryAfter	WmPublic. Document type that specifies the minimum duration in milliseconds that Integration Server waits before retransmitting a message.
pub.soap.wsa.submission:to	WmPublic. Document type that defines the contents of the <code>wsa:To</code> WS-Addressing header.
pub.soap.wsa.submission:schema_wsa_submission	WmPublic. A schema containing the elements from <code>http://schemas.xmlsoap.org/ws/2004/08/addressing</code> namespace.
pub.soap.wsrn:closeSequence	WmPublic. Closes a reliable messaging sequence.
pub.soap.wsrn:createSequence	WmPublic. Sends a request to a reliable messaging destination to create a new reliable messaging sequence.
pub.soap.wsrn:sendAcknowledgementRequest	WmPublic. Requests an acknowledgment for a message sequence.
pub.soap.wsrn:terminateSequence	WmPublic. Terminates a reliable messaging sequence.
pub.soap.wsrn:waitUntilSequenceCompleted	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>Document 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 pub.soap.utils:QName 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>	String Namespace name for the web service operation.
<i>localName</i>	String Name of the web service operation on which the handler is being written.
<i>bodyDocument</i>	Document 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>String 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"> ■ <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. ■ <code>false</code> to have the service use the values you specify in the <i>operationQName</i> parameters. This is the default. <p>Note: 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>String 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"> ■ <code>true</code> to validate <i>inputBodyBlock/bodyDocument</i> . If the validation fails, an exception is thrown. ■ <code>false</code> to skip validating <i>inputBodyBlock/bodyDocument</i> . This is the default.

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>	Document Contains the fault code and possible subcodes.

<u>Key</u>	<u>Descriptions</u>
------------	---------------------

<i>namespaceName</i>	String Namespace name for the SOAP fault code.
----------------------	---

<i>localName</i>	String Code that identifies the fault.
------------------	---

<i>subCodes</i>	<p>Document List 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"> ■ <i>namespaceName</i> for the subcode ■ <i>localName</i> that identifies the subcode
-----------------	---

<i>reasons</i>	Document List 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>	String Text explaining the cause of the fault.
--------------	---

	<i>@lang</i>	String Optional. Language for the human readable description.
	<i>node</i>	String Optional. URI to the SOAP node where the fault occurred.
	<i>role</i>	String Optional. Role in which the node was operating at the point the fault occurred.
	<i>detail</i>	Document Optional. Application-specific details about the SOAP fault.
<i>validate</i>		<p>String 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"> ■ <code>true</code> to validate <i>soapFault/detail</i> . If the validation fails, an exception is thrown. ■ <code>false</code> to skip validating <i>soapFault/detail</i> This is the default.

Output Parameters

<i>status</i>	<p>String Flag indicating the outcome of the service. A value of:</p> <ul style="list-style-type: none"> ■ <code>true</code> indicates that adding the fault block succeeded. ■ <code>false</code> indicates that adding the fault block failed. <p>Note: If the SOAP message already contains a fault block, the service sets <i>status</i> to false.</p>
---------------	--

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>String 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"> ■ <code>true</code> to indicate that the attribute <code>mustUnderstand="true"</code> will be added

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*.

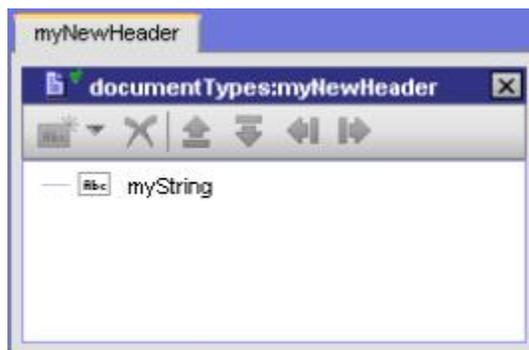
For more information about the `mustUnderstand` and `actor` attributes in SOAP 1.1, see the *Simple Object Access Protocol (SOAP) 1.1 - W3C Note 08 May 2000* at <http://www.w3.org/TR/SOAP/>.

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.

wSDLString **String** Optional. WSDL document as a string.

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>targetPackageName</i>	String Name of the package in which to place the generated IS document types on Integration Server.
<i>targetFolderName</i>	String 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>	<p>String Optional. Flag indicating whether the service should generate the documents corresponding to headers in the SOAP messages described in the WSDL document. Set to:</p> <ul style="list-style-type: none"> ■ <code>true</code> to generate the documents. This is the default. ■ <code>false</code> to skip generating the documents.
<i>generateBodyDocs</i>	<p>String 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:</p> <ul style="list-style-type: none"> ■ <code>true</code> to generate the documents. ■ <code>false</code> to skip generating the documents. This is the default.
<i>generateFaultDocs</i>	<p>String 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:</p> <ul style="list-style-type: none"> ■ <code>true</code> to generate the documents. ■ <code>false</code> to skip generating the documents. This is the default.
<i>generateXOPObjectForBase64Binary</i>	<p>String 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:</p> <ul style="list-style-type: none"> ■ <code>true</code> to generate fields of type <code>com.wm.util.XOPObject</code>. ■ <code>false</code> to skip generating fields of type <code>com.wm.util.XOPObject</code>. This is the default.
<i>contentModelCompliance</i>	<p>String Optional. Flag that specifies how strictly the service represents content models from the XML Schema definition in the generated IS document types. Set to:</p> <ul style="list-style-type: none"> ■ <code>strict</code> 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

messageContext

Object Message context containing the SOAP message from which to retrieve the 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 SOAP body block to be retrieved from the SOAP message.

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 body block; that is, the conversion will not consider style or use.

validate

String Optional. Flag that indicates whether you want the service to validate the document returned in the *outputBodyBlock/bodyDocument* parameter 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 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>Document 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 pub.soap.util:QName document type.</p>						
	<table border="1"> <thead> <tr> <th>Key</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td><i>namespaceName</i></td> <td>String Namespace name for the web service operation.</td> </tr> <tr> <td><i>localName</i></td> <td>String Name of the web service operation on which the handler is being written.</td> </tr> </tbody> </table>	Key	Description	<i>namespaceName</i>	String Namespace name for the web service operation.	<i>localName</i>	String Name of the web service operation on which the handler is being written.
Key	Description						
<i>namespaceName</i>	String Namespace name for the web service operation.						
<i>localName</i>	String 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>	Document The retrieved fault block.
------------------	--

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

<i>code</i>	Document Contains the fault code and possible subcodes.
-------------	--

<u>Key</u>	<u>Descriptions</u>
------------	---------------------

<i>namespaceName</i>	String Namespace name for the SOAP fault code.						
<i>localName</i>	String Code that identifies the fault.						
<i>subCodes</i>	<p>Document List Subcodes that provide further detail. Each Document in the <i>subCodes</i> Document List contains:</p> <ul style="list-style-type: none"> ■ <i>namespaceName</i> for the subcode ■ <i>localName</i> that identifies the subcode 						
<i>reasons</i>	Document List 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 style="text-decoration: underline;">Key</th> <th style="text-decoration: underline;">Descriptions</th> </tr> </thead> <tbody> <tr> <td><i>*body</i></td> <td>String Text explaining the cause of the fault.</td> </tr> <tr> <td><i>@lang</i></td> <td>String Optional. Language for the human readable description.</td> </tr> </tbody> </table>	Key	Descriptions	<i>*body</i>	String Text explaining the cause of the fault.	<i>@lang</i>	String Optional. Language for the human readable description.
Key	Descriptions						
<i>*body</i>	String Text explaining the cause of the fault.						
<i>@lang</i>	String Optional. Language for the human readable description.						
<i>node</i>	String URI to the SOAP node where the fault occurred.						
<i>role</i>	String Role in which the node was operating at the point the fault occurred.						
<i>detail</i>	Document 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

messageContext **Object** Message context containing the SOAP message from which to retrieve 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 Fully qualified name of the IS document type that specifies the structure to apply to the content of the 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.

Integration Server uses the QName of the first field of the document type to determine which header block to retrieve from the SOAP message.

validate

String Optional. Flag that indicates whether you want the service to validate the document returned in the *outputHeaderBlock/headerDocument* parameter against the IS document type specified in the *documentType* input parameter.

Set *validate* to:

- `true` to validate *outputHeaderBlock/headerDocument*. If the validation fails, an exception is thrown.
- `false` to skip validating *outputHeaderBlock/headerDocument*. This is the default.

Output Parameters

outputHeaderBlock

Document List 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.

Key	Description
<i>mustUnderstand</i>	String Flag indicating whether the <code>mustUnderstand</code> attribute is present in the retrieved header block. A value of:

	<ul style="list-style-type: none"> ■ <code>true</code> indicates that the header block contained <code>mustUnderstand="true"</code>. ■ <code>false</code> indicates that the <code>mustUnderstand</code> attribute was absent from the header block or <code>mustUnderstand="false"</code>.
<i>role</i>	<p>String 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>Document 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

messageContext **Object** Message context containing the SOAP message from which to retrieve 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.

<i>QName</i>	<p>Document Optional. Qualified name (namespace name and local name) of the header element to retrieve.</p> <p>The <i>QName</i> document references the pub.soap.utils:QName document type. If you do not specify <i>QName</i>, you must specify <i>documentType</i>.</p>
<i>documentType</i>	<p>String 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>
```

```

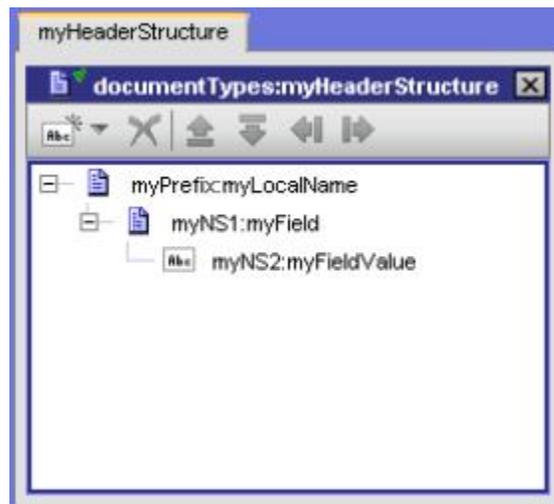
<SOAP-ENV:Body>
...
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

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> <code>userHandlerNamespaceName</code>
	<i>localName</i> <code>userHandlerLocalName</code>
<i>documentType</i>	<code>documentTypes:myHeaderStructure</code>

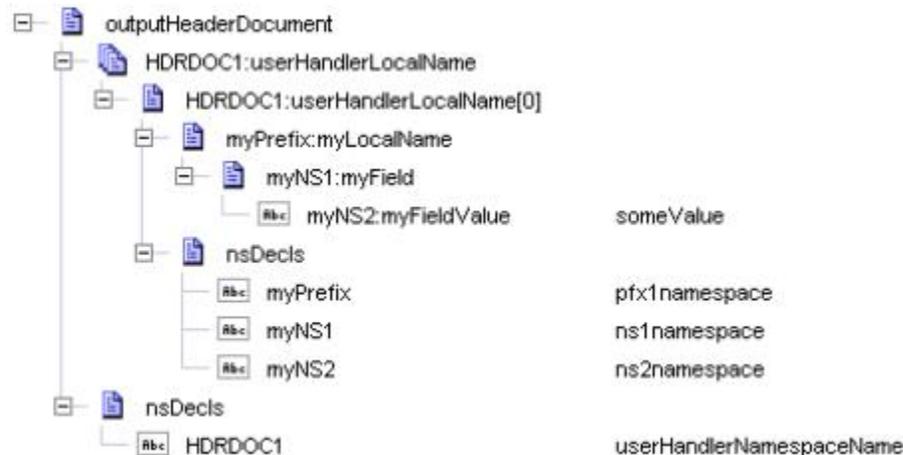
The structure of `documentTypes:myHeaderStructure` looks like this:



The prefixes in `documentTypes:myHeaderStructure` refer to the following namespaces.

Prefix	Namespace
<code>myPrefix</code>	<code>pfx1namespace</code>
<code>myNS1</code>	<code>ns1namespace</code>
<code>myNS2</code>	<code>ns2namespace</code>

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

*messageAddressing
Properties*

Document Value of the specified message addressing property.

<u>Key</u>	<u>Description</u>						
<i>messageID</i>	String Conditional. Unique identifier of the SOAP message.						
<i>relatesTo</i>	Document List 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>String Message ID of the related message.</td> </tr> <tr> <td><i>relationshipType</i></td> <td>String Conditional. The relationship type.</td> </tr> </tbody> </table>	<u>Key</u>	<u>Description</u>	<i>value</i>	String Message ID of the related message.	<i>relationshipType</i>	String Conditional. The relationship type.
<u>Key</u>	<u>Description</u>						
<i>value</i>	String Message ID of the related message.						
<i>relationshipType</i>	String Conditional. The relationship type.						
<i>action</i>	String Conditional. WS-Addressing action specified in the message addressing property of the SOAP message.						
<i>to</i>	<p>Document Conditional. The endpoint reference that specifies the address of the intended receiver of the SOAP message. The <i>to</i> endpoint reference includes:</p> <ul style="list-style-type: none"> ■ <i>attributes</i>, which includes namespaceName, localname, and their values. ■ <i>address</i> and its attributes and values. ■ <i>referenceParameters</i> ■ <i>metadata</i> and its attributes and elements ■ <i>extensibleElements</i>, which are any other elements usually provided for future extensions. 						
<i>from</i>	Document Conditional. The endpoint reference that specifies the source of the SOAP message. The <i>from</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.

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

<i>messageContext</i>	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.
<i>key</i>	String Name of the selected property whose value to retrieve.

Output Parameters

<i>value</i>	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

<i>messageContext</i>	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.

For this provider handler service...

servicePipeline contains...

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		<p>The output pipeline of the endpoint service that corresponds to the invoked web service operation.</p> <p><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.</p>
Fault handler service		<p>The output pipeline of the endpoint service that corresponds to the invoked web service operation.</p> <p><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.</p>

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

<i>soapAction</i>	String SOAP action associated with the SOAP message request.
<i>binderName</i>	String Name of the WSBinder being invoked.
<i>binding</i>	String Binding against which the service was invoked.
<i>portType</i>	String Port type against which the service was invoked.
<i>operationQName</i>	Document Qualified name for the web service wrapper. The service only returns <i>operationQName</i> for RPC/Encoded and RPC/Literal services.

Key	Description
<i>namespaceName</i>	String Namespace name for the web service operation.
<i>localName</i>	String Name of the web service operation that was invoked.

<i>style</i>	<p>String Style of the SOAP message. Possible values are:</p> <ul style="list-style-type: none"> ■ document ■ rpc
<i>use</i>	<p>String Use of the SOAP message. Possible values are:</p> <ul style="list-style-type: none"> ■ literal ■ encoded
<i>inbound</i>	<p>String Flag indicating whether the message is incoming or outgoing. A value of:</p> <ul style="list-style-type: none"> ■ <code>true</code> indicates that the SOAP message is incoming. ■ <code>false</code> indicates that the SOAP message is outgoing.
<i>soapVersion</i>	<p>String SOAP version of the message. Possible values are:</p> <ul style="list-style-type: none"> ■ SOAP11

■ SOAP12

pub.soap.handler:handlerSpec

WmPublic. Specification to use as the signature for a service that acts as a header handler.

Input Parameters

messageContext **Object** 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

statusCode **String** Specifies the status of the handler service execution which indicates the action Integration Server will take next. The *statusCode* 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>	String Descriptive name assigned to the consumer handler when it was registered.
<i>className</i>	String Class name of the handler.
<i>policyType</i>	String Conditional. Policy type of the handler. <i>policyType</i> does not apply to service handlers.
<i>handleRequestService</i>	String 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.
<i>handleResponseService</i>	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.
<i>handleFaultService</i>	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>	String Namespace portion of the header's qualified name.
<i>localName</i>	String 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>	String Descriptive name given to the provider handler when it was registered.
<i>className</i>	String Class name of the handler.
<i>policyType</i>	String Conditional. Policy type of the handler. <i>policyType</i> does not apply to service handlers.

handleRequestService **String** 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.

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 provider 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 provider handler.

Headers **Document List** List of QNames for the headers registered with the provider handler.

<u>Key</u>	<u>Description</u>
<i>namespace</i>	String Namespace portion of the header's qualified name.
<i>localName</i>	String 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

descriptiveName **String** Name to assign to the consumer header handler. Each consumer header handler must have a unique name.

QNameList **Document List** Optional. Qualified names of the headers on which the handler operates. In the document list, each document references the [pub.soap.utils:QName](#) document type.

handleRequestService **String** Optional. Fully qualified name of the service to use as the request header handler.

handleResponseService **String** Optional. Fully qualified name of the service to use as the response header handler.

handleFaultService **String** 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 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: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>	String Name to assign to the provider header handler. Each provider header handler must have a unique name.
<i>QNameList</i>	Document List Optional. Qualified names of the headers on which the handler operates. In the document list, each document references the pub.soap.utils:QName document type.
<i>handleRequestService</i>	String Optional. Fully qualified name of the service to use as the request header handler.
<i>handleResponseService</i>	String Optional. Fully qualified name of the service to use as the response header handler.
<i>handleFaultService</i>	String 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

messageContext

Object Message context that contains the SOAP message from which to remove a header 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 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.

QName

Object Optional. Qualified name of the header block to remove. The *QName* document references the [pub.soap.util:QName](#) document type.

Note: Either *QName* or *documentType* must be supplied.

documentType **String** 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 QName of the first element in the IS document type to determine which header block to remove.

Note: Either *QName* or *documentType* must be supplied.

index **String** Optional. An integer representing the position of the header block entry with the specified QName to remove. A SOAP message can contain multiple header blocks with the same QName. If you specify an index, Integration Server removes that occurrence of the header block. Note that 0 (zero) represents the first header block with the specified QName.

If *index* is not provided, Integration Server removes all the header blocks with a matching QName.

For example, if *QName* is myNSName:myLocalName and *index* is 1, Integration Server removes the second occurrence of the header block with the QName myNSName:myLocalName.

Output Parameters

status **String** Indicates whether Integration Server removed the header block.

Key	Description
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>	Object Message context that contains the SOAP message from which to remove a header element (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 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.
<i>QName</i>	Object Qualified name of the header element (block) to remove. The <i>QName</i> document references the pub.soap.utils:QName document type.

Output Parameters

<i>status</i>	String Indicates whether Integration Server successfully removed the header element. A value of: <ul style="list-style-type: none">■ <code>True</code> indicates that Integration Server removed the header element.■ <code>False</code> indicates that Integration Server did not remove the header element because the SOAP message did not contain a header element that matched the provided <i>QName</i>.
---------------	--

See Also

[pub.soap.handler:addHeaderElement](#)
[pub.soap.handler:getHeaderElement](#)

pub.soap.handler:removeProperty

WmPublic. Removes a property from a message context.

Input Parameters

<i>messageContext</i>	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.
<i>key</i>	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>	Object 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>	String Name of the property to set.
<i>value</i>	Object 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

descriptiveName **String** 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

descriptiveName **String** 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>	Document List 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>	String Text explaining the cause of the fault.
--------------	---

<i>@lang</i>	String Optional. Language for the human readable description.
--------------	--

<i>code</i>	Document Optional. Contains the fault code and possible subcodes.
-------------	--

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

<i>namespaceName</i>	String Namespace name for the SOAP fault code.
----------------------	---

<i>localName</i>	String Code that identifies the fault.
------------------	---

<i>subCodes</i>	Document List Optional. Subcodes that provide further detail.
-----------------	--

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

<i>namespaceName</i>	String Namespace name for the subcode.
----------------------	---

<i>localName</i>	String Code that identifies the subcode.
------------------	---

<i>node</i>	String Optional. URI to the SOAP node where the fault occurred.
-------------	--

role **String** Optional. Role in which the node was operating at the point the fault occurred.

Output Parameters

status **String** Flag indicating whether the fault block was updated successfully. A value of:

- `true` indicates that the fault block was successfully updated.
- `false` indicates that either a fault block is not present in the *messageContext* parameter or that the fault block was not successfully updated.

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>	String Process directive that is assigned to the SOAP processor.
<i>svcName</i>	String Fully qualified name of the service that functions as the SOAP processor.
<i>descriptiveName</i>	String 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>String 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"> ■ <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. This setting overrides the server's global <code>watt.server.SOAP.validateSOAPMessage</code> setting. ■ indicates that messages are not validated by the SOAP message handler. This setting overrides the server's global <code>watt.server.SOAP.validateSOAPMessage</code> setting. <p>If <i>validateSOAPMessage</i> is null, message validation for the processor is determined by the server's <code>watt.server.SOAP.validateSOAPMessage</code> setting.</p>

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

<i>soapRequestData</i>	Object SOAP object containing the SOAP request submitted to the Integration Server by a client.
<i>soapResponseData</i>	Object Empty SOAP object that the service will use to compose the SOAP response message.

Output Parameters

<i>soapResponseData</i>	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>	String Process directive that you want to assign to the SOAP processor. Note: Use only letters, digits, or the characters <code>-_!~*'()</code> in the name you specify in <i>directive</i> .
<i>svcName</i>	String Fully qualified name of the service that you are registering as a SOAP processor.
<i>descriptiveName</i>	String Descriptive comment for this SOAP processor. This comment is shown when you run the utility service

[pub.soap.processor:list](#) to get a list of the registered SOAP processors.

validateSOAPMessage **String** Optional. Flag indicating whether the SOAP message handler validates the SOAP messages that this processor sends and receives. Set to:

- `true` 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 `watt.server.SOAP.validateSOAPMessage` setting.

- `false` to bypass validation on messages sent and received by this SOAP processor.

This setting overrides the server's global `watt.server.SOAP.validateSOAPMessage` setting.

Or, leave *validateSOAPMessage* null to validate messages according to the Integration Server's `watt.server.SOAP.validateSOAPMessage` setting. This is the default.

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_msgQueue`
`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>	Object SOAP object to which you want the trailer added.
<i>trailer</i>	com.wm.lang.xml.Node 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

<i>soapProtocol</i>	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.
<i>fault</i>	Document The generic SOAP fault that is to be converted. The <i>fault</i> 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>	String 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>	Document 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>	String Conditional. A code that identifies the fault. This field corresponds to the SOAP 1.1 <code>faultcode</code> element. This field is set based on the value of the <code>code/namespaceName</code> field of the <i>fault</i> input parameter.

When *namespaceName* contains:

- **http://schemas.xmlsoap.org/soap/envelope/**, which is the standard namespace name for a SOAP 1.1 Envelope, *faultcode* is set to the following:

SOAP-ENV:*localName*

- **Any other value**, *faultcode* is set to the following:

{*namespaceName*}*localName*

In the above, *localName* is the value of the *code/localName* field and *namespaceName* is value of the *code/namespaceName* field of the *fault* input parameter

faultstring **String** Conditional. A human readable explanation of the fault. This field corresponds to the SOAP 1.1 *faultstring* element.

The service maps the **body* value from the first *reasons* document of the *fault* input parameter.

faultactor **String** Conditional. Information about the cause of the fault. This field corresponds to the SOAP 1.1 *faultactor* element.

The service maps the value from the *role* field of the *fault* input parameter.

detail **Document** Conditional. Application-specific details about the SOAP fault. This field corresponds to the SOAP 1.1 *detail* element.

The service maps the value from the *detail* field of the *fault* input parameter.

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.

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

SOAP- ENV:Code	Document Conditional. Contains the fault code.
SOAP- ENV:Value	<p>String A code that identifies the fault. This corresponds to the SOAP 1.2 <code>Code</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. When <code>namespaceName</code> contains:</p> <ul style="list-style-type: none"> ■ http://www.w3.org/2003/05/soap-envelope, which is, the standard namespace name for a SOAP 1.2 Envelope, <code>faultcode</code> is set to the following: SOAP-ENV:<code>localName</code> ■ Any other value, <code>faultcode</code> is set to the following: {<code>namespaceName</code>}<code>localName</code> <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>
SOAP- ENV:Reason	Document Conditional. Document containing the reason for the SOAP fault. This corresponds to the SOAP 1.2 <code>Reason</code> element.
SOAP- Env:Text	<p>Document Conditional. Document containing the human readable explanation of the cause of the fault. The service maps the first document of the <code>reasons</code> field of the <code>fault</code> input parameter.</p>
	<p>@XML:lang String Conditional. Specifies the language for the human</p>

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

<i>encoding</i>	String Optional. Specifies the encoding method. Default value is UTF-8.
<i>addEmptyHeader</i>	String Optional. Specifies whether to create an empty header entry along with the SOAP envelope and body entries in the SOAP message. Set to: <ul style="list-style-type: none"> ■ <code>True</code> to create an empty header entry in the SOAP message. ■ <code>False</code> 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.

<u>Key</u>	<u>Description</u>
<i>base64String</i>	String Optional. The base64-encoded string from which you want to generate the <code>com.wm.util.XOPObject</code> instance.
<i>bytes</i>	byte [] Optional. The byte array from which you want to generate the <code>com.wm.util.XOPObject</code> instance.
<i>stream</i>	Object Optional. The <code>InputStream</code> from which you want to generate the <code>com.wm.util.XOPObject</code> 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 790](#)
[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:

Key	Description
<i>namespaceName</i>	String Namespace portion of the node's qualified name.
<i>localName</i>	String 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.

Select...	To...
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>	String 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>	byte [] 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>	Object 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.utis: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.utis: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>	Object SOAP object containing the header entry that you want to delete.
<i>headerEntry</i>	com.wm.lang.xml.Node Optional. The header entry that you want to remove from <i>soapData</i> . (You would obtain the node with the pub.soap.utils:getHeaderEntries service.) Note: 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>	String 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

<i>soapData</i>	Object SOAP object containing the trailer that you want to delete.
<i>trailer</i>	com.wm.lang.xml.Node Optional. The trailer that you want to remove from <i>soapData</i> . (You would obtain the node with the pub.soap.utils:getTrailers service.) Note: You can use <i>trailer</i> or <i>index</i> to specify the trailer that you want <code>removeTrailer</code> to delete.
<i>index</i>	String Optional. Index of the trailer that you want to remove (where index 0 represents the first trailer). <i>index</i> is ignored if <i>trailer</i> 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

<i>wsdName</i>	String The name of the web service descriptor for which you want to set the effective policy. The <code>setWSDEffectivePolicy</code> 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 <code>true</code>).
<i>handlerName</i>	String The name of the handler for which you want to set the effective policy.
<i>effectivePolicyID</i>	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.setWSDEffectivePolicy](#) 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>	String Optional. Namespace name for the SOAP fault code.
<i>localName</i>	String Code that identifies the fault.
<i>subCodes</i>	Document List Optional. Subcodes that provide further detail. Each Document in the <i>subCodes</i> Document List contains: <ul style="list-style-type: none"> ■ <i>namespaceName</i> for the subcode ■ <i>localName</i> that identifies the subcode

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.

<u>Key</u>	<u>Description</u>
<i>*body</i>	String Text explaining the cause of the fault.

<i>@lang</i>	String Optional. Language for the human readable description.
<i>node</i>	String Optional. URI to the SOAP node where the fault occurred.
<i>role</i>	String Optional. Role in which the node was operating at the point the fault occurred.
<i>detail</i>	Document 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>	java.io.InputStream SOAP message that is to be converted to a SOAP object.
<i>soapProtocol</i>	<p>String 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"> ■ 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

<i>soapData</i>	Object SOAP object representation of the SOAP message in <i>stream</i> .
-----------------	---

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

<i>string</i>	String SOAP message that is to be converted to a SOAP object.
<i>soapProtocol</i>	String 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: <ul style="list-style-type: none">■ 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.
<i>addEmptyHeader</i>	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.

Key	Description
<i>*body</i>	String 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.

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

<i>wsa:Address</i>	Document Contains the end point URI for the fault message.
--------------------	---

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

<i>*body</i>	String The end point URI for the fault message.
--------------	---

wsa:ReferenceParameters **Document** Contains the set of reference parameter elements.

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

<i>*any</i>	Object List The reference parameter elements.
-------------	--

wsa:Metadata **Document** Contains the set of metadata elements.

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

<i>*any</i>	Object List 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

<i>wsa:From</i>	Document Contains the details about the source of the message.																												
	<table> <thead> <tr> <th><u>Key</u></th> <th><u>Description</u></th> </tr> </thead> <tbody> <tr> <td><i>wsa:Address</i></td> <td>Document Contains the address of the source of the message.</td> </tr> <tr> <td></td> <td> <table> <thead> <tr> <th><u>Key</u></th> <th><u>Description</u></th> </tr> </thead> <tbody> <tr> <td><i>*body</i></td> <td>String The address of the source of the message.</td> </tr> </tbody> </table> </td> </tr> <tr> <td><i>wsa:Reference Parameters</i></td> <td>Document Contains the set of reference parameter elements.</td> </tr> <tr> <td></td> <td> <table> <thead> <tr> <th><u>Key</u></th> <th><u>Description</u></th> </tr> </thead> <tbody> <tr> <td><i>*any</i></td> <td>Object List The reference parameter elements.</td> </tr> </tbody> </table> </td> </tr> <tr> <td><i>wsa:Metadata</i></td> <td>Document Contains the set of metadata elements.</td> </tr> <tr> <td></td> <td> <table> <thead> <tr> <th><u>Key</u></th> <th><u>Description</u></th> </tr> </thead> <tbody> <tr> <td><i>*any</i></td> <td>Object List The metadata elements.</td> </tr> </tbody> </table> </td> </tr> <tr> <td><i>*any</i></td> <td>Object List Contains other extensible elements, if any.</td> </tr> </tbody> </table>	<u>Key</u>	<u>Description</u>	<i>wsa:Address</i>	Document 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>String The address of the source of the message.</td> </tr> </tbody> </table>	<u>Key</u>	<u>Description</u>	<i>*body</i>	String The address of the source of the message.	<i>wsa:Reference Parameters</i>	Document 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>Object List The reference parameter elements.</td> </tr> </tbody> </table>	<u>Key</u>	<u>Description</u>	<i>*any</i>	Object List The reference parameter elements.	<i>wsa:Metadata</i>	Document 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>Object List The metadata elements.</td> </tr> </tbody> </table>	<u>Key</u>	<u>Description</u>	<i>*any</i>	Object List The metadata elements.	<i>*any</i>	Object List Contains other extensible elements, if any.
<u>Key</u>	<u>Description</u>																												
<i>wsa:Address</i>	Document 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>String The address of the source of the message.</td> </tr> </tbody> </table>	<u>Key</u>	<u>Description</u>	<i>*body</i>	String The address of the source of the message.																								
<u>Key</u>	<u>Description</u>																												
<i>*body</i>	String The address of the source of the message.																												
<i>wsa:Reference Parameters</i>	Document 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>Object List The reference parameter elements.</td> </tr> </tbody> </table>	<u>Key</u>	<u>Description</u>	<i>*any</i>	Object List The reference parameter elements.																								
<u>Key</u>	<u>Description</u>																												
<i>*any</i>	Object List The reference parameter elements.																												
<i>wsa:Metadata</i>	Document 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>Object List The metadata elements.</td> </tr> </tbody> </table>	<u>Key</u>	<u>Description</u>	<i>*any</i>	Object List The metadata elements.																								
<u>Key</u>	<u>Description</u>																												
<i>*any</i>	Object List The metadata elements.																												
<i>*any</i>	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.

<u>Key</u>	<u>Description</u>
<i>*body</i>	String 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>	Document Optional. Element that provides the details about the [action] that caused the problem.
<u>Key</u>	<u>Description</u>

body* **String Optional. The [action] that caused the problem.

wsa:SoapAction **String** 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.

Key	Description
<i>*body</i>	String 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>	String 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>	String Optional. The relationship type.
<i>*body</i>	String < <code>wsa:MessageID</code> > 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

<i>wsa:ReplyTo</i>	Document Contains the address of the intended receiver of the response message.																												
	<table> <thead> <tr> <th><u>Key</u></th> <th><u>Description</u></th> </tr> </thead> <tbody> <tr> <td><i>wsa:Address</i></td> <td>Document Contains the end point URI for the response message.</td> </tr> <tr> <td></td> <td> <table> <thead> <tr> <th><u>Key</u></th> <th><u>Description</u></th> </tr> </thead> <tbody> <tr> <td><i>*body</i></td> <td>String Optional. The end point URI for the response message.</td> </tr> </tbody> </table> </td> </tr> <tr> <td><i>wsa:ReferenceParameters</i></td> <td>Document Optional. Contains the set of reference parameter elements.</td> </tr> <tr> <td></td> <td> <table> <thead> <tr> <th><u>Key</u></th> <th><u>Description</u></th> </tr> </thead> <tbody> <tr> <td><i>*any</i></td> <td>Object List Optional. The reference parameter elements.</td> </tr> </tbody> </table> </td> </tr> <tr> <td><i>wsa:Metadata</i></td> <td>Document Optional. Contains the set of metadata elements.</td> </tr> <tr> <td></td> <td> <table> <thead> <tr> <th><u>Key</u></th> <th><u>Description</u></th> </tr> </thead> <tbody> <tr> <td><i>*any</i></td> <td>Object List Optional. The metadata elements.</td> </tr> </tbody> </table> </td> </tr> <tr> <td><i>*any</i></td> <td>Object List Optional. Contains other extensible elements, if any.</td> </tr> </tbody> </table>	<u>Key</u>	<u>Description</u>	<i>wsa:Address</i>	Document 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>String Optional. The end point URI for the response message.</td> </tr> </tbody> </table>	<u>Key</u>	<u>Description</u>	<i>*body</i>	String Optional. The end point URI for the response message.	<i>wsa:ReferenceParameters</i>	Document Optional. 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>Object List Optional. The reference parameter elements.</td> </tr> </tbody> </table>	<u>Key</u>	<u>Description</u>	<i>*any</i>	Object List Optional. The reference parameter elements.	<i>wsa:Metadata</i>	Document Optional. 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>Object List Optional. The metadata elements.</td> </tr> </tbody> </table>	<u>Key</u>	<u>Description</u>	<i>*any</i>	Object List Optional. The metadata elements.	<i>*any</i>	Object List Optional. Contains other extensible elements, if any.
<u>Key</u>	<u>Description</u>																												
<i>wsa:Address</i>	Document 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>String Optional. The end point URI for the response message.</td> </tr> </tbody> </table>	<u>Key</u>	<u>Description</u>	<i>*body</i>	String Optional. The end point URI for the response message.																								
<u>Key</u>	<u>Description</u>																												
<i>*body</i>	String Optional. The end point URI for the response message.																												
<i>wsa:ReferenceParameters</i>	Document Optional. 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>Object List Optional. The reference parameter elements.</td> </tr> </tbody> </table>	<u>Key</u>	<u>Description</u>	<i>*any</i>	Object List Optional. The reference parameter elements.																								
<u>Key</u>	<u>Description</u>																												
<i>*any</i>	Object List Optional. The reference parameter elements.																												
<i>wsa:Metadata</i>	Document Optional. 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>Object List Optional. The metadata elements.</td> </tr> </tbody> </table>	<u>Key</u>	<u>Description</u>	<i>*any</i>	Object List Optional. The metadata elements.																								
<u>Key</u>	<u>Description</u>																												
<i>*any</i>	Object List Optional. The metadata elements.																												
<i>*any</i>	Object List 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

wsa:RetryAfter **Document** Contains the `wsa:RetryAfter` header of a SOAP message.

Key	Description
<i>*body</i>	String 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.

<u>Key</u>	<u>Description</u>
<i>*body</i>	String 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>	String 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>	Document Contains the end point URI for the fault message.
--------------------	---

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

<i>*body</i>	String Optional. The end point URI for the fault message.
--------------	--

<i>wsa:Reference Properties</i>	Document Optional. The properties that are required to identify the entity or resource being conveyed.
---------------------------------	---

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

<i>*any</i>	Object List Optional. The property elements.
-------------	---

<i>wsa:Reference Parameters</i>	Document Optional. Contains the set of reference parameter elements.
---------------------------------	---

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

	<i>*any</i>	Object List Optional. The reference parameter elements.						
<i>wsa:PortType</i>		Document 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;">Key</th> <th style="border-bottom: 1px solid black;">Description</th> </tr> </thead> <tbody> <tr> <td><i>*body</i></td> <td>Object List Optional. The QName of the primary <code>portType</code> of the endpoint being conveyed.</td> </tr> </tbody> </table>	Key	Description	<i>*body</i>	Object List Optional. The QName of the primary <code>portType</code> of the endpoint being conveyed.		
Key	Description							
<i>*body</i>	Object List Optional. The QName of the primary <code>portType</code> of the endpoint being conveyed.							
<i>wsa:Service Name</i>		Document 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;">Key</th> <th style="border-bottom: 1px solid black;">Description</th> </tr> </thead> <tbody> <tr> <td><i>@PortName</i></td> <td>String Optional. The name of the <code><wsdl:port></code> definition that corresponds to the endpoint being referenced.</td> </tr> <tr> <td><i>*body</i></td> <td>String Optional. The <code><wsdl:service></code> definition that contains a WSDL description of the endpoint being referenced.</td> </tr> </tbody> </table>	Key	Description	<i>@PortName</i>	String Optional. The name of the <code><wsdl:port></code> definition that corresponds to the endpoint being referenced.	<i>*body</i>	String Optional. The <code><wsdl:service></code> definition that contains a WSDL description of the endpoint being referenced.
Key	Description							
<i>@PortName</i>	String Optional. The name of the <code><wsdl:port></code> definition that corresponds to the endpoint being referenced.							
<i>*body</i>	String Optional. The <code><wsdl:service></code> definition that contains a WSDL description of the endpoint being referenced.							
	<i>*any</i>	Object List 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

<i>wsa:From</i>	Document Contains the details about the source of the message.																												
	<table> <thead> <tr> <th><u>Key</u></th> <th><u>Description</u></th> </tr> </thead> <tbody> <tr> <td><i>wsa:Address</i></td> <td>Document Contains the details about the source of the message.</td> </tr> <tr> <td></td> <td> <table> <thead> <tr> <th><u>Key</u></th> <th><u>Description</u></th> </tr> </thead> <tbody> <tr> <td><i>*body</i></td> <td>String Optional. The address of the source of the message.</td> </tr> </tbody> </table> </td> </tr> <tr> <td><i>wsa:Reference Properties</i></td> <td>Document Optional. The properties that are required to identify the entity or resource being conveyed.</td> </tr> <tr> <td></td> <td> <table> <thead> <tr> <th><u>Key</u></th> <th><u>Description</u></th> </tr> </thead> <tbody> <tr> <td><i>*any</i></td> <td>Object List Optional. The property elements.</td> </tr> </tbody> </table> </td> </tr> <tr> <td><i>wsa:Reference Parameters</i></td> <td>Document Optional. Contains the set of reference parameter elements.</td> </tr> <tr> <td></td> <td> <table> <thead> <tr> <th><u>Key</u></th> <th><u>Description</u></th> </tr> </thead> <tbody> <tr> <td><i>*any</i></td> <td>Object List Optional. The reference parameter elements.</td> </tr> </tbody> </table> </td> </tr> <tr> <td><i>wsa:PortType</i></td> <td>Document Optional. Contains the QName of the primary portType of the endpoint being conveyed.</td> </tr> </tbody> </table>	<u>Key</u>	<u>Description</u>	<i>wsa:Address</i>	Document Contains the details about 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>String Optional. The address of the source of the message.</td> </tr> </tbody> </table>	<u>Key</u>	<u>Description</u>	<i>*body</i>	String Optional. The address of the source of the message.	<i>wsa:Reference Properties</i>	Document Optional. The properties that are required to identify the entity or resource being conveyed.		<table> <thead> <tr> <th><u>Key</u></th> <th><u>Description</u></th> </tr> </thead> <tbody> <tr> <td><i>*any</i></td> <td>Object List Optional. The property elements.</td> </tr> </tbody> </table>	<u>Key</u>	<u>Description</u>	<i>*any</i>	Object List Optional. The property elements.	<i>wsa:Reference Parameters</i>	Document Optional. 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>Object List Optional. The reference parameter elements.</td> </tr> </tbody> </table>	<u>Key</u>	<u>Description</u>	<i>*any</i>	Object List Optional. The reference parameter elements.	<i>wsa:PortType</i>	Document Optional. Contains the QName of the primary portType of the endpoint being conveyed.
<u>Key</u>	<u>Description</u>																												
<i>wsa:Address</i>	Document Contains the details about 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>String Optional. The address of the source of the message.</td> </tr> </tbody> </table>	<u>Key</u>	<u>Description</u>	<i>*body</i>	String Optional. The address of the source of the message.																								
<u>Key</u>	<u>Description</u>																												
<i>*body</i>	String Optional. The address of the source of the message.																												
<i>wsa:Reference Properties</i>	Document Optional. The properties that are required to identify the entity or resource being conveyed.																												
	<table> <thead> <tr> <th><u>Key</u></th> <th><u>Description</u></th> </tr> </thead> <tbody> <tr> <td><i>*any</i></td> <td>Object List Optional. The property elements.</td> </tr> </tbody> </table>	<u>Key</u>	<u>Description</u>	<i>*any</i>	Object List Optional. The property elements.																								
<u>Key</u>	<u>Description</u>																												
<i>*any</i>	Object List Optional. The property elements.																												
<i>wsa:Reference Parameters</i>	Document Optional. 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>Object List Optional. The reference parameter elements.</td> </tr> </tbody> </table>	<u>Key</u>	<u>Description</u>	<i>*any</i>	Object List Optional. The reference parameter elements.																								
<u>Key</u>	<u>Description</u>																												
<i>*any</i>	Object List Optional. The reference parameter elements.																												
<i>wsa:PortType</i>	Document Optional. Contains the QName of the primary portType of the endpoint being conveyed.																												

	<u>Key</u>	<u>Description</u>
	<i>*body</i>	Object List Optional. The QName of the primary <code>portType</code> of the endpoint being conveyed.
<i>wsa:Service Name</i>		Document Optional. Represents the QName identifying the WSDL service element that contains the definition of the endpoint being conveyed.
	<u>Key</u>	<u>Description</u>
	<i>@PortName</i>	String Optional. The name of the <code><wsdl:port></code> definition that corresponds to the endpoint being referenced.
	<i>*body</i>	String Optional. The <code><wsdl:service></code> definition that contains a WSDL description of the endpoint being referenced.
<i>*any</i>		Object List 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.

Key	Description
<i>*body</i>	String 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.

Key	Description
<i>@RelationshipType</i>	String Optional. The relationship type.
<i>*body</i>	String < <code>wsa:MessageID</code> > 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>	Document Contains the end point URI for the response.
--------------------	--

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

<i>*body</i>	String Optional. The end point URI for the fault message.
--------------	--

<i>wsa:Reference Properties</i>	Document Optional. The properties that are required to identify the entity or resource being conveyed.
---------------------------------	---

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

<i>*any</i>	Object List Optional. The property elements.
-------------	---

<i>wsa:Reference Parameters</i>	Document Optional. Contains the set of reference parameter elements.
---------------------------------	---

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

	<i>*any</i>	Object List Optional. The reference parameter elements.						
<i>wsa:PortType</i>		Document 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>Object List Optional. The QName of the primary <code>portType</code> of the endpoint being conveyed.</td> </tr> </tbody> </table>	Key	Description	<i>*body</i>	Object List Optional. The QName of the primary <code>portType</code> of the endpoint being conveyed.		
Key	Description							
<i>*body</i>	Object List Optional. The QName of the primary <code>portType</code> of the endpoint being conveyed.							
<i>wsa:Service Name</i>		Document 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>String Optional. The name of the <code><wsdl:port></code> definition that corresponds to the endpoint being referenced.</td> </tr> <tr> <td><i>*body</i></td> <td>String Optional. The <code><wsdl:service></code> definition that contains a WSDL description of the endpoint being referenced.</td> </tr> </tbody> </table>	Key	Description	<i>@PortName</i>	String Optional. The name of the <code><wsdl:port></code> definition that corresponds to the endpoint being referenced.	<i>*body</i>	String Optional. The <code><wsdl:service></code> definition that contains a WSDL description of the endpoint being referenced.
Key	Description							
<i>@PortName</i>	String Optional. The name of the <code><wsdl:port></code> definition that corresponds to the endpoint being referenced.							
<i>*body</i>	String Optional. The <code><wsdl:service></code> definition that contains a WSDL description of the endpoint being referenced.							
	<i>*any</i>	Object List 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>	String 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
-----	-------------

body* **String 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.wsmr: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.wsmr:createSequence` service or as the `reliableMessagingInfo/responseReliableMessagingProperties/serverSequenceId` output parameter of a web service connector.

Output Parameters

fault **Document** Conditional. Contents of the fault block.
The fault document references the `pub.soap.utils: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>	Document Conditional. Header fields from the request message issued by the reliable messaging source.
<i>responseHeaders</i>	<p>Document 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>	Document Conditional. Status code from the request.
<i>statusMessage</i>	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

<i>consumerWebServiceDescriptorName</i>	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.

Key	Description				
<i>address</i>	Document Document specifying the address to which the acknowledgment is to be sent.				
	<table border="1"> <thead> <tr> <th>Key</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td><i>value</i></td> <td>String Address to which the acknowledgment is to be sent.</td> </tr> </tbody> </table>	Key	Description	<i>value</i>	String Address to which the acknowledgment is to be sent.
Key	Description				
<i>value</i>	String 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.

Key	Description
<i>transport</i>	<p>Document 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.</p> <p>You only need to provide credentials in <i>transport</i> if the endpoint URL specifies HTTPS and you want to overwrite the credentials specified in the consumer endpoint alias assigned to the binder.</p>

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

<i>type</i>	<p>String Optional. Type of authentication required by the reliable messaging client. If <i>type</i> is not specified, Integration Server uses Basic.</p>
-------------	---

Note: If any value other than **Basic** is specified, Integration Server ignores the credentials provided in *user*, *pass*, and *serverCerts*.

<i>user</i>	<p>String Optional. User name used to authenticate the reliable messaging client at the HTTP or HTTPS transport level on the reliable messaging server.</p>
-------------	--

<i>pass</i>	<p>String Optional. Password used to authenticate the reliable messaging client to the reliable messaging server.</p>
-------------	--

<i>serverCerts</i>	<p>Document Optional. The private key and certificate chain of the message signer.</p>
--------------------	---

- *keyStoreAlias* **String** Alias to the keystore that contains

the private key used to securely connect to the reliable messaging server.

- *keyAlias* **String** 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 *keyStoreAlias*.

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.

_url **String** Optional. URL to use as the endpoint URL for the web service. If supplied, the value of *_url* overwrites the endpoint URL in the original WSDL.

transportHeaders **Document** Optional. Transport header fields that you want to explicitly set in the request issued by the reliable messaging client.

Specify a key in *transportHeaders* 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 *transportHeaders* must be of type **String**. For information about using *transportHeaders*, including a description of the default Integration Server behavior, see *Web Services Developer's Guide*.

Output Parameters

serverSequenceId **String** 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.

fault

Document Conditional. Contents of the fault block.

The fault document references the "[pub.soap.utils:soapFault](#)" on [page 804](#) 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>	Document Conditional. Header fields from the request message issued by the reliable messaging source.
<i>responseHeaders</i>	<p>Document 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>	Document Conditional. Status code from the request.

statusMessage **Document** 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 804](#) 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>	Document Conditional. Header fields from the request message issued by the reliable messaging source.
<i>responseHeaders</i>	Document 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.

<i>status</i>	Document Conditional. Status code from the request.
<i>statusMessage</i>	Document 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

<i>fault</i>	Document Conditional. Contents of the fault block. The fault document references the " pub.soap.utils:soapFault " on page 804 document type.
<i>transportInfo</i>	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>	Document Conditional. Header fields from the request message issued by the reliable messaging source.
<i>responseHeaders</i>	<p>Document 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>	Document Conditional. Status code from the request.
<i>statusMessage</i>	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.wsmr: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

fault **Document** Conditional. Contents of the fault block.
The fault document references the "[pub.soap.utils:soapFault](#)" on [page 804](#) 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>	Document Conditional. Header fields from the request message issued by the reliable messaging source.
<i>responseHeaders</i>	Document 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.

32 Storage Folder

■ About the Storage Elements	838
■ Locking Considerations	839
■ Sample Flow Service for Checkpoint Restart	841
■ Summary of Elements in this Folder	842

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 840.

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 840.

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

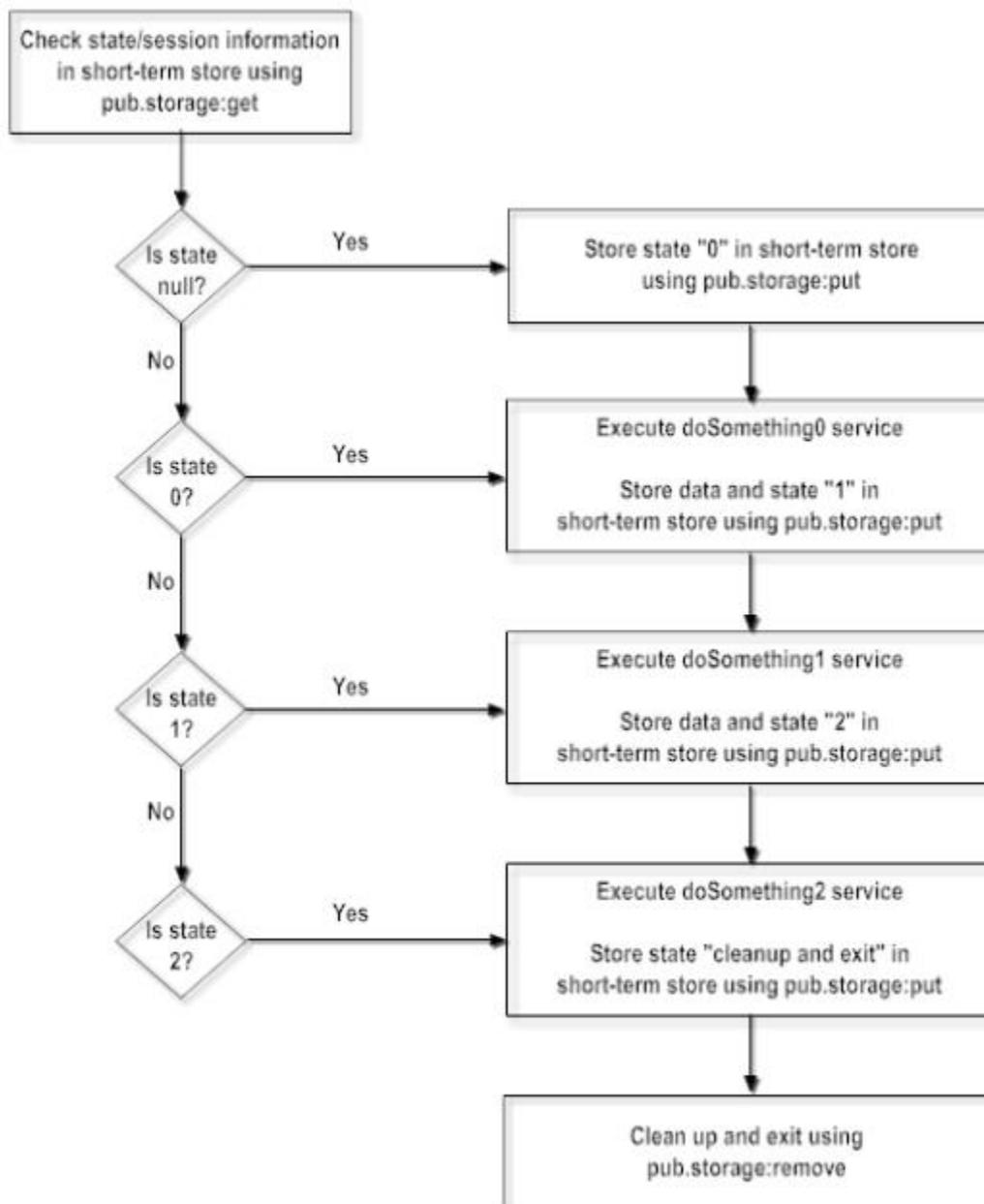
You can control how long Integration Server will wait to obtain a lock and how long it will hold a lock by using the following server properties:

- 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
pub.storage:add	WmPublic. Inserts a new entry into a data store.
pub.storage:closeStore	WmPublic. <i>Obsolete</i> – Closes a data store and unregisters the data store with the server.
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.
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.
pub.storage:keys	WmPublic. Obtains a list of all the keys in a data store.
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.
pub.storage:lock	WmPublic. Locks an entry and/or data store on behalf of the thread invoking this service.
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.
pub.storage:registerStore	WmPublic. <i>Obsolete</i> – Opens or creates a data store and registers the store with the server.
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.
pub.storage:remove	WmPublic. Removes an entry from a data store.

Element	Package and Description
pub.storage:shutdown	WmPublic. Releases internal resources used by the <code>pub.storage</code> services. This service is run automatically when the WmPublic package is unloaded and should <i>not</i> be explicitly invoked by a client.
pub.storage:startup	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.
pub.storage:unlock	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>	String Name of the data store in which to insert the entry.
<i>key</i>	String Key under which the entry is to be inserted.
<i>value</i>	Document Value (IData object) to be inserted.

Output Parameters

<i>result</i>	<p>String Flag indicating whether the entry was successfully added. A value of:</p> <ul style="list-style-type: none"> ■ <code>true</code> indicates that the new entry was inserted successfully. ■ <code>false</code> indicates that the entry was not inserted (usually because an entry for <i>key</i> already exists).
<i>error</i>	String 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>	String Name of the data store from which you want to retrieve the entry.
<i>key</i>	String Key of the entry whose value you want to retrieve.
<i>waitLength</i>	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. 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 Settings > Extended Settings screen on the Integration Server Administrator.
<i>lockMode</i>	<p>String Optional. Type of lock you want to place on the entry. Set to:</p> <ul style="list-style-type: none"> ■ Exclusive 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.

- `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 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.

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

<i>holder</i>	<p>String Optional. Identifies the holder whose pub.storage locks are to be listed. The format is "_DataStore_<sessionId:threadId>", where:</p> <ul style="list-style-type: none"> ■ <i>sessionId</i> is a unique, internally generated identifier for the client's session in Integration Server. ■ <i>threadId</i> is a unique, internally generated identifier for the client's thread in Integration Server.
<i>target</i>	<p>String Optional. Identifies the target whose pub.storage locks are to be listed.</p>

Output Parameters

locks **Document List** The list of pub.storage locks. This output variable can be null.

Value	Description
id	String The internal ID of the lock
target	String Item that is locked, specified as a data store name or the key for an entry
holder	String Holder of the lock. This value is generated internally by the pub.storage services.
type	String "EXCLUSIVE" or "SHARE"
count	String Number of lock holders sharing this lock
time	String 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>	String Name of the data store containing the entry.
<i>key</i>	String 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">■ 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.■ 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. If both <i>storeName</i> and <i>key</i> are specified and you request: <ul style="list-style-type: none">■ A shared lock, the service obtains a shared lock on the data store and the entry.■ An exclusive lock, the service obtains a shared lock on the data store and an exclusive lock on the entry.
<i>waitLength</i>	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. 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 **Settings > Extended Settings** screen on the Integration Server Administrator.

lockMode

String Optional. Type of lock you want to place on the entry or data store. Set to:

- **Exclusive** 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.

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

<i>storeName</i>	String Name of the data store into which you want to insert or update the entry.
<i>value</i>	Document Value (IData object) to be inserted or updated.
<i>waitLength</i>	<p>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 <code>watt.server.storage.lock.maxWait</code> property. You can update this property by using the Settings > Extended Settings screen on the Integration Server Administrator.</p> <p>This parameter is used only when your service did not explicitly lock the entry beforehand.</p>
<i>key</i>	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

<i>holders</i>	String List Optional. Holders whose pub.storage locks are to be released.
<i>ids</i>	String Optional. Ids whose pub.storage locks are to be released.

Output Parameters

<i>count</i>	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>	String Name of the data store from which to remove an entry.
<i>key</i>	String Key of the entry that you want to remove.
<i>waitLength</i>	String 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 Settings > Extended Settings screen on the Integration Server Administrator.

Output Parameters

<i>result</i>	String Flag indicating whether the entry was successfully removed. A value of: <ul style="list-style-type: none">■ <code>true</code> indicates that the entry was removed successfully.■ <code>false</code> indicates that the entry was not removed (usually because an entry for key does not exist).
---------------	---

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>	String Name of the data store in which to unlock an entry.
<i>key</i>	String 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 856

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
pub.string:base64Decode	WmPublic. Decodes a Base-64 encoded string into a sequence of bytes.
pub.string:base64Encode	WmPublic. Converts a sequence of bytes into a Base64-encoded String.
pub.string:bytesToString	WmPublic. Converts a sequence of bytes to a String.
pub.string:concat	WmPublic. Concatenates two strings.
pub.string:HTMLDecode	WmPublic. Replaces HTML character entities with native characters.
pub.string:HTMLEncode	WmPublic. Replaces HTML-sensitive characters with equivalent HTML character entities.
pub.string:indexOf	WmPublic. Returns the index of the first occurrence of a sequence of characters in a string.
pub.string:length	WmPublic. Returns the length of a string.
pub.string:lookupDictionary	WmPublic. Looks up a given key in a hash table and returns the string to which that key is mapped.
pub.string:lookupTable	WmPublic. Locates a key in a String Table and returns the string to which that key is mapped.
pub.string:makeString	WmPublic. Builds a single string by concatenating the elements of a String List.

Element	Package and Description
pub.string:messageFormat	WmPublic. Formats an array of strings into a given message pattern.
pub.string:numericFormat	WmPublic. Formats a number into a given numeric pattern.
pub.string:objectToString	WmPublic. Converts an object to string representation 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.
pub.string:padRight	WmPublic. Pads a string to a specified length by adding pad characters to the end of the string.
pub.string:replace	WmPublic. Replaces all occurrences of a specified substring with a substitute string.
pub.string:stringToBytes	WmPublic. Converts a string to a byte array.
pub.string:substring	WmPublic. Returns a substring of a given string.
pub.string:tokenize	WmPublic. Tokenizes a string using specified delimiter characters and generates a String List from the resulting tokens.
pub.string:toLowerCase	WmPublic. Converts all characters in a given string to lowercase.
pub.string:toUpperCase	WmPublic. Converts all characters in a given string to uppercase.
pub.string:trim	WmPublic. Trims leading and trailing white space from a given string.
pub.string:URLDecode	WmPublic. Decodes a URL-encoded string.
pub.string:URLEncode	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

bytes **byte[]** Sequence of bytes to encode into a Base64-encoded String.

useNewLine **String** Optional. Flag indicating whether to retain or remove the line breaks. Set to:

- `true` to retain the line breaks. This is the default.
- `false` to remove the line breaks.

encoding **String** Optional. Specifies the encoding method. Default value is ASCII.

Output Parameters

value **String** 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>	byte[] Sequence of bytes to convert to a String.
<i>encoding</i>	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. To use the default encoding, set <i>encoding</i> to <code>autoDetect</code> .

Output Parameters

<i>string</i>	String String representation of the contents of <i>bytes</i> .
---------------	---

pub.string:concat

WmPublic. Concatenates two strings.

Input Parameters

<i>inString1</i>	String String to which you want to concatenate another string.
<i>inString2</i>	String String to concatenate to <i>inString1</i> .

Output Parameters

<i>value</i>	String 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>
>	>
<	<
&	&
"	"

Input Parameters

inString **String** An HTML-encoded String.

Output Parameters

value **String** Result from decoding the contents of *inString*. Any HTML character entities that existed in *inString* will appear as native characters in *value*.

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>
>	>
<	<

<u>Replaces this native language character...</u>	<u>With...</u>
&	&
"	"
`	'

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

inString **String** String in which you want to locate a sequence of characters.

subString **String** Sequence of characters to locate.

fromIndex **String** Optional. Index of *inString* from which to start the search. If no value is specified, this parameter contains 0 to indicate the beginning of the string.

Output Parameters

value **String** Index of the first occurrence of *subString* in *inString* . If no occurrence is found, this parameter contains -1.

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

<i>lookupTable</i>	String [] [] A multi-row, multi-column string table in which to search.
<i>keyColumnIndex</i>	String Index of the "key" column. Default is 0.
<i>valueColumnIndex</i>	String Index of the "value" column. Default is 1.
<i>key</i>	String Key to locate. Note: The key is case sensitive.
<i>ignoreCase</i>	String Optional. Flag indicating whether to perform a case-sensitive or case-insensitive search. Set to: <ul style="list-style-type: none">■ <code>true</code> to perform a case-insensitive search.■ <code>false</code> to perform a case-sensitive search. This is the default.
<i>useRegex</i>	String Optional. Flag indicating whether the values in the table are to be interpreted as regular expressions. Note: 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">■ <code>true</code> to interpret the key column values in the table as regular expressions.■ <code>false</code> to interpret the key column values in the table as literal values (that is, not regular expressions). This is the default.

Output Parameters

<i>value</i>	String 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>	String List Strings to concatenate.
<i>separator</i>	String String to insert between each non-null element in <i>elementList</i> .

Output Parameters

<i>value</i>	String Result from concatenating the strings in <i>elementList</i> . Strings are separated by the characters specified in <i>separator</i> .
--------------	---

pub.string:messageFormat

WmPublic. Formats an array of strings into a given message pattern.

Input Parameters

<i>pattern</i>	String Message that includes "placeholders" where elements from <i>argumentList</i> are to be inserted. The message can contain any sequence of characters. Use the { <i>n</i> } placeholder to insert elements from <i>argumentList</i> , where <i>n</i> 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*.

<i>argumentList</i>	String List Optional. List of strings to use to populate <i>pattern</i> . If <i>argumentList</i> is not supplied, the service will not replace placeholders in <i>pattern</i> 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.

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:

Pattern	Description
<code>#,###</code>	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

inString **String** String that you want to pad.

padString **String** Characters to use to pad *inString* .

length **String** Total length of the resulting string, including pad characters.

Output Parameters

value **String** Contents of *inString* preceded by as many pad characters as needed so that the total length of the string equals *length* .

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`.

If <i>length</i> equals...	Then <i>value</i> will contain...
-----------------------------------	--

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>	String String that you want to pad.
<i>padString</i>	String Characters to use to pad <i>inString</i> .
<i>length</i>	String Total length of the resulting string, including pad characters.

Output Parameters

<i>value</i>	String 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`.

<u>If <i>length</i> equals...</u>	<u>Then <i>value</i> will contain...</u>
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>	String String containing the substring to replace.
<i>searchString</i>	String Substring to replace within <i>inString</i> .
<i>replaceString</i>	String 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>	String 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">■ <code>true</code> to indicate that <i>searchString</i> is a regular expression.■ <code>false</code> to indicate that <i>searchString</i> is not a regular expression. This is the default.

Output Parameters

<i>value</i>	String Contents of <i>inString</i> with replacements made.
--------------	---

pub.string:stringToBytes

WmPublic. Converts a string to a byte array.

Input Parameters

<i>string</i>	String String to convert to a byte[].
<i>encoding</i>	String 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 `autoDetect`. If you specify an unsupported encoding, an exception will be thrown.

Output Parameters

bytes **byte[]** Contents of *string* represented as a `byte[]`.

pub.string.substring

WmPublic. Returns a substring of a given string.

Input Parameters

inString **String** String from which to extract a substring.

beginIndex **String** Beginning index of the substring to extract (inclusive).

endIndex **String** Ending index of the substring to extract (exclusive). If this parameter is null or empty, the substring will extend to the end of *inString*.

Output Parameters

value **String** Substring from *beginIndex* and extending to the character at *endIndex* - 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

inString **String** String you want to tokenize (that is, break into delimited chunks).

delim **String** Delimiter characters. If null or empty, the service uses the default delimiters `\t\n\r`, where `t`, `n`, and `r` represent the white space characters tab, new line, and carriage return).

Output Parameters

valueList **String List** Strings containing the tokens extracted from *inString*.

pub.string:toLower

WmPublic. Converts all characters in a given string to lowercase.

Input Parameters

inString **String** String to convert.

language **String** Optional. Lowercase, two-letter ISO-639 code. If this parameter is null, the system default is used.

country **String** Optional. Uppercase, two-letter ISO-3166 code. If this parameter is null, the system default is used.

variant **String** Optional. Vendor and browser-specific code. If null, this parameter is ignored.

Output Parameters

value **String** Contents of *inString*, with all uppercase characters converted to lowercase.

pub.string:toUpper

WmPublic. Converts all characters in a given string to uppercase.

Input Parameters

inString **String** String to convert.

<i>language</i>	String Optional. Lowercase, two-letter ISO-639 code. If this parameter is null, the system default is used.
<i>country</i>	String Optional. Uppercase, two-letter ISO-3166 code. If this parameter is null, the system default is used.
<i>variant</i>	String Optional. Vendor and browser-specific code. If null, this parameter is ignored.

Output Parameters

<i>value</i>	String 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

<i>inString</i>	String String to trim.
-----------------	-------------------------------

Output Parameters

<i>value</i>	String Contents of <i>inString</i> with white space trimmed from both ends.
--------------	--

pub.string:URLDecode

WmPublic. Decodes a URL-encoded string.

Input Parameters

<i>inString</i>	String URL-encoded string to decode.
-----------------	---

Output Parameters

<i>value</i>	String Result from decoding <i>inString</i> . If <i>inString</i> contained plus (+) signs, they will appear in <i>value</i> as spaces. If <i>inString</i>
--------------	--

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 876

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>	String Name of the key. Waiting services with the same <i>key</i> will receive notification and input from this service.
<i>value</i>	Document Input for the waiting services.

Output Parameters

<i>notified</i>	String 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>	String 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>	String Length of time, in seconds, the service waits for notification. If the request times out, an exception is thrown. Note: A <i>time</i> value of -1 or 0 results in undefined behavior.
<i>exclusive</i>	String 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">■ <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.■ <code>no</code> to allow multiple services to wait for notification. This is the default.

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 880

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
pub.synchronization.latch:closeLatch	WmPublic. Closes the latch for a resource.
pub.synchronization.latch:isLatchClosed	WmPublic. Checks a resource's latch status.
pub.synchronization.latch:openLatch	WmPublic. Opens the latch for a resource.
pub.synchronization.xref:createXReference	WmPublic. Creates a cross-reference between a canonical key and a native ID.
pub.synchronization.xref:deleteByObjectId	WmPublic. Removes all cross-reference records associated with a particular process or synchronization.
pub.synchronization.xref:deleteXReference	WmPublic. Deletes a cross-reference record from the cross-reference table.
pub.synchronization.xref:getCanonicalKey	WmPublic. Retrieves the canonical key for a specified native ID.
pub.synchronization.xref:getNativeId	WmPublic. Retrieves the native ID of a resource record associated with a canonical key.
pub.synchronization.xref:insertXReference	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>	String 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>	String The canonical key. A unique identifier for the canonical document used in the synchronization.
<i>objectId</i>	String 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>	String 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>	String The canonical key. A unique identifier for the canonical document used in the synchronization.
<i>objectId</i>	String 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>	String 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>	String The canonical key. A unique identifier for the canonical document used in the synchronization.
<i>objectId</i>	String 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>	String A unique identifier for the resource (application) for which you want to create a cross-reference to a canonical key.
<i>nativeId</i>	String A unique identifier for the resource record for which you want to create a cross-reference to a canonical key.
<i>canonicalKey</i>	String 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>	String 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>	String 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>	String The canonical key. A unique identifier for the canonical document for which you want to delete a cross-reference.
<i>objectId</i>	String 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>	String A unique identifier for the resource (application) that contains the native ID for which you want to retrieve a canonical key.
<i>nativeId</i>	String A unique identifier for the resource record for which you want to obtain the canonical key.
<i>objectId</i>	String 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>	String 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>	String A unique identifier for the resource from which you want to retrieve the native ID associated with the provided canonical key.
<i>canonicalKey</i>	String The canonical key for which you want to obtain the corresponding native ID.
<i>objectId</i>	String 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>	String 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>	String 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>	String A unique identifier for the resource record with which you want to establish a cross-reference to <i>canonicalKey</i> .
<i>canonicalKey</i>	String The canonical key with which you want to establish a cross-reference to <i>nativeId</i> .
<i>objectId</i>	String 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	890
--	-----

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
pub.trigger:createJMSTrigger	WmPublic. Creates a JMS trigger.
pub.trigger:createTrigger	WmPublic. Creates a webMethods messaging trigger.
pub.trigger:deleteJMSTrigger	WmPublic. Deletes a JMS trigger.
pub.trigger:deleteTrigger	WmPublic. Deletes a webMethods messaging trigger.
pub.trigger:disableJMSTriggers	WmPublic. Disables one or more JMS triggers.
pub.trigger:enableJMSTriggers	WmPublic. Enables one or more JMS triggers.
pub.trigger:resourceMonitoringSpec	WmPublic. Specification for the signature of a resource monitoring service.
pub.trigger:resumeProcessing	WmPublic. Resumes document processing for the specified webMethods messaging trigger.
pub.trigger:resumeRetrieval	WmPublic. Resumes retrieval of documents from the messaging provider for a specific webMethods messaging trigger.

Element	Package and Description
pub.trigger:suspendJMSTriggers	WmPublic. Suspends one or more JMS triggers.
pub.trigger:suspendProcessing	WmPublic. Suspends document processing for the specified webMethods messaging trigger.
pub.trigger:suspendRetrieval	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

<i>triggerName</i>	<p>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.</p> <p>Note: For a list of reserved words and symbols for element names, see <i>webMethods Service Development Help</i>.</p>
<i>package</i>	<p>String Name of the package in which you want to save the trigger.</p>
<i>aliasName</i>	<p>String Name of the JMS connection alias that you want this JMS trigger to use to receive messages from the JMS provider.</p> <p>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.</p> <p>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.</p>
<i>jmsTriggerType</i>	<p>String Type of JMS trigger. Specify:</p> <ul style="list-style-type: none"> ■ <code>Standard</code> 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>String Flag indicating whether the new JMS trigger is enabled or disabled. Set to:</p> <ul style="list-style-type: none"> ■ <code>true</code> to create the JMS trigger in an enabled state. ■ <code>false</code> to create the JMS trigger in a disabled state. This is the default.
<i>joinTimeout</i>	<p>String 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> <p>Note: You can specify a <i>joinTimeout</i> for a standard JMS trigger only. SOAP-JMS triggers cannot have joins.</p>
<i>joinType</i>	<p>String 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> <p>You only need to set <i>joinType</i> if the JMS trigger receives messages from multiple destinations.</p> <p>Note: You can specify a <i>joinType</i> for a standard JMS trigger only. SOAP-JMS triggers can</p>

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)

<i>retryInterval</i>	<p>Note: <i>maxRetryAttempts</i> applies to non-transacted JMS triggers only.</p> <p>String Optional. Length of time Integration Server waits between retry attempts. The default is 10 seconds.</p> <p>Note: <i>retryInterval</i> applies to non-transacted JMS triggers only.</p>
<i>onTransientError</i>	<p>String Flag indicating how Integration Server handles transient errors for the JMS trigger.</p> <p>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 <code>ISRuntimeException</code>.</p> <p>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.</p> <p>Specify one of the following:</p> <ul style="list-style-type: none">■ <code>Throw Exception/ Recover Only</code> 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 <code>ISRuntimeException</code>. 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.■ <code>Suspend and Retry Later/ Suspend and Recover</code> 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.

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.

<i>dupDetection</i>	<p>String Flag indicating whether exactly-once processing is enabled for the JMS trigger. Set to:</p> <ul style="list-style-type: none">■ <code>true</code> to specify that exactly-once processing is provided for messages received by this trigger.■ <code>false</code> to specify that exactly-once processing is not provided for messages received by this trigger. This is the default.
<i>dupHistory</i>	<p>String Flag indicating whether a document history database will be maintained and used to determine whether a message is a duplicate. Set to:</p> <ul style="list-style-type: none">■ <code>true</code> to indicate that Integration Server uses a document history database as part of exactly-once processing.■ <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.
<i>dupHistoryTTL</i>	<p>String Optional Number of milliseconds that the document history database maintains an entry for a document processed by this trigger.</p> <p>The default is 2 hours (7200000 milliseconds).</p>
<i>dupResolverSvcName</i>	<p>String Optional. Specifies the service that you created to determine whether message's status is New, Duplicate, or In Doubt.</p>
<i>prefetchSize</i>	<p>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.</p> <p>The default is 10.</p> <p>Note: This parameter applies only when working with the webMethods Broker as a JMS provider.</p>

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.

Key	Description
<i>destination</i>	<p>String 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>String Optional. Type of destination from which the JMS trigger receives messages. Set to:</p> <ul style="list-style-type: none"> ■ Queue to specify that the destination is a queue. This is the default.

	<ul style="list-style-type: none">■ <code>Topic</code> to specify that the destination is a topic.
<i>messageSelector</i>	String 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.
<i> durableSubscriberName</i>	String 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. Note: <i> durableSubscriberName</i> applies when <i> destinationType</i> is set to <code>Topic</code> only.
<i> durableSubscriberNoLocal</i>	String Optional. Flag indicating whether the JMS trigger ignores messages sent by the same Integration Server on which the JMS trigger resides. Set to: <ul style="list-style-type: none">■ <code>true</code> to indicate that the JMS trigger ignores messages sent by the same Integration Server on which the JMS trigger resides.■ <code>false</code> 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: <i> durableSubscriberNoLocal</i> applies when <i> destinationType</i> is set to <code>Topic</code> 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>	String Name for the routing rule.
<i> serviceName </i>	String Fully qualified name of the service Integration Server invokes when it receives a message from one of the specified destinations.
<i> filter </i>	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>	<p>String 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.</p> <p>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.</p> <p>Set <i>joinTimeOut</i> to -1 to indicate that the join condition never expires.</p> <p>The default is 1 day.</p>
<i>queueCapacity</i>	<p>String Maximum number of documents that Integration Server maintains in the queue for this trigger.</p> <p>The default is 10.</p>
<i>queueRefillLevel</i>	<p>String Number of unprocessed documents that must remain in the trigger queue before Integration</p>

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:

- `ThrowException` 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.

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.
- maxExecutionThreads* **String** 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.
- dupDetection* **String** Indicates whether Integration Server performs exactly-once processing for guaranteed documents received by this trigger. Set to:
- `true` to indicate that Integration Server performs exactly-once processing for guaranteed documents received by this trigger.
 - `false` to indicate that exactly-once processing is not performed. This is the default.
- dupHistory* **String** Indicates whether Integration Server uses a document history database as part of performing exactly-once processing. 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 Number of milliseconds that the document history database maintains an entry for a document processed by this trigger.

The default is 2 hours.

dupResolverSvcName

String 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>String 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>String 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>String 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"> ■ N/A to indicate this is not a join condition. ■ 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.

This is the default join type.

- **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.
- **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 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>	String Fully qualified name of the publishable document type or message to which the trigger subscribes.
<i>filter</i>	String Optional. Filter that you want applied to each received document. The trigger processes only those documents that meet the filter criteria. 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.

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.

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: At 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, 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.
- 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>	String List Specifies the JMS triggers that you want to disable.
<i>applyChangeAcrossCluster</i>	String 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">■ <code>true</code> to disable the specified JMS triggers on all the nodes in the cluster. <p>Note: 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>.</p> <ul style="list-style-type: none">■ <code>false</code> to disable the JMS triggers on the local Integration Server only. This is the default.

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>	String List Specifies the JMS triggers that you want to enable.
<i>applyChangeAcrossCluster</i>	<p>String 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"> ■ <code>true</code> to enable the specified JMS triggers on all the nodes in the cluster. <p>Note: 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"> ■ <code>false</code> to enable the JMS triggers on the local Integration Server only. This is the default.

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.

The *applyChangeAcrossCluster* parameter applies only to triggers that receive documents from the Broker.

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>	String Fully qualified name of the webMethods messaging trigger for which you want to resume document retrieval.
<i>persistChange</i>	String Optional. Flag indicating whether the document retrieval change should be permanent or temporary. Set to: <ul style="list-style-type: none">■ <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.■ <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.
<i>applyChangeAcrossCluster</i>	String. 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">■ <code>true</code> to resume document retrieval for the specified trigger on all the servers in the cluster. <div data-bbox="573 1520 1367 1766" style="background-color: #f0f0f0; padding: 10px;"><p>Note: 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">■ <code>false</code> to indicate that document retrieval for this trigger should be resumed on the local Integration Server only. This is the default.

The *applyChangeAcrossCluster* parameter applies only to triggers that receive documents from the Broker.

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>	String List Specifies the JMS triggers that you want to suspend.
<i>applyChangeAcrossCluster</i>	<p>String 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"> ■ <code>true</code> to suspend the specified JMS triggers on all the nodes in the cluster. <div style="background-color: #f0f0f0; padding: 10px; margin: 10px 0;"> <p>Note: 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"> ■ <code>false</code> to suspend the JMS triggers on the local Integration Server only. This is the default.

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>	String Fully qualified name of the webMethods messaging trigger for which you want to suspend document processing.
<i>persistChange</i>	String Optional. Flag indicating whether the document processing change should be permanent or temporary. Set to: <ul style="list-style-type: none">■ <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.■ <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.
<i>applyChangeAcross Cluster</i>	String 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">■ <code>true</code> to suspend document processing for the specified trigger on all the servers in the cluster.

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.

The *applyChangeAcrossCluster* parameter applies only to triggers that receive documents from the Broker.

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 Broker.

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>	String Fully qualified name of the webMethods messaging trigger for which you want to suspend document retrieval.
<i>persistChange</i>	<p>String Optional. Flag indicating whether the document retrieval change should be permanent or temporary. Set to:</p> <ul style="list-style-type: none"> ■ <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.

- `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 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.

The *applyChangeAcrossCluster* parameter applies only to triggers that receive documents from the Broker.

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 in the trigger's client queue on the Broker. 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 Broker discards any volatile documents in that trigger's client queue.

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	936
--	-----

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
pub.tx:init	WmPublic. Initializes the processing of inbound guaranteed delivery requests.
pub.tx:shutdown	WmPublic. Stops the processing of inbound guaranteed delivery requests.
pub.tx:resetOutbound	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>	String 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>	String 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	940
--	-----

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
pub.universalName:find	WmPublic. Returns the fully qualified service name for an explicit universal name.
pub.universalName.findDocumentType	WmPublic. Returns the fully qualified document type name for a provided explicit universal name
pub.universalName:list	WmPublic. Returns a list of services in the current universal-name registry.
pub.universalName:listAll	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>	String Namespace portion of the universal name.
<i>localName</i>	String Local portion of the universal name.

Output Parameters

<i>svcName</i>	String 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:

Key	Description
<i>universalName</i>	Document The universal name associated with the entry. This document contains the following information:

	<u>Key</u>	<u>Description</u>
	<i>namespaceName</i>	String Namespace portion of the universal name.
	<i>localName</i>	String Local portion of the universal name.
<i>svcName</i>		String 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>	Document Universal name associated with the entry. This document contains the following information:

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

	<i>namespaceName</i>	String Namespace portion of the universal name.
	<i>localName</i>	String Local portion of the universal name.
<i>svcName</i>		String 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	946
--	-----

The utils folder contains utility services.

Summary of Elements in this Folder

The following elements are available in this folder:

Element	Package and Description
pub.utils:deepClone	WmPublic. Clones an object using the default Java serialization mechanism.
pub.utils:executeOSCommand	WmPublic. Executes an operating system command such as <code>dir</code> in Windows or <code>ls</code> in UNIX.
pub.utils:generateUUID	WmPublic. Generates a random Universally Unique Identifier (UUID).
pub.utils:getServerProperty	WmPublic. Retrieves the value of a specified server property.
pub.utils.ws:setCompatibilityModeFalse	WmPublic. Changes the value of the Pre-8.2 compatibility mode 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
<code>allowedOSCommands</code>	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 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>).
<code>allowedWorkingDirectories</code>	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>String 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>Important: 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>String List 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>String List 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>String List 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>	<p>String The exit value of the executed command. The value 0 (zero) indicates normal termination.</p>
<i>outputMessage</i>	<p>String Output of the executed command.</p>
<i>errorMessage</i>	<p>String Errors that occurred during command execution.</p>

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

UUID **String** A randomly generated Universally Unique Identifier (UUID).

pub.utils:getServerProperty

WmPublic. Retrieves the value of a specified server property.

Input Parameters

propertyName **String** The name of the server property whose value you want to retrieve (for example, `watt.server.SOAP.directive`).

defaultValue **String** Optional. The default value to return if the server property specified in *propertyName* 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.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.

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>	String Package that contains the updated web service descriptor.
<i>name</i>	String Fully qualified name of the updated web service descriptor.
<i>warnings</i>	String List List of any warning that occurred when the service changed the Pre-8.2 compatibility mode 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.

Key	Description
<i>package</i>	String Package containing the web service descriptor that could not be changed.
<i>name</i>	String Fully qualified name of the web service descriptor that could not be changed because of an error.
<i>warnings</i>	String List List of the warnings that occurred when the service attempted to change the Pre-8.2 compatibility mode property from true to false.
<i>errors</i>	String List List of the errors that prevented the service from changing the value of the Pre-8.2 compatibility mode 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.

Key	Description
<i>package</i>	String Package containing the web service descriptor.
<i>name</i>	String Fully qualified name of the web service descriptors for which Pre-8.2 compatibility mode property was already set to false.

Usage Notes

Even though the *convertAllPackages* and *packageNameNames* input parameters are optional, you must either set *convertAllPackages* to `true` or specify packages in *packageNameNames*.

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	956
--	-----

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*.

Summary of Elements in this Folder

The following elements are available in this folder:

Element	Package and Description
pub.vcs.admin:getUsers	WmVCS. Returns a list of the Designer user accounts that are associated with a corresponding version control system (VCS) user account on the VCS server.
pub.vcs.admin:removeCurrentUser	WmVCS. 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.
pub.vcs.admin:removeMultipleUsers	WmVCS. Removes the specified Designer user accounts from the list of users associated with a version control system (VCS) user account.
pub.vcs.admin:setCurrentUser	WmVCS. Associates a version control system (VCS) user account with the currently logged in Designer user account.
pub.vcs.admin:setMultipleUsers	WmVCS. Associates a version control system (VCS) user account with the specified Designer user accounts.

pub.vcs.admin:getUsers

WmVCS. 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.

<u>Key</u>	<u>Description</u>
<i>devName</i>	String The name of the Designer user account.
<i>vcsName</i>	String 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. 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. 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 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. 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. 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:

Key	Description
<i>devName</i>	String The name of the Designer user account.
<i>vcsName</i>	String The name of an existing VCS user account on the VCS server.
<i>vcsPassword</i>	String 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	962
--	-----

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
pub.xml:documentToXMLString	WmPublic. Converts a document (IData object) to an XML string.
pub.xml:freeXMLNode	WmPublic. Frees the resources allocated to a given XML node.
pub.xml:getNextXMLNode	WmPublic. Gets the next XML node from a NodeIterator.
pub.xml:getXMLNodeIterator	WmPublic. Creates and returns a NodeIterator.
pub.xml:getXMLNodeType	WmPublic. Returns information about an XML node.
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.
pub.xml:loadXMLNode	WmPublic. Retrieves an XML document via HTTP or HTTPS, parses it using the legacy XML parser, and produces an XML node.
pub.xml:queryXMLNode	WmPublic. Queries an XML node.
pub.xml:xmlNodeToDocument	WmPublic. Converts an XML node to a document (an IData object).
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.
pub.xml:xmlStringToXMLNode	WmPublic. Converts an XML document (represented as a String, byte[], or

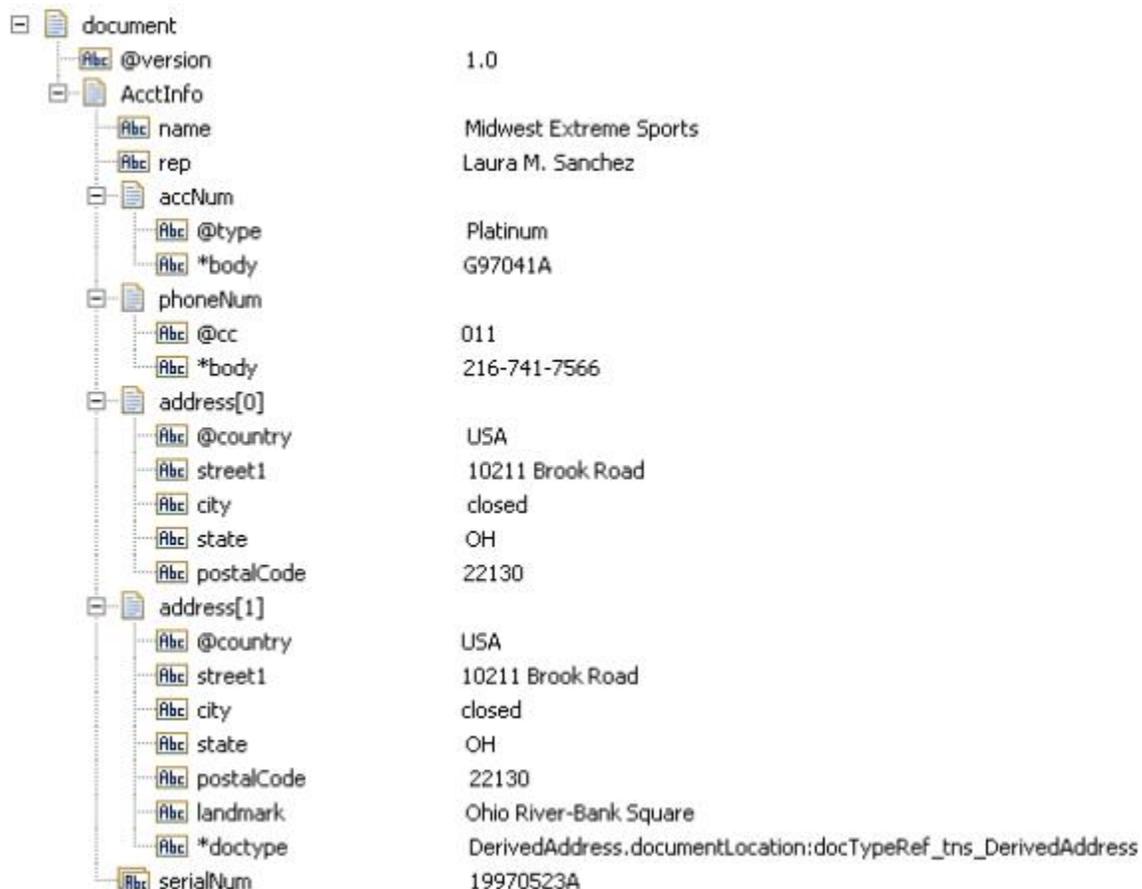
Element	Package and Description
	InputStream) to an XML node using the legacy XML parser.

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.

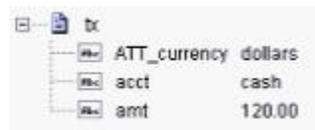
- Fields that are not String or Document based (for example, Floats or Integers) are converted to XML values using the underlying object's `toString` method.

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/bmonitor"
```

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 < 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>	String Optional. System identifier for the DTD, if any.
<i>publicID</i>	String Optional. Public identifier for the DTD, if any.
<i>rootNSPrefix</i>	String Optional. Namespace prefix of the <i>rootLocalName</i> , if any.
<i>rootLocalName</i>	String 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:



```

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.



```

address1
  @country USA
  street1 10211 Brook Road
  city    closed
  state   OH
  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* :



```

phoneNum
  @cc 011
  *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

<i>rootNode</i>	<p>XML node or enhanced XML node whose resources you want to release. This parameter supports the following types of input:</p> <ul style="list-style-type: none"> ■ 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. <p>Specify the same type of input that you supplied to <code>pub.xml:getXMLNodeIterator</code>.</p>
-----------------	--

Output Parameters

None.

pub.xml:getNextXMLNode

WmPublic. Gets the next XML node from a NodeIterator.

Input Parameters

<i>iterator</i>	com.wm.app.b2b.util.NodeIterator NodeIterator from which to retrieve the next node.
-----------------	--

Output Parameters

<i>next</i>	Document Conditional. The requested node. Will be null when the NodeIterator has no more nodes to return. Otherwise, <i>next</i> will contain the following:
-------------	---

Key	Description
<i>name</i>	String 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

<i>node</i>	<p>The XML node or enhanced XML node for which you want to produce a <code>NodeIterator</code>. This parameter supports the following types of input:</p> <ul style="list-style-type: none"> ■ com.wm.lang.xml.Node XML node for which you want to produce a <code>NodeIterator</code>. The node can represent either an XML document or an element of an XML document; however, if the <code>NodeIterator</code> 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 <code>NodeIterator</code>. An enhanced XML node can be produced by <code>pub.xml:loadEnhancedXMLNode</code>, <code>pub.xml:xmlStringToEnhancedXMLNode</code>, or a content handler that receives an XML document in a request for which <code>xmlFormat</code> is set to enhanced.
<i>criteria</i>	<p>String List Optional. Pattern strings identifying the nodes that the iterator is to return. A pattern string may take either the form <code><localName></code> or the form <code><prefix>:<localName></code>.</p> <p>When a pattern takes the first form, it identifies an element whose local name is <code><localName></code> and that belongs to the default XML namespace. When a pattern takes the second form, it identifies an element whose local name is <code><localName></code> and whose XML namespace is given by the prefix <code><prefix></code>. If the input parameter <code>nsDecls</code> declares this prefix, the namespace URI of the element must match the URI declared for the prefix. If the prefix is not declared in <code>nsDecls</code>, the prefix is matched against prefixes found in the XML.</p> <p><code><prefix></code> and <code><localName></code> can each optionally take the value <code>"*"</code> (asterisk) to match any namespace or local name.</p>

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 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*.

Key	Description
<i>type</i>	String 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>	String 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> .

	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>	<p>String 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>

data **Document** Optional. The data that you want `pub.xml:loadEnhancedXMLNode` 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>Document 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"> ■ URL-encode name=value pair, so you do not need to URL-encode the values you specify in <i>args</i> . ■ Insert the "&" character between pairs, so you do not need to include it in <i>args</i> .

- 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-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: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>String Optional. Determines whether Integration Server performs DTD validation. Set to:</p> <ul style="list-style-type: none"> ■ <code>true</code> to perform DTD validation. ■ <code>false</code> to disable DTD validation. This is the default.
<i>isNamespaceAware</i>	<p>String 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"> ■ <code>true</code> to enable namespace processing. This is the default. ■ <code>false</code> to disable namespace processing.
<i>isCoalescing</i>	<p>String Optional. Determines whether Integration Server coalesces adjacent character data while parsing the XML document. Set to:</p> <ul style="list-style-type: none"> ■ <code>true</code> to coalesce adjacent character data. ■ <code>false</code> to indicate that Integration Server does not coalesce adjacent character data. This is the default.
<i>isReplacingEntityReferences</i>	<p>String 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.

*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 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.

<i>status</i>	<p>String Conditional. The HTTP status code returned by the target server if an HTTP error occurs when loading the requested URL.</p> <p>The <code>pub.xml:loadEnhancedXMLNode</code> service returns <i>status</i> when an HTTP error occurs and <i>failOnHTTPError</i> is set to false.</p>
<i>statusMessage</i>	<p>String Conditional. The HTTP status message returned by the target server if an HTTP error occurs when loading the requested URL.</p> <p>The <code>pub.xml:loadEnhancedXMLNode</code> service returns <i>statusMessage</i> when an HTTP error occurs and <i>failOnHTTPError</i> is set to false.</p>

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>String 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>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 <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"> ■ 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 |

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>	String 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>	String 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>	String 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>Document 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"> ■ URL-encode name=value pair, so you do not need to URL-encode the values you specify in <i>args</i>. ■ Insert the "&" character between pairs, so you do not need to include it in <i>args</i>. ■ 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>. <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>String Table 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 `E2B/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 too 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>	String Name to assign to resulting value.																
<i>resultType</i>	String 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> <tr> <td>String[]</td> <td>String List</td> </tr> <tr> <td>String[][]</td> <td>String Table</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	String[]	String List	String[][]	String Table
<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>	String Query identifying the data to be extracted from <i>node</i> .
<i>queryType</i>	<p>String Query language in which <i>query</i> is expressed. Valid values are <code>WQL</code> and <code>XQL</code>.</p> <p>If the content of <i>node</i> is an enhanced XML node, you must set <i>queryType</i> to <code>XQL</code>.</p>
<i>onnull</i>	<p>String Code indicating what you want <code>queryXMLNode</code> to do when the result is null. Set to one of the following:</p> <ul style="list-style-type: none"> ■ <code>continue</code> to indicate that all result values are acceptable for this query (including null). ■ <code>fail</code> to indicate that the service should fail if the result of this query is null and continue in all other cases. ■ <code>succeed</code> to indicate that the service should continue if the result of this query is null and fail in all other cases.
<i>fields</i>	Document List 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).

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>	<code>originator</code>
<i>rep</i>	<code>originator</code>

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:



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

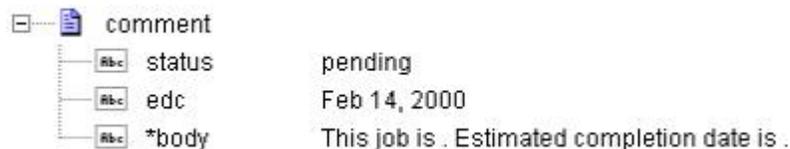
String Optional. Flag specifying how mixed-content elements (elements containing both text values and child elements) are to be converted. The following is an example of a mixed-content element:

```

<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.

Following are examples of XML documents and the documents (IData objects) that `xmlNodeToDocument` would produce.

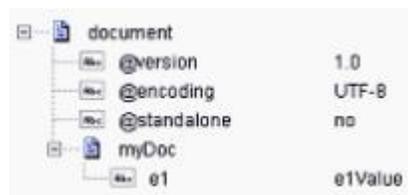
XML Document

```
<myDoc>
<e1>e1Value</e1>
</myDoc>
```

Output from xmlNodeToDocument



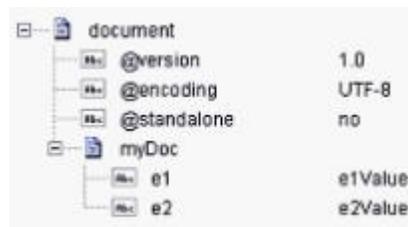
```
<?xml version="1.0" encoding="UTF-8"
standalone="no"?>
<myDoc>
<e1>e1Value</e1>
</myDoc>
```



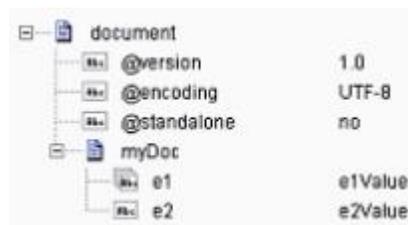
```
<?xml version="1.0" encoding="UTF-8"
standalone="no"?>
<myDoc>
<e1 e1Attr="attrValue">e1Value</e1>
</myDoc>
```



```
<?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>
```



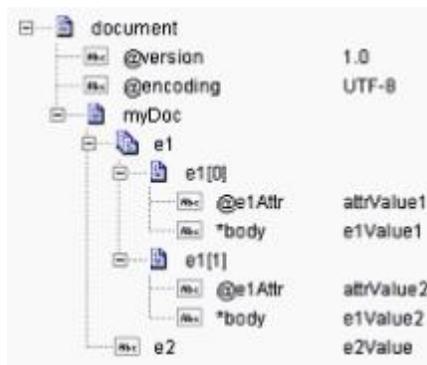
XML Document

```
<?xml version="1.0"
encoding="UTF-8"?>
<myDoc>
<e1 e1Attr="attrValue1">e1Value1</e1>
<e2>e2Value</e2>
<e1 e1Attr="attrValue2">e1Value2</e1>
</myDoc>
```

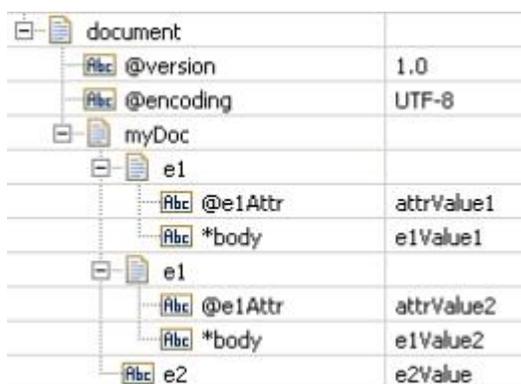
```
<?xml version="1.0"
encoding="UTF-8"?>
<myDoc>
<e1 e1Attr="attrValue1">e1Value1</e1>
<e2>e2Value</e2>
<e1 e1Attr="attrValue2">e1Value2</e1>
</myDoc>
```

```
<?xml version="1.0"
encoding="UTF-8"?>
<myDoc>
<e1 e1Attr="attrValue1">e1Value1</e1>
<e2>e2Value</e2>
<e1 e1Attr="attrValue2">e1Value2</e1>
</myDoc>
```

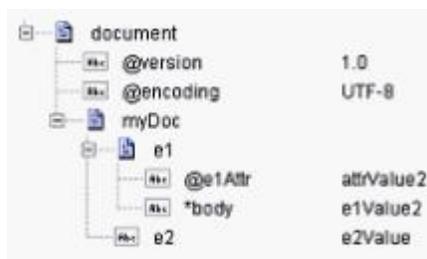
Output from xmlNodeToDocument



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.



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.



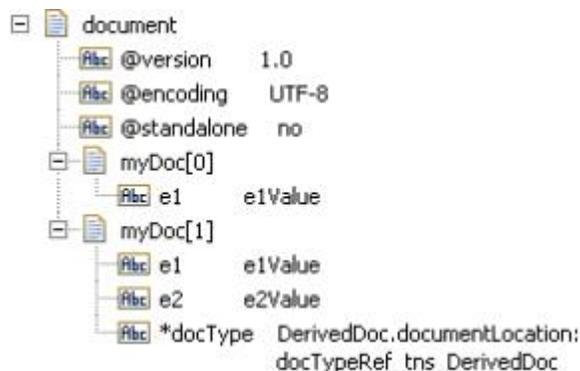
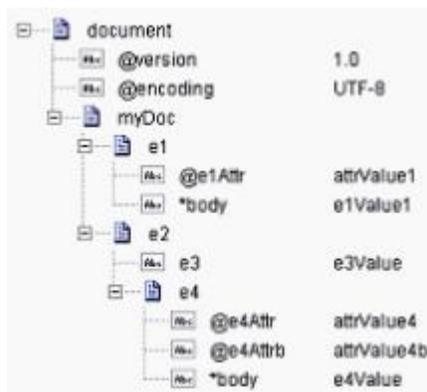
XML Document

```
<?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 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

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.



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

<i>xmldata</i>	<p>String Optional. String containing the XML document to convert to an XML node.</p> <p>Note: If you specify <i>xmldata</i>, do not specify <i>\$filedata</i> or <i>\$filestream</i>.</p>
<i>\$filedata</i>	<p>byte[] Optional. byte[] containing the XML document to convert to an XML node.</p> <p>Note: If you specify <i>\$filedata</i>, do not specify <i>xmldata</i> or <i>\$filestream</i>.</p>
<i>\$filestream</i>	<p>java.io.InputStream Optional. InputStream containing the XML document to convert to an XML node.</p> <p>Note: If you specify <i>\$filestream</i>, do not specify <i>xmldata</i> or <i>\$filedata</i>.</p>
<i>encoding</i>	<p>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 <code>autoDetect</code> (the default, if <i>encoding</i> is not specified).</p>
<i>inputProcessing</i>	<p>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 <code>javax.xml.stream.XMLInputFactory</code> class.</p>

Key	Description
<i>isValidating</i>	<p>String Optional. Determines whether Integration Server performs DTD validation. Set to:</p> <ul style="list-style-type: none"> ■ <code>true</code> to perform DTD validation. ■ <code>false</code> to disable DTD validation. This is the default.
<i>isNamespaceAware</i>	<p>String 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">■ <code>true</code> to enable namespace processing. This is the default.■ <code>false</code> to disable namespace processing.
<i>isCoalescing</i>	<p>String Optional. Determines whether Integration Server coalesces adjacent character data while parsing the XML document. Set to:</p> <ul style="list-style-type: none">■ <code>true</code> to coalesce adjacent character data.■ <code>false</code> to indicate that Integration Server does not coalesce adjacent character data. This is the default.
<i>isReplacingEntityReferences</i>	<p>String 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">■ <code>true</code> to replace entity references. This is the default.■ <code>false</code> to indicate entity references will not be replaced.
<i>isSupportingExternalEntities</i>	<p>String Optional. Determines whether Integration Server resolves external parsed entities while parsing the XML document. Set to:</p> <ul style="list-style-type: none">■ <code>true</code> to resolve external parsed entities.■ <code>false</code> to indicate Integration Server does not support external parsed entities. <p>The JVM in which Integration Server runs determines the default.</p>
<i>supportDTD</i>	<p>String Optional. Determines whether Integration Server supports DTDs while parsing the XML document. Set to:</p> <ul style="list-style-type: none">■ <code>true</code> to support DTDs while parsing the XML document. This is the default.■ <code>false</code> 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>String Optional. String containing the XML document to convert to an XML node.</p> <p>Note: If you specify <i>xmldata</i>, do not specify <i>\$filedata</i> or <i>\$filestream</i>.</p>
<i>\$filedata</i>	<p>byte[] Optional. byte[] containing the XML document to convert to an XML node.</p> <p>Note: If you specify <i>\$filedata</i>, do not specify <i>xmldata</i> or <i>\$filestream</i>.</p>
<i>\$filestream</i>	<p>java.io.InputStream Optional. InputStream containing the XML document to convert to an XML node.</p> <p>Note: If you specify <i>\$filestream</i>, do not specify <i>xmldata</i> or <i>\$filedata</i>.</p>
<i>encoding</i>	<p>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</p>

encoding, specify `autoDetect` (the default, if *encoding* is not specified).

expandDTD

String Optional. Flag indicating whether references to parameter entities in the XML document's DTD are to be processed. Set to:

- `true` to expand references to parameter entities to their full definition.
- `false` to ignore references to parameter entities. This is the default.

isXML

String Optional. Flag specifying whether the input document is XML or HTML. (`xmlStringToXMLNode` must know this so that it can parse the document correctly.) Set to:

- `autoDetect` to parse the document based on its type. When you use this option, `xmlStringToXMLNode` detects 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.

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 XSLT Folder

- Summary of Elements in this Folder 1014

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
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.
pub.xslt.Cache:removeAllTemplates	WmXSLT. Clears the XSLT stylesheet cache.
pub.xslt.Cache:removeTemplate	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>	String URI (simple file path or URL) for the XSLT stylesheet to use.
<i>systemId</i>	String 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>	String 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>	byte[] 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>	Input stream 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>	Document 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>	String Tells Designer what to transform the XML into. Must be one of these values: <ul style="list-style-type: none">■ <i>bytes</i> to transforms the XML into a byte array.■ <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.■ <i>xmlNode</i> to transforms the XML into an XML node.
<i>outFileName</i>	String 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>	Boolean. 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">■ <i>true</i> to use the <code>org.apache.xalan.xsltc.trax.TransformerFactoryImpl</code> class as a transformer factory.■ <i>false</i> to use the transformer factory that is specified on the home page of the WmXSLT package. This is the default. 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 490
- access permissions assigned to built-in services 24
- ACLs assigned to built-in services 24
- action 808
- activatePackage 542
- addBodyBlock 720
- addBodyEntry 776
- addBodyPart 513
- addComplexTask 621
- addFaultBlock 722
- addFloatList 490
- addFloats 491
- addHeaderBlock 724
- addHeaderEntry 777
- adding
 - body parts to MIME messages 513
 - documents to list 481
 - entries into a data store 843
 - MIME message headers 517
 - numeric values 490, 491, 492, 493
 - strings to list 482
- addIntList 492
- addInts 493
- addItemToVector 480
- addMimeHeader 517
- addObjects 493
- addOnetimeTask 624
- addReleaseRegistryEntry 603
- addRepeatingTask 626
- addSubscriber 241
- addTrailer 779
- alarm specification 245
- alarmInfo document type 246
- appending data to a remote file 92
- appendToDocumentList 481
- appendToStringList 482
- arithmetic services 488
- ART folder 28
- ART services 28
- asynchronous call to a remote server 599
- asynchronous request/reply
 - description of 581
- asynchronous request/reply
 - delivering request 565
 - description of 566

- publishing request 579
 - publishing request to a specific client 565
 - retrieving reply 587
- audit specification 246
- auditError event 248
- auditErrorInfo document type 249
- auditInfo document type 250

B

- backupPackage 543
- base64 Decode 858
- base64Encode 858
- binding
 - output template to Values object 616
- BPEL folder 507, 935
- byte 386
- byte array from string 869
- bytesToDocument 223
- bytesToFile 300
- bytesToStream 387
- bytesToString 859

C

- Cache folder 53
- calculateDateDifference 181
- call 194
- calling stored procedures 194
- callStackItem document type 251
- cancelTask 629
- canonical document, definition 883
- canonical keys
 - creating 883
 - inserting 887
 - retrieving 885
- cd 93, 135
- cdgrp 136
- cdls 93, 107
- certificate chain 661
- certs-only S/MIME Message
 - creating 695
 - extracting certificates from 700
- changing the working directory 93, 135
- character sequence, indexing first occurrence 861
- checkFileExistence 301
- chmod 136
- chown 137
- clear pipeline 350
- clearing transactional state 196

- clearKeyAndChain 661
 - clearTransaction 196
 - Client folder 84
 - close 388
 - close connection 197
 - closeAll 198
 - closeLatch 880
 - closeStore 844
 - closing database connection 197
 - committing changes to a database 198
 - commitTransaction 49
 - compatibility mode 949
 - complex tasks
 - adding to scheduler 621
 - updating to scheduler 637
 - components
 - XML schema definitions 655
 - concatenating
 - array of strings 864
 - strings 859
 - connections
 - creating 199
 - containsKey 380
 - content type, getting from MIME message 525
 - converting
 - string list to document list 483
 - XML nodes to IData objects 996
 - converting to
 - InputStream 386
 - convertToString service 314
 - convertToValues service 319
 - copyFile 302
 - createByteArray 388
 - createCertsOnlyData 695
 - createDocumentType service 329
 - createEncryptedData 695
 - createFFDictionary service 329
 - createHashtable 381
 - createJMSTrigger 891
 - createMessageDigest 676
 - createMimeData 519
 - createSignedAndEncryptedData 554, 696
 - createSignedData 555, 698
 - createSoapData 785
 - createTrigger 904
 - createXReference 883
 - createXSD 649
 - creating MIME messages 519
 - cross-references
 - creating 883
 - deleting by object ID 884
 - deleting individual records 884
 - inserting 887
 - current VCS user
 - removing 957
 - setting 958
 - currentNanoTime 182
- D**
- dateTimeFormat 185
 - data
 - converting 386
 - waiting for delivery 877
 - data stores
 - adding entries 843
 - closing 844
 - deleting 844
 - inserting entries 850
 - registering 851
 - removing entries 852
 - retrieving values 845
 - unlocking entries 850, 853
 - unregistering 844
 - updating entries 850
 - database connection, closing 197, 198
 - database services 192
 - databases
 - clearing transactional state 196
 - closing connections 197, 198
 - committing changes 198
 - creating connections 199
 - deleting rows 201
 - discarding changes 216
 - invoking stored procedures 194
 - querying 215
 - retrieving names of stored procedures 208
 - retrieving tables 211
 - starting a transaction 217
 - datatypes in XML schemas 655, 655
 - Date folder 175
 - dateBuild 183
 - dates
 - converting formats 185
 - formatting 187
 - invalid dates 180
 - returning current 188, 188
 - symbols used for 176

- time zones 177
- dateTimeBuild 184
- debug log 350
- debugging services 348
- decoding URL-encoded strings 872
- decrypting
 - MIME messages 700
 - S/MIME messages 556
- deepClone 946
- default key, associating with invoked services 661
- deleteById 884
- deleteFFDictionary service 330
- deleteFFDictionaryEntry service 331
- deleteFFSchema service 332
- deleteFile 303
- deleteJMSTrigger 917
- deleteReleaseRegistryEntry 604
- deleteStore 844
- deleteSubscriber 252
- deleteXReference 884
- deleting
 - cross-reference by object ID 884
 - cross-references 884
 - data stores 844
 - files in working directory 94
 - rows from a database 201
 - subscribers from subscription list 252
- deliver service 563
- deliverAndWait service 564
- digital signatures
 - attaching to MIME messages 698
 - verifying 557, 675, 701
- dir 95
- directory
 - changing 93
- disableConnection 31
- disableListener 34
- disableListenerNotification 39
- disablePackage 544
- disablePollingNotification 39
- disablePublishEvents 40
- distributeViaFTP 605
- distributeViaSvcPull 606
- distributeViaSvcPush 607
- divideFloats 494
- divideInts 495
- divideObjects 496
- document types
 - sessionStartInfo 290
- documentResolverSpec specification 567
- Document folder 221
- document processing
 - resuming 922
 - suspending 929
- document resolver spec, for JMS messages 408
- document retrieval
 - resuming 925
 - suspending 931
- document services 222
- document types
 - alarmInfo 246
 - audit error event information 249
 - audit event information 250
 - callStackItem 251
 - exception information 260
 - gdEndInfo 262
 - gdStartInfo 263
 - portStatusInfo 281
 - QName 798
 - replicationInfo 282
 - sessionEndInfo 287
 - sessionExpireInfo 289
 - statInfo 292
 - txEndInfo 295
 - txStartInfo 296
- documentation
 - using effectively 21
- documentListToDocument 224
- documents
 - adding to list 481
 - constructing from a list of documents 224
 - converting a byte array to a document 223
 - converting a document to an array of bytes 226
 - converting list from string list 483
 - converting to a String 230
 - converting to JSON strings 475
 - converting to XML strings 963
 - creating from JSON streams 476
 - creating from JSON strings 477
 - delivering 563
 - delivering and waiting for reply 564
 - expanding contents into a list of documents 228
 - publishing 575
 - publishing and waiting for reply 578
 - replying to 582
 - resolving status of 567
 - retrieving redelivery count 574
 - synchronizing 585, 586

- waiting for 587
 - documentToBytes 226
 - documentToDocumentList 228
 - documentToJSONString 475
 - documentToXMLString 963
 - documentToXMLValues 230
 - dynamic text generation 616
- E**
- eda
 - event 253
 - eventToDocument 255
 - schema_event 257
 - elapsedNanoTime 187
 - elements, number of in a list 483
 - enableConnection 32
 - enableListener 35
 - enableListenerNotification 40
 - enablePackage 545
 - enablePollingNotification 41
 - enablePublishEvents 41
 - encoding schema, SOAP 775
 - encoding schema, SOAP, version 1.2 775
 - encoding URL strings 873
 - encrypting MIME messages 554, 695
 - ending guaranteed delivery transactions 595
 - Enterprise Gateway specification 664
 - entries
 - inserting into repository 843
 - envelope schema, SOAP 775
 - envelope schema, SOAP, version 1.2 775
 - envelope stream, generating from MIME message 525
 - envelope, for published documents 569
 - error event 257, 257
 - error notification document type 588
 - error service 588
 - event
 - creating subscription for 241
 - sending 277
 - event document type 253
 - Event folder 237
 - event handlers
 - get list of subscribers 264
 - event handlers 238
 - alarm specification 245
 - alarmInfo document type 246
 - audit specification 246
 - auditError specification 248
 - auditErrorInfo document type 249
 - auditInfo document type 250
 - callStackItem document type 251
 - exception specification 258
 - exceptionInfo document type 260
 - gdEnd specification 261
 - gdEndInfo document type 262
 - gdStart specification 262
 - gdStartInfo document type 263
 - get list of event types 264
 - modify subscriber info 272
 - portStatus specification 280
 - portStatusInfo document type 281
 - reload event manager settings 281
 - replication specification 282
 - replicationInfo document type 282
 - save event manager settings 283
 - sessionEnd specification 287
 - sessionEndInfo document type 287
 - sessionExpire specification 288
 - sessionExpireInfo document type 289
 - sessionStart specification 289
 - sessionStartInfo document type 290
 - stat specification 291
 - statInfo document type 292
 - subscriptions for 241
 - txEnd specification 294
 - txEndInfo document type 295
 - txStart specification 295
 - txStartInfo document type 296
 - unsubscribing from an event 252
 - eventToDocument 255, 276
 - exception specification 258
 - exceptionInfo document type 260
 - exceptions for retry 368
 - exceptions, last trapped in a flow 351
 - execSQL 203
 - executeOSCommand 947
 - exitUnableToUnderstand 788
 - extensions
 - files with no extension, using with FTP put command 107
 - extracting MIME content 522
 - extracting MIME message headers 523, 527
- F**
- faultTo 808

- FFDictionary IS document type 332
- FFSchema IS document type 335
- fields, removing from pipeline 350
- File folder 297
- file object, templates 614
- file services 298, 398
- files
 - appending data to 92
 - change owner 137
 - change permissions 136
 - deleting 94
 - listing 93, 95, 102, 107, 140
 - multiple delete 103
 - multiple get 104
 - multiple transfer 105
 - renaming 108
 - retrieving 95
 - saving pipeline contents 358
 - transferring 106
 - transferring SFTP 141
- findDependants service 338
- finding universal names 940
- findReferences service 338
- fire-and-forget call 598
- floating point numbers
 - adding 490
 - dividing 494
 - multiplying 497
 - subtracting 502
- Flow folder 347
- flow services 348
- forcing a response string 360
- formatDate 187
- FormatService service 325
- formatting
 - array of strings 864
 - number to numeric pattern 865
- freeXMLNode 970
- from 810
- FTP
 - changing directory and listing files 93, 107
 - changing working directory 93, 135
 - client actions 88
 - delete file 94
 - executing non-standard FTP commands 108
 - files with no extension 107
 - filetransfer 106
 - ftp_no_extensionkey 107
 - listing files 95, 102, 140

- login 98
- logout 101, 140
- multiple file delete 103
- multiple file get 104
- multiple file transfer 105
- renaming files 108
- retrieving files 95
- session info 109
- transferring files 106

FTP servers, submitting requests to 110

G

- gdEnd specification 261
- gdEndInfo document type 262
- gdStart specification 262
- gdStartInfo document type 263
- generateDocumentTypesFromWSDL 731
- generateReplicationEvent 607
- generateUUID 949
- get 381
- get (FTP) files 95
- get (SFTP) files 138
- getActor 788
- getBody 789
- getBodyBlock 734
- getBodyBlockQNames 735
- getBodyEntries 790
- getBodyPartContent 522
- getBodyPartHeader 523
- getCanonicalKey 885
- getCertificateInfo 677
- getConnectionStatistics 32
- getContentType 525
- getCurrentDate 188
- getCurrentDateString 188
- getDocument 790
- getEncoding 791
- getEnvelopeStream 525
- getEventTypes 264
- getFaultBlock 736
- getFFDictionaryAsXML service 339
- getFFDictionaryEntryAsXML service 340
- getFFSchemaAsXML service 340
- getFile 303
- getHeader 792
- getHeaderEntries 792
- getInitialSOAPRequest 744
- getLastError 351

getLocalReleasedList 607
 getMessageAddressingProperties 745
 getMimeHeader 527
 getMustUnderstand 793
 getNativeId 886
 getNextXMLNode 971
 getNumParts 528
 getPoolInfo 398
 getPrimaryContentType 529
 getProclInfo 207
 getProcs 208
 getQName 794
 getRedeliveryCount service 574
 getRemoteReleasedList 608
 getSession 353
 getStatus 595
 getSubContentType 530
 getSubscribers 264
 getSupportedEncodings 328
 getTableInfo 210
 getTables 211
 getTaskIDs 629
 getTaskInfo 630
 getTrailer service 795
 getTransportInfo 353
 getUsers 956
 getWebServiceInvocationProperties 751
 getWorkingDays 189
 getXMLNodeIterator 972
 getXMLNodeType 975
 guaranteed delivery services 592
 guaranteed delivery transactions

- ending 595
- getting status of 595
- invoking service 596
- restarting 597
- retrieving results of 597
- starting 598

 guaranteed one-way call 598

H

handling large flat files

- iterator variable of convertToValues service 320

 Hashtable folder 379
 hashtable services 380
 header fields, adding to MIME object 517
 headers, changing HTTP 110
 HTMLDecode 860

HTMLEncode 860
 HTTP

- changing header variables 110
- response code 364
- response header 365
- response headers 368
- submitting requests to servers 110

I

IData objects

- converting to JSON strings 475
- converting to XML strings 963
- creating from JSON streams 476
- creating from JSON strings 477

 IDs, retrieving list of 629
 indexOf 861
 InputStream, converting to byte 386
 inserting table rows 213
 insertXReference 887
 installPackage 545
 integers

- adding 492
- dividing 495
- multiplying 499
- subtracting 504

 invalid dates 180
 invoke, guaranteed delivery service 596
 invokeService 354
 invoking

- remote services 592
- report services 614
- stored procedures 194

 invoking a service

- guaranteed delivery 599
- protocols used 370
- remote webMethods Integration Server 593

 IO services 386
 IS Internal 838
 isLatchClosed 881
 iterating through XML nodes 972

J

JDBC folder 397
 JDBC pools 398
 JDBC-related services 192
 JMS

- sending an event 277

 JMS folder 401

- JMS messages
 - send batch 454
 - sending multiple 454
 - JMS trigger
 - create 891
 - delete 917
 - journal event 269, 270
 - JSON streams
 - converting to IData objects 476
 - JSON strings
 - converting to IData objects 477
 - creating from documents 475
 - jsonStreamToDocument 476
 - jsonStringToDocument 477
- K**
- key string lookup 862, 863
 - keys, obtaining a list in a data store 846
- L**
- large flat file handling
 - iterator variable of convertToValues service 320
 - latch operations
 - checking latches 881
 - closing 880
 - opening 882
 - length
 - of a string 862
 - list
 - adding, retrieving, or replacing elements 480
 - number of elements 483
 - registered consumer handlers 755
 - registered processors 770
 - registered provider handlers 756
 - universal names 941
 - list services 480, 480
 - listAdapterConnections 33
 - listAdapterListenerNotifications 42
 - listAdapterListeners 36
 - listAdapterPollingNotifications 42
 - listAdapterServices 47
 - listAll 942
 - listFFDictionaryEntries service 341
 - listFiles 305
 - listKeys 381, 672
 - listRegisteredAdapters 31
 - loadEnhancedXMLNode 975
 - loadPKCS7CertChain 678
 - loadXMLNode 985
 - locking a repository entry 847, 848, 851
 - locking considerations for pub.storage services 839
 - log off of FTP server 101, 140
 - logging
 - pipeline fields 369
 - login to FTP server 98
 - login to SFTP server 139
 - lookupDictionary 862
 - lookupTable 863
 - lowercase, converting to 871
 - ls 102, 140
- M**
- makeString 864
 - mark 389
 - markSupported 390
 - Math folder 487
 - math services 488
 - max 496
 - mdelete 103
 - mergeHeaderAndBody 531
 - messageFormat 864
 - messageID 811
 - mget 104
 - MIME messages
 - adding body parts to 513
 - adding headers to 517
 - creating 519
 - creating encrypted messages 695
 - creating signed messages 698
 - decrypting 700
 - discovering content type 525
 - discovering number of parts 528
 - discovering subtype 530
 - generating message stream 525
 - getting content from 522
 - getting headers from 527
 - getting part headers from 523
 - getting top-level portion 529
 - merging HTTP response into InputStream 531
 - sending through SMTP 145
 - signing and encrypting 554, 696
 - verifying signed messages 701
 - MIME services 512
 - min 497
 - mkdir 141
 - modifySubscriber 272

moveFile 305
mput 105
multiple VCS users
 removing 958
 setting 959
multiplyFloatList 497
multiplyFloats 498
multiplyIntList 499
multiplyInts 500
multiplyObjects 501

N

namespace components 656, 656, 656
native ID, retrieving 886
nerv
 eventToDocument 276
node iterator
 creating 972
 freeXMLNode 970
 getNextXMLNode 971
 getXMLNodeIterator 972
notification error document type 588
notifying services 876
notifyPackageRelease 609
numeric values
 dividing 495
numeric values
 adding 490, 491, 492, 493
 dividing 496
 multiplying 497, 498, 499, 500, 501
 subtracting 502, 504
numericFormat 865

O

OAuth client 131
OAuth folder 533
objects
 adding 493
 dividing 496
 multiplying 501
 subtracting 504
objectToString 866
obtaining information about 398
one-time tasks
 adding to scheduler 624
 updating to scheduler 640
one-way call, guaranteed 598
openLatch 882

outbound passwords 658
 creating a WmSecureString 678
 internal vs. public 670
 listing keys 672
 removing 672
 retrieving 671
 setting 669
 updating 673
output templates 614

P

package management services 542
Package Release Registry
 adding entries 603
 deleting entries 604
 obtaining local server list 607
package replication services 602
packageCreation 610
packages
 activating 542
 backing up 543
 disabling 544
 enabling 545
 installing 545
 recovering 547
 reloading 548
padLeft 867
padRight 868
pattern strings, date patterns 176
pipeline
 applying templates to 616
 inserting Session object 353
 removing fields 350
 restoring previously saved 354, 355
 saving contents to a file 358
 saving into memory 357
 tracing 369
 using previously saved keys and values 356
 validating against a document type 654
pipeline services 348
PKCS7 signatures 550
PKCS7 SignedData objects 551
 creating 673
PKI profiles 550
portStatus specification 280
portStatusInfo document type 281
Pre-8.2 compatibility mode property 949
problemAction 811

- problemHeaderQName 812
- problemIRI 812
- processCertsOnlyData 700
- processEncryptedData 556, 700
- processing
 - resuming for triggers 922
 - suspending for triggers 929
- processMessage 772
- processRPCMessage 772
- processSignedData 557, 701
- protocols, retrieving information about 370
- protocols, retrieving information about 353
- pseudorandom number generator 501
- pub.b.client.sftp
 - chmod 136
- pub.b.publish.notification
 - error 588
- pub
 - flow
 - setResponse2 362
- 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.client
 - ftp 88, 89
 - http 110
 - smtp 145
 - soapClient 149
 - soapHTTP 167
 - soapRPC 170
- pub.client.ftp
 - append 92
 - cd 93
 - cdls 93
 - delete 94
 - dir 95
 - get 95
 - login 98
 - logout 101, 140
 - ls 102, 140
 - mdelete 103
 - mget 104
 - mput 105, 105
 - put 106, 141
 - putCompletedNotification 107
 - quote 108
 - rename 108
 - sessioninfo 109
- pub.client.oauth
 - executeRequest 131
- pub.client.sftp
 - cd 135
 - chgrp 136
 - chown 137
 - get 138
 - login 139
 - mkdir 141
 - pwd 143
 - rename 143
 - rm 144
 - rmdir 144
 - symlink 145
- pub.date
 - calculateDateDifference 181
 - currentNanoTime 182
 - dateBuild 183
 - dateTimeBuild 184

- dateTimeFormat 185
- elapsedNanoTime 187
- formatDate 187
- getCurrentDate 188
- getCurrentDateString 188
- getWorkingDays 189
- pub.db
 - call 194
 - clearTransaction 196
 - close 197
 - closeAll 198
 - commit 198
 - connect 199
 - delete 201
 - execSQL 203
 - getProclInfo 207
 - getProcs 208
 - getTableInfo 210
 - getTables 211
 - insert 213
 - query 215
 - rollback 216
 - startTransaction 217
 - update 218
- pub.document
 - bytesToDocument 223
 - documentListToDocument 224
 - documentToBytes 226
 - documentToDocumentList 228
 - documentToXMLValues 230
 - XMLValuesToDocument 234
- pub.event.eda
 - event 253
- pub.event
 - addSubscriber 241
 - alarm 245
 - alarmInfo 246
 - audit 246
 - auditError 248
 - auditErrorInfo 249
 - auditInfo 250
 - callstackItem 251
 - deleteSubscriber 252
 - error 257
 - errorInfo 257
 - exception 258
 - exceptionInfo 260
 - gdEnd 261
 - gdEndInfo 262
 - gdStart 262
 - gdStartInfo 263
 - getEventTypes 264
 - getSubscribers 264
 - journal 269
 - journalInfo 270
 - modifySubscriber 272
 - portStatus 280
 - portStatusInfo 281
 - reloadEventManagerSettings 281
 - replication 282
 - replicationInfo 282
 - saveEventManagerSettings 283
 - security 284
 - securityInfo 285
 - sessionEnd 287
 - sessionEndInfo 287
 - sessionExpire 288
 - sessionExpireInfo 289
 - sessionStart 289
 - sessionStartInfo 290
 - stat 291
 - statInfo 292
 - txEnd 294
 - txEndInfo 295
 - txStart 295
 - txStartInfo 296
- pub.event.eda
 - eventToDocument 255
 - schema_event 257
- pub.event.nerv
 - eventToDocument 276
 - send 277
- pub.file
 - bytesToFile 300
 - checkFileExistence 301
 - copyFile 302
 - deleteFile 303
 - getFile 303
 - listFiles 305
 - moveFile 305
 - readerToFile 307
 - streamToFile 308
 - stringToFile 308
- pub.flatFile
 - convertToString 314
 - convertToValues 319
 - FormatService 325
 - getSupportedEncodings 328

- pub.flatFile.generate
 - createDocumentType 329
 - createFFDictionary 329
 - deleteFFDictionary 330
 - deleteFFDictionaryEntry 331
 - deleteFFSchema 332
 - FFDictionary IS document type 332
 - FFSchema IS document type 335
 - findDependants 338
 - findReferences 338
 - getFFDictionaryAsXML 339
 - getFFDictionaryEntryAsXML 340
 - getFFSchemaAsXML 340
 - listFFDictionaryEntries 341
 - saveXMLAsFFDictionary 342
 - saveXMLASFFSchema 343
 - updateFFDictionaryEntryFromXML 345
- pub.flow
 - clearPipeline 350
 - debugLog 350
 - getLastError 351
 - getRetryCount 352, 903
 - getSession 353
 - getTransportInfo 353
 - invokeService 354
 - restorePipeline 355
 - restorePipelineFromFile 356
 - savePipeline 357
 - savePipelineToFile 358
 - setCustomContextID 360
 - setResponse 360
 - setResponseCode 364
 - setResponseHeader 365
 - setResponseHeaders 368
 - throwExceptionForRetry 368
 - tracePipeline 369
 - transportInfo 370
- pub.hashtable
 - containsKey 380
 - createHashtable 381
 - get 381
 - listKeys 381
 - put 382
 - remove 382
 - size 383
- pub.io
 - bytesToStream 387
 - close 388, 388
 - createByteArray 388, 388
 - mark 389
 - markSupported 390
 - read 391, 391
 - readAsString 391
 - readerToString 392
 - reset 393, 393
 - skip 393, 393
 - streamToBytes 394, 394
 - streamToReader 395
 - streamToString 395
 - stringToReader 396
 - stringToStream 396
- pub.jms
 - documentResolverSpec 408
 - reply 417
 - send 425
- pub.jms.wmjms
 - sendStream 470
- pub.json
 - documentToJSONString 475
 - jsonStreamToDocument 476
 - jsonStringToDocument 477
- pub.list
 - addItemToVector 480
 - appendToDocumentList 481
 - appendToStringList 482
 - sizeOfList 483
 - stringListToDocumentList 483
 - vectorToArray 484
- pub.math
 - absoluteValue 490
 - addFloatList 490
 - addFloats 491
 - addIntList 492
 - addInts 493
 - addObjects 493
 - divideFloats 494
 - divideInts 495
 - divideObjects 496
 - max 496
 - min 497
 - multiplyFloatList 497
 - multiplyFloats 498
 - multiplyIntList 499
 - multiplyInts 500
 - multiplyObjects 501
 - randomDouble 501
 - roundNumber 502
 - subtractFloats 502

- subtractInts 504
- subtractObjects 504
- toNumber 505
- pub.mime
 - addBodyPart 513
 - addMimeHeader 517
 - createMimeData 519
 - getBodyPartContent 522
 - getBodyPartHeader 523
 - getContentType 525
 - getEnvelopeStream 525
 - getMimeHeader 527
 - getNumParts 528
 - getPrimaryContentType 529
 - getSubContentType 530
 - mergeHeaderAndBody 531
- pub.packages
 - backupPackage 543
 - disablePackage 544
 - enablePackage 545
 - installPackage 545
 - recoverPackage 547
 - reloadPackage 548
- pub.pki.pkcs7
 - sign 551
 - verify 552
- pub.pki.smime
 - createSignedAndEncryptedData 554
 - createSignedData 555
 - processEncryptedData 556
 - processSignedData 557
- pub.publish
 - deliver 563
 - deliverAndWait 564
 - documentResolverSpec 567
 - envelope 569
 - getRedeliveryCount 574
 - publish 575
 - publishAndWait 578
 - reply 582
 - syncDocuments 585
 - syncToProvider 586
 - waitForReply 587, 587
- pub.remote
 - invoke 593
- pub.remote.gd
 - end 595
 - getStatus 595
 - invoke 596
- restart 597
- retrieve 597
- send 598
- start 598
- submit 599
- pub.replicator
 - addReleaseRegistryEntry 603
 - deleteReleaseRegistryEntry 604
 - distributeViaFTP 605
 - distributeViaSvcPull 606
 - distributeViaSvcPush 607
 - generateReplicationEvent 607
 - getLocalReleasedList 607
 - getRemoteReleasedList 608
 - notifyPackageRelease 609
 - packageCreation 610
- pub.report
 - runFileTemplate 614
 - runFileTemplateOnPipe 615
 - runStringTemplate 615
 - runStringTemplateOnPipe 616
 - runTemplate 616
 - runTemplateOnPipe 617
- pub.scheduler
 - addComplexTask 621
 - addOnetimeTask 624
 - addRepeatingTask 626
 - cancelTask 629
 - getTaskIDs 629
 - getTaskInfo 630
 - getUserTaskList 634
 - migrateTasksToJDBC 634
 - resumeTask 635
 - suspendTask 636
 - updateComplexTask 637
 - updateOneTimeTask 640
 - updateRepeatingTask 642
- pub.schema
 - createXSD 649
 - validate 651
 - validatePipeline 654
- pub.schema.w3c 655
 - datatypes 655
 - structures 655
 - xml 656
 - xsi 656
- pub.security
 - clearKeyAndChain 661
 - setKeyAndChain 662

- setKeyAndChainFromBytes 663
- pub.security.enterpriseGateway
 - alertSpec 664
- pub.security.keystore
 - getCertificate 666
 - getKeyAndChain 667, 667
 - getTrustedCertificates 667
 - setKeyAndChain 668
- pub.security.keystore.pkcs7
 - sign 668
- pub.security.outboundPasswords
 - listKeys 672
 - removePassword 672
 - setPassword 669
 - updatePassword 673
- pub.security.pkcs7
 - sign 673
 - verify 675
- pub.security.util
 - convertSecureString 679
 - createMessageDigest 676
 - createSecureString 678
 - destroySecureString 680
 - getCertificateInfo 677
 - loadPKCS7CertChain 678
- pub.smime
 - createCertsOnlyData 695
 - createEncryptedData 695
 - createSignedAndEncryptedData 696
 - createSignedData 698
 - processCertsOnlyData 700
 - processEncryptedData 700
 - processSignedData 701
- pub.smime.keystore
 - createSignedAndEncryptedData 704
 - createSignedData 706
 - processEncryptedData 706
- pub.soap.p.wsa
 - to 815
- pub.soap.handler
 - registerConsumer 760
- pub.soap.handler
 - addBodyBlock 720
 - addFaultBlock 722
 - addHeaderBlock 724
 - generateDocumentTypesFromWSDL 731
 - getBodyBlock 734
 - getBodyBlockQNames 735
 - getFaultBlock 736
 - getInitialSOAPRequest 744
 - getMessageAddressingProperties 745
 - getServicePipeline 748
 - getWebServiceInvocationProperties 751
 - hasFaultMessage 754
 - listConsumer 755
 - listProvider 756
 - registerConsumer 728, 741, 747, 750, 753, 757, 758, 764, 765, 766, 767
 - registerProvider 758
 - removeBodyBlock 761
 - removeHeaderBlock 762
 - unregisterConsumer 767
 - unregisterProvider 768
 - updateFaultBlock 768
- pub.soap.processor
 - list 770
 - processMessage 772
 - processRPCMessage 772
 - registerProcessor 773
 - unregisterProcessor 774
- pub.soap.schema
 - encoding 775
 - encoding_1_2 775
 - envelope 775
 - envelope_1_2 775
- pub.soap.utils
 - stringToSoapData 806
- pub.soap.utils
 - addBodyEntry 776
 - addHeaderEntry 777
 - addTrailer 779
 - createSoapData 785
 - exitUnableToUnderstand 788
 - getActor 788
 - getBody 789
 - getBodyEntries 790
 - getDocument 790
 - getEncoding 791
 - getHeader 792
 - getHeaderEntries 792
 - getMustUnderstand 793
 - getQName 794
 - getTrailer 795
 - QName 798
 - removeBodyEntry 798
 - removeHeaderEntry 799
 - removeTrailer 800
 - requestResponseSpec 801

- soapDataToBytes 803
- soapDataToString 803
- streamToSoapData 805
- validateSoapData 807
- pub.soap.wsa
 - action 808
 - faultTo 808
 - from 810
 - messageID 811
 - problemAction 811
 - problemHeaderQName 812
 - problemIRI 812
 - relatesTo 813
 - replyTo 814
 - retryAfter 815
 - schema_wsa 816
- pub.soap.wsa.submission
 - action 816
 - faultTo 817
 - from 819
 - messageID 820
 - relatesTo 821
 - replyTo 822
 - retryAfter 824
 - schema_wsa_submission 825
 - to 824
- pub.soap.wsrn
 - closeSequence 825
 - createSequence 826
 - sendAcknowledgementRequest 831
 - waitUntilSequenceCompleted 833
- pub.storage
 - add 843
 - checkpoint restart example 841
 - closeStore 844
 - data store locking 839
 - deleteStore 844
 - entry locking 839
 - get 845
 - keys 846
 - listLocks 847
 - lock 848
 - lock duration 840
 - lock wait time 840
 - put 850
 - registerStore 851
 - releaseLocks 851
 - remove 852
 - short-term store 838
 - unlock 853
- pub.string
 - base64Decode 858
 - base64Encode 858
 - bytesToString 859
 - concat 859
 - HTMLDecode 860
 - HTMLEncode 860
 - indexOf 861
 - length 862
 - lookupDictionary 862
 - lookupTable 863
 - makeString 864
 - messageFormat 864
 - numericFormat 865
 - objectToString 866
 - padLeft 867
 - padRight 868
 - replace 869
 - stringToBytes 869
 - substring 870
 - tokenize 870
 - toLowerCase 871
 - toUpperCase 871
 - trim 872
 - URLDecode 872
 - URLEncode 873
- pub.sync
 - notify 876
 - shutdown 877
 - wait 877
- pub.synchronization.latch
 - closeLatch 880
 - isLatchClosed 881
 - openLatch 882
- pub.synchronization.xref
 - createXReference 883
 - deleteByObjectId 884
 - deleteXReference 884
 - getCanonicalKey 885
 - getNativeId 886
 - insertXReference 887
- pub.trigger
 - createJMSTrigger 891
 - createTrigger 904
 - deleteJMSTrigger 917
 - resumeProcessing 922
 - resumeRetrieval 925
 - suspendProcessing 929

- suspendRetrieval 931
 - pub.universalName
 - find 940
 - list 941
 - listAll 942
 - pub.utils
 - deepClone 946
 - executeOSCommand 947
 - generateUUID 949
 - getServerProperty
 - retrieves value of a server property 949
 - pub.utils.ws
 - setCompatibilityModeFalse 949
 - pub.vcs.admin
 - getUsers 956
 - removeCurrentUser 957
 - removeMultipleUsers 958
 - setCurrentUser 958
 - setMultipleUsers 959
 - pub.xml
 - documentToXMLString 963
 - freeXMLNode 970
 - getNextXMLNode 971
 - getXMLNodeIterator 972
 - getXMLNodeType 975
 - loadEnhancedXMLNode 975
 - loadXMLNode 985
 - queryXMLNode 993
 - xmlNodeToDocument 996
 - xmlStringToEnhancedXMLNode 1006
 - xmlStringToXMLNode 1010
 - pub.xslt.Transformations
 - transformSerialXML 1014
 - publish service 575
 - publish/subscribe, latching and cross-referencing operations 880
 - publishable document types
 - synchronizing 586
 - publishAndWait service 578
 - publishing packages 602
 - addReleaseRegistryEntry 603
 - deleteReleaseRegistryEntry 604
 - distributeViaFTP 605
 - distributeViaSvcPull 606
 - distributeViaSvcPush 607
 - generateReplicationEvent 607
 - getLocalReleasedList 607
 - getRemoteReleasedList 608
 - notifyPackageRelease 609
 - packageCreation 610
 - put 382
 - put (FTP) files 106
 - put (SFTP) files 141
- ## Q
- QName document type 798
 - queryConnectionState 34
 - querying a database 215
 - queryListenerNotificationState 43
 - queryListenerState 37
 - queryPollingNotificationState 44
 - queryXMLNode 993
 - quote 108
- ## R
- random number generator 501
 - randomDouble 501
 - raw offset for time zones 177
 - read 391
 - readAsString 391
 - readerToFile 307
 - readerToString 392
 - records, deleting particular type 884
 - recoverPackage 547
 - recurring tasks, adding to scheduler 626
 - redelivery count, retrieving for documents 574
 - registering
 - consumer handlers 747
 - consumer handlers 728, 741, 750, 753, 757, 758, 760, 764, 765, 766, 767
 - providerhandlers 758
 - unregistering consumer handlers 767
 - unregistering provider handlers 768
 - registering data stores 851
 - registerProcessor 773
 - registerStore 851
 - relatesTo 813
 - release registry. 603
 - reloadEventManagerSettings 281
 - reloadPackage 548
 - remote file, appending data to 92
 - remote procedure call, submitting 170
 - remote servers, invoking services on 592, 593
 - remove 382
 - removeBodyBlock 761
 - removeBodyEntry 798
 - removeCurrentUser 957

- removeHeaderBlock 762
 - removeHeaderEntry 799
 - removeMultipleUsers 958
 - removeTrailer 800
 - removing entries from data stores 852
 - renaming files on an FTP server 108
 - renaming files on an SFTP server 143
 - repeating tasks, updating to scheduler 642
 - replacing strings 869
 - replication specification 282
 - replicationInfo document type 282
 - replicator services 602
 - reply documents
 - for join conditions 584
 - retrieving for asynchronous request 587
 - sending 582
 - reply service 582
 - replyTo 814
 - report services 614
 - repository
 - closing data store 844
 - creating/opening 851
 - deleting data store 844
 - inserting entries 843
 - inserting/updating entries 850
 - list of keys in 846
 - locking entries 847, 848, 851
 - registering 851
 - removing entries 852
 - retrieving keys from 846
 - retrieving values from 845
 - unlocking entries 853
 - request/reply model
 - deliverAndWait service 566
 - delivering a request 564
 - multiple reply documents 581
 - publishAndWait service 581
 - publishing request 578
 - publishing request to a specific client 564
 - retrieving reply 587
 - sending replies 582
 - waitForReply service 587
 - waiting for reply 579
 - requestResponseSpec 801
 - reset 393
 - response string, forcing 360
 - restarting a guaranteed delivery transaction 597
 - restorePipeline 355
 - restorePipelineFromFile 356
 - resumeListener 37
 - resumePollingNotification 45
 - resumeTask 634, 634, 635
 - resuming
 - document processing for triggers 922
 - document retrieval for triggers 925
 - retrieval
 - resuming for triggers 925
 - suspen ding for tri ggers 931
 - retrieving
 - documents from a local file system 298, 380
 - files from FTP server 95
 - files from SFTP server 138
 - MIME content 522
 - MIME message headers 523, 527
 - results of guaranteed delivery transaction 597
 - retrieving reply 587
 - retry count, retrieving 352
 - retry exception 368
 - retry limit
 - setting to zero 903
 - retryAfter 815
 - rm 144
 - rmdir 144
 - rollback 216
 - rollbackTransaction 50
 - roundNumber 502
 - RPC, submitting 170
 - runFileTemplate 614
 - runFileTemplateOnPipe 615
 - runStringTemplate 615
 - runStringTemplateOnPipe 616
 - runTemplate 616
 - runTemplateOnPipe 617
- ## S
- S/MIME messages
 - creating certs-only message 695
 - creating encrypted messages 695
 - creating signed messages 698
 - creating using PKI profiles 550
 - decrypting 556, 700
 - extracting certificates from certs-only message 700
 - verifying signed messages 557, 701
 - S/MIME services 694
 - saveEventManagerSettings 283
 - savePipeline 357

- savePipelineToFile 358
- saveXML AsFFDictionary service 342
- saveXMLASFFSchema service 343
- sc hemas
 - schema_ws a_submission 825
- scheduler
 - adding complex tasks 621
 - adding one-time tasks 624
 - canceling tasks 629
 - recurring tasks 626
 - resuming tasks 635
 - retrieving list of task IDs 629
 - retrieving task info 630
 - suspending tasks 636
 - updating complex tasks 637
 - updating one-time tasks 640
 - updating repeating tasks 642
- scheduler services 620
- schema
 - for events 257
- schema domain 652
- schema services 648
- schemas
 - components 655
 - creating 649
 - datatypes 655
 - for validating 651
 - namespace components 656, 656
 - retrieving tables 211
 - SOAP encoding schema, version 1.2 775
 - SOAP encoding schema 775
 - SOAP envelope schema 775
 - SOAP envelope schema, version 1.2 775
 - wsa 816
 - XML datatypes 655
- sending a guaranteed call 598
- sending STOR or STOU commands to remove server 89, 105, 106
- sendStream 470
- server log
 - pipeline field names and values 369
 - writing to 350
- services
 - default access permissions 24
 - listing of 21
- Session object, inserting in pipeline 353
- sessionEnd specification 287
- sessionEndInfo document type 287
- sessionExpireInfo document type 289
- sessionExpire specification 288
- sessionInfo 109
- sessionStart specification 289
- sessionStartInfo document type 290
- setAdapterServiceNodeConnection 48
- setCurrentUser 958
- setCustomContextID 360
- setKeyAndChain 662, 663
- setListenerNodeConnection 38
- setListenerNotificationNodeListener 46
- setMultipleUsers 959
- setPollingNotificationNodeConnection 46
- setResponse 360
- setResponse2 362
- setResponseCode 364
- setResponseHeader 365
- setResponseHeaders 368
- setTransactionTimeout 50
- SFTP
 - changing file owner 137
 - changing owner group 136
 - changingpermissions 136
 - file transfer 141
 - login 139
 - remove files 144
 - remove directory 144
 - renaming files 143
 - retrieving files 138
 - symbolic link between files 145
 - transferring files 141
 - view remote working directory 143
- sftp working directory, changing 135
- short-term store 838
- shutdown service 877
- signed data object, creating 673
- SignedData objects 551
- signing and encrypting MIME messages 554
- signing MIME messages 555, 698, 701
- size 383
- sizeOfList 483
- skip 393
- SMTP servers, submitting requests to 110
- SMTP, sending MIME messages 145
- SOAP
 - actor attribute 788
 - adding body entries 776
 - adding header entries 777
 - converting SOAP object to Byte Array 803
 - converting SOAP object to String 803

- converting stream to SOAP object 805
- converting string to SOAP object 806
- creating SOAP objects 785
- encoding schema, version 1.2 775
- encoding schema 775
- envelope schema 775
- envelope schema, version 1.2 775
- executing the default processor 772
- executing the default RPC processor 772
- getting body entries 789, 790, 795
- getting encoding entries 791
- getting entire message 790
- getting header entries 792
- getting QNames 794
- listing handlers, consumer 755
- listing handlers, provider 756
- listing processors 770
- mustUnderstand attribute 788, 793
- registering a processor 773
- registering handlers, consumer 728, 741, 747, 750, 753, 757, 758, 760, 764, 765, 766, 767
- registering handlers, provider 758
- removing body entries 798
- removing header entries 799
- removing trailers 800
- sending a SOAP message 167
- sending a SOAP message over HTTP 149
- specification for custom processor 801
- specification for target service 801
- submitting an RPC request 170
- unregistering handlers, provider 768
- unregistering handlers, consumer 767
- unregistering processors 774
- updating Fault Code and Fault String 768
- validating SOAP objects 807
- WS-Addressing wsa
 - faultTo 808
- WS-Addressing wsa
 - from 810
- WS-Addressing wsa
 - action 808
 - message ID 811
 - problemAction 811
 - problemHeaderQName 812
 - problemURI 812
 - relatesTo 813
 - replyTo 814
 - retryAfter 815
 - to 815
- WS-Addressing wsa.submission
 - action 816
 - faultTo 817
 - from 819
 - message ID 820
 - relatesTo 821
 - replyTo 822
 - retryAfter 824
 - to 824
- SOAP messages, inserting trailer 779
- SOAP services 710
- soapClient 149
- soapDataToBytes 803
- soapDataToString 803
- soapHTTP 167
- soapRPC 170
- special characters
 - handling in XML strings 966
- specification
 - audit error event 248
- specifications
 - alarm event handler 245
 - audit event handler 246
 - exception event handler 258
 - gdEnd 261
 - gdStart 262
 - portStatus event handler 280
 - replication event handler 282
 - sessionEnd event handler 287
 - sessionExpire event handler 288
 - sessionStart event handler 289
 - stat event handler 291
 - txEnd event handler 294
 - txStart event handler 295
- SQL statements, executing 203
- starting a guaranteed delivery transaction 598
- startTransaction 51, 217
- stat specification 291
- statInfo document type 292
- status of guaranteed delivery transactions 595
- STOR or STOU commands, sending to remote server
 - using pub.client
 - ftp service 89
 - using pub.client.ftp
 - mput service 105
 - put service 106
- storage services 838, 838
- stored procedures

- invoking 194
 - retrieving information about 207
 - retrieving names of 208
 - streamToBytes 394, 394
 - streamToFile 308
 - streamToReader 395
 - streamToSoapData 805
 - streamToString 395
 - string array, formatting 864
 - stringListToDocumentList 483
 - strings
 - adding to list 482
 - base64 decoding 858
 - base64 encoding 858
 - byte array conversion 859
 - concatenating 859
 - concatenating an array of strings 864
 - converted from documents 230
 - converting list to document list 483
 - converting to byte array 869
 - converting to lower case 871
 - converting to uppercase 871
 - decoding URL-encoded 872
 - HTML to native characters 860
 - length of 862
 - native characters to HTML 860
 - padding beginning 867
 - padding end 868
 - replacing all 869
 - returning substring of 870
 - tokenizing into an array 870
 - transformation services 856
 - trimming white space 872
 - URL-encoding 873
 - XML values to a document 234
 - stringToBytes 869
 - stringToFile 308
 - stringToReader 396
 - stringToSoapData 806
 - stringToStream 396
 - submission action 816
 - submission
 - faultTo 817
 - from 819
 - messageID 820
 - relatesTo 821
 - replyTo 822
 - retryAfter 824
 - to 824
 - submitting an asynchronous call to a remote server 599
 - subscribing to packages 602
 - subscriptions, creating 241
 - substring 870
 - indexing first occurrence 861
 - subtractFloats 502
 - subtractInts 504
 - subtractObjects 504
 - subtype, getting from MIME message 530
 - sum of numeric values 490, 491, 492, 493
 - suspended tasks, resuming 635
 - suspending
 - document processing for triggers 929
 - document retrieval for triggers 931
 - suspendListener 38
 - suspendPollingNotification 47
 - suspendTask 636
 - symlink 145
 - Synchronization folder 879
 - synchronization services
 - coordinating the execution of services 876
 - performing latching and cross-referencing operations 880
 - synchronous call to a remote server 596
 - synchronous request / r eply
 - delivering a re quest 565
 - synchronous request/reply
 - description of 566, 581
 - publishing a request 579
 - syncToBroker 585
 - syncToProvider 586
- T**
- tables
 - inserting rows 213
 - removing rows 201
 - retrieving column information 210
 - retrieving names of 211
 - retrieving rows from 215
 - updating all rows 218
 - tasks
 - adding complex to scheduler 621
 - adding one-time to scheduler 624
 - adding recurring to scheduler 626
 - migrating to external database 634
 - obtaining information about 630
 - obtaining list of 634

- removing from scheduler 629
 - resuming 635
 - retrieving IDs for 629
 - suspending 636
 - updating complex to scheduler 637
 - updating one-time to scheduler 640
 - updating repeating to scheduler 642
 - templates
 - applying to a document 614
 - applying to pipeline 615, 616
 - from string objects 615
 - time zones 177
 - time, converting formats 185
 - to 815
 - tokenizing a string 870
 - toLower 871
 - toNumber 505
 - toUpper 871
 - tracePipeline 369
 - trailers, adding to SOAP messages 779
 - transactional state, clearing 196
 - transactions, ending guaranteed delivery 595
 - transforming XML 1014
 - transient error, description of 369
 - transportInfo 370
 - trigger rs
 - suspending document retrieval 931
 - Trigger folder 889
 - trigger services
 - performance 903
 - retry count 352
 - retry count, retrieving 903
 - retrying 368
 - triggers
 - creating JMS 891
 - deleting JMS 917
 - management services 890
 - resuming document processing 922
 - resuming document retrieval 925
 - retry, setting to 0 903
 - suspending processing 929
 - trimming white space 872
 - txEnd specification 294
 - txEndInfo document type 295
 - txStart specification 295
 - txStartInfo document type 296
- U**
- universal name services 940
 - universal names
 - finding service names for 940
 - listing the registry 941
 - unlocking entries in data stores 853
 - unregisterProcessor 774
 - unsubscribing from an event 252
 - updateComplexTask 637
 - updateFFDictionaryEntryFromXML service 345
 - updateOneTimeTask 640
 - updateRepeatingTask 642
 - updating
 - rows in a data base table 218
 - uppercase, converting to 871
 - URLDecode 872
 - URLEncode 873
 - user management, VCS 956
 - get user names 956
 - remove current user 957
 - remove multiple users 958
 - set current user 958
 - set multiple users 959
 - Utils services
 - retrieves value of server property 946
- V**
- validatePipeline 654
 - validateSoapData 807
 - validating an object 651
 - VCS folder 955
 - VCS Integration feature 956
 - getting users 956
 - remove current user 957
 - remove multiple users 958
 - set current user 958
 - set multiple users 959
 - vectorToArray 484
 - verifying digital signatures 675
 - verifying signed data objects 552
- W**
- waitForReply service 587
 - waiting for delivery from a notifying service 877
 - watt.security.opc.AllowInternalPasswordAccess 670
 - Web service descriptor
 - consumer
 - registering the handler 765

- Web service descriptor
 - consumer
 - registering the handler 757
 - consumer
 - registering the handler 728, 741, 747, 750, 753, 758, 760, 764, 766, 767
 - unregistering the handler 767
 - Pre-8.2 compatibility mode property 949
 - provider
 - registering the handler 758
 - unregistering the handler 768
- white space, trimming 872
- WmSecureString 678
 - converting 679
 - creating 678
 - destroying 680
- working directory, changing 93
- wsm
 - closeSequence 825
 - createSequence 826
 - sendAcknowledgementRequest 831
 - waitUntilSequenceCompleted 833

- X**
- XML
 - datatypes 655
 - pub.schema.w3c 656, 656
- XML documents
 - converting to IData objects 996
 - converting to XML nodes 1006, 1010
- XML nodes
 - creating from XML string 1006, 1010
 - extracting data from 993
 - freeing 970
 - getting type info 975
 - loading from URL 975, 985
 - querying 993
- XML schemas
 - components 655
 - datatypes 655
- XML services 962
- XML strings
 - creating from documents 963
- XML strings, creating from documents
 - handling special characters 966
- XML values
 - converting from a document 230
- xmlNodeToDocument 996
- xmlStringToEnhancedXMLNode 1006
- xmlStringToXMLNode 1010
- XMLValuesToDocument 234
- XQL queries 993
- XSLT folder 1013
- XSLT services
 - maintaining the stylesheet cache 1014
 - transforming an XML stream 1014