TIBCO® MDM

Workflow Reference

Version 9.3.2
May 2025

Document Updated: September 2025

Contents

Workflow Overview

Workflows are a core component of TIBCO MDM. They are used to initiate and manage business processes.

Note: It is a good practice to use the TIBCO MDM Studio Process Designer component which provides a user-friendly graphical interface to design your workflows, and that you manually create/edit workflows only for advanced use.

For more information about using the Process Designer, see the *TIBCO MDM Studio - Process Designer User's Guide*.

For best practices of workflows, see *TIBCO MDM Best Practices*.

Workflow Process Selection

Workflows are initiated by submitting a workflow request to the TIBCO MDM Process Engine.

Workflow requests are XML documents created in response to an external event, that is, receipt of a JMS message or modification of data using the UI. The process engine selects a workflow based on data present in the workflow request. The main decision factors are (but not limited to):

- **Sender enterprise** – The enterprise that created the workflow request.
- **Receiver enterprise** –The enterprise that must process the workflow. For most of the workflow requests, the sender and receiver enterprises are the same. However, enterprises can be different in a multi-enterprise configuration or for requests received from other applications.
- **Partner organization** –The organization within the receiver enterprise for which this workflow is targeted.
- **Document type and subtype**–Key request identifier, indicates what type of request it is.

Workflow selection is done using business process rules. This provides greater flexibility; changes are hot deployed (you don't need to restart the application server) and workflow selection can be exported and imported across instances. Workflow selection configuration is done through the UI.

For details about using business process rules for workflow selection, see Workflow Process Selection Using Business Process Rule Template.

Multiple Workflow Selection

In case of multiple workflow selection, two processes are selected and these processes are executed consecutively.

All out-of-box workflows that are used in multiple workflow selection scenarios have been modeled into sync-subflows. With these changes, multiple workflow selection behavior has been modeled as follows:

- The selection of the second process (P2) is initiated by the first process (P1) as a synchronous subflow.
- Once the second process (P2) completes, it returns control to the invoking process (P1) which sets the event status to Success (or error) and completes its execution.
- A SubFlowFlag parameter is passed from the first process to the second process, and this parameter ensures that the second process (P2) does not update any event status.

The following workflows (multiple process selection) are applicable for GDSN only (if you have the GDSN plug-in installed).

GDSN Out-of-the-box Workflows Modified for Multiple Process Selection

Feature	Workflow/Process 1	Workflow/Process 2
1Sync	wfout1sync62cic26v1	wfin26prodnotifbasicv3
CIM2CIM	wfoutveloselcin26v2	wfin26prodnotifretailerv4 wfin26prodnotifprocessv2
WWRE	wfoutagentrics50cin26v2	wfin26prodnotifretailerv4 wfin26prodnotifprocessv2 wfin26prodnotifv2 wfin26prodnotifbasicv3 wfin26rfcinv2

Note: The table denotes those out-of-box workflows that have been modified for multiple process selection - this is usually in case of 1Sync, CIM2CIM, and WWRE.

For example, for CIM2CIM, the first process wfoutveloselcin26v2 is a workflow to translate the incoming CIM2CIM message to mlXML format. The second process called is either wfin26prodnotifretailerv4 or wfin26prodnotifprocessv2, both of which are workflows to process incoming record messages and save data - except that wfin26prodnotifretailerv4 is a variation targeted towards integration hubs.

Out-of-the-box Workflows

The out-of-the-box workflows are stored in *$MQ_COMMON_DIR*/standard/workflow. These workflows are updated and maintained by TIBCO. Ensure that you do not modify the standard workflows.

If you must change the supplied workflows, copy them to another directory before you modify them. The out-of-the-box workflows are sample implementation of common business processes for master data management.

For a complete list of out-of-the-box workflows, see *Installation and Configuration*.

Changes to Out-of-the-Box Workflows

The following out-of-the-box workflows are modified:

- Synchronization (wfin26catsynchv7.xml): The workflow file is modified to skip IdentifyProtocolOperations activity, to improve the performance, and to avoid the generation of product logs, which are rarely used during local publish. The following new transition is added from the ConverRecordsToOutput to the ApplySynchRulebase activity:

```
<Transition FromActivity="ConvertRecordsToOutputFormat" ToActivity="ApplySynchRuleBase">
```

```
<Description>Process or skip Identify Protocol Operations</Description>
<Rule>
<Parameter name="SkipIdentifyProtocolOps" type="string" eval="xpath" direction="in"
source="//BuyerCatalogHeader/Extension[@name='SkipIdentifyProtocolOps']/Value/text()">inDoc</Parameter>
<Parameter name="result" type="boolean" direction="out"/>
<Condition format="java"><![CDATA[
com.tibco.mdm.workflow.engine.transition.WfSharedConditionTransition.isYes
(SkipIdentifyProtocolOps);]]></Condition>
</Rule>
</Transition>
<Transition FromActivity="ConvertRecordsToOutputFormat" ToActivity="IdentifyProtocolOperations"/>
```

- The SkipIdentifyProtocolOps parameter is controlled by the **SkipIdentifyProtocolOps** check box available on the Create Synchronization Profile page. For more information about using this option, see *User's Guide*.

  Note: The SkipIdentifyProtocolOps parameter or the check box option is available in only the MDM software edition and not in the GDSN Plug-in. Therefore, even though the transition is available in the standard out-of-box workflows, the synchronization workflow operation in the GDSN software edition is not changed.

- Factsheet Generation (wfin26productfactsheetpdfv2.xml): The Factsheet Generation workflow is modified so that its output (a PDF document) is explicitly passed to the OutDocument. See also, Example: Defining and Using Out Parameters with Workflows

Workflow Components

The workflow is defined as XML. The XML must conform to Workflow.xsd schema available in the *$MQ_HOME*/schema/XW/1.3 directory.

The workflow definition provides the following structure:

> **Note:** It is a good practice to use the TIBCO MDM Studio Process Designer component to define processes instead of manually creating or editing.

- **Workflow header:** See Workflow Header.

  Tags global to the entire workflow are defined in this section. This includes filename, input, and output parameters, and a textual description of the workflow.

- **Workflow inputs:** See Workflow Inputs and Outputs.

  A workflow can have any number of input and output parameters. The attribute direction indicates whether this is an 'in'-going or 'out'-going parameter. Ingoing parameters are the ones that were sent to the workflow by the outside environment, and outgoing parameters are the ones the workflow returns.

- **Activities:** Activities define what is to be done. Activities can be listed in no particular order. The first activity to execute is identified by the <Start/> tag, which can be specified for only one activity in the workflow. Activities are composed of a set of global tags, and a set of activity specific tags.

  For a complete description of all activities and the legal tags, see Workflow Activities.

- **Transitions:** Transitions tie activities together.

  Transitions must always be defined after all activities are defined. For each activity, there can be more than one transition to other activities. Transitions are evaluated in the order in which they are specified and the first transition which evaluates to true is executed.

Sample Workflow File

The general form of a workflow XML file is as follows:

```
<?xml version="1.0"?>
<!--edited with XML Spy v3.5 NT (http://www.xmlspy.com) (Tibco) -->
<!--Cancel Workflow.
$Revision: 70941 $
-->
<Workflow Version = "1.3" xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation = "Workflow.xsd">
  <Owner>TIBCO</Owner>
  <Name>cancelworkflowv1</Name>
  <Description>Sub process for event cancellation</Description>
  <Parameter direction = "in" name = "inDoc" type = "document" eval = "variable">1</Parameter>
  <Parameter direction = "out" name = "outDoc1" type = "document" eval = "variable"/>
  <Activity Name = "CloseAllWorkitems">
    <Start/>
    <Action>ManageWorkItem</Action>
    <Description lang = "en">Close all the open work items and set the status to
Cancelled</Description>
    <Execution>SYNCHR</Execution>
    <Parameter direction = "in" eval = "constant" type = "string" name =
"WorkItemAction">CLOSE</Parameter>
    <Parameter direction = "in" eval = "constant" type = "string" name =
"WorkItemStatus">CANCELLED</Parameter>
    <Parameter direction = "in" name = "InDocument" type = "document" eval =
"variable">inDoc</Parameter>
  </Activity>
  <!--End of Add approval loop -->
  <Activity Name = "SetStatusToCancelled">
    <Action>UpdateEvent</Action>
    <Description lang = "en">Set the event state to Cancelled/Done.</Description>
    <Parameter direction = "in" name = "eventStatus" type = "string" eval =
"constant">CANCELLED</Parameter>
    <Parameter direction = "in" type = "string" eval = "constant" name =
"eventState">DONE</Parameter>
  </Activity>
  <Activity Name = "SetStatusToError">
    <Action>UpdateEvent</Action>
    <Description lang = "en">Set the event state to Error/Done</Description>
    <Parameter direction = "in" name = "eventStatus" type = "string" eval =
"constant">ERROR</Parameter>
    <Parameter direction = "in" type = "string" eval = "constant" name =
"eventState">DONE</Parameter>
  </Activity>
  <Activity Name = "UpdateRecordStateAsRejected">
    <Action>UpdateRecordState</Action>
    <Description lang = "en">Set the record status as rejected</Description>
    <Parameter direction = "in" name = "Status" type = "string" eval =
```

```
"constant">REJECTED</Parameter>
   <Parameter direction = "in" name = "InDocument" type = "document" eval =
"variable">inDoc</Parameter>
  </Activity>
  <Activity Name = "RejectDueToWorkflowError">
   <Action>Send</Action>
   <Description lang = "en">Error occurred during workflow     processing</Description>
   <Parameter direction = "in" eval = "rule" source = "Message or Workflow Completion" type =
"string" name = "Protocol">inDoc</Parameter>
   <Parameter direction = "in" eval = "rule" source = "Message or Workflow Completion" type =
"long" name = "Address">inDoc</Parameter>
   <Parameter direction = "in" name = "Sender" type = "string" eval =
"constant">support@tibco.com</Parameter>
   <Parameter direction = "in" name = "Presentation" type = "string" eval = "constant">

com.tibco.mdm.ui.workflow.engine.emailtemplates.EmailProductE ditWorkflowError</Parameter>
   <Parameter direction = "in" eval = "constant" type = "string" name =
"Form">standard/forms/fm26ca.xml</Parameter>
   <Parameter direction = "in" type = "document" eval = "variable" name =
"InDocument">inDoc</Parameter>
  </Activity>
  <!--Determining whether this is modify or delete-->
  <Transition FromActivity = "CloseAllWorkitems" ToActivity = "UpdateRecordStateAsRejected">
   <Description>If type is CatEdit only, then set the record status as rejected</Description>
   <Rule>
    <Parameter direction = "in" name = "doctype" type = "string" eval = "xpath" source =
"/Message/Body/Document/@type">inDoc</Parameter>
    <Parameter direction = "in" name = "inputParamString" eval = "constant" type =
"string">CatEdit</Parameter>
    <Parameter name = "result" type = "boolean" direction = "out"/>
    <Condition format = "java">
     <![CDATA[
com.tibco.mdm.workflow.engine.transition.WfSharedConditionTransition.equalsIgnoreCase
(doctype, inputParamString);
    </Condition>
   </Rule>
  </Transition>
  <Transition FromActivity = "CloseAllWorkitems" ToActivity = "SetStatusToCancelled"/>
  <Transition FromActivity = "UpdateRecordStateAsRejected" ToActivity =
"SetStatusToCancelled"/>
  <Transition type = "error" FromActivity = "Any" ToActivity = "RejectDueToWorkflowError"/>
  <Transition FromActivity = "RejectDueToWorkflowError" ToActivity = "SetStatusToError"/>
</Workflow>
```

Workflow Header

The following tags defined in the workflow header provide the Meta information of a workflow and its interaction with the outside environment, which is either input or output.

Workflow Global Tags

Tag	Semantics	Signature	Required?	Valid Values
Description	A textual description of the business process.	<Description> #PCDATA </Description>	N	Any arbitrary string.
Execution	Reserved. Always specify SYNCHR.	<Execution> #PCDATA </Execution>	N	SYNCHR
Name	The name of the workflow file. See Recommended Naming Conventions (Name tag) for the recommended naming conventions. This name is used for debugging purposes only.	<Name> #PCDATA </Name>	Y	Must match the name of the workflow file without the .XML extension.
Owner	Specify the author who defined this process definition. This is a documentation field. For example, all	<Owner> #PCDATA </Owner>	Y	Description of the owner, usually an enterprise. This field is for information only, you can use it to define

Tag	Semantics	Signature	Required?	Valid Values
	samples provided by TIBCO are owned by TIBCO.			a logical owner of the workflows.
Context	Context identifies the "run as" user when a business process is executed. The workflow is deemed to be executed by the user identified in the context.	<Context> #PCDATA </Context>	N	Sender Receiver (default) When 'Sender' is specified, the business process is executed for the sender specified in the workflow request message. When 'Receiver' is specified, the business process is executed for the receiver specified in the workflow request message.

Recommended Naming Conventions (Name tag)

Follow the recommended naming convention.

The following is the recommended naming convention:

- Business Process definitions used to process workflow requests generated by the

application: wfin*mlxmlVersionNumbershort name for the workflow actionv workflowfile version number*.xml. For example, wfin26productaddapproavalv2.xml is the workflow for processing record add. It is based on mlXML version 2.6 and this file has version 2.

- Business Processes which are used to process workflow requests/messages received from other systems: wfout *name of the sender, if anyversion number of protocol used by sendershort name for the workflow actionmlxmlVersion numberworkflowfile version number*.xml. For example wfoutsap30cin26.xml is used to process messages received by the application from SAP. The protocol version used for SAP is 3.0 and the mlXML version is 2.6. w

Workflow Definition Modifications

When a change is required to an existing business process definition, you can use either one of these of the methods.

If the workflow definition file is replaced without changing the name of the file, this indicates that the process definition can be applied to workflows which have already started. As the process definition is cached in-memory, the changed definition would apply if:

1. The application is restarted, thus reinitializing the workflow definition.
2. The application's local cache is flushed using the Cache Manager utility.
3. The property com.tibco.cim.optimize.configfilecheck is set to true.

   Note: Workflow definition changes apply only to new workflows started or previously suspended workflows that are restarted after the process definition file is replaced. It does not affect in progress workflows.

   If the changed workflow definition file is saved with a new name, the new definition is not used for any workflows which have already started before the change. Older workflows continue to use the previous definition even if they are suspended and the workflow is restarted or the server is restarted. The new definition is used only when:

4. The 'Process Definition Selection' rule is changed to map process selection to a new name. To populate the new workflow file name in the 'Process Definition Selection' rule, see the section Customizing Workflow Selection.

5. The application cache is flushed to effect the changes or the server is restarted.

This method must be used when you want to change any currently executing workflows, typically to correct defects.

Workflow Inputs and Outputs

A business process definition can have any number of input and output parameters.

The attribute direction indicates whether it is an ‘incoming’ or ’outgoing’ parameter. Incoming parameters are the ones sent to the workflow by the outside environment, and outgoing parameters are the ones the business process returns.

The following parameter is a special parameter which is mandatory and must always be present exactly as specified. This parameter is used to pass input workflow request document.

```
<Parameter direction="in" type="document" name="inboundname">1</Parameter>
```

The following parameter is also a special parameter which is optional and is used to pass any record list.

```
<Parameter direction="in" type="recordlist" name="inRecordList">2</Parameter>
```

These two parameters can be followed by any number of input parameters. These parameters must be specified only if the workflow is called from another workflow (as a subflow).

The parameter type indicates how the workflow must regard the value specified, in this case the number 1. Incoming documents are numbered in the order passed in so that the first gets the number 1, the second 2, and so forth. The name attribute defines the internal name that the parameter are referred to in the rest of the workflow. In the provided example, inRecordList will now be used to refer to the first incoming record list.

You can specify any number of input/output parameters. The parameter specification must be defined similar to any activity parameter.

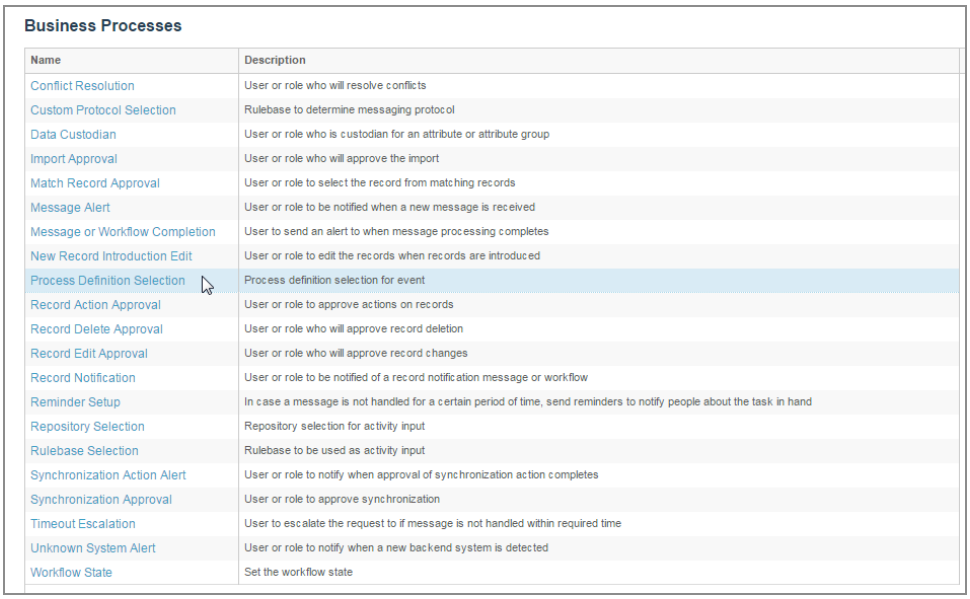
For example:

```
<Parameter direction="in" type="string" eval="variable" name="inStatus">
testInStatus</Parameter>
<Parameter direction="out" eval="variable" type="string" name="status">finalStatus</Parameter>
```

For more information about how to define these parameters, see Parameter Definition.

Workflow Process Selection Using Business Process Rule Template

In **Business Processes**, use the **Process Definition Selection** rule template to define rules for your workflows.



Conditions in Rule Template

The "Process Name" workflow name is selected based on conditions.

The following are standard conditions in the default rule template:

Conditions in Rule Template

Condition Name	Description
Document Type	Key request identifier, indicates the type of request.
Document Sub Type	Key request identifier, indicates the type of request.
Repository Name	The name of the Repository.
Synchronization Profile	The synchronization profile.
Sender Organization	The enterprise that created the workflow request.
Receiver Organization	The enterprise that must process the workflow.
Note: For most of the workflow requests, the sender and receiver enterprises are the same. However, enterprises can be different in a multi-enterprise configuration or for requests received from other applications.	
Receiver organization type	The organization within the receiver enterprise for which this workflow is targeted.
Software Edition	The edition of the software (MDM or GDSN).
Document format	The format of the document (such as mlXML).
Resubmit (Optional)	used to resubmit events.
Dequeued (Optional)	used to de-queue events.
Redeliver (Optional)	used for workflow failover

Workflows - TIBCO MDM

The following figure shows the various TIBCO MDM processes along with supported Document Type, Document Sub Type, Repository, and so on.

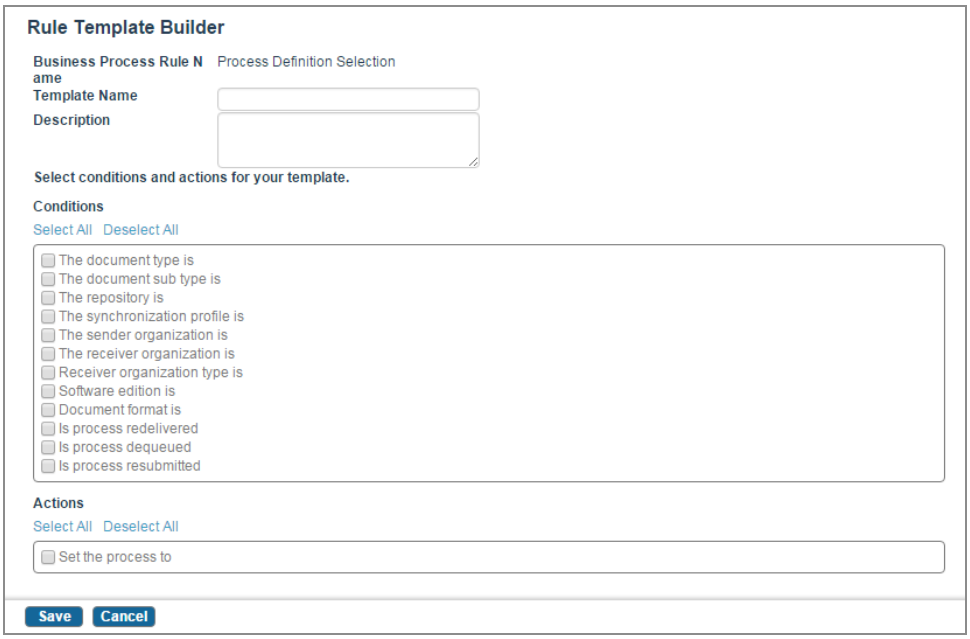
For example, the wfin26catmultipartysynchv2.xml - Multiparty sync process workflow must have the following:

- Document Sub Type as Multi Party Publish.
- Document Type, Repository, Synchronization Profile, Default Sender, Default Receiver, Organization Type, Software Edition, Document Format as Any.

Document Type	Document Sub Type	Repository	Synchronization Profile	Default Sender	Default Receiver	Organization Type	Software Edition	Document Format	Process Definition Name	Default
Purge Historical Data	Any	Any	Any	Any	Any	Any	Any	Any	wfin26purgev3.xml - Purge process	No
Synchronization	Multi Party Publish	Any	Any	Any	Any	Any	Any	Any	wfin26catmultipartysynchv2.xml - Multiparty sync process	No
Synchronization	Catalog Data dump	Any	Any	Any	Any	Any	Any	Any	wfin26catsynchdbdumpv3.xml - DB dump process	No
Synchronization	Any	Any	Any	Any	Any	Any	Any	Any	wfin26catsynchv7.xml - Synchronization process	No
Backend Record Edit Response Notification	Any	Any	Any	Any	Any	Any	Any	Any	wfin26BackEndIntegrationV1_Sample2.xml - Back end integration sample2 process	No
Record Edit	Record Add	Any	Any	Any	Any	Any	Any	Any	wfin26productaddapprovalv3.xml - New Record Introduction Process	No
Record Edit	Backend Record Add Notification	Any	Any	Any	Any	Any	Any	Any	wfin26BackEndIntegrationV1_Sample1.xml - Back end integration sample1 process	No
Record Edit	Record Review	Any	Any	Any	Any	Any	Any	Any	wfin26catactionv2.xml - Record Synchronization and Approval for retailers process	No
Record Edit	Record Reject	Any	Any	Any	Any	Any	Any	Any	wfin26catactionv2.xml - Record Synchronization and Approval for retailers process	No
Record Edit	Record Accept	Any	Any	Any	Any	Any	Any	Any	wfin26catactionv2.xml - Record Synchronization and Approval for retailers process	No
Record Edit	Record Synchronized	Any	Any	Any	Any	Any	Any	Any	wfin26catactionv2.xml - Record Synchronization and Approval for retailers process	No
Record Edit	Import Sub-Workflow	Any	Any	Any	Any	Any	Any	Any	wfin26catsourceimportv2.xml - Import sub process	No
Record Edit	Generate Factsheet	Any	Any	Any	Any	Any	Any	Any	wfin26productfactsheetpdfv2.xml - Fact sheet process	No
Record Edit	Record Restore	Any	Any	Any	Any	Any	Any	Any	wfin26productediapprovalv3.xml - Record Change/Edit process	No
Record Edit	Any	Any	Any	Any	Any	Any	Any	Any	wfin26productediapprovalv3.xml - Record Change/Edit process	No
Mass Update	Advanced Mass Update	Any	Any	Any	Any	Any	Any	Any	wfin26catmassupdatev1.xml - Mass Update process	No
Mass Update	Basic Mass Update	Any	Any	Any	Any	Any	Any	Any	wfin26catmassupdatev1.xml - Mass Update process	No
Upload/Import	Import Classification	Any	Any	Any	Any	Any	Any	Any	wfin24classimpv2.xml - Classification Import process	No
Upload/Import	Load And Extract Classification	Any	Any	Any	Any	Any	Any	Any	wfin24classimpv2.xml - Classification Import process	No
Upload/Import	Refresh And Import Classification	Any	Any	Any	Any	Any	Any	Any	wfin24classimpv2.xml - Classification Import process	No
Upload/Import	Import Records	Any	Any	Any	Any	Any	Any	Any	wfin26catsourcev5.xml - Import process	No
Meta Data Request	Any	Any	Any	Any	Any	Any	Any	Any	wfin26dataservicev2.xml - Meta Data Request Process	No
Upload/Import	Load	Any	Any	Any	Any	Any	Any	Any	wfin26catsourcev5.xml - Import process	No
Upload/Import	Load Datasource and Import Records	Any	Any	Any	Any	Any	Any	Any	wfin26catsourcev5.xml - Process for data source upload and import	No

Rules Creation

You can create rules using the rule template. Provide all the details and then set the process to the required workflow.



Migration to Business Process Rules for Workflow Selection

The old workflow process selection method must be migrated to the new Business process Rule method.

Any existing workflow manager configuration must be also migrated to use business rules.

Migrating Older Enterprises

For existing enterprises, you must run rule migration so that a new process definition rule is created in the corresponding enterprise.

You are also required to select the organization for which process definition selection must be created.

Follow these steps to migrate older (prior to 8.0) enterprises:

Procedure

1. Run the appropriate migration script from the following location:

   Oracle: *$MQ_HOME*/db/oracle/migration/Migrate72_80/processSelection.sql

   DB2:*$MQ_HOME*/db/db2/migration/Migrate72_80/processSelection.db2
2. Copy velosel\rules into your *$MQ_COMMON-DIR*.
3. Make an entry of all the organizations that must be migrated (except super user) in orglist-processSelection.txt.

```
$ ./MigrateRules.sh  -addNewProcessSelectionRules
$ ./MigrateRules.sh  -addDefaultTemlate
```

Migrating Custom Processes

Procedure

1. Make an entry of all organizations that must be migrated (except super user) in orglist-processSelection.txt.
2. Add the doctype and docsubtype defined which are not out-of-box. Ensure they are available in the DOMAINENTRY table.
3. Run the following: $ ./MigrateRules.sh –migrateCustomProcesses

Customizing Workflow Selection

Procedure

1. Add an entry of the workflow in the CONFIGURATIONDEFINITION table. Set the organization ID of the organization for which customization is to be done. For example:INSERT INTO CONFIGURATIONDEFINITION(ID, "TYPE", OWNERID, GLOBAL, "NAME", SELECTOR, DESCRIPTION, DEFINITIONTYPE, DEFINITION, ACTIVE, MODMEMBERID, MODDATE, MODVERSION)VALUES ('4', 'PROCESSNAME', '1', 'Y', 'wfin26productaddapprovalv3.xml', 'WORKFLOW', 'wfin26productaddapprovalv3.xml', 'File', 'standard/workflowwfin26productaddapprovalv3.xml', 'Y', '1', TO_DATE('15-03-2006 04:01:00 pm','DD-MM-YYYY HH:MI:SS AM'), '1');

> **Note:** If your workflows are available in a folder other than *$MQ_COMMON_DIR*/standard, ensure that you have provided the appropriate path in the query.

2. If the DOCTYPE or DOCSUBTYPE is different or not provided out-of-the-box, add an entry in the DOMAINENTRY table with domaintype as DOCTYPE/DOCSUBTYPE. For example:INSERT INTO DOMAINENTRY (DOMAINTYPE, VALUE, DESCRIPTION) VALUES ('DOCSUBTYPE', 'Loader', 'Loader'); INSERT INTO DOMAINENTRY (DOMAINTYPE, VALUE, DESCRIPTION) VALUES ('DOCTYPE', 'DBLoader', 'DBLoader');
3. Log in to the organization.
4. Select the process definition selection. If the out-of-box condition is not per the requirement, add a new condition provided in the templates.
5. Create a rule through the rule creation menu.

Result

> **Note:** Rules are executed in order of definition. So, a default rule could end up overriding an explicit one, if it is placed first.
>
> For example, if a default rule with doctype=Record Edit and docsubtype=any is specified before an explicit rule with doctype=Record Edit and docsubtype=Record, the default rule gets executed.

Workflow File Refresh

During implementation, process definition might change and it is sometimes desired that changes be reloaded automatically.

This can be set up using the **Configurator** > **Miscellaneous** > **Quick Configuration Update** option to **true**.

- **true**: Check whether the file has changed every time it is used. This picks up updated files immediately.
- **false**: Do NOT check if the file has changed. This optimizes performance but new files are not picked up until the system is restarted or cache is cleared. Use once

the system has stabilized, for example, for a production installation.

Note: Modifications apply to new workflows or any running workflows if suspended and restarted.

Parameter Definition

Parameters are arguments passed into activities/workflows and outputs returned from activities/workflows.

A parameter definition provides the following components:

Parameter Components

Attribute	Description
direction	The direction of the parameter. • **Valid Value:** in out • **Signature:** direction='#CDATA'
type	The type of the parameter. • **Valid Value:** string long boolean arraylist document recordlist file date timestamp • **Signature:** type='#CDATA'
name	The internal name of the parameter. • **Valid Value:** Any string value. • **Signature:** name='#CDATA'
eval (Optional)	How to evaluate the current parameter before binding it. If eval is not specified, constant is assumed. • **Valid Value:** constant variable rule xpath property lookup catalog event userprofile system compute • **Signature:** eval='#CDATA'
source	Specifies source of parameter. Required when eval is rule, xpath, lookup, catalog. Following the table-eval-rule, the source must point to a valid business process rule name.

Attribute	Description
	• If eval = catalog, the source is the property of the catalog. Also, the catalog is automatically selected using the indocument. The indocument must resolve following xpaths: /Message/Body/Document/ BusinessDocument,CatalogAction, CatalogActionHeader/ CatalogReference[@type='Catalog']/ RevisionID/DBID /Message/Body/Document/ BusinessDocument/CatalogAction/ CatalogActionHeader/CatalogReference[@type='Catalog'] /RevisionID/Version • If eval = xpath, source must be the xpath to be evaluated. • If eval = lookup, source must be a valid domain look up of the format 'domainType,domainValue'. It picks up the first non-null value from domainValue table in this order - EMAILID,PHONENUMBERID, FTPID,HTTPID,ADDRESSID, MEMBERID,ORGANIZATIONID,VALUE • **Valid Values:** SourceOrganizationID VersionOption IsXMLFormat,FileGenerationOptionMasterCatalogID,Master-CatalogVersion, DateTimeCatalogID, CatalogName, CatalogDescription, OutputMapID, ChannelID, ClassificationSchemeID, SubsetRuleID,TransformRuleBase TransferModeDestinationAddress CatalogFileNameZipOutput, IncrementalLastPublishedDate, LastPublishedStatusProductCount, ValidFromDateValidToDate, SupplierDomainDefaultLanguageUNUOM, CharSetDeliver CatalogDefaultCurrencyItemAdd, ItemDelete ItemPublish, InitialLoadWithdrawPublication DelistPublicationItemAccept, ItemReview, ItemReject, ItemSynchronize, ItemCorrectionModMemberID, TradingPartnerGLN, SenderGLNID SenderGLNTradingPartner

Attribute	Description
	GLNIDTradingPartnerDomain ModVersion, ModDateActive, Action, DeliverToSelectionType, NamedVersionIDCascade, XSLTFileName • **Signature:** source='#CDATA'

Note: CDATA appears in attribute declarations and specifies that an attribute contains character data that is not parsed. '&', '<' and the quote characters are used for delimiting the attribute value and have a special meaning in attributes of this type.

Parameter Direction

Either the value is passed in to the activity, or it is generated as output. If the same name is used for input/output, the output value overwrites the input value.

This technique is fairly common, because it designates one variable to hold the "current value" so regardless of which path was taken through the workflow, you are always assured that value holds the current value.

Parameter Type

The type specifies what the expected type of the variable is. The native parameter types are: "string", "long", "boolean", and "arraylist".

There are also two object types which correspond to specific implementations of objects in the system: "document" and "recordlist".

- **string**: String variables can be used wherever strings are expected, such as file names and keyword parameters. Valid values are alphanumeric characters of any length.
- **long**: This variable can hold numeric values (including decimal numbers).
- **boolean**: This variable can take the value true or false (case insensitive).
- **arraylist**: An ArrayList can hold more than one object. For example, it can store a

set of strings and so on.

- **recordlist**: An object defined by TIBCO MDM to represent a list of records. The record list can be created explicitly. (using the ManageRecordCollection activity) or implicitly by many activities. This object is defined by:
    - VersionOption: Lets you choose records based on state.
    - Base object.
    - Repository: The record data is read from the repository.
    - Catalog: The record data is read from the catalog, in the output format selected in the catalog.
    - Relationship Names: Record relationships that are included while creating a record bundle.
    - VersionDate: The data (records and relationships) are selected based on the date specified.

      Record List can be writable or read only. A read-only record list is explicitly created by specifying a “setReadOnly” flag in the ManageRecordCollection activity. Such record lists cannot be modified. So, they must be set as input to the activities which modify the record list.

      The record List contains bundles of records. That is, for every record included in the record list, all activities would process the record bundle created as the included record as “root”, and all records related to this record based on specified relationships.
- **document**: The Document object is an encapsulation of the mlXML document and can used as a DOM. The usual method to access any data in the document is XPATH.
    - Name: This corresponds to the internal name of the parameter. This cannot be changed by the user.
    - Eval: The evaluation attribute specifies where you get an argument from. The location/source of the argument value is completely divorced from the usage of the value. Therefore, the parameter value can come from anywhere, as long as it ends up with a valid value.
- **file**: Defines parameter of type File.
- **date**: Defines parameter of type Date.
- **timestamp**: Defines parameter of type Timestamp.

Parameter Evaluation

The evaluation attribute specifies where you get an argument from. The location or source of the argument value is completely divorced from the usage of the value.

Therefore, the parameter value can come from anywhere, as long as it ends up with a valid value:

- **constant**

  The constant value is typed directly into the workflow definition.

  ```
  <Parameter direction=”in|out” type=”any” eval=”constant”
  name=”InternalName”>VALUE</Parameter>
  ```

- **variable**

  The workflow maintains a “workflow context” which stores all evaluated variables. Any of these variables can be accessed by specifying the variable name.

  ```
  <Parameter direction=”in” type=”any” eval=”variable”
  name=”InternalName”>VariableName</Parameter>
  ```

- **rule**

  The rule parameter executes a Business Process Rule, whose name is specified in the source attribute. The document that is passed in is used as the source document for any xpath evaluations specified in the Business Process Rule.

  ```
  <Parameter direction=”in” type=”any” eval=”rule” source=”Business Process Rule”
  name=”InternalName”>DocumentVariable</Parameter>
  ```

- **xpath**

  The xpath evaluation runs an xpath into the document specified in the tag. The source is the xpath to execute.

  ```
  <Parameter direction=”in” type=”any” eval=”xpath” source=”/mlXml/blah”
  name=”InternalName”>DocumentVariable</Parameter>
  ```

- **property**

  The property evaluation executes a get method on the specified object.

```
<Parameter direction=”in” type=”any” eval=”property”
name=”InternalName”>ObjectVariable.Method</Parameter>
```

```
<Parameter direction="in" eval="property" type="string"
name="File">ItemAddModifyDoc.FileName</Parameter>
```

The second example executes the method getFileName on the object stored in the variable ItemAddModifyDoc. Note that the “get” is suffixed to the method specified in the parameter. This evaluation method should be used with care, since method names are not always guaranteed to remain constant between application versions.

The specification follows Java Bean standards. The format is objectName.propertyName. The object name should point to a valid object name available in the workflow context, that is, workdoc could point to a valid document. The object should support a method get<propertyName>.

- **lookup**

  Two forms of lookup exist:

```
<Parameter direction=”in” type=”any” eval=”lookup” name=”InternalName” source=”
{domaintype,domainvalue}>DocumentVariable</Parameter>
<Parameter direction=”in” type=”any” eval=”lookup” name=”InternalName” source=”
{{ownerxpath1, ownerxpath2}, domaintype,domainvalue}>DocumentVariable</Parameter>
```

  This looks up the value in the DOMAINVALUE table for a given ownerID, domaintype, domainvalue. If no ownerxpath is specified, the ID of the context organization is used.

  For example, the following looks up the default Trading Partner for the current organization.

```
<Parameter direction="in" name="DefaultBuyer" type="long" eval="lookup" source="
{COMPANYBUYERORG,DEFAULTORDERBUYER}">inDoc</Parameter>
```

  This example looks up a value in the Marketplace (i.e. channel) organization specified in the document header.

```
<Parameter direction="in" name="CatalogOutputMap" type="string" eval="lookup" source="
```

```
{{//MessageHeader[@role='Channel']/Organization/PartyID/DBID},
SYNCHRONIZATION,DEFAULTOUTPUTMAP}">inDoc</Parameter>
```

- **catalog**

  When doing a publish using a “Catalog” definition, you can access the values in that Catalog definition.

  The following list of attributes is accessible from the catalog.

Attribute Name	Description
Active	is it Active (Y,N).
AddCancel	Cancel request to synchronize with integration hub.
AddCorrection	Add Correction request to synchronize with integration hub.
AddDIsContinue	Discontinue request to synchronize with integration hub.
AddReinstate	Add request to synchronize with integration hub which is either discontinued or canceled.
BuyerDomain	Organization of the trading partner.
BuyerGln	GLN used by the trading partner.
BuyerID	Organization ID of the Trading partner.
CatalogFileName	Name.
CharSet	Character set selected in format specific attributes.
ClassificationSchemeID	Selected Classification scheme ID.

Attribute Name	Description
DefaultCurrency	Currency selected in format specific attributes.
DefaultLanguage	Language selected in format specific attributes.
DeliverCatalog	Is catalog delivery requested using specific channel attributes? (Y, N).
DeliverTo	Value selected in DeliverTo.
DestinationAddress	Address selected.
EditionDescription	Synchronization profile description.
EditionID	Selected Synchronization profile ID.
EditionName	Selected Synchronization profile name.
FileGenerationOption	Is File generation requested (Y, N).
IncludeRejectedRecords	Can the Rejected records be Synchronized (Y,N).
Incremental	Is incremental Synchronization requested? (Y, N).
InitialLoad	Is Reload (Initial Load) requested? (Y, N).
IsXMLFormat	Is the output to be generated as an XML file? (True or false)
ItemAccept	Is Accept Requested? (Y, N).
ItemAdd	Is Add Requested? (Y, N).
ItemCorrection	Is Item Correction Requested? (Y, N).

Attribute Name	Description
ItemDelete	Is Delete Requested (Y, N).
ItemPublish	Is Publish Requested (Y, N).
ItemReject	Is Reject Requested (Y, N).
ItemReview	Is Review Requested (Y, N).
ItemSynchronize	Is Synchronize Requested (Y, N).
MarketPlaceID	Organization ID of the integration hub.
MasterCatalogID	Repository ID.
MasterCatalogVersionDateTime	Version date to pick the record data.
ModDate	Latest Modification Date.
ModMemberID	Member ID who has last modified the record.
ModVersion	Version of the record.
OutputMapID	Output Map ID.
PublishCorrection	Correction request to the trading partner.
PublishNew	ADD request to a tradingPartner.
RFCINFlag	Request to Supplier for a CIN.
SelectionType	Selection Type: Trading Partner, Trading Partner on a channel or Channel.
SourceOrganizationID	Organization ID of the sender.
SubsetRuleID	Subset Rule selected.

Attribute Name	Description
SupplierDomain	Credential domain for sender.
SupplierGLN	Credential value of the sender.
SupplierGLNID	Credential ID (internal ID assigned) of the sender.
TransferMode	Transfer mode: FTP, EMAIL or NONE.
TransformRulebase	Rulebase file name.
ValidFromDate	Valid from.
ValidToDate	Valid to.
VersionOption	Version selection: CONFIRMED, LATEST, ALL.
XSLTFileName	XSLT file uploaded for transform of XML.

To access one of the parameters, set eval to “catalog” and the source to the Parameter name. For example:

```
<Parameter direction="in" name="VersionOption" type="string" eval="catalog"
source="VersionOption">inDoc</Parameter>
```

When eval is set to catalog, the source is the property of the catalog. Also, the catalog is automatically selected using the indocument. The indocument must be able to resolve the following xpaths:
/Message/Body/Document/BusinessDocument/CatalogAction/CatalogActionHeader/CatalogReference[@type='Catalog']/RevisionID/DBID
/Message/Body/Document/BusinessDocument/CatalogAction/CatalogActionHeader/CatalogReference[@type='Catalog']/RevisionID/Version

- **event** Using the event evaluation, you can access attributes of the event. The following is a list of attributes that you can access from the event:

Attribute Name	Description
For information about valid values of these attributes, see UpdateEvent Parameters and Valid Execution Modes	
eventID	An ID of the current event.
eventDescriptor	The descriptor for the event.
eventState	The latest state of the event.
eventStatus	The current status of the event.
processID	The current process ID.
eventDate	The start date of an event.
eventParentID	Event ID of the parent event that initiates the current event.

```
<Parameter direction="in" type="long" eval="event" name="eventID1"
source="eventID"></Parameter>
        <Parameter direction="in" type="string" eval="event" name="eventDescriptor1"
source="eventDescriptor"></Parameter>
        <Parameter direction="in" type="string" eval="event" name="eventState1"
source="eventState"></Parameter>
        <Parameter direction="in" type="string" eval="event" name="eventStatus1"
source="eventStatus"></Parameter>
        <Parameter direction="in" type="long" eval="event" name="processID1"
source="processID"></Parameter>
        <Parameter direction="in" type="long" eval="event" name="eventParentID1"
source="eventParentID"></Parameter>
```

- **userprofile**

  Using the userprofile evaluation, you can access attributes from the user profile. The following is a list of attributes that you can access from the userprofile:

Attribute Name	Description
username	The name of a user who initiated an event.
enterpriseID	An ID of an enterprise.
enterpriseName	The name of an enterprise.
enterpriseInternalName	The internal name of an enterprise.
enterpriseVertical	Enterprise vertical.
organizationID	An ID of the organization.
organizationName	The name of an organization.
organizationType	The type of an organization. The types are Retailer, Buyer, or Supplier.
userID	An ID of the user.
formattedUserName	The format is lastName + "," + firstName + " " + middleName.

```
<Parameter direction="in" type="string" eval="userprofile" name="userName1" source="userName"></Parameter>

    <Parameter
direction="in" type="long" eval="userprofile" name="enterpriseID1" source="enterpriseID"></Parameter>

    <Parameter direction="in" type="long" eval="userprofile" name="userID1"  source="userID"></Parameter>

    <Parameter
direction="in" type="long" eval="userprofile" name="organizationID1" source=organizationID"></Parameter>
```

```
<Parameter direction="in" type="string" eval="userprofile" name="organizationName1" source="organizationName" ></Parameter>
<Parameter direction="in" type="string" eval="userprofile" name="organizationType1" source="organizationType"></Parameter>
<Parameter direction="in" type="string" eval="userprofile" name="enterpriseName1" source="enterpriseName"></Parameter>
<Parameter direction="in" type="string" eval="userprofile" name="myvalue" source="myvalue"></Parameter>
```

- **system**

  Using the system evaluation, you can access attributes from the instance. The following is a list of attributes that you can access from the system:

Attribute Name	Description
systemDate	Refers to the system date. The recommended data type is string.
systemTimestamp	Refers to the system date and time. The recommended data type is timestamp.
hostname	Refers to the localhost. The recommended data type is string.
hostAddr	Refers to the host IP address. The recommended data type is string.
nodeID	Refers to the ID of the instance. For example, Member 1. The recommended data type is string.

Attribute Name	Description
systemTimestampLong	The recommended data type is long.
homeDir	Refers to the MQ_HOME directory. The recommended data type is string.
commonDir	Refers to the MQ_COMMON_DIR directory. The recommended data type is string.
logDir	Refers to the MQ_LOG directory. The recommended data type is string.

```
<Parameter direction="in" type="string" eval="system" name="systemDate1"
source="systemDate"></Parameter>
<Parameter direction="in" type="timestamp" eval="system" name="systemTimestamp1"
source="systemTimestamp"></Parameter>
<Parameter direction="in" type="string" eval="system" name="hostname1"
source="hostname"></Parameter>
<Parameter direction="in" type="string" eval="system" name="hostAddr1"
source="hostAddr"></Parameter>
<Parameter direction="in" type="string" eval="system" name="nodeID1"
source="nodeID"></Parameter>
<Parameter direction="in" type="long" eval="system" name="systemTimestampLong1"
source="systemTimestampLong"></Parameter>
<Parameter direction="in" type="string" eval="system" name="mqhome"
source="homeDir"></Parameter>
<Parameter direction="in" type="string" eval="system" name="commondir"
source="commonDir"></Parameter>
<Parameter direction="in" type="string" eval="system" name="mqlog"
source="logDir"></Parameter>
```

Note: If you do not use the recommended data type, an attempt is made to convert the data type to the specified data type. For example, if the systemDate data type is specified as string, the data is an output as string instead of date.

- Compute

  Using compute, you can calculate work item expiry date based on the record

attributes. You need to define the Rulebase parameter which is used to compute the target expiry date. The formula of the actual expiry date:

```
the expiry date = Launch Date(output from rulebase) - RemindBeforeNumberOfDays
```

Based on the value of ReminderNumber (reminders generated so far), configure the 'Reminder Setup' rule to retrieve the values of RemindBeforeNumberOfDays and email addresses to send the reminder email. Set the rules in such a way that for each Reminder number, only one value for RemindBeforeNumberOfDays is returned (although any number of email addresses can be returned). When a record is edited or a work item revivified, expiry date is recomputed and updated into the work item. If the recomputed expiry date differs from the old expiry date, the ReminderNumber is set to zero. For more information about compute, see the ExpiryType parameter in CreateWorkItem Parameters and Valid Execution Modes.

Parameters to Activity (local)

Each activity defines required and optional local parameters that can be passed to it.

Parameters to Workflow (global)

A set of variables can be defined with the scope of the entire workflow definition. Any activity or transition inside the workflow process has access to the global parameters.

An activity would typically use a global parameter by mapping it onto a local parameter and a transition would use it in the code section of its condition.

All activities take 0 or more input parameters, and might or might not return any output parameters. The syntax for activity parameters is similar to workflow parameters, except that input activity parameters allow more types.

Additional Parameters

You can pass in additional parameters using some activities. They are included in detail with the description of each activity.

- Inputs into a rulebase

  Using some activities such as EvaluateRulebase and IdentifyProtocolOperations you can pass in parameters that match input variables declared in the rulebase. These are then passed into the rulebase for further processing.

- Inputs to a form template

  Email templates can specify tags that are not specified in the form file. In this case, the tag value is retrieved from inputs to the Send activity.

- Store as name/value pairs

  Some activities such as UpdateEvent or CreateWorkitem take additional parameters and store them in “detail” tables. These can then be accessed by the same names by other components.

- To track execution paths

  Any variable can be added to an activity to track the execution paths. Let’s say you have a branch in the workflow depending on a certain action: delete, modify, or add. Then further down in the workflow you might want to send an email. One of the tags in the email might be the action that was required. Rather than have to code the branch again, just add a new parameter called MessageAction (any name not already in use is acceptable) to the activities in the first branch. Set parameters to “Delete”, “Modify”, or “Add”. It has no immediate use in the first activity, but is set in the workflow context. Then, when it is time to send the email, this parameter can be retrieved from the workflow context, and the correct action is passed to the Email template.

Removal of Parameter from Workflow State

Once a parameter is output of an activity, it is added to the workflow state and is not removed from the state.

To explicitly remove the parameter from the state, you can assign an empty value to the parameter or reuse the parameter name for other purposes.

The total size of workflow state must not exceed 4000 bytes. As the number of activities increase, the workflow state size increases and might exceed 4000 bytes. This results in workflow failure as the process engine truncates the state. When this happens, you can:

- Split the workflow and create subflows. The state of the subflow is not included into

the calling workflow except for the output parameters returned by the subflow. See Calling a Workflow for more details.

- Remove some state variables by assigning an empty value.

  Null Values and Repeated Parameters

When an input parameter is specified to an activity, the application allows the same parameter to be derived from more than one source.

You can define the parameter more than once, each time the value being derived from a different source, that is, rule or constant.

If a parameter with an identical name is repeated in the activity, the process engine follows the following algorithm:

“Evaluate each parameter from top to bottom and stop when one returns a non-null value”.

Example of Null Values and Repeated Parameters

Placeholder for Short Description.

```
<Activity Name="InitiateSynchProducts">
  <Action>SpawnWorkflow</Action>
  <Description lang="en">Start publish workflows</Description>
  <Parameter direction="in" type="string" eval="constant"
  name="eventState">SPAWNWORKFLOW</Parameter>
  <Parameter direction="in" name="InDocument" type="document"
  eval="variable">workDoc</Parameter>
  <Parameter direction="in" name="DocumentTemplate" type="string"
  eval="constant">standard/template/tm26catpubwoutcatv1.xml</Parameter>
  <Parameter direction="in" name="ProcessID" type="string"
  eval="constant">standard/workflow/wfin26catsynchv4</Parameter>
  <Parameter direction="in" eval="constant" name="Form"
  type="string">standard/forms/fm26catpubtemplatev1.xml</Parameter>
  <Parameter direction="in" name="MasterCatalog" eval="xpath" type="long"
  source="/Message/Body/Document/BusinessDocument/CatalogAction/CatalogActi   onHeader/C
atalogReference[@type='MasterCatalog']/RevisionID/DBID/text()">   workDoc</Parameter>
  <Parameter direction="in" name="MasterCatalog" eval="xpath" type="long"
  source="/Message/Body/Document/BusinessDocument/CatalogAction/CatalogActi   onHeader/M
asterCatalog/RevisionID/DBID/text()">workDoc</Parameter>
  <Parameter direction="in" name="RecordKey" type="arraylist"
  eval="variable">SynchProductKeyArray</Parameter>
  <Parameter direction="in" name="SenderCredential" type="arraylist"
  eval="variable">SynchSenderArray</Parameter>
  <Parameter direction="in" name="ChannelCredential" type="arraylist"
  eval="variable">SynchChannelArray</Parameter>
```

```
<Parameter direction="in" name="TradingPartnerCredential" type="arraylist"
eval="variable">SynchTradingPartnerArray</Parameter>
<Parameter direction="in" name="RelationshipName" type="string"
eval="constant">Contains</Parameter>
<Parameter direction="in" name="CatalogOutputMap" type="string"
eval="constant">EAN.UCC</Parameter>
</Activity>
```

Event Details in Activity

The UpdateEvent activity can be used to update event attributes such as the Event state, event status, event description, and event type.

See Workflow Activities for more details on the UpdateEvent activity.

You can add eventState, eventStatus, eventType, and eventDescriptor parameters to any activity and it updates the corresponding event details in the activity. The purpose of this is to make it easy to update event information, so that a person looking at the Event can easily figure out what state it is in at any given time.

Note: eventState, eventStatus, eventType, and eventDescriptor are optional parameters for all activities.

Example for Event Details in Activity

```
<Parameter direction="in" type="string" eval="constant"
name="eventState">GENERATEFILE</Parameter>
<Parameter direction="in" type="string" eval="constant"
name="eventStatus">SUCCESS</Parameter>
<Parameter direction="in" type="string" eval="constant"
name="eventDescriptor">CatEditAdd</Parameter>
<Parameter direction="in" type="string" eval="constant" name="eventType">USER</Parameter>
```

The event state is updated before the activity is executed. Event state must be one of the pre-defined states. As this list keeps growing, check the existing states by running the following query:

```
Select * from domainentry where domaintype = ‘EVENTSTATE’ order by value
```

It is possible to add more entries to this list. To do this, execute the following SQL:

```
insert into domainentry (domaintype, value, description) values ('EVENTSTATE', '_TEST',
'Description of the state')
```

Ensure that you always add ‘_’ to your value to avoid any conflict with predefined values (more will be added in future).

Once the state is added to this table, you can use the workflow to update the state during any activity.

Output Parameters

You can save and access workflow output parameters.

APIs are provided (see APIs) to access these parameters from external systems.

You can define any number of output parameters (out parameters defined in the workflow globally), all those with non-null values are saved. Parameters are saved for each workflow, and for both main workflows and subflows. All parameters other than those of type arraylist are supported.

In case of in-memory workflows, out parameters are saved as part of the workflow transaction (provided the saveonsucess flag is set to true). In case of regular workflows, out parameters are saved in a separate transaction after the last activity is executed. See also, Changes to Out-of-the-Box Workflows

Example: Defining and Using Out Parameters with Workflows

Consider out-of-the-box factsheet generation workflow (which is one of two workflows that have been modified), and highlight the changes made to support and use out parameters. Map the output of one activity to the output of the workflow by defining out parameters and assigning values to them in intermediate activities.

Original workflow

This is the original factsheet generation workflow, where the translate activity generates a PDF (in $MQ_COMMON_DIR) that is output of the workflow.

```
<?xml version="1.0"?>
<!-- edited with XML Spy v3.5 NT (http://www.xmlspy.com) (Tibco) -->
<!--
Sample workflow for PDF fact sheet generation
-->
<Workflow Version="1.3" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="Workflow.xsd">
  <Owner>TIBCO</Owner>
  <Name>wfin26productfactsheetpdfv2</Name>
  <Description>Sample workflow for generation of PDF data    sheets</Description>
  <Context>Sender</Context>
  <Parameter direction="in" name="inDoc" type="document"    eval="variable">1</Parameter>
  <Parameter direction="out" name="outDoc1" type="document"    eval="variable"/>
<!-- Edit loop -->
  <Activity Name="UpdateEventStartSync">
    <Start />
    <Action>UpdateEvent</Action>
    <Description lang="en">Update event state to indicate start of      workflow</Description>
    <Parameter direction="in" name="eventDescriptor" type="string"
    eval="constant">FACTSHEET</Parameter>
    <Parameter direction="in" type="string" eval="constant"
    name="eventState">START</Parameter>
    <Parameter direction="in" name="InDocument" type="document"
    eval="variable">inDoc</Parameter>
  </Activity>
<Activity Name="GetItemData">
    <Action>GetRecord</Action>
    <Description lang="en">Get record data</Description>
    <Parameter direction="in" type="string" eval="constant"
    name="eventState">GETPRODUCTINFO</Parameter>
    <Parameter direction="in" name="InDocument" type="document"
    eval="variable">inDoc</Parameter>
    <Parameter direction="in" eval="constant" type="string"
    name="Agency">SOURCE</Parameter>
    <Parameter direction="in" eval="xpath" type="long" name="MasterCatalog"
    source="/Message/Body/Document/BusinessDocument/CatalogAction/CatalogA     ctionHeade
r/MasterCatalog/RevisionID/DBID/text()">inDoc</Parameter>
    <Parameter direction="out" eval="variable" type="document"
    name="OutDocument">workDoc</Parameter>
  </Activity>
<!-- End of Add approval loop -->
```

```
<Activity Name="SetStatusToSuccess">
    <Action>UpdateEvent</Action>
    <Description lang="en">Set the event state to      Success/Done</Description>
    <Parameter direction="in" name="eventStatus" type="string"
    eval="constant">SUCCESS</Parameter>
    <Parameter direction="in" type="string" eval="constant"
    name="eventState">DONE</Parameter>
  </Activity>
  <Activity Name="SetStatusToError">
    <Action>UpdateEvent</Action>
    <Description lang="en">Set the event state to Error/Done</Description>
    <Parameter direction="in" name="eventStatus" type="string"
    eval="constant">ERROR</Parameter>
    <Parameter direction="in" type="string" eval="constant"
    name="eventState">DONE</Parameter>
  </Activity>
  <Activity Name="CreatePDFDataSheet">
    <Action>Translate</Action>
    <Description>Translate the record data to to PDF datasheet</Description>
    <Parameter direction="in" type="string" eval="constant"
    name="eventState">TRANSLATE</Parameter>
    <Parameter direction="in" name="Derived" type="string"      eval="constant">true</Parameter>
    <Parameter direction="in" eval="constant" type="string"
    name="Translator">XML2PDF</Parameter>
    <Parameter direction="in" eval="constant" type="string"
    name="Map">standard/maps/mpfrom26topdfprodspecv1</Parameter>
    <Parameter direction="in" eval="constant" type="string"
    name="OutputFormat">PDF</Parameter>
    <Parameter direction="in" eval="variable" type="document"
    name="InDocument">workDoc</Parameter>
    <Parameter direction="out" eval="variable" type="document"
    name="OutDocument">pdfDataDoc</Parameter>
  </Activity>
  <Transition FromActivity="UpdateEventStartSync" ToActivity="GetItemData"/>
  <Transition FromActivity="GetItemData" ToActivity="CreatePDFDataSheet" />
  <Transition FromActivity="CreatePDFDataSheet"    ToActivity="SetStatusToSuccess" />
  <Transition type="error" FromActivity="Any" ToActivity="SetStatusToError"    />
</Workflow>
```

Modified Workflow

This is the modified factsheet generation workflow, where you can define an out parameter at the workflow level, and in the Translate activity (used to convert the record date to a PDF datasheet), the outdocument is mapped to this out parameter.

```
<?xml version="1.0"?>
<!-- edited with XML Spy v3.5 NT (http://www.xmlspy.com) (Tibco) -->
<!--Sample workflow for PDF fact sheet generation -->
<Workflow Version="1.3" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="Workflow.xsd">
  <Owner>TIBCO</Owner>
  <Name>wfin26productfactsheetpdfv2</Name>
  <Description>Sample workflow for generation of PDF data    sheets</Description>
  <Context>Sender</Context>
  <Parameter direction="in" name="inDoc" type="document"    eval="variable">1</Parameter>
  <Parameter direction="out" name="outDoc1" type="document"    eval="variable"/>
<!-- Edit loop -->
  <Activity Name="UpdateEventStartSync">
    <Start />
    <Action>UpdateEvent</Action>
    <Description lang="en">Update event state to indicate start of      workflow</Description>
    <Parameter direction="in" name="eventDescriptor" type="string"
    eval="constant">FACTSHEET</Parameter>
    <Parameter direction="in" type="string" eval="constant"
    name="eventState">START</Parameter>
    <Parameter direction="in" name="InDocument" type="document"
    eval="variable">inDoc</Parameter>
  </Activity>
  <Activity Name="GetItemData">
    <Action>GetRecord</Action>
    <Description lang="en">Get record data</Description>
    <Parameter direction="in" type="string" eval="constant"
    name="eventState">GETPRODUCTINFO</Parameter>
    <Parameter direction="in" name="InDocument" type="document"
    eval="variable">inDoc</Parameter>
    <Parameter direction="in" eval="constant" type="string"
    name="Agency">SOURCE</Parameter>
    <Parameter direction="in" eval="xpath" type="long" name="MasterCatalog"
    source="/Message/Body/Document/BusinessDocument/CatalogAction/CatalogA     ctionHeade
r/MasterCatalog/RevisionID/DBID/text()">inDoc</Parameter>
    <Parameter direction="out" eval="variable" type="document"
    name="OutDocument">workDoc</Parameter>
  </Activity>
<!-- End of Add approval loop -->
  <Activity Name="SetStatusToSuccess">
    <Action>UpdateEvent</Action>
    <Description lang="en">Set the event state to      Success/Done</Description>
    <Parameter direction="in" name="eventStatus" type="string"
    eval="constant">SUCCESS</Parameter>
    <Parameter direction="in" type="string" eval="constant"
    name="eventState">DONE</Parameter>
```

```
  </Activity>
  <Activity Name="SetStatusToError">
    <Action>UpdateEvent</Action>
    <Description lang="en">Set the event state to Error/Done</Description>
    <Parameter direction="in" name="eventStatus" type="string"
    eval="constant">ERROR</Parameter>
    <Parameter direction="in" type="string" eval="constant"
    name="eventState">DONE</Parameter>
  </Activity>
  <Activity Name="CreatePDFDataSheet">
    <Action>Translate</Action>
    <Description>Translate the record data to PDF datasheet</Description>
    <Parameter direction="in" type="string" eval="constant"
    name="eventState">TRANSLATE</Parameter>
    <Parameter direction="in" name="Derived" type="string"      eval="constant">true</Parameter>
    <Parameter direction="in" eval="constant" type="string"
    name="Translator">XML2PDF</Parameter>
    <Parameter direction="in" eval="constant" type="string"
    name="Map">standard/maps/mpfrom26topdfprodspecv1</Parameter>
    <Parameter direction="in" eval="constant" type="string"
    name="OutputFormat">PDF</Parameter>
    <Parameter direction="in" eval="variable" type="document"
    name="InDocument">workDoc</Parameter>
    <Parameter direction="out" eval="variable" type="document"
    name="OutDocument">outDoc1</Parameter>
  </Activity>
  <Transition FromActivity="UpdateEventStartSync" ToActivity="GetItemData"/>
  <Transition FromActivity="GetItemData" ToActivity="CreatePDFDataSheet" />
  <Transition FromActivity="CreatePDFDataSheet"    ToActivity="SetStatusToSuccess" />
  <Transition type="error" FromActivity="Any" ToActivity="SetStatusToError"    />
</Workflow>
```

APIs

APIs have been exposed so you can access workflow output parameters.

The following APIs are exposed:

- EventBusinessDelegate.getWorkflowOutput (in eventID)

  This API accepts the Event ID as input and returns a list of all workflow outputs (as a formatted string processID, Name, Value, Type). Here, Name represents the parameter name and Type represents the type of the value associated.

- DocumentBusinessDelegate.downloadWorkflowOutputDocumentOrFile(String processId,

String name, String value, String valueType)

This API can be used to download workflow outputs of type FILE or DOCUMENT. It returns byte.

Workflow Activities

Workflow activities are atomic work units which can be connected with each other to define a business process.

Workflow activities share a common set of tags that define their execution mode and interaction with the rest of the workflow system.

An activity is defined by a Name which is any string of size 80 or less characters. This name (English only) must convey the purpose of the activity and appears in the Event Log.

A workflow cannot have duplicate actions.

```
<Activity Name="InternalEditMoveToFirst">
  <Start/>
  <Action>StateTransition</Action>
  <Description lang="en">Set workflow to next step.</Description>
  <Execution>SYNCHR</Execution>
  <Parameter direction="in" name="InDocument" type="document" eval="variable">inDoc</Parameter>
  <Parameter direction="out" name="ConversationState" eval="variable"
type="string">InternalEditNextState</Parameter>
</Activity>
```

Workflow Activity Semantics

Tag	Semantics	Signature
Start (Only one activity can have a start tag)	Specifies the first activity to execute. No valid values.	<Start/>
Action	The type of activity to execute. **Valid Value** See list of activities. The name specified	<Action>#PCDATA</Action>
Description (Optional)	A textual description of the activity. The description must contain 80 or less characters, else an error occurs. It is a good practice to provide the description. The description shows up in the Event Details and can be in any language. **Valid Value** Any arbitrary string.	<Description>#PCDATA</Description>
Execution	The execution mode. SYNCHR - the process engine waits for the activity to complete before proceeding. ASYNCHR - the process engine does not wait for the activity to complete. Failure of the activity might be ignored by the process engine.	<Execution>#PCDATA</Execution>

Tag	Semantics	Signature
	Valid Value SYNCHR and ASYNCHR. SYNCHR is supported for all activity types. Check details of each activity to verify if it supports ASYNCHR.	

Note: #PCDATA is a token used in an element declaration to declare the element as having mixed content (character data, or character data mixed with other elements). The content of the element is parsed; '&' and '<' have a special meaning and must be escaped if they are not the start of markup.

Workflow Context Variables

Two workflow context variables are added in the workflow to handle errors.

- CIM_errorCode: Refers to an error code.
- CIM_errorDoc: Document that is used to trace errors. You can download this document from the Event Log page.

These variables are populated by the workflow engine when an error occurs while processing the workflow. The workflow designer can use these parameters to define flow or process error details. You can access these variables from any workflow activity as follows:

```
<Parameter direction="in" type="string" eval="variable" name="testerrorCode">CIM_
errorCode</Parameter>
<Parameter direction="in" type="document" eval="variable" name="testerrorDoc">CIM_
errorDoc</Parameter>
```

Generic Output Parameter StepID and ReferenceStepID

Some activities require an optional input ReferenceStepID that points to a previous execution step.

This is done by capturing the parameter StepID from the required step. This output parameter StepID is generated for each activity, and it points to a unique number identifying each workflow step.

In the following example, the activity CreateWorkItem needs a reference to the CompareRecord step.

```
<Activity Name="CompareRecord">
  <Action>CompareRecord</Action>
  <Description lang="en">Compare with previous confirmed version and generate an output
document</Description>
  <Parameter direction="in" name="InDocument" type="document"
eval="variable">workDoc</Parameter>
  <Parameter direction="in" name="CompareKeyword" type="string" eval="constant">PREVIOUS_
CONFIRMED_VERSION</Parameter>
  <Parameter direction="in" name="FullCompareFlag" type="string"
eval="constant">True</Parameter>
  <Parameter direction="out" eval="variable" type="document"
name="OutDocument">workDoc</Parameter>
  <Parameter direction="out" name="StepID" eval="variable" type="long">pl1</Parameter>
</Activity>
<Activity Name="InternalEditWorkItem">
  <Action>CreateWorkItem</Action>
  <Description lang="en">Create workitem for edit of the new record</Description>
  <Execution>SYNCHR</Execution>
  <Parameter direction="in" eval="constant" type="string" name="Intent">Edit</Parameter>
  <Parameter direction="in" eval="constant" type="string"
name="RecordAttributeName">GTIN</Parameter>
  <Parameter direction="in" eval="constant" type="string"
name="RecordAttributeName1">SHORTDESC</Parameter>
  <Parameter direction="in" eval="constant" type="string"
name="RecordAttributeName2">UOM</Parameter>
  <Parameter direction="in" name="ReferenceStepID" eval="variable"
type="long">pl1</Parameter>
  <Parameter direction="in" eval="rule" source="New Record Introduction Edit" type="long"
name="ParticipantID">workDoc</Parameter>
  <Parameter direction="in" eval="rule" source="New Record Introduction Edit" type="string"
name="ParticipantType">workDoc</Parameter>
  <Parameter direction="in" eval="constant" type="string"
```

```
name="Form">standard/forms/fm26ca.xml</Parameter>
  <Parameter direction="in" eval="constant" type="string"
name="MailPresentation">com.tibco.mdm.ui.workflow.engine.emailtemplates.
  EmailProductAddEditWorkItem</Parameter>
  <Parameter direction="in" eval="constant" type="string" name="ReassignMailPresentation">
  com.tibco.mdm.ui.workflow.engine.emailtem plates.EmailReassignWorkItem</Parameter>
  <Parameter direction="in" eval="constant" type="string" name="DelegationMailPresentation">

com.tibco.mdm.ui.workflow.engine.email templates.EmailWorkItemDelegationNotification</Param
eter>
  <Parameter direction="in" eval="constant" type="string" name="FormPresentation">
  com.tibco.mdm.ui.workflow.engine.workitem.templates. ProductAddEdit</Parameter>
  <Parameter direction="in" eval="constant" type="string"
name="TaskType">CATALOGMESSAGE</Parameter>
  <Parameter direction="in" eval="variable" type="document"
name="InDocument">workDoc</Parameter>
  <Parameter direction="out" eval="variable" type="document"
name="OutDocument">wiDoc</Parameter>
  <Parameter direction="out" eval="variable" type="boolean"
name="WorkItemCreated">editFlag</Parameter>
  <Parameter direction="out" eval="variable" type="boolean"
name="MergedDocument">workDoc</Parameter>
  <Parameter direction="in" eval="constant" type="string"
name="ExpiryType">RELATIVE</Parameter>
  <Parameter direction="in" eval="constant" type="string"
name="ExpiryDate">1:0:0:0</Parameter>
</Activity>
```

Common Parameters

This section lists common parameters applicable to several activities.

Parameters Common Across Many Workflow Activities

Name	Semantics	Type	Valid Values	Multiplicity
AsynProcessIndicator	Indicates whether the activity must be processed synchronously or asynchronously. This must not be confused with activity definition <Execution>SYNCHR</Execution>. This indicator is a suggestion to the process engine to optimize execution of the activity. The activity might initiate processing using multiple threads. This parameter works with RecordPerAsyncCall to determine if parallel processing must be initiated. When the records to be imported are large in number, they can be divided into groups and imported by different asynchronous calls. When the records to be processed are less than the RecordPerAsyncCall, the AsynProcessIndicator is false even if it is specified as true in the workflow.	boolean	True (Default) False	0..1
IncludeDraftRecords	Specifies that if there are any draft record versions associated with the event, such versions must be considered. For example: `<Parameter direction="in" name="IncludeDraftRecords" type="boolean" eval="constant">false</Parameter>` If not specified, default is true to include draft records. This parameter might be set to false if the designer is certain that no draft records must be included. This would result in slightly better performance.	boolean	True (Default) False	1
InDocument	The document to process (typically the workflow request document) in the workflow. There can currently only be one document.	document	mIXML document.	1
InRecordList	Specifies which records to work on. If a record collection is passed as input to this activity, this activity processes the record collection. For example: `<Parameter direction="in" name="InRecordList" type="recordlist" eval="variable">inRL</Pa rameter>`	recordlist	Valid variable. Holds the records to be processed.	1

Name	Semantics	Type	Valid Values	Multiplicity
MasterCatalog (Optional)	The name of the repository.	long or string	Any existing Repository ID or Name.	0..1
OutDocument(Optional)	The output or returned document.	document	N/A	N/A
OutRecordList(Optional)	Output Record List.	recordlist	Output records.	1
RecordPerAsyncCall	If the activity is being processed asynchronously, this parameter indicates records to be processed asynchronously called message. The configuration file parameter can be set using the Configurator (**Optimization** > **Records Per Asynchronous Call**). The order of configuration is: Workflow file > Configuration file > Default value.	long	Any number. 20 (Default) The default value is 100.	1
BundlePerAsyncCall (Optional)	When the records are processed using a record Collection (which is a set of bundles), this parameter must be specified. Either BundlePerAsyncCall or RecordPerAsyncCall is used depending on whether bundles are being processed or not. If the activity is being processed asynchronously, this parameter indicates the bundles to be processed per asynchronously called message. Set the following option to 10 in the Configurator: **Optimization** > **Bundles Per Asynchronous Call**. The order of configuration is: Workflow file > Configuration file > Default value.	long	Any number. 10 (Default)	1
Rulebase(Optional)	The Rulebase to execute. Relative to $MQ_COMMON_DIR. You can also specify an absolute path, however, it is not a good practice to do so. If you do not specify any rulebase file, the standard catalogvalidation.xml file is used. The file is located at $MQ_COMMON_DIR\<enterprise_name>catalog>master><repositoryID>. However, to use this file, you must specify the true value for the CatalogValidation parameter of the EvaluateRulebase activity.	string	Existing Rulebase file.	1
VersionOption	Specifies the version of a record to be considered for processing.	string	CONFIRMED (Default) LATEST ALL	1
ErrorSeverity(Optional)	Defines the error severity. Any severity higher than the specified severity are considered errors and reported using the ValidationErrors output parameter. This parameter applies when a rulebase is used in the or EvaluateRuleBase activity	long	0 - 9. 0 : nothing is considered an error. 9 : everything is considered an error.	0..n

Name	Semantics	Type	Valid Values	Multiplicity
			The default value is 9.	
eventState	Determines the latest state of the workflow.	string	Valid value differs per activity.	0..1
WorkflowPriority (Optional)	Defines the priority for any workflow initiated from the activity or any message to external JMS queue by Send activity.	long	0 – 9. 0 – Workflow with 0 is considered low priority. 9 – Workflow with 9 is considered high priority. The default value is 4.	0 or 1
FEDOption(Optional)	Indicates whether the exported data contains Future Effective Date records.	string	The valid values are: **O**: If specified, export data includes only Future Effective Date records. **N**: If specified, export data does not include Future Effective Date records. **I**: If specified, export data includes Future Effective Date records and non Future Effective Date records.	0..1

AddressCleansing Activity

Use the AddressCleansing activity to cleanse geocode bulk records on add or update operations by using the configured GeoAnalytics service.

Consider a scenario where a customer has incoming location data into TIBCO MDM, which includes an address information. For example, any location-based services where a vendor has pickup or delivery or shipping addresses or a bank has customer addresses. This data either captured manually or otherwise, might not be accurate always, and must be entered quickly into the system without having time to correct it.

For correction of such records having address information, bulk geocoding or cleansing services can be used. For example, the incoming data includes Country=US and ZIP=10307. After the address is cleansed, the following response is returned where City and State information is also included and which is the most accurate address: City=New York, State=NY, Country=US, and ZIP=10307. Use import functionality when records are being added in large volumes.

This activity works on the input records list from the workflow, saves the modified record after geocoding, and then provides an output, that is, success or the rejected records list. You can use the RejectInvalid parameter to accept or reject the records which failed during address cleansing or validation. Use the VersionPolicy parameter for the record version correction and for avoiding creation of a new version of the records.

AddressCleansing Parameters and Valid Execution Modes

The valid execution mode for AddressCleansing is SYNCHR.

The parameters of AddressCleansing are as follows:

AddressCleansing Parameters

Name	Semantics	Type	Valid Values	Multiplicity
Direction: In				

Name	Semantics	Type	Valid Values	Multiplicity
eventState	Determines the latest state of the workflow.	string	Predefined state: DONE. Custom states can be defined in the DOMAINENTRY table.	0..1
MasterCatalog	See Common Parameters	string		
InRecordList	See Common Parameters	recordlist		
Mode	Sets the Geocoding mode to FORWARD. if you select FORWARD, the address points to the latitude and longitude mentioned on the map.	string	The valid and default value is FORWARD.	
RejectInvalid	If Geocoding or cleansing for any record fails, selecting this option decides whether to reject or accept an invalid initial input data. Failed records are added to rejectRecordList	boolean	The valid values are true or false. The default value is true.	

Name	Semantics	Type	Valid Values	Multiplicity
	(OutRecordList2) and succeeded records are added to workRecordList (OutRecordList) and are passed on further in the workflow.			
VersionPolicy (Optional)	Indicates how the record versions are managed. The allowed VersionPolicy values are NEW, CORRECT, OPTIMIZE. • NEW: By using this option, you can create a new version of a record. This is the default value. • CORRECT : By using this option, you can update the existing record	string	• NEW (Default): creates a new version • CORRECT: updates an existing record version, no new version. If policy rule allows correction. • OPTIMIZE: this is a combination of both NEW and CORRECT. Based on the choice,a record can be corrected or a new	0..1

Name	Semantics	Type	Valid Values	Multiplicity
	versions. New version is not created, only if the old record version state and new record version state match. • OPTIMIZE: By using this option, you can optimize the best choice between the CORRECT and NEW options. New version gets created based on the choice selected. The following rules are used when VersionPolicy=		version is created.	

Name	Semantics	Type	Valid Values	Multiplicity
	OPTIMIZE: • A new record version is created when the old and new record state do not match. • A new record version is created when the old record state is REJECTED. • A new record version is created when the old record version is DELETED. • The existing record versions are corrected when both the old and new record			

Name	Semantics	Type	Valid Values	Multiplicity
	state match. • For everything else, the existing record versions are corrected.			
InDocument	See Common Parameters	document		
Direction: Out				
OutDocument	See Common Parameters	document		
OutRecordList	See Common Parameters	recordlist		
OutRecordList2	List of rejected records which failed during address validation.	recordlist		
RecordPerAsyncCall	See Common Parameters	long		

- Set the RecordPerAsyncCall parameter based on the configuration of GeoAnalytics service. For information about the GeoAnalytics configuration properties, see the section, "Configuration Properties of TIBCO GeoAnalytics" in *TIBCO MDM System Administration*.

Name	Semantics	Type	Valid Values	Multiplicity
	• With Maporama service, 200 records in one batch request are recommended for Geocoding.			
AsynProcessIndicator	See Common Parameters	boolean		

Example for AddressCleansing Activity

```
<Activity Name="AddressCleansing">
    <Action>AddressCleansing</Action>
    <Description lang="en">Cleanse or fill the address attributes by either Geocoding or reverse-geocoding</Description>
    <Parameter name="eventState" direction="in" type="string" eval="constant">ADDRCLEANSING</Parameter>
    <Parameter direction="in" eval="xpath" type="string" name="MasterCatalog" source
="/Message/Body/Document/BusinessDocument/CatalogAction/CatalogActionHeader/MasterCatalog/RevisionID/BaseNam
e/text()">inDoc</Parameter>
    <Parameter direction="in" name="InDocument" type="document" eval="variable">inDoc</Parameter>
    <Parameter direction="in" name="InRecordList" type="recordlist" eval="variable">workRecordList</Parameter>
    <!-- MODE: FORWARD - Address to Point/LatLng -->
    <Parameter direction="in" name="Mode" type="string" eval="constant">FORWARD</Parameter>
    <!-- Parameter direction="in" name="AsynProcessIndicator" type="boolean" eval="constant">true/Parameter-->
  <!-- Parameter direction="in" name="VersionPolicy" type="string" eval="constant">CORRECT/Parameter-->
```

```
<!-- Reject or accept invalid initial input data, if it fails to Geocode/cleanse to correct address. Defaults to true i.e. reject -->
    <Parameter direction="in" name="RejectInvalid"  type="boolean" eval="constant">true</Parameter>
    <Parameter direction="out" name="OutDocument" type="document" eval="variable">workDoc</Parameter>
    <Parameter direction="out" name="OutRecordList" type="recordlist" eval="variable">workRecordList</Parameter>
    <Parameter direction="out" name="OutRecordList2" type="recordlist" eval="variable">rejectRecordList</Parameter>
</Activity>
```

CheckHierarchyState Activity

The CheckHierarchyState activity works in two modes based on the value of CheckHierarchyState required parameter. The value of CheckHierarchyState parameter can either be LockHierarchy or CheckLockState.

- If you set the value to LockHierarchy, the activity acquires a lock on the hierarchy available in InDocument for the current user and an event. If the hierarchy is already locked for or by the current user or member, the activity updates the lock for the current Event ID. If the activity fails to lock the hierarchy available in InDocument for the current user or member, or an event ID, an exception occurs.
- If you set the value to CheckLockState, the activity only confirms that the hierarchy available in InDocument is locked by the current user or member running the workflow and current event. If the hierarchy available in InDocument is locked by another member or user, an exception occurs and the user cannot modify the hierarchy.

CheckHierarchyState Parameters and Valid Execution Modes

The valid execution mode for CheckHierarchyState is SYNCHR.

The parameters of CheckHierarchyState are as follows:

CheckHierarchyState Parameters

Name	Semantics	Type	Valid Values	Multiplicity
Direction: In				
InDocument (Mandatory)	See Common Parameters.	Document	N/A	N/A
eventState	See Common Parameters.	N/A	LOCKHIERARCHY	0..1
CheckHierarchyState	Specify either LockHierarchy or CheckLockState.	String	Default is CheckLockState.	1
Direction: Out				
RejectionParticipantID	The user or member who owns the lock on the hierarchy that is available in InDocument is the participant user in case of a rejection work item.	Long	Participant ID in rejection work item for hierarchy approval rejection	1
OutDocument	See Common Parameters.	Document	N/A	N/A

Example for CheckHierarchyState Activity

The following example locks hierarchy for the current user and event.

```
<Activity Name="LockHierarchy">
<Action>CheckHierarchyState</Action>
<Description lang="en">Lock Hierarchy for current User and Event</Description>
<Parameter direction="in" type="string" eval="constant"
name="eventState">LOCKHIERARCHY</Parameter>
<Parameter direction="in" name="InDocument" type="document"
eval="variable">workDoc</Parameter>
<!-- CheckHierarchyState: CheckLockState/LockHierarchy. Default - CheckLockState -->
<Parameter direction="in" name="CheckHierarchyState" type="string"
eval="constant">LockHierarchy</Parameter>
<!-- The owner of Hierarchy lock is Participant User in case of Rejection. -->
<Parameter direction="out" eval="variable" type="long"
name="RejectionParticipantID">RejectionParticipantID</Parameter>
<Parameter direction="out" eval="variable" type="document"
name="OutDocument">workDoc</Parameter>
</Activity>
```

The following example checks if hierarchy lock is owned by the current user and event.

```
<Activity Name="CheckHierarchyState">
<Action>CheckHierarchyState</Action>
<Description lang="en">Check (lock) state of the hierarchy</Description>
<Parameter direction="in" type="string" eval="constant"
name="eventState">CHECKHIERARCHYSTATE</Parameter>
<Parameter direction="in" name="InDocument" type="document"
eval="variable">workDoc</Parameter>
<!-- CheckHierarchyState: CheckLockState/LockHierarchy Default - CheckLockState -->
<Parameter direction="in" name="CheckHierarchyState" type="string"
eval="constant">CheckLockState</Parameter>
<!-- The owner of Hierarchy lock is Participant User in case of Rejection. -->
<Parameter direction="out" eval="variable" type="long"
name="RejectionParticipantID">RejectionParticipantID</Parameter>
<Parameter direction="out" eval="variable" type="document"
name="OutDocument">workDoc</Parameter>
</Activity>
```

CheckMessageStatus Activity

CheckMessageStatus activity can be used to determine if an incoming message is associated with any suspended workflow or not. A response to any request sent earlier would find the suspended workflow.

The activity uses the messageID to correlate the message to the corresponding workflow and finds the status of the workflow. It is possible that the workflow has already completed if the response is received later than the configured time.

The activity can accept a list of message IDs, and then concatenate the IDs to create a correlationID.

Based on the Message Status and Process Status, decisions can be made about the manner in which to process the incoming message.

CheckMessageStatus Parameters and Valid Execution Modes

The valid execution mode for CheckMessageStatus is SYNCHR.

The parameters of CheckMessageStatus are as follows:

CheckMessageStatus Parameters

Name	Semantics	Type	Valid Values	Multi-plicity
Direction: In				
MessageID# (Optional)	MessageID of the Message that was sent to the Application. Replace # with an integer, for example, MessageID1, MessageID2. The activity concatenates them. The lookup stops when the next messageID# is not found, that is, there must be no gap in the sequence.	string	Is generally an Xpath into the received message.	0..1
SelectionMethod (Optional)	Used to decide if the status of the first or last application task must be returned for the given message ID. Application tasks are internal work items which keep track of requests. Typically, one request would result in one application task. However, this might change as time-outs happen.	string	First Last (Default)	1
Direction: Out				
MessageID	The Message ID for which status is retrieved. It is a concatenation of all MessageID# values passed to the activity.	string		1
MessageStatus (Optional)	Status of the first or last (since there can be more than one) Application Task corresponding to the MessageID.	string	NOTFOUND OPEN CLOSED OPEN_TIMEDOUT CLOSED_TIMEDOUT	1
ProcessID (Optional)	ProcessID of the last Application Task found.	long	-1 if no waiting application task found, that is if MessageStatus = NOTFOUND. Else, the process instance ID of the last Application Task found.	1
ProcessStatus (Optional)	Status of the process.	string	Available only if a ProcessID > 0. Status can be: INPROGRESS - process is running.	1

Name	Semantics	Type	Valid Values	Multi-plicity
			WAIT-TASK - suspended and waiting for restart event. WAIT-WORKITEM – suspended and waiting for work item action. COMPLETED - done with processing. CANCELLED – process is cancelled.	

Example for CheckMessageStatus Activity

If your messageID was of the pattern: ProductID + '-' + ProductExt.

```
<Activity Name="CheckMessageStatus">
  <Action>CheckMessageStatus</Action>
  <Description lang="en">Check the status of the message</Description>
  <Parameter direction="in" name="MessageID1" type="string" eval="xpath"
  source="/Message/Body/Document/BusinessDocument/CatalogAction/CatalogActi   onDetails/C
atalogItem[1]/PartNumber/GlobalPartNumber/ProdID/IDNumber/text   ()">workDoc</Parameter>
  <Parameter direction="in" eval="constant" type="string"    name="MessageID2">-</Parameter>
  <Parameter direction="in" name="MessageID3" type="string" eval="xpath"
  source="/Message/Body/Document/BusinessDocument/CatalogAction/CatalogActi   onDetails/C
atalogItem[1]/PartNumber/GlobalPartNumber/ProdID/IDExtension/t   ext()">workDoc</Parameter>
  <Parameter direction="out" eval="variable" type="string"
  name="MessageStatus">msgStatus</Parameter>
  <Parameter direction="out" eval="variable" type="string"
  name="ProcessStatus">processStatus</Parameter>
  <Parameter direction="out" eval="variable" type="long"
  name="ProcessID">processId</Parameter>
  <Parameter direction="out" eval="variable" type="string"
  name="MessageID">msgID</Parameter>
</Activity>
```

CheckpointWorkflow

Use the CheckpointWorkflow activity to save process states to database at workflow milestones in case of in-memory workflows.

For in-memory workflows, the workflow state is not saved to the database during workflow execution, and in case of failure or recovery scenarios, workflow execution starts from the first activity in the workflow. CheckpointWorkflow saves minimal data required for recovery, and in case of failure, workflow execution starts with the next activity.

For more information, see CheckpointWorkflow Activity.

CheckpointWorkflow Valid Execution Mode and Example

The valid execution mode for CheckMessageStatus is SYNCHR.

Example for CheckpointWorkflow

```
<Activity Name="Milestone1">
  <Action>CheckpointWorkflow</Action>
  <Description lang="en">Save workflow state after milestone1</Description>
</Activity>
```

CheckRecordInWorkflow

Use the CheckRecordInWorkflow activity to determine if records are in workflow or if records exist.

- If the CheckRecordState parameter is set to CheckInWorkflow, the activity determines if the records passed in are in workflow.
- If the CheckRecordState parameter is set to CheckExists, the activity determines if records passed in exist in the corresponding repository.

The activity accepts records to test using the following two ways:

- Record list – If a record list is passed in, a record list is output.
- mlXML document – If a document is passed in, a document is output.

Note: For the CheckRecordInWorkflow activity, the InDocument or InRecordList parameter must be specified. For workflows created before 7.0, you must add or uncomment (if already present in the definition) the InDocument parameter. For example,

In the workflow, replace:

```
<!--<Parameter direction="in" name="InDocument" type="document"
eval="variable">inDoc</Parameter>-->
```

with:

```
<Parameter direction="in" name="InDocument" type="document"
eval="variable">inDoc</Parameter>
```

CheckRecordInWorkflow Parameters and Valid Execution Modes

The valid execution mode for CheckRecordInWorkflow is SYNCHR.

The parameters of CheckRecordInWorkflow are as follows:

CheckRecordInWorkflow Parameters

Name	Semantics	Type	Valid Values	Multiplicity
Direction: In				
CheckRecordState (Mandatory)	Determines which test to run on input records.	string	CheckInWorkflow CheckExists	1
InDocument (Optional)	See Common Parameters. Either the InDocument or			

Name	Semantics	Type	Valid Values	Multiplicity
	InRecordList parameter must be specified. If both are specified, the record collection takes precedence.			
InRecordList (Optional)	See Common Parameters. Either the InDocument or InRecordList parameter must be specified.			1
Direction: Out				
OutDocument (Optional)	See Common Parameters.			
OutRecordList (Optional)	See Common Parameters.			
RecordsProcessed (Optional)	The number of records tested by the activity.	long	Any integer.	1

CheckRecordState Input and Output options

The input and output options for CheckRecordState activity are as follows:

CheckRecordState Input and Output Options

CheckRecordState Value	Input	Output
CheckInWorkflow	record list	A record list of all the bundles with records in Workflow.
CheckInWorkflow (Optional)	mIXML document	The input mIXML Document with the ItemState/InWorkflow node for each CatalogItem updated with true if the record is in workflow, and false if the record is not in workflow.
CheckExists(Optional)	record list	A record list of all the bundles with records that do not exist in the repository for the record list. The repository checked is the repository in the record list.
CheckExists(Optional)	mIXML document	The input mIXML Document with the ItemState/Exists node for each CatalogItem updated with true if record exists in repository, and false if record does not exist in repository. The repository checked is determined from the MasterCatalog tag in the CatalogItem node or in the CatalogActionHeader node.

Example for CheckRecordInWorkflow Activity

```
<Activity Name="CheckRecordInWorkflow"><Start/>
  <Action>CheckRecordInWorkflow</Action>
  <Description lang="en">Check state of the record</Description>
  <Parameter direction="in" name="InRecordList" type="recordlist"
   eval="variable">recordList</Parameter>
  <Parameter direction="in" name="CheckRecordState" type="string"
   eval="constant">CheckExists</Parameter>
  <Parameter direction="out" name="OutRecordList" type="recordlist"
   eval="variable">recordsNotExists</Parameter>
  <Parameter direction="out" name="RecordsProcessed" type="long"
   eval="variable">recordsProcessed</Parameter>
  <Parameter direction="out" name="OutDocument"
```

```
  type="document" eval="variable">recordsNotExists</Parameter>
</Activity>
```

CompareRecord

The CompareRecord activity compares two versions of a record and generates a difference report. The difference is added to the mIXML output document. The <ActionLog> tag contains all comparison data of a record.

The comparison data includes:

- Record Attributes
- Relationships
- Relationship Attributes
- Relationship Target Records
- Classifications – Any changes in the classification data is displayed in the <ChangedClassificationData> element. The change in classification is represented in the <Classification> element. Every classification scheme that is available in the current or previous version of a record is displayed as a new <Classification> element in the mIXML document.

  The classification data is compared scheme wise using the action attribute. Changes in the classification code are compared and shown in the hierarchical format. The current and previous classification code hierarchy is compared using the <CurrentCodeHierarchy> and <OriginalCodeHierarchy> elements respectively under its respective <Classification> tag.

  > **Note:** If the classification record does not include any changes, classification information is not displayed in the mIXML document.

  The classification action performed on each classification scheme is denoted using the action attribute. The following values are displayed for the action attribute:

- Added
- Modified
- Deleted

Added Value for the Action Attribute

The record was not previously classified. However, if you classify the current version of a record, the Added value is displayed.

Example for the Added Value of the Action Attribute

The following partial XML message shows the format generated for the added classification scheme:

```
<ActionLog>
 <ChangedClassificationData>
  <Classification>
   <ClassificationScheme action="Added">
    <RevisionID>
     <BaseName>FOOD</BaseName>
    <Version>1</Version>
    <DBID>336919</DBID>
   </RevisionID>
  </ClassificationScheme>
  <CurrentCodeHierarchy>
   <ClassificationCode>
    <Code>Bakery</Code>
    <Value>BAKERY</Value>
    <ClassificationCode>
     <Code>Bread</Code>
     <Value>BREAD</Value>
    </ClassificationCode>
   </ClassificationCode>
  </CurrentCodeHierarchy>
  <OriginalCodeHierarchy />
 </Classification>
 </ChangedClassificationData>
</ActionLog>
```

Modified Value for the Action Attribute

The record was previously classified. However, if you change the classification of the current version of a record, the Modified value is displayed.

Example for the Modified Value of the Action Attribute

The following partial XML message shows the format generated for the modified classification scheme:

```
<ActionLog>
 <ChangedClassificationData>
  <Classification>
   <ClassificationScheme action="Modified">
    <RevisionID>
     <BaseName>FOOD</BaseName>
     <Version>1</Version>
     <DBID>336919</DBID>
    </RevisionID>
   </ClassificationScheme>
   <CurrentCodeHierarchy>
    <ClassificationCode>
     <Code>Bakery</Code>
     <Value>BAKERY</Value>
     <ClassificationCode>
      <Code>Cake</Code>
      <Value>CAKE</Value>
     </ClassificationCode>
    </ClassificationCode>
    <ClassificationCode>
     <Code>Dairy</Code>
     <Value>DAIRY</Value>
     <ClassificationCode>
      <Code>Milk</Code>
      <Value>MILK</Value>
     </ClassificationCode>
    </ClassificationCode>
   </CurrentCodeHierarchy>
   <OriginalCodeHierarchy>
    <ClassificationCode>
     <Code>Dairy</Code>
     <Value>DAIRY</Value>
     <ClassificationCode>
      <Code>Milk</Code>
      <Value>MILK</Value>
     </ClassificationCode>
    </ClassificationCode>
   </OriginalCodeHierarchy>
  </Classification>
```

```
 </ChangedClassificationData>
</ActionLog>
```

Deleted Value for the Action Attribute

The record was previously classified. However, if you remove the specified classification of the current version of a record, the Deleted value is displayed.

Example for the Deleted Value of the Action Attribute

The following partial XML message shows the format generated for the deleted classification scheme:

```
<ActionLog>
 <ChangedClassificationData>
  <Classification>
   <ClassificationScheme action="Deleted">
    <RevisionID>
     <BaseName>FOOD</BaseName>
     <Version>1</Version>
     <DBID>336919</DBID>
    </RevisionID>
   </ClassificationScheme>
   <CurrentCodeHierarchy />
   <OriginalCodeHierarchy>
    <ClassificationCode>
     <Code>Bakery</Code>
     <Value>BAKERY</Value>
     <ClassificationCode>
      <Code>Bread</Code>
      <Value>BREAD</Value>
     </ClassificationCode>
    </ClassificationCode>
   </OriginalCodeHierarchy>
  </Classification>
 </ChangedClassificationData>
</ActionLog>
```

CompareRecord Parameters and Valid Execution Modes

The Compare Record activity is not enabled for DRAFT records. Therefore, you cannot compare the previous draft version of a record. You can only compare with the previous confirmed or last version. When the last version is indicated, the last confirmed or unconfirmed version is selected.

The valid execution mode for CompareRecord is SYNCHR.

The parameters of CompareRecord are as follows:

CompareRecord Parameters

Name	Semantics	Type	Valid Values	Multi-plicity
Direction: In				
InDocument (Mandatory)	See Common Parameters.			
CompareKeyword (Optional)	The version to compare with. You can compare with the last valid confirmed version or last version (confirmed or unconfirmed). There is no default.	string	PREVIOUS_CONFIRMED_VERSION (Default) PREVIOUS_VERSION	1
FullCompareFlag (Optional)	Indicates if the output must include fields which have not changed. If true, even if there is no difference in the attributes of the two products, attributes are listed.	string	True False (Default)	1
CompareMode (Optional)	Indicates the difference between related records if they are added, modified, or removed. To know the difference between related records, specify the BUNDLE value. Consider the scenario, if the Customer repository is related to the Address repository, and a1 record in the Customer repository is related to b1 record in Address repository. • If you modify the a1 record by creating a new b1 related record. The output document shows the details of the newly added related record • If you modify the a1 record by deleting the b1 related record. The output document shows the information about the deleted related record. The difference details are displayed under <ChangedRelationshipData> element in the <ActionLog/> tag.	string	The valid value is BUNDLE.	1
Direction: Out				

Name	Semantics	Type	Valid Values	Multi-plicity
OutDocument (Optional)	See Common Parameters. A copy of InDocument with differences. If no differences found, this returns null.			

Example for CompareRecord Activity

```
<Activity Name="CompareRecord"><Start/>
  <Action>CompareRecord</Action>
  <Description lang="en">Compare with previous confirmed version and generate an output
document</Description>
  <Parameter direction="in" name="InDocument" type="document"
   eval="variable">workDoc</Parameter>
  <Parameter direction="in" name="CompareKeyword" type="string"
   eval="constant">PREVIOUS_CONFIRMED_VERSION</Parameter>
  <Parameter direction="in" name="FullCompareFlag" type="string"
   eval="constant">True</Parameter>
  <Parameter direction="out" eval="variable" type="document"
   name="OutDocument">workDoc</Parameter>
   <Parameter direction="out" name="StepID" eval="variable" type="long">pl1</Parameter>
   <Parameter direction="in" name="CompareMode" type="string"
   eval="constant">BUNDLE</Parameter>
</Activity>
```

ConvertRecordToOutput Activity

ConvertRecordToOutput activity works on the repository and its related repositories.

Related data from different catalogs is converted to the specified output formats using related output maps. Converted data is stored in output map tables and can be used by subsequent activities.

ConvertRecordToOutput Parameters and Valid Execution Modes

The valid execution mode for ConvertRecordToOutput is SYNCHR or ASYNCHR.

The parameters of ConvertRecordToOutput are as follows:

ConvertRecordToOutput Parameters

Name	Semantics	Type	Valid Values	Multiplicity
Direction: In				
AsynProcessIndicator (Optional)	See Common Parameters.			
BundlePerAsyncCall (Optional)	See Common Parameters.			
CatalogOutputMap(Optional)	The name or ID of the CatalogOutputMap to use. If not specified, the activity tries to evaluate (from the Trading Partner) the catalog associated with the event, else tries to derive it from the event details. If none of the methods yield a valid value, it throws an exception.	string or long	Any catalog output map name or ID.	1
InDocument(Mandatory)	See Common Parameters.			
InRecordList(Mandatory)	Collection of records to process.	recordlist	A valid variable.	1
VersionOption(Optional)	See Common Parameters.			1
FEDOption(Optional)	See Common Parameters.			
Direction: Out				
DataTruncationWarningFlag (Optional)	Flag indicating data Truncation. It is set if any record data is truncated & failed records are added to the data truncation log.	boolean	True False	1
OutRecordList(Optional)	See Common Parameters.			

Example for ConvertRecordToOutput Activity

```
<Activity Name="ConvertRecordToOutput"><Start/>
  <Action>ConvertRecordsToOutput</Action>
  <Description lang="en">Convert records data from catalog to output    format.</Description>
  <Parameter direction="in" type="string" eval="constant"
   name="eventState">CONVERTTOOUTPUT</Parameter>
  <Parameter direction="in" name="InDocument" type="document"
   eval="variable">inDoc</Parameter>
  <Parameter direction="in" name="InRecordList" type="recordlist"
   eval="variable">workRecordList</Parameter>
  <Parameter direction="in" name="CatalogOutputMap" type="long"    eval="catalog"
source="OutputMapID">inDoc</Parameter>
  <Parameter direction="out" name="OutRecordList" type="recordlist"
   eval="variable">workRecordList</Parameter>
  <Parameter direction="out" name="StepID" eval="variable"    type="long">prlog1</Parameter>
  <Parameter direction="out" name="DataTruncationWarningFlag"    eval="variable"
type="boolean">dataTruncationWarningFlag</Parameter>
  <Parameter direction="in" name="BundlePerAsyncCall" type="long"
   eval="constant">10</Parameter>
  <Parameter direction="in" name="AsynProcessIndicator" type="boolean"
   eval="constant">true</Parameter>
</Activity>
```

CreateNamedVersion Activity

CreateNamedVersion activity creates a named version of the repository after the Incremental Export Records operation is successfully completed.

During incremental export, the named version tag for subset and repository is created separately. For information about named versions, see the *Repositories Main Screen* section in the TIBCO MDM User's Guide.

CreateNamedVersion Parameters and Valid Execution Modes

The valid execution mode for CreateNamedVersion is SYNCHR

The parameters of CreateNamedVersion are as follows:

CreateNamedVersion Parameters

Name	Semantics	Type	Valid Values	Multiplicity
Direction: In				
Name (Mandatory)	Refers to the name that is used to create the named version.	string	Any specified name for create named version	1
MasterCatalog (Optional)	Refers to repository name.	long or string	Any existing Repository ID or Name.	0..1
InDocument(Mandatory)	Refers to mIXML document. See Common Parameters.	Document	mIXML document	1
NamedVersionTimestamp (Optional)	Refers to the Date string using which named version is created. You must specify yyyy-MM-dd HH:mm:ss format for the Date string. For example, 2011-03-29 16:45:40	string	Default value is current date and time.	0..1

Example for CreateNamedVersion Activity

```
<Activity Name="CreateNamedVersion">
<Action>CreateNamedVersion</Action>
<Description>Create named version after DB dump is successful</Description>
<Execution>SYNCHR</Execution>
<Parameter direction="in" name="InDocument" type="document"
eval="variable">inDoc</Parameter>
<Parameter name="Name" direction="in" eval="variable"
type="string">EvaluationName</Parameter>
<Parameter name="NamedVersionTimestamp" direction="in" eval="variable"
type="string">EvaluationTimestamp</Parameter>
</Activity>
```

CreateOutputFile Activity

CreateOutputFile activity generates an output file containing record data in different formats for a list of processed records.

This activity works on the records from a repository. If the repository has relationships with other repositories, any related records from other repositories are also processed. If any classification schemes are associated with the output maps used for synchronization, the records classification data file is also generated.

A zipped output is generated containing the following files:

- If the ExecutionMode parameter is DBDump, records from the specified repository are exported without any transformation. Otherwise, records are exported after transforming them to the output format using selected output maps. Multiple output files might be generated for record data from different repositories depending on the used output maps.

  Record data output file name format is as follows:

  OUTPUT_*SynchronizationFormatName_timestamp*.txt

- One relationship file containing the record relationship information.

  Relationship output file name format is as follows:

  Relationship_*timestamp*.txt

- One or more relationship attribute files are generated, depending on the number of relationships involved to output the relationship attributes data for the record

relationships.

Relationship attribute data file name format is as follows:

RelationshipAttribute_*RelationshipName_timestamp*.txt

- One or more classification output files might be generated containing records classification data, if the records are classified into classification schemes associated with the output maps used for synchronization.

  Classification output file name format is as follows:

  RepositoryName_ClassificationSchemeName_timestamp.txt

- One index file containing information about generated output files.

  Index file name format is as follows:

  Index_*timestamp*.txt

CreateOutputFile Parameters and Valid Execution Modes

The valid execution mode for CreateOutputFile is SYNCHR.

The parameters of CreateOutputFile are as follows:

CreateOutputFile Parameters

Name	Semantics	Type	Valid Values	Multi-plicity
Direction: In				
AsynProcessIndicator (Optional)	See Common Parameters.			
BundlePerAsyncCall (Optional)	See Common Parameters.			
ExecutionMode (Optional)	Specifies the execution mode, that is, DBDump or any other mode.	string	DBDump or Any String	1

Name	Semantics	Type	Valid Values	Multi-plicity
	The execution mode DBDump is used to get a dump of a specific repository without applying any repository format on it. That is, the database values are extracted and presented in an output file with column names as headers. The default mode is to use the catalog format to output data. You can specify any value other than DBDump in that case.			
InDocument (Mandatory)	See Common Parameters.			
InRecordList (Mandatory)	See Common Parameters.			
FEDOption (Optional)	See Common Parameters.			
ReferenceStepID (Optional)	Reference Process Log ID. Required only if ExecutionMode is not equal to DBDump.	long	Any number.	1
Direction: Out				
OutDocument(Optional)	See Common Parameters.			
RecordsProcessed (Optional)	Number of Records written to the file.	long	Any number.	1

Example for CreateOutputFile Activity

```
<Activity Name="CreateOutputFile">
 <Start/>
 <Action>CreateOutputFile</Action>
 <Description lang="en">Write output in file.</Description>
 <Parameter direction="in" type="string" eval="constant"
 name="eventState">GENERATEFILE</Parameter>
 <Parameter direction="in" name="InDocument" type="document"
 eval="variable">inDoc</Parameter>
 <Parameter direction="in" name="InRecordList" type="recordlist"
 eval="variable">workRecordList</Parameter>
 <Parameter direction="out" name="OutDocument" type="document"
 eval="variable">fileDoc</Parameter>
 <Parameter direction="in" name="BundlePerAsyncCall" type="long"
 eval="constant">10</Parameter>
 <Parameter direction="in" name="AsynProcessIndicator" type="boolean"
 eval="constant">true</Parameter>
</Activity>
```

CreateWorkItem Activity

This activity creates a work item in the specified user's inbox. The created work items can be either notifications (ASYNCHR) or action items (SYNCHR).

- When ASYNCHR is specified, a work item is created. The workflow continues its execution.
- When SYNCHR is specified, workflow execution suspends and waits for a response from the user.

For workflows that have more than one participant (role or member, or a combination of both), the semantics are currently as follows:

- When work items are assigned to roles, all users assigned to such roles receive work items.
- The number of work items that must be submitted to mark as the end of a step, can be configured in the workflow.
- If the work item definition includes the MailPresentation and MailType tags, all participants are notified.

When a work item is created, you can:

- Add the custom attributes to a work item.
- Add keywords to tag a work item.

CreateWorkItem Parameters and Valid Execution Modes

The valid execution mode for CreateWorkItem is SYNCHR or ASYNCHR.

The parameters of CreateWorkItem are as follows:

CreateWorkItem Parameters

Name	Semantics	Type	Valid Values	Multi-plicity
Direction: In				
CUSTOM_*name*(Optional)	You can save any arbitrary value into work item details. These values can be used for reports, or in customization of inbox descriptions. Name is any arbitrary name.	string		0..n
CUSTOMDisplayServlet (Optional)	Forwards request to custom servlet if TaskType is CATALOGMERGERECORD. This parameter is used only for customization.	string	Name of the custom servlet that is defined in the Web.xml file.	0..1
Error severity(Optional)	See Common Parameters.			
ExpiryType(Optional)	Type of expiry date. Time is specified as DD:hh:mm:ss. for relative and as MM:DD:hh:mm:ss for absolute. Relativeloop is an extension of relative in which timeouts are attempted several times. COMPUTE is a generic way to calculate work item expiry date based on record attributes. The Rulebase parameter must be defined; it is used to compute the target expiry date. The actual expiry date is `expiry date = Launch Date( output from rulebase ) - RemindBeforeNumberOfDays` Based on the value of ReminderNumber (reminders generated so far), the 'Reminder Setup' rule can be configured to get values of RemindBeforeNumberOfDays and email addresses to send the reminder email to. Rules must be set such that for each Reminder number, only one value for RemindBeforeNumberOfDays would be returned (although any number of email addresses can be returned). When a record is edited or a work item revivified, expiry date is recomputed and updated into the work item. If the recompute expiry date differs from the old expiry date, the Reminder number is set to zero.	string	RELATIVE RELATIVELOOP ABSOLUTE COMPUTE	0..n
ExpiryDate(Optional)	Relative or absolute time when this work item is no longer valid. If such a time occurs	string	See ExpiryType.	0..1

Name	Semantics	Type	Valid Values	Multi-plicity
	without the work item being closed, the workflow is revived with a TIMEDOUT event which can be polled in the outgoing transitions of this activity. The work item timeout is computed at regular intervals when MqRevivify runs. Depending upon the process interval time of MqRevivify, the work item timeout might happen after the specified duration has expired. For example: expiryDate=24 hrs, MqRevivify interval=30 min, the timeout might take place after 24 hrs 30 mins.			
Form (Optional)	The filename of a work item form. Use the filename for passing parameters between the workflow and the UI and for specifying how to render the fields of the document (readonly and readwrite).	string	A file on disk. In general, form files reside in MQ_COMMON_DIR/<standard or enterprisename>/form	1
FormPresentation(Mandatory)	The fully qualified classname of an XMLC generated HTML class file to use for presenting the document.	string	A valid class name (can be either custom or predefined).	1
MailPresentation(Optional)	The name of an XMLC file used as the body of an email to send to a work item participant for notification. The XMLC page can have an anchor element called inboxUrl. The href attribute of an email sets to the address of the inboxUrl as specified in the Configurator.	string	A class file generated using XMLC.	0..1
ReassignMailPresentation (Optional)	The name of an XMLC file used as the body of an email to send reassignment email to a work item participant for notification.	string	A class file generated using XMLC.	0..1
DelegationPresentation (Optional)	The name of an XMLC file used as the body of an email to send delegation email to a work item participant for notification.	string	A class file generated using XMLC.	0..1
MailType(Optional)	The type of mail to send. If LINK is specified, a link to the inbox is sent. In future, other types (such as sending the entire HTML form) will be available.	string	LINK	0..1
InDocument(Mandatory)	See Common Parameters.			
InformationThreshold(Optional)	Threshold used to hide/show information messages generated during rulebase validation.	long	0 - 9. Default value is 1, show all information messages with level greater than 1.	0..n

Name	Semantics	Type	Valid Values	Multi-plicity
Intent(Optional)	Intent of the work item.	string	Any String. Out-of-workflow defines Edit, Approval, Correction, Associate TradingPartner and Conflict Resolution.	0..n
MaxNumberOfTimeouts (Optional)	Number of times the work item can timeout.	long	Any whole number, if not specified, ONE is assumed. This parameter is applicable for RELATIVELOOP and COMPUTE only.	0..n
ParticipantID(Mandatory)	Specify the person or role that must participate in this work item. If ParticipantType is ROLE, this parameter corresponds to a RoleID. If ParticipantType is MEMBER, ParticipantID is a MemberID.	long	Any MemberID or RoleID as appropriate.	1..n
ParticipantType(Optional)	The type of participant. If you do not specify the value for this parameter, by default MEMBER is considered.	string	When statically specified, the type can be MEMBER or ROLE, and the value must be a valid member or role in the supplier's organization.	1..n
RecordAttributeName orRecordAttributeName# (Optional)	Name of the record attribute which must be extracted from the records associated with the work item. **Attention:** This option is kept for backward compatibility. With the 9.1.0 release or later, if you have selected **Searchable** and **Display in Record List** options while creating attributes, those attributes are automatically configured to be associated with work items.	string	The name of the attribute for which value must be extracted for Inbox search. You must specify the value in upper case. For example, <Parameter direction="in" eval="constant" type="string" name="RecordAttributeName">ATTRIBUTE1</Parameter> <Parameter direction="in" eval="constant" type="string" name="RecordAttributeName1"> ATTRIBUTE2</Parameter>	0..5
RemindBeforeNumberOfDays (Optional)	Number of days before the expiry a reminder be generated.	long	Any whole number, if not specified, ONE is assumed. This parameter is applicable for COMPUTE only.	0..n
Rulebase(Optional)	See Common Parameters. Rulebase to be used for computation of expiry when method is COMPUTE.	string	Rulebase for computation of expiry. Workflow fails if the rulebase parameter is not specified.	0..n
SkipPrevOwners(Optional)	Flag to indicate that skip creating another work item for users who have already had a work item in this event.	string	**All** to indicate that work items must not be created for users who had at least one work item in this event. **Last** to indicate skipping work items for the those users who had work items in the last work item step. If this parameter is not specified, no skipping is done.	1

Name	Semantics	Type	Valid Values	Multi-plicity
ValidateUsingRuleBase (Optional)	Flag for rulebase evaluation by the activity (validations are stored with processlogid and associated with work item). If this flag is disabled, rulebase evaluation is done on the UI.	boolean	True (default) False.	
WarningThreshold(Optional)	Threshold used to hide/show warning messages generated during rulebase validation.	long	0-99 Default value is 99, show all warning messages.	0..n

Keyword or Keyword#	Tag work items by adding any number of keywords.	string	The value of a keyword. For example,
			`<Parameter name="Keyword" direction="in"eval="constant"type="String" >Priority_Critical</Parameter>`
			`<Parameter name="Keyword1" direction="in"eval="constant"type="String" >AutoFocus</Parameter>`
				0..n

PerspectiveName (Optional)	Specify the perspective name defined in a repository. • If the perspective name is defined, the record bundle is loaded with a subset of relationships that is defined in the perspective. This improves the bundle loading performance. • If you specify the PerspectiveName parameter, the specified perspective must exist in the specified MasterCatalog parameter or MasterCatalog present in an input document of the activity, otherwise an error is displayed. • The perspective name specified in the activity gets precedence. • If the perspective name is not specified in the activity, it is checked in the mlXML document. • If you specify RelationshipName as well as the PerspectiveName in the activity, the perspective name gets precedence. **Note:** If the MasterCatalog parameter is not specified, its value is retrieved from the mlXML document.	string	Any valid perspective name. For example: `<Parameter direction="in" name="PerspectiveName" type="string" eval="constant" >perspective_address</Parameter>` **Note:** • Multiple perspective names are not allowed. • If you specify an incorrect perspective name, an error is displayed.	0..1
Direction: Out				
FailedDelegations(Optional)	The number of work items which could not be delegated when delegation was	long		0..1

Name	Semantics	Type	Valid Values	Multi-plicity
	indicated in user profiles.			
InitialParticipantNumber (Optional)	Count of the participants which were initially considered for creation of work items. The parameter is part of the output only when the number of participants for which work items are created are not same as the number of participants passed to the activity. This can happen in case of delegation, use of roles, and invalid users.	long		0..1
MergedDocument	A copy of InDocument with workitem data merged from the workitem form result document, which is generated when a workitem is submitted.	document	Any document	1
NumberCreated(Optional)	The number of work items created.	long		0..1
OutDocument(Optional)	See Common Parameters.			
ParticipantName(Optional)	Names of participants for whom work items were created.	string		0..1
ReminderNumber(Optional)	Number of reminders generated so far.	long		0..1
TimedOutParticipantID (Optional)	The person/memberID for which a timeout or reminder was generated.	long		0..1
UsersSelected(Optional)	Names of the participants initially selected for creation of work items.	string		0..1
OutRecordList	This parameter is applicable only when a work item is created for mass update. Contains the record list for approved records in the mass update work item.	recordlist	output records	
OutRecordList2	This parameter is applicable only when a work item is created for mass update. Contains the record list for approved records in the mass update work item.	recordlist	output records	

Example for CreateWorkItem Activity

The following example evaluates participants based on a Business Process Rule, Import Approval, and creates work items in the Inbox of those users.

The work item expires after one day, in case the user does not approve or reject it within that time.

```
<Activity Name="ImportApprovalWorkitem">
 <Description lang="en">Create a workitem for import approval</Description>
      <Execution>SYNCHR</Executionn>
      <Parameter
direction="in" type="string" eval="constant" name="eventState">IMPORTAPPROVALWORKITEM</Parameter>
 <Parameter direction="in" name="Intent" eval="constant" type="string">Approval</Parameter>
      <Parameter direction="in" name="ReferenceStepID" eval="variable" type="long">pl2</Parameter>
      <Parameter direction="in" eval="rule" source="Import
Approval" type="long"   name="ParticipantID">workDoc</Parameter>
 <Parameter direction="in" eval="rule" source="Master Catalog Import
Approval" type="string" name="ParticipantType">workDoc</Parameter>
      <Parameter direction="in" eval="constant" type="string" name="FormPresentation">
 com.tibco.mdm.ui.workflow.engine.workitem.templates.CatalogImportApproval</Parameter>
      <Parameter direction="in" eval="constant" type="string" name="TaskType">CATALOGIMPORTRECORD</Parameter>
      <Parameter direction="in" eval="variable" type="document" name="InDocument">workDoc</Parameter>
      <Parameter direction="in" eval="constant" type="string" name="ExpiryType">RELATIVE</Parameter>
```

```
        <Parameter direction="in" eval="constant" type="string" name="ExpiryDate">1:0:0:0</Parameter>
        <Parameter direction="out" eval="variable" type="document" name="OutDocument">wiDoc</Parameter>
        <Parameter direction="out" eval="variable" type="boolean" name="WorkItemCreated">approvalFlag</Parameter>
        <Parameter direction="out" eval="variable" type="boolean" name="MergedDocument">workDoc</Parameter>
        <Parameter direction="out" name="StepID" eval="variable" type="long">pl1</Parameter>
    </Activity>
```

Based on the output from activity, loop through the ImportApprovalWorkitem activity till all work items are approved or one of them is rejected.

```
<Transition FromActivity="ImportApprovalWorkitem" ToActivity="ImportApprovalWorkitem">
  <Description lang="en">Test that all created Import Master Catalog approval workitems have been completed or someone has rejected  workitem.</Description>
  <Rule>
    <Parameter name="wicreated" type="boolean" eval="variable" direction="in">approvalFlag</Parameter>
    <Parameter name="numberCreated" type="long" eval="variable" direction="in">NumberCreated</Parameter>
    <Parameter name="numberCompleted" type="long" eval="variable" direction="in">NumberCompleted</Parameter>
    <Parameter name="response" type="string" eval="variable" direction="in">FormResult</Parameter>
    <Parameter name="result" type="boolean" direction="out" />
    <Condition format="bsh">
```

```
        <![CDATA[

                result = ((wicreated) &&  (numberCompleted < numberCreated) &&  (!((response != null) &&
(response.equalsIgnoreCase("Reject")))));

                System.out.println("Transition ImportMasterCatalogInternalApprovalWorkitem to
ImportMasterCatalogInternalApprovalWorkitem result - " + result + " Created = " + numberCreated + " Completed = " +
numberCompleted + " Response = " + response);

    </Condition>

  </Rule>

</Transition>
```

Example for CreateWorkItem Activity for Hierarchy Approval

The following example evaluates participants based on the Hierarchy Create Approval business process rule and creates hierarchy approval work items for creating hierarchy in the Inbox of those users.

```
<Activity Name="HierarchyMgmtCreateApproval">
<Action>CreateWorkItem</Action>
<Description lang="en">Create workitem for approval of the creation of Hierarchy</Description>
<Execution>SYNCHR</Execution>
<Parameter direction="in" eval="constant" type="string" name="Intent">Approval</Parameter>
<Parameter direction="in" type="string" eval="constant"
name="eventState">HIERARCHYCREATEAPPROVAL</Parameter>
<Parameter direction="in" eval="rule" source="Hierarchy Create Approval" type="long"
name="ParticipantID">workDoc</Parameter>
<Parameter direction="in" eval="rule" source="Hierarchy Create Approval" type="long"
name="ParticipantType">workDoc</Parameter>
<Parameter direction="in" eval="constant" type="string" name="FormPresentation">
com.tibco.mdm.ui.workflow.engine.workitem.templates.HierarchyEditApprovalSummary
</Parameter>
<Parameter direction="in" eval="constant" type="string"
name="Form">standard/forms/fm26ca.xml</Parameter>
<Parameter direction="in" eval="constant" type="string" name="MailPresentation">
com.tibco.mdm.ui.workflow.engine.emailtemplates.EmailHierarchyAddEditWorkItem
</Parameter>
<Parameter direction="in" eval="constant" type="string"
name="TaskType">HIERARCHYAPPROVAL</Parameter>
<Parameter direction="in" eval="variable" type="document"
name="InDocument">workDoc</Parameter>
<Parameter direction="out" eval="variable" type="document"
name="OutDocument">wiDoc</Parameter>
<Parameter direction="out" eval="variable" type="boolean"
name="WorkItemCreated">approvalFlag</Parameter>
<Parameter direction="out" eval="variable" type="boolean"
name="MergedDocument">workDoc</Parameter>
<Parameter direction="out" name="StepID" eval="variable" type="long">pl1</Parameter>
<Parameter direction="in" eval="constant" type="string"
name="ExpiryType">RELATIVE</Parameter>
<Parameter direction="in" eval="constant" type="string" name="ExpiryDate">1:0:0:0</Parameter>
</Activity>
```

The following example evaluates participants based on the Hierarchy Edit Approval business process rule and creates hierarchy approval work items for modify hierarchy and for split and merge operations in the Inbox of those users.

```
<Activity Name="HierarchyMgmtEditApproval">
<Action>CreateWorkItem</Action>
<Description lang="en">Create workitem for approval of the modification of
Hierarchy</Description>
<Execution>SYNCHR</Execution>
<Parameter direction="in" eval="constant" type="string" name="Intent">Approval</Parameter>
<Parameter direction="in" type="string" eval="constant"
name="eventState">HIERARCHYEDITAPPROVAL</Parameter>
<Parameter direction="in" eval="rule" source="Hierarchy Edit Approval" type="long"
name="ParticipantID">workDoc</Parameter>
<Parameter direction="in" eval="constant" type="string"
name="FormPresentation">com.tibco.mdm.ui.workflow.engine.
workitem.templates.HierarchyEditApprovalSummary</Parameter>
<Parameter direction="in" eval="constant" type="string"
name="Form">standard/forms/fm26ca.xml</Parameter>
<Parameter direction="in" eval="constant" type="string"
name="MailPresentation">com.tibco.mdm.ui.
workflow.engine.emailtemplates.EmailHierarchyAddEditWorkItem</Parameter>
<Parameter direction="in" eval="constant" type="string"
name="TaskType">HIERARCHYAPPROVAL</Parameter>
<Parameter direction="in" eval="variable" type="document"
name="InDocument">workDoc</Parameter>
<Parameter direction="out" eval="variable" type="document"
name="OutDocument">wiDoc</Parameter>
<Parameter direction="out" eval="variable" type="boolean"
name="WorkItemCreated">approvalFlag</Parameter>
<Parameter direction="out" eval="variable" type="boolean"
name="MergedDocument">workDoc</Parameter>
<Parameter direction="out" name="StepID" eval="variable" type="long">pl1</Parameter>
<Parameter direction="in" eval="constant" type="string"
name="ExpiryType">RELATIVE</Parameter>
<Parameter direction="in" eval="constant" type="string" name="ExpiryDate">1:0:0:0</Parameter>
</Activity>
```

The following example evaluates participants based on the Hierarchy Link Approval business process rule and creates hierarchy approval work items for the link and unlink operation in the Inbox of those users.

```
<Activity Name="HierarchyMgmtLinkApproval">
<Action>CreateWorkItem</Action>
<Description lang="en">Create workitem for approval of the hierarchy to record link
```

```
operations</Description>
<Execution>SYNCHR</Execution>
<Parameter direction="in" eval="constant" type="string" name="Intent">Approval</Parameter>
<Parameter direction="in" type="string" eval="constant"
name="eventState">HIERARCHYLINKAPPROVAL</Parameter>
<Parameter direction="in" eval="rule" source="Hierarchy Link Approval" type="long"
name="ParticipantID">workDoc</Parameter>
<Parameter direction="in" eval="constant" type="string"
name="FormPresentation">com.tibco.mdm.ui.
workflow.engine.workitem.templates.HierarchyEditApprovalSummary</Parameter>
<Parameter direction="in" eval="constant" type="string"
name="Form">standard/forms/fm26ca.xml</Parameter>
<Parameter direction="in" eval="constant" type="string"
name="MailPresentation">com.tibco.mdm.ui.
workflow.engine.emailtemplates.EmailHierarchyAddEditWorkItem</Parameter>
<Parameter direction="in" eval="constant" type="string"
name="TaskType">HIERARCHYAPPROVAL</Parameter>
<Parameter direction="in" eval="variable" type="document"
name="InDocument">workDoc</Parameter>
<Parameter direction="out" eval="variable" type="document"
name="OutDocument">wiDoc</Parameter>
<Parameter direction="out" eval="variable" type="boolean"
name="WorkItemCreated">approvalFlag</Parameter>
<Parameter direction="out" eval="variable" type="boolean"
name="MergedDocument">workDoc</Parameter>
<Parameter direction="out" name="StepID" eval="variable" type="long">pl1</Parameter>
<Parameter direction="in" eval="constant" type="string"
name="ExpiryType">RELATIVE</Parameter>
<Parameter direction="in" eval="constant" type="string" name="ExpiryDate">1:0:0:0</Parameter>
</Activity>
```

The following example evaluates participants based on the Hierarchy Delete Approval business process rule and creates hierarchy approval work items for the delete hierarchy in the Inbox of those users.

```
<Activity Name="HierarchyMgmtDeleteApproval">
<Action>CreateWorkItem</Action>
<Description lang="en">Create workitem for approval of the deletion of Hierarchy</Description>
<Execution>SYNCHR</Execution>
<Parameter direction="in" type="string" eval="constant"
name="eventState">HIERARCHYDELETEAPPROVAL</Parameter>
<Parameter direction="in" name="Intent" eval="constant" type="string">Notify</Parameter>
<Parameter direction="in" eval="rule" source="Hierarchy Delete Approval" type="long"
name="ParticipantID">workDoc</Parameter>
<Parameter direction="in" eval="constant" type="string"
```

```
name="FormPresentation">com.tibco.mdm.ui.workflow.
engine.workitem.templates.HierarchyDeleteApprovalSummary</Parameter>
<Parameter direction="in" eval="constant" type="string"
name="Form">standard/forms/fm26ca.xml</Parameter>
<Parameter direction="in" eval="constant" type="string"
name="MailPresentation">com.tibco.mdm.ui.workflow.
engine.emailtemplates.EmailHierarchyDeleteWorkItem</Parameter>
<Parameter direction="in" eval="constant" type="string"
name="TaskType">HIERARCHYAPPROVAL</Parameter>
<Parameter direction="in" eval="variable" type="document"
name="InDocument">workDoc</Parameter>
<Parameter direction="out" eval="variable" type="document"
name="OutDocument">wiDoc</Parameter>
<Parameter direction="out" eval="variable" type="boolean"
name="WorkItemCreated">approvalFlag</Parameter>
<Parameter direction="out" eval="variable" type="boolean"
name="MergedDocument">workDoc</Parameter>
<Parameter direction="out" name="StepID" eval="variable" type="long">pl1</Parameter>
<Parameter direction="in" eval="constant" type="string"
name="ExpiryType">RELATIVE</Parameter>
<Parameter direction="in" eval="constant" type="string" name="ExpiryDate">1:0:0:0</Parameter>
</Activity>
```

How to Assign Default Presentation Parameters

You can assign default presentation parameters using CONFIGURATIONDEFINITION and WORKFLOWFORM seed data.

To assign defaults, the FormPresentation parameter must be present in the WORKFLOWFORM table. The inputs of this parameter are MailPresentation, ReassignMailPresentation, and DelegationMailPresentation. These parameters are optional. If not specified, values are defaulted using the WORKFLOWFORM table

- The default FormPresentation parameter is retrieved from WORKFLOWFORM.FORMID corresponding to the FormPresentation entry. The file referred by FORMID must exist in the CONFIGURATIONDEFINTION table.
- The default MailPresentation parameter is retrieved from the MAILTEMPLATEID column.
- The default ReassignMailPresentation parameter is retrieved from the REASSIGNMAILTEMPLATEID column.
- The default DelegationMailPresentation parameter is retrieved from the

DELEGMAILTEMPLATEID column.

The MAILTEMPLATEID, REASSIGNMAILTEMPLATEID, and DELEGMAILTEMPLATEID refer to entries stored in the WORKFLOWFORM table itself.

Note that the Form is defaulted during work item creation and the value is copied over. Hence, any changes to default does not affect the existing work items. However, for other inputs (mail templates), any changes to defaults affect existing work items as well.

Note: For all out of the box work item forms, seed data is populated to retain backward compatibility with previous releases.

Changing Seed Data to Customize the Forms and Templates

You can change seed data to customize the forms and templates.

Procedure

1. For a custom FormPresentation, add an entry in the WORKFLOWFORM table. While adding an entry, you must consider the following points:
   - Note that the type = TASK.
   - Ensure that the ID is unique and is more than maximum IDs assigned to any entry and it is more than 10000.
   - Ensure that entry corresponding to 'FormPresentation' points to custom mails and form.

     For example,

     ```
     INSERT INTO WORKFLOWFORM ( ID, VERSION, OWNERORGANIZATIONID,
     NAME, DEFINITION, ACTIVE, MODMEMBERID,MODDATE, MODVERSION, TYPE,
     ACTIONABLE, reassignmailtemplateid, formid, delegmailtemplateid, mailTemplateID,
     )
     VALUES (22000, 1, 1, 'Mass update approval',
     'com.tibco.mdm.ui.gi.JSXAPPS.MassupdateWorkItem', 'Y', 101,
     TO_Date('03-15-2010 01:32:30 PM', 'MM/DD/YYYY HH:MI:SS AM'), 1, TASK , 'Y',
     21000, 21001, 21002, 21003);
     ```
2. For a new Form, add an entry in the CONFIGURATIONDEFINITION table. While

adding an entry, ensure that the name and ID are unique.

For example,

```
INSERT INTO CONFIGURATIONDEFINITION(ID, "TYPE", OWNERID, GLOBAL, "NAME",
SELECTOR, DESCRIPTION, DEFINITIONTYPE, DEFINITION, ACTIVE, MODMEMBERID,
MODDATE, MODVERSION)
VALUES ('33', 'FORM', '1', 'Y', 'fmretailer.xml', 'WORKFLOW', 'Form for retailer workitems',
'File', 'standard/forms/fmretailer.xml', 'Y', '1',
TO_DATE('15-03-2010 04:01:00 pm','DD-MM-YYYY HH:MI:SS AM'), '1');
```

3. For a new email template, add an entry in the WORFLOWFORM table. While adding an entry, you must consider the following points:
   - Ensure the ID and name are unique.
   - Ensure that the ID is more than maximum IDs assigned to any entry and it is more than 10000.

     For example,

     ```
     INSERT INTO WORKFLOWFORM ( ID, VERSION, OWNERORGANIZATIONID,
     NAME, DEFINITION, ACTIVE,
     MODMEMBERID,MODDATE, MODVERSION, TYPE, ACTIONABLE )
     VALUES ( 21000, 1, 1, 'Email for xxx',

     'com.tibco.mdm.ui.workflow.engine.emailitemplates.EmailWorkItemDelegationNotifica
     tion', 'Y', 101,
     TO_Date('03-15-2010 01:32:30 PM', 'MM/DD/YYYY HH:MI:SS AM'), 1, 'EMAIL' , 'Y');
     ```

     To change the FORMID or any of the email templates assigned to the existing entries, update the existing entries in the WORKFLOWFORM table.

     Note:
     - After the entries are changed, you must clear the cache as the updated data is cached.
     - If default forms and email templates are changed, existing work items are impacted.

DeleteRecord Activity

Use the DeleteRecord activity to delete one or more records specified in the input.

As part of the import workflow, the ImportDataDeleteRecord parameter in this activity applies for deletion of records. When importing records, this activity is invoked sequentially after the ImportCatalogRecords activity and is followed by the ExtractRelationship Activity activity.

DeleteRecord Parameters and Valid Execution Modes

The valid execution mode for DeleteRecord is SYNCHR or ASYNCHR.

The parameters of DeleteRecord are as follows:

DeleteRecord Parameters

Name	Semantics	Type	Valid Values	Multi-plicity
Direction: In				
IncludeDraftRecords (Optional)	See Common Parameters.			
InDocument(Optional)	See Common Parameters. One of the following parameters must be specified: ProductIds, RecordKey, InRecordList or InDocument.			
InRecordList(Optional)	See Common Parameters. One of the following parameters must be specified: ProductIds, RecordKey, InRecordList or InDocument.			
MasterCatalog(Optional)	Specify this parameter, if InDocument is specified and InRecordList, ProductIDs, RecordKeys are not specified. The activity tries to get the repository name in the following order: From InParameter. From InDocument. An exception is thrown and the workflow exits in the following conditions: • repository name is not found in both InParameter and InDocument. • repository name does not resolve to a valid repository.	string	Any string.	0..1
PerspectiveName (Optional)	See CreateWorkItem Parameters and Valid Execution Modes.			
ProductExtns(Optional)	Indicates:	arraylist or string	A valid value for extension as an array or string.	0..n

Name	Semantics	Type	Valid Values	Multi-plicity
	• Product extensions of the records to be processed (Arraylist). For example: <Parameter direction="in" name="ProductExtns" type="arraylist" eval="variable" >ProductIDKeyArray1</Parameter> • Individual product extension (string). For example: <Parameter direction="in" name="ProductExtns" type="string" eval="constant">w33</Parameter> This input is used only if ProductIds is specified. If an arraylist is specified, the number of entries in the array must be the same as the number of entries in productIds.		An array is processed only if the input for productId is Array.	
ProductIds(Optional)	This is a required parameter only when you want to process the record from the input. You must specify either this parameter or the RecordKey parameter. If both are specified, recordKey is processed. If none of them are specified, the activity fails. One of the following parameters must be specified: ProductIds, RecordKey, InRecordList or InDocument. This parameter indicates: • Product IDs of the records to be processed (arraylist). For example: <Parameter direction="in" name="ProductIds" type="arraylist" eval="variable" >ProductIDKeyArray1</Parameter> • Individual product ID (string). For example: <Parameter direction="in" name="ProductIds" type="string" eval="constant">w33</Parameter>	arraylist or string	Valid ProductIDs for the records that exist in the system.	0..1
RecordKey(Optional)	You must specify either this parameter or the ProductIds parameter. If both are specified, recordKey is processed. If none of them are specified, the activity fails. One of the following parameters must be specified: ProductIds, RecordKey, InRecordList or InDocument.	arraylist	List of record keys for the records to be deleted.	0..1
RelationshipName (Optional)	See Common Parameters. If relationship names are specified, the activity tries to build the bundle and process it. That is, the bundle of the specified relationship is built and each record in the bundle is marked for deletion.	string	ALL or a valid relationship name applicable for the repository.	0..n

Name	Semantics	Type	Valid Values	Multi-plicity
	This parameter accepts ALL as a valid input. If ALL is specified, all the relationships applied to the catalog (including forward and reverse) are considered for deletion.			
VersionOption(Optional)	See Common Parameters.			0..1
ImportDataDeleteRecord	This parameter applies when the DeleteRecord activity is invoked by ImportCatalogRecords activity as part of the import process. This parameter is used to delete records during import.	boolean	True False	0..1
Direction: Out				
RecordsDeleted(Optional)	This parameter contains the total number of records deleted by this activity.	long		1
RecordsProcessed (Optional)	This parameter contains the total number of records processed by this activity.	long		1

Example for DeleteRecord Activity

```
<Activity Name="DeleteRecord">
 <Start/>
 <Action>DeleteRecord</Action>
 <Description lang="en">Delete records</Description>
 <Parameter direction="in" type="string" eval="constant"
 name="eventState">DELETERECORDS</Parameter>
 <Parameter direction="in" name="InDocument" type="document"   eval="variable"> inDoc
</Parameter>
 <Parameter direction="in" name="RecordKey" type="arraylist"
 eval="variable">ProductKeyArray</Parameter>
 <Parameter direction="in" name="MasterCatalog" eval="constant"
 type="String">XYZ</Parameter>
 <Parameter direction="in" name="VersionOption" type="string"
 eval="constant">LATEST</Parameter>
 <Parameter direction="out" eval="variable" type="long"
 name="RecordsProcessed">RecordsProcessed</Parameter>
 <Parameter direction="out" eval="variable" type="long"
 name="RecordsDeleted">RecordsDeleted</Parameter>
 <Parameter direction="in" name="RelationshipName" type="string"
 eval="constant">Contains</Parameter>
 <Parameter direction="in" name="IncludeDraftRecords" type="boolean"
 eval="constant">false</Parameter>
</Activity>
```

Example when DeleteRecord is invoked as part of the import process.

```
<Activity Name="DeleteRecord">
 <Action>DeleteRecord</Action>
 <Description lang="en">Delete records for import</Description>
 <Parameter direction="in" type="string" eval="constant"
 name="eventState">IMPORTDELETEPROD</Parameter>
 <Parameter direction="in" name="ImportDataDeleteRecord" type="boolean"
 eval="constant">true</Parameter>
 <Parameter direction="in" eval="xpath" type="string" name="MasterCatalog"
 source="/Message/Body/Document/BusinessDocument/CatalogAction/CatalogAc
 tionHeader/MasterCatalog/RevisionID/BaseName/text()">inDoc</Parameter>
 <!--Parameter direction="in" eval="constant" type="long"
 name="RecordPerAsyncCall">10</Parameter-->
 <!--Parameter direction="in" name="AsynProcessIndicator" type="boolean"
 eval="constant">true</Parameter-->
</Activity>
```

Rules for Processing DeleteRecord Activity

This activity processes only one of the input parameters to identify the records. If more than one input record identification information is provided, the first input parameter which is not null is processed and other inputs are ignored.

The order of precedence for record identification is as follows:

1. Product IDs
2. Record Keys
3. RecordCollection
4. Indocument

- When productId and RecordKey mode is specified, any related records are also deleted. Related records are searched based on relationships specified. If no relationship names are specified, no related records are deleted. To delete all related records, specify RelationshipName as ALL.
- When RecordCollection is used to select records, the relationships defined when record collection was created are used to identify related records. Relationship names, if specified, are ignored.
- When inDocument is used to select records, only records included in indocument are deleted. No other records are processed. Relationship names, if specified, are ignored.
- If RecordCollection is specified, IncludeDraftRecords is ignored. Record versions are automatically identified using RecordCollection. For all other inputs, IncludeDraftRecords influences the version to be deleted. When this parameter is set to true, if there is any valid draft version associated with the event, it is deleted. If this flag is set to false, record versions are selected based on the VersionOption parameter. The default is true.
- VersionOption influences the record versions selected for deletion. If VersionOption is set to LATEST, the last confirmed or unconfirmed version is selected. Note that if includeDraftRecords is set to true, instead of last confirmed and unconfirmed version, any valid draft might be selected. If versionOption is set CONFIRMED, only the last confirmed and undeleted version is selected. Again, if includeDraftRecords is set to true, instead of the last confirmed version, any valid draft might be selected. The default value is CONFIRMED.

- If the VersionOption flag is not specified, the default value of includeUnconfirmed record is true. If includeUnconfirmed record is false, only confirmed records are used.

DuplicateMessageCheck Activity

Use the DuplicateMessageCheck activity to verify duplicate messages and avoid their processing.

To accomplish this, the activity uses a PayloadID. Any message that contains a PayloadID, that is, unique within the receiving organization is considered ‘new’. If the PayloadID provided to the activity was previously received, the activity sets the IsDuplicate parameter to true. However, if the Fatal parameter is set to true, the activity throws an Exception - an ‘error’ transition can be configured to handle this.

DuplicateMessageCheck Parameters and Valid Execution Modes

The valid execution mode for DuplicateMessageCheck is SYNCHR.

The parameters of DuplicateMessageCheck are as follows:

DuplicateMessageCheck Parameters

Name	Semantics	Type	Valid Values	Multiplicity
Direction: In				
PayloadID(Mandatory)	The unique PayloadID of the message.	string		1
MarketOrganizationID (Mandatory)	The organization ID of the marketplace/sender.	string		1
Fatal(Optional)	Causes the activity to	boolean	True	0..1

Name	Semantics	Type	Valid Values	Multiplicity
	throw an exception if set to “True” and a duplicate document is found. The absence of this tag defaults to “True”.		False	
InDocument (Mandatory)	See Common Parameters.			
Direction: Out				
IsDuplicate(Optional)	If Fatal is set to “False”, this parameter is set to “True” if the document is indeed a duplicate, “False”, otherwise.	boolean	True False	0..1

Example for DuplicateMessageCheck Activity

In this example, you can find the PayloadID by using an XPath expression on doc1, and the market organization.

If a duplicate was found, an exception is thrown as Fatal is True.

```
<Activity Name="DuplicateMessageCheck">
  <Start/>
  <Description>Check if this document was received before</Description>
  <Execution>SYNCHR</Execution>
  <Action>DuplicateMessageCheck</Action>
  <Parameter direction="in" type="boolean" eval="constant"   name="Fatal">True</Parameter>
  <Parameter direction="in" type="string" eval="xpath" name="PayloadID"
  source="//mlXML/@payloadID">doc1</Parameter>
  <Parameter direction="in" type="string" eval="xpath"   name="MarketOrganizationID"
  source="//mlXML/Header/HeaderDetails[@HeaderOrigin='Sender']/Organization/  @dbid"> doc1
</Parameter>
```

```
 <Parameter direction="in" type="document" eval="constant"
 name="InDocument">doc1</Parameter>
</Activity>
```

EvaluateRuleBase Activity

The EvaluateRuleBase activity lets you run an XML rulebase. The rulebase can validate records, cleanse and populate data in the records, and return evaluated data to the workflow.

Primarily, you can use the following two modes:

- Process one or more records specified by input parameters.
- Run once. This allows you to execute any validation, task or function once for all records. This is indicated by specifying ExecutionMode as NoRecords. The parameter ExecutionMode lets you select the Run once mode. You can execute a custom function, or any SQL based rulebase which operates on all the records in one execution.

Note: To use the Connect action with the EvaluateRulebase activity, you must set the following parameters in the ManageRecordCollection Activity activity:

- Specify the RelationshipName. The name must match with the name of a relationship specified in the Connect action of a rulebase.
- Set the value of the BundlingOption parameter to true.

  For example,

```
<Parameter direction="in" name="RelationshipName" type="string"
eval="constant">CustomerToAddress</Parameter>
<Parameter direction="in" name="BundlingOption" type="boolean"
eval="constant">true</Parameter>
```

EvaluateRuleBase Parameters and Valid Execution Modes

The valid execution mode for EvaluateRuleBase is SYNCHR.

The parameters of EvaluateRuleBase are as follows:

EvaluateRuleBase Parameters

Name	Semantics	Type	Valid Values	Multi-plicity
Direction: In				
AllowDuplicateInOutput (Optional)	Configures whether to allow duplicates in output variables.	boolean	True (Default) False	0..1
AllowDuplicateInReturn (Optional)	Configures whether to allow duplicates in the ResultArray.	boolean	True False (Default)	0..1
AsynProcessIndicator (Optional)	See Common Parameters.			
BundlePerAsyncCall (Optional)	See Common Parameters.			
CatalogValidation (Optional)	Whether to execute the catalog's catalogvalidation.xml file. Only works if the Rulebase parameter is not specified.	boolean	True False (default)	0..1
ErrorSeverity(Optional)	See Common Parameters.			
EvaluateChildren (Optional)	Controls the rulebase evaluation on child records. By default (true), the rulebase is evaluated on child records present in the bundle. If this parameter is set to false, the rulebase is applied on the root record and not on the child record. **Limitation**: This parameter only works if a record list is passed and not when a document is passed.	boolean	True (default) False	0..1
ExecutionMode(Optional)	Lets you select the Run once mode. You can execute a custom function, or any SQL based rulebase which operates on all the records in one execution.	string	NoRecords AllRecords (default) NoRecords means that	0..1

Name	Semantics	Type	Valid Values	Multi-plicity
			the rulebase is executed only once without considering any records.	
<InputNames>(Optional)	Any values can be passed to the activity. These values are available to the rulebase. The name of the variable must match the declaration in the rulebase: <declare usage=”input”>	string	Must be declared as input variables in the rulebase.	0..n
IncludeDeletedRecords (Optional)	When this parameter is set to true, deleted records are also processed.	boolean	True False (default).	0..1
InDocument(Optional)	See Common Parameters.	document		1
InRecordList	See Common Parameters.			
LogOption(Optional)	Determines what information is logged and how it is to be logged. You can combine the options as per your requirement. For example, if you want errors to be reported in a file, specify FE.	string	A - Errors are stored in an AttributeLog database table. E - Only errors are reported (warnings and information messages are not reported). F - Errors are reported in a file.	0..1
OverwriteRelationships (Optional)	If specified as True and relationship names are passed, the relationships of RecordCollection are overwritten.	Boolean	True False (Default)	1
PerspectiveName (Optional)	See CreateWorkItem Parameters and Valid Execution Modes.			
RecordPerAsyncCall (Optional)	See Common Parameters.			
RelationshipName (Optional)	See Common Parameters. When processing rootrecord, the recordBundle is created for the rootRecord with all relationships as defined in the			

Name	Semantics	Type	Valid Values	Multi-plicity
	Database and the rulebase is applied on all the records in the bundle. If a rulebase is to be applied on a bundle with specific relationships, the name of the relationship can be specified in the workflow file using this parameter. A record bundle is then created for the specified relationshipName only. The order of configuration is: Workflow file > Configuration file > Default value.			
RemoveRecord(Optional)	Whether to remove a record from the RecordList if an error occurs. If the parameter is set to FATAL, records with fatal errors are removed (as specified by Severity). This option is applicable only if InRecordList is present.	string	NONE FATAL (default)	0..1
Rulebase(Optional)	See Common Parameters.			
SaveFlag(Optional)	Whether to save the product if transformations are done.	string	SAVE NOSAVE	0..1
Severity#(Optional)	Additional Severity cut-offs. Start at 1. Replace # with an integer, for example, Severity1, Severity2. Severity1 Severity2 ... Severity9	long	0 - 99.	0-9
StandardValidation (Optional)	Whether to perform Standard Validations in addition to those specified in the Rulebase.	Boolean	True False (default).	1
ValidationExclusion (Optional)	Indicates the relationships to be excluded while applying validations on record bundle.	String	Default value is null.	
VersionPolicy (Optional)	See ImportCatalogRecords Parameters and Valid Execution Modes.			
Direction: Out				
OutDocument(Optional)	See Common Parameters.			

Name	Semantics	Type	Valid Values	Multi-plicity
ValidationErrors(Optional)	Number of Validation errors encountered when checking all products with severity <= errorSeverity.	long	N/A	1
ValidationErrors# (Optional)	Count in Severity Buckets. Replace # with an integer, for example, ValidationError1, ValidationError2. ValidationError1 = Severity <s<= Severity1, ValidationError2 = Severity1 <s<= Severity2, … Validation(n) = SeverityN-1<s<=SeverityN Validation(Last) = s > Severity(Last-1) Number of ValidationError counts will be one more than the Severity specifications, to allow for last (greater than) entry.	long	N/A	0-9
ResultArray(Optional)	Array of Return values from the Rulebase. Values for each Record is concatenated. By default, duplicates are NOT allowed unless AllowDuplicateInReturn is set to True.	arraylist	N/A	0..n
<OutputName>(Optional)	Output Variables from the Rulebase. Name must match the <declare usage=”output”> tag in Rulebase. By default, duplicates are allowed unless AllowDuplicateInOutput is set to “False”.	arraylist	Must be declared as “output” variables in the Rulebase.	1
RecordsProcessed (Optional)	Count of Number of Records processed by the activity.	long		1
OutRecordList(Optional)	See Common Parameters.			
OutRecordList2(Optional)	List of records that failed validation.	recordlist	Records that fail validation.	1
StaleDataError(Optional)	The StaleDataError is thrown from concurrent record updates by multiple users or workflow threads. If this parameter is specified and if the StaleDataError occurs, this parameter is set to true and the workflow continues. If this parameter is not specified and there is an error, an exception is thrown. To prevent the StaleDataError that might occur due to concurrent modifications done through UI or web services, select true	boolean	True False (Default)	1

Name	Semantics	Type	Valid Values	Multi-plicity
	for the Enable Save Lock (com.tibco.cim.record.useSaveLock) property in the Configurator. The property enables locks to avoid concurrent modification of parent and its child record. Additionally, you can specify the time period for which you can wait to save the record that is concurrently being modified by another user. Specify the time period for the Specify Timeout property in the Configurator. By default, the time period is 300 seconds.			

Example for EvaluateRuleBase Activity

```
<Activity Name="ApplySynchRuleBase">
 <Start/>
 <Action>EvaluateRuleBase</Action>
 <Description lang="en">Apply Validation and transformation  rules.</Description>
 <Parameter direction="in" type="string" eval="constant"
 name="eventState">EvaluateRuleBase</Parameter>
 <Parameter direction="in" name="Rulebase" eval="catalog" type="string"
 source="TransformRuleBase">inDoc</Parameter>
 <Parameter direction="in" name="InDocument" type="document"
 eval="variable">inDoc</Parameter>
 <Parameter direction="in" name="InRecordList" type="recordlist"
 eval="variable">workRecordList</Parameter>
 <Parameter direction="in" name="EvaluateChildren" type="boolean"
 eval="constant">false</Parameter>
 <!-- Severity: Validations with severity < input Severity are considered  Fatal errors. The rest are
considered Warnings. -->
 <Parameter direction="in" name="Severity" type="long"  eval="constant">9</Parameter>
 <!-- SaveFlag: NONE - Do NOT remove records with errors.  FATAL - Remove  records with Fatal
errors (see Severity) -->
 <Parameter direction="in" name="RemoveRecord" type="string"
 eval="constant">FATAL</Parameter>
 <!-- SaveFlag indicates if any changes to attributes should be saved in the  database. Values are:
SAVE,NOSAVE -->
 <Parameter direction="in" name="SaveFlag" type="string"  eval="constant">SAVE</Parameter>
 <!-- LogOption: A - AttributeLog, L - Execution Log, F - Log File -->
 <Parameter direction="in" name="LogOption" type="string"  eval="constant">F</Parameter>
 <!-- Number of FATAL errors (see Severity) -->
 <Parameter direction="out" name="ValidationErrors" type="long"
 eval="variable">fatalErrors</Parameter>
 <!-- Number of Warnings (see Severity) -->
 <Parameter direction="out" name="ValidationErrors1" type="long"
 eval="variable">warningErrors</Parameter>
 <Parameter direction="in" name="BundlePerAsyncCall" type="long"
 eval="constant">10</Parameter>
 <Parameter direction="in" name="RelationshipName" type="string"
 eval="constant">CONTAINS</Parameter>
</Activity>
```

EvaluateMassUpdateRulebase Activity

The EvaluateMassUpdateRulebase activity is a wrapper activity around the EvaluateRulebase activity. This activity retrieves rulebase input parameters and values from the mIXML inDocument.

This activity contains all the parameters of the EvaluateRuleBase Activity activity and in addition includes one input and one output parameter.

EvaluateMassUpdateRulebase Parameters and Valid Execution Modes

The valid execution mode for EvaluateMassUpdateRulebase is SYNCHR.

The parameters of EvaluateMassUpdateRulebase are as follows:

EvaluateMassUpdateRulebase Parameters

Name	Semantics	Type	Valid Values	Multiplicity
Direction: In				
status	Specifies the status of the new record version.	string	Draft.	0..1
Direction: Out				
UpdatedRecordsCount	Lists the number of records transformed by the EvaluateMassupdateRulebase activity.	long	Any	0..1

Example for EvaluateMassUpdateRulebase Activity

```
<Activity Name="ApplyMassupdateRulebase">
<Action>EvaluateMassupdateRulebase</Action>
<Description>Apply mass update rulebase</Description>
<Execution>SYNCHR</Execution>
<Parameter name="eventState" direction="in" eval="constant"
type="string">EVALUATERULEBASE</Parameter>
<Parameter name="InDocument" direction="in" eval="variable"
type="document">inDoc</Parameter>
<Parameter name="Rulebase" direction="in" eval="xpath"
source="//CatalogActionHeader/MassUpdateRulebase/@name" type="string">inDoc</Parameter>
<Parameter name="InRecordList" direction="in" eval="variable"
type="recordlist">workRecordList</Parameter>
<Parameter name="Severity" direction="in" eval="constant" type="long">9</Parameter>
<Parameter name="RemoveRecord" direction="in" eval="constant"
type="string">FATAL</Parameter>
<Parameter name="SaveFlag" direction="in" eval="constant" type="string">SAVE</Parameter>
<Parameter name="LogOption" direction="in" eval="constant" type="string">F</Parameter>
<Parameter name="Status" direction="in" eval="constant" type="string">DRAFT</Parameter>
<Parameter name="ValidationErrors" direction="out" eval="variable"
type="long">fatalErrors</Parameter>
<Parameter name="ValidationErrors1" direction="out" eval="variable"
type="long">warningErrors</Parameter>
<Parameter name="OutRecordList" direction="out" eval="variable"
type="recordlist">workRecordList</Parameter>
<Parameter name="UpdatedRecordsCount" direction="out" eval="variable"
type="long">UpdatedRecordsCount</Parameter>
</Activity>
```

EvaluateSubset Activity

The EvaluateSubset activity applies a Subset Rule on previously selected records to create a smaller set. The activity can be used along with a catalog or repository. When used with a Catalog, the Subset Rule to be applied is identified when the catalog is defined.

When used with a repository, the Subset Rule name is supplied as input to the activity, and you can have more than one Subset Rule applied one after another.

The EvaluateSubset activity supports incremental evaluation of subset. This activity accepts a named version prefix and verifies the last duration when the subset was executed. If incremental is true, the EvaluateSubset activity returns added or updated records after the last evaluation.

- If data is exported for a complete repository, Named Version name is formed as Prefix%.
- If data is exported for a subset, Named Version name is formed as Prefix_ SubsetID%.

  In both cases, Prefix refers to the name given to the named version and % refers to timestamp.

EvaluateSubset Parameters and Valid Execution Modes

The valid execution mode for EvaluateSubset is SYNCHR.

The parameters of EvaluateSubset are as follows:

EvaluateSubset Parameters

Name	Semantics	Type	Valid Values	Multi-plicity
Direction: In				
InDocument(Mandatory)	See Common Parameters.			
RelationshipName# (Optional)	See Common Parameters.			
Subset(Optional)	Subset Rule name.	string	Name of the Subset Rule to be applied. Must be specified for only non sync events (i.e events which are not for Catalog Synchronization). If specified, the Subset Rule name takes priority over catalog definition.	0..1
NamedVersionPrefix (Optional)	Prefix used while creating the named version. This parameter is used in case of incremental export.	string	Any string	0..1
IgnoreMetadataChange	Ignores the change in repository metadata. In case of incremental export, if IgnoreMetadataChange is set to false and repository metadata changes, all data is exported even though the same data is exported earlier.	boolean	True (Default) False	0..1
MasterCatalog(Optional)	See Common Parameters.			
InRecordList(Optional)	Record list.	recordlist	Record list. If specified, the Subset Rule is applied on the record list. If no record collection is specified, Subset Rule is applied for all records in the repository. Also note that if no Subset Rule is applicable (that is, no Subset Rule name is specified or the event is for catalog synchronization and no Subset Rule is selected in catalog definition), all records in the repository are selected.	0..1
PerspectiveName (Optional)	See CreateWorkItem Parameters and Valid Execution Modes.			

Name	Semantics	Type	Valid Values	Multi-plicity
VersionOption(Optional)	Record version selection option.	string	CONFIRMED (Default) LATEST Specifies the record version to be selected. If there are any draft records in the workflow, they are included automatically. This parameter is applicable only if the inrecord collection is null. For use with Catalog, it is a good practice to not override the versionOption specified in the catalog definition.	0..1
FEDOption(Optional)	See Common Parameters.			
FEDDate(Optional)	Refers to the date used to process the records between current date and future date (FED date). The FED date is always greater than the current date.	string	The valid value is date in format **yyyy-MM-dd HH:mm:ss.S**. If no date is specified, then the default (dummy) date is "3999-12-31".	0..1
CutoffDate	Set to filter records after a certain date.	string	The valid value is date specified in the following date format: yyyy-MM-dd HH:mm:ss. For example, 2020-07-17 17:50:00.	0..1
Direction: Out				
EvaluationTimestamp	Refers to the timestamp at which the subset is evaluated. The timestamp format is yyyy-MM-dd hh:mm:ss. This timestamp is passed to the CreateNamedVersion activity in the Incremental Export workflow. The timestamp is generated only if the NamedVersionPrefix is specified.	string	The valid value is date and time. The default value is current date and time.	0..1
EvaluationName	Refers to the name of NamedVersion, which is used to verify the last export date or time. This EvaluationName is passed to the CreateNamedVersion activity in the Incremental Export workflow. The EvaluationName is generated only if the NamedVersionPrefix is specified.	string	Any valid name of the named version.	0..1
OutRecordList(Optional)	See Common Parameters.			
RecordsProcessed (Optional)	Number of records in outRecordList.	long		1

Example for EvaluateSubset Activity

The example shows typical use. In this example, the Subset Rule is used from the catalog, that is, no Subset Rule name is specified.

Note: versionOption is selected from the catalog.

```
<Activity Name="EvaluateSubset">
 <Start/>
 <Action>EvaluateSubset</Action>
 <Description lang="en">Apply Subset.</Description>
 <Parameter direction="in" type="string" eval="constant"  name="eventState">SUBSET</Parameter>
 <Parameter direction="in" name="InDocument" type="document"  eval="variable">inDoc</Parameter>
 <Parameter
direction="in" name="MasterCatalog" type="string" eval="xpath"  source="//CatalogActionHeader/CatalogReference
[@type='MasterCatalog']/Revi  sionID/DBID/text()">inDoc</Parameter>
 <Parameter direction="in" name="RelationshipName" type="string"  eval="constant">Contains</Parameter>
 <Parameter direction="in" name="RelationshipName1" type="string"  eval="constant">ContainedBy</Parameter>
 <Parameter
direction="in" name="VersionOption" type="string" eval="catalog"  source="VersionOption">inDoc</Parameter>
 <Parameter name="CutoffDate" direction="in" eval="constant" type="string">2020-07-17 17:50:00</Parameter>
 <!-- If you want to cascade subsetting, specify master catalog, subset and  record collection -->
```

```
<!--Parameter direction="in" name="Subset" type="string"   eval="constant">MYSUBSET</Parameter-->
<Parameter name="NamedVersionPrefix" direction="in" eval="constant"   type="string">DBDUMP</Parameter>
<!--<Parameter name="IgnoreMetadataChange" direction="in" eval="constant"   type="boolean">false</Parameter>-->
<!--Parameter direction="in" name="MasterCatalog" type="string"   eval="constant">MYCATALOG</Parameter-->
<Parameter direction="out" name="OutRecordList" type="recordlist"   eval="variable">workRecordList</Parameter>
<Parameter direction="out" name=" RecordsProcessed" type="long"   eval="variable">recordsProcessed</Parameter>
</Activity>
```

ExecuteExternalCommand Activity

By using the ExecuteExternalCommand activity, you can carry out a number of shell operations such as copy, JAR, mkdir, and so on. Internally, ExecuteExternalCommand uses Jakarta Ant commands, so the specification of that tool provides a full description of all the valid commands for this activity.

ExecuteExternalCommand uses Java reflection to send values to an Ant command, and to actually execute it. It is the responsibility of the workflow designer to supply the correct attribute names and values to ExecuteExternalCommand in the parameter tags.

ExecuteExternalCommand Parameters and Valid Execution Modes

The valid execution mode for ExecuteExternalCommand is SYNCHR.

The parameters of ExecuteExternalCommand are as follows:

ExecuteExternalCommand Parameters

Name	Semantics	Type	Valid Values	Multiplicity
Direction: In				
Command (Mandatory)	Specify the name of the ant command to call in all lowercase letters.	string	See Legal ANT Commands list.	1

Depending on the command, the ExecuteExternalCommand activity needs other parameters as well. These parameters are of type property and have the names of methods available to that Ant command.

Legal ANT Commands

The list is not exhaustive. See the ANT manual for all available commands.

Legal ANT Commands

Command	Description
copy	Copy a file from one directory to another.
copydir	Copy a complete directory to another location.
delete	Delete a file.
gzip	Gzip a file.
get	Get a URL from the web.
tar	Tar a set of files from a directory.

Parameters for Copy Command

The “To” parameters are evaluated in the order that follows, and the results are concatenated to make the full path. Although they are individually optional, there is an error if no valid directory or filename is constructed.

Parameters for Copy command

Name	Semantics	Type	Valid Values	Multi-plicity
Direction: In				
Command (Mandatory)	Specify the name of the Ant command to call in all lowercase letters.	string	copy getfile	1
File(Optional)	The full path of the source file to copy.	string	A path to a file.	1
Name(Optional)	Name.	string	Any string	1
Todir(Optional)	Directory path. If none is specified, use the directory of the input file.	string	Directory Name.	1
Tofile(Optional)	Filename segment. If not there, use input filename.	string	Legal filename string.	1
TofileExt (Optional)	An optional extension to add to the destination file. A “.” is automatically added between the filename and the extension.	string	Legal filename string.	1
TofilePrefix (Optional)	Filename segment. Ignored if not there.	string	Legal filename string.	1
TofileSuffix (Optional)	Filename segment. Ignored if not there.	string	Legal filename string.	1

Name	Semantics	Type	Valid Values	Multi-plicity
Torelative (Optional)	Environment variable.	string	By interpreting the path as an environment variable, it allows for using directories mounted on PCs and UNIX workstations, or mounted using different names on different machines.	1
Direction: Out				
OutDocument (Optional)	See Common Parameters.			

Example for Copy Command

This example can be used to copy an mlXML document designated by doc0 to /temp/.

```
<Activity Name="CopyFile"><Start/>
  <Action>ExecuteExternalCommand</Action>
  <Execution>SYNCHR</Execution>
  <Parameter direction="in" eval="constant" type="string"    name="Command">copy</Parameter>
  <Parameter direction="in" eval="property" type="string"    name="File">doc0</Parameter>
  <Parameter direction="in" eval="constant" type="string"    name="Torelative">MQ_COMMON_
DIR</Parameter>
  <Parameter direction="in" eval="constant" type="string"
   name="Todir">PO/outgoing/</Parameter>
  <Parameter direction="in" eval="constant" type="string"
   name="TofilePrefix">Order</Parameter>
  <Parameter direction="in" eval="constant" type="string"    name="TofileExt">855</Parameter>
  <Parameter direction="out" name="OutDocument"   type="document">OutputDoc</Parameter>
</Activity>
```

Parameters for GetFile Command

The GetFile command retrieves a file so that it can be referenced by the workflow engine.

Parameters for GetFile command

Name	Semantics	Type	Valid Values	Multi-plicity
Direction: In				
Command	Specify the name of the Ant command to call in all lowercase letters.	string	copy getfile	1
Todir	Directory path. If none is specified, use the directory of the input file.	string	Directory Name.	1
Tofile	Filename segment. If not there, use input filename.	string	Legal filename string.	1
TofileExt (Optional)	An optional extension to add to the destination file. A “.” is automatically added between the filename and the extension.	string	Legal filename string.	1
TofilePrefix (Optional)	Filename segment. Ignore if not there.	string	Legal filename string.	1
TofileSuffix (Optional)	Filename segment. Ignored if not there.	string	Legal filename string.	1
Torelative (Optional)	Environment variable.	string	By interpreting the path as an environment variable, it allows for using directories mounted on PCs and UNIX	1

Name	Semantics	Type	Valid Values	Multi-plicity
			workstations, or mounted using different names on different machines.	
Direction: Out				
OutDocument (Optional)	See Common Parameters.			

Example for GetFile command

Use this example to copy an mlXML document designated by doc0 to /temp/.

```
<Activity Name="CopyFile"><Start/>
  <Action>ExecuteExternalCommand</Action>
  <Execution>SYNCHR</Execution>
  <Parameter direction="in" eval="constant" type="string"   name="Command">copy</Parameter>
  <Parameter direction="in" eval="property" type="string"   name="File">doc0</Parameter>
  <Parameter direction="in" eval="constant" type="string"   name="Torelative">MQ_COMMON_
DIR</Parameter>
  <Parameter direction="in" eval="constant" type="string"
   name="Todir">PO/outgoing/</Parameter>
  <Parameter direction="in" eval="constant" type="string"
   name="TofilePrefix">Order</Parameter>
  <Parameter direction="in" eval="constant" type="string"   name="TofileExt">855</Parameter>
</Activity>
```

ExtractDataToDelimitedFile Activity

This activity marks record bundles and extracts data to a file in the delimited format.

The file contains a list of processed records.

- Marking record bundles: This activity works on a record collection. If records in the record collection contain relationships with other repositories, the related records from other repositories are also processed.

- Extracting data to a file: A zipped output is generated containing the following files:
    - Repository data file for each repository. If multi-value attributes are defined on a repository, the repository data file also contains multi-value data.
    - Relationship file containing record relationship information
    - Relationship attribute file for each relationship

ExtractDataToDelimitedFile Parameters

The parameters of ExtractDataToDelimitedFile are as follows:

ExtractDataToDelimitedFile Parameters

Name	Semantics	Type	Valid Values	Multiplicity
Direction: In				
RelationshipDepth	The depth of related records that are exported.	long	Any integer Default value is 10.	0..1
ExportAttributes (Optional)	Attribute names that are exported.	string	Default value is all. Valid value is displayable attributes.	0..1
Delimiter	The delimiter for data in a file. The delimiter must be a single character. In case of multi-character string, the first character is used as a delimiter.	string	Default value is a (,) comma.	0..1
TextQualifier	The qualifier used to differentiate the field data from the delimiter.	string	Any valid text qualifier. Default value is (") double quotes.	0..1
DataFileExtension	An extension of the exported data files.	string	Any valid file extension. Default value is CSV.	0..1
DateFormat	The date format that is used to format date type attributes. Only valid Java date formats are supported. For valid date formats, see the "SimpleDateFormat" section on the Oracle documentation site.	string	Default value is the user date format. If the user date format is empty, the yyyy-MM-dd format is used.	0..1
DropTempTables	Specifies whether to drop temporary tables generated by the activity.	boolean	True and False. Default value is true. If true is specified, the temporary tables are dropped.	0..1
BackwardCompatible	Specifies whether the generated output is backward compatible. If you want 8.0 backward compatible output, specify the value to true.	boolean	True and False. Default value is false. If you specify true, the generated output is backward	0..1

Name	Semantics	Type	Valid Values	Multiplicity
			compatible with 8.0.	
InRecordList	A record list that contains root records.	recordlist	Any valid record list. No default value.	1
Direction: Out				
OutDocument (Optional)	Refers to the zipped file that contains exported data. See Common Parameters.			

Note: The ExtractDataToDelimitedFile activity also uses the VersionOption and RelationshipName parameters of the EvaluateSubset activity. These parameters are associated with the record collection that determine record version and related records to be exported. For more information, see EvaluateSubset Activity.

Example for ExtractDataToDelimitedFile Activity

```
<Activity Name="ExtractDB">
 <Action>ExtractDataToDelimitedFile</Action>
  <Description lang="en">Mark related records and write output in     file</Description>
  <Parameter direction="in" type="string" eval="constant"
   name="eventState">GENERATEFILE</Parameter>
  <Parameter direction="in" type="long" eval="constant"
   name="RelationshipDepth">10</Parameter>
  <Parameter direction="in" type="string" eval="constant"     name="Delimiter">,</Parameter>
  <Parameter direction="in" type="string" eval="constant"     name="TextQualifier">"</Parameter>
  <Parameter direction="in" type="string" eval="constant"
   name="DataFileExtension">csv</Parameter>
  <Parameter direction="in" type="string" eval="constant"     name="DateFormat">yyyy-MM-
dd</Parameter>
  <Parameter direction="in" type="boolean" eval="constant"
   name="DropTempTables">true</Parameter>
  <Parameter direction="in" name="InDocument" type="document"
   eval="variable">inDoc</Parameter>
  <Parameter direction="in" name="InRecordList" type="recordlist"
   eval="variable">workRecordList</Parameter>
  <Parameter direction="out" name="OutDocument" type="document"
   eval="variable">fileDoc</Parameter>
</Activity>
```

ExtractRelationship Activity

The ExtractRelationship activity is invoked as part of the import workflow, to extract and process relationships for records being imported.

The ExtractRelationship activity is always called during import and if there are no relationships to be processed, it proceeds to the next activity.

The ExtractRelationship activity is followed by the PurgeStaging Activity.

ExtractRelationship Parameters and Valid Execution Modes

The valid execution mode for ExtractRelationship is SYNCHR or ASYNCHR.

The parameters of ExtractRelationship are as follows:

ExtractRelationship Parameters

Name	Semantics	Type	Valid Values	
Direction: In				
eventState	Determines the latest state of the workflow.	string	Predefined state: DONE. Custom states can be defined in the DOMAINENTRY table.	0..1
MasterCatalog	See Common Parameters.	string		
RecordPerAsyncCall	See Common Parameters.	long		
ImportRelationshipsOnly	Identifies the Import type used in different activities during the workflow execution. For more information, see the section, "Viewing Input Maps" in *TIBCO MDM User's Guide*.	string	An XPATH expression to get this value from the InDocument.	1
DeleteRelationshipsOnly (Optional)	Deletes the relationships between two records. For more information, see the section,	boolean	True False (Default)	0..1

Name	Semantics	Type	Valid Values	
	"Viewing Input Maps" in *TIBCO MDM User's Guide*.			
IsLoggingRequired	Enables the additional logging for all imported relationships. If you set the true value for this parameter, a log file is generated in the $MQ_COMMON_DIR/Temp directory. You can also download the log file from the Event Details page. It is displayed in the Additional Data section.	boolean	True False (Default)	1

Example for ExtractRelationship Activity

```
<Activity Name="ExtractRelationship">
  <Action>ExtractRelationship</Action>
  <Description lang="en">Process record relationship for import</Description>
  <Parameter direction="in" type="string" eval="constant"
  name="eventState">IMPORTRELATION</Parameter>
  <Parameter direction="in" eval="xpath" type="string" name="MasterCatalog"
```

```
 source="/Message/Body/Document/BusinessDocument/CatalogAction/  CatalogActionHeader/M
asterCatalog/RevisionID/BaseName/text()">inDoc  </Parameter>
 <Parameter direction="in" eval="xpath" type="string"   name="ImportRelationshipsOnly"
 source="/Message/Body/Document/BusinessDocument/CatalogAction/CatalogActio  nHeader/M
asterCatalog/Extension[@name='ImportRelationshipsOnly']/Value/text  ()">inDoc</Parameter>
 <!--Parameter direction="in" eval="constant" type="long"
 name="RecordPerAsyncCall">10</Parameter-->
 <!--Parameter direction="in" name="AsynProcessIndicator" type="boolean"
 eval="constant">true</Parameter-->
 <Parameter direction="in" name="IsLoggingRequired" type="boolean"
 eval="constant">true</Parameter>
</Activity>
```

FireWorkflow Activity

Using the FireWorkflow activity, a named workflow can be started. The activity interface is similar to the subflow activity, but the workflow fired is a separate event.

The FireWorkflow activity is different from the SpawnWorkflow activity because it does not iterate over records.

The FireWorkflow activity processes inputs, passes them to a new workflow, and then fires the workflow. The fired workflow runs as an independent workflow and does not return any output to the caller. The following are the main features of the FireWorkflow activity:

- Bypasses workflow selection business rule
- Supports in-memory workflow execution
- Does not support sequencing, that is, no registration order or registration keys
- Supports failover and restart

  Failover might generate duplicate events. Failover is handled as follows:

  - Before firing the FireWorkflow activity, an implicit checkpoint is added to ensure that the workflow is committed before the activity. For more information about the checkpoint workflow, see CheckpointWorkflow Activity.
  - The configuration property Save state before sending workflow message (com.tibco.cim.optimization.process.savebeforesend) is added in the Configurator. If you set it to true, the workflow state that is created before sending the message is persisted to the database. Else, the workflow state is persisted

before the workflow execution starts, based on the in-memory status of a workflow.

- Failover marker is created before the workflow initiation and set it as completed.
- On re-executing the workflow after failover.

FireWorkflow Parameters and Valid Execution Modes

The valid execution mode for FireWorkflow is ASYNCHR.

The parameters of FireWorkflow are as follows:

FireWorkflow Parameters

Name	Semantics	Type	Valid Values	Multiplicity
Direction: In				
LinkParentEvent (Optional)	Establishes link to a parent event. You can link the new event to the caller event to track the status of an event through Event Log (Associated Events).	boolean	True (Default) False	0..1
ProcessID (Mandatory)	The name of the workflow to execute. If specified, the workflow is retrieved from the specified location. The location must be specified relative to MQ_COMMON_DIR.	string	Any workflow name relative to MQ_COMMON_DIR.	1
PartitioningKey	Specify to ensure that	string	Any attribute	0..n

Name	Semantics	Type	Valid Values	Multiplicity
	the new data created is correctly partitioned. For more information, see the Application Partitioning chapter of *TIBCO MDM System Administration*.		of INTEGER data type.	
WorkflowPriority (Optional)	See Common Parameters. If the priority is not specified, event priority is inherited by the new message.			
InDocument (Mandatory)	See Common Parameters.			
Direction: Out				
InitiatedEventID	An ID of the event fired.	long		

GetAssocEventsSummary Activity

The GetAssocEventsSummary activity provides a summary of associated events. Associated events are the events spawned by another workflow.

You can use this activity in

- Parent workflow to obtain summary of all spawned children events or for example, import workflow can inquire summary of all children workflow. This is the SummaryOption of "Children".
- Any children workflow to get a summary of all spawned children by the parent workflow (siblings). For example, when import starts record add workflow, this activity can be used in record add workflow to find summary of all the events

started by the parent (import workflow). This is the default SummaryOption of 'Parent'.

As workflow status can change while summary is being computed, if the summary is going to be used to decide if some other operation must be done, it is advisable to request a lock using LockOption parameter. If this parameter is not specified, default is FALSE - no lock is taken. The 'true' parameter forces a distributed lock (The lock Key is parent ID for SummaryOption = Parent, and current event id for SummaryOption = Children). Lock is recommended for SummaryOption = Parent but rarely needed for SummaryOption = Parent).

The activity must not be used with SummaryOption = Parent for an event which does not have a parent event. This is an error.

Note that when it runs with SummaryOption = Children, the summary includes all the spawned events initiated in all the previous spawn activities. (there could be more than one spawn activity). whereas when it runs in SummaryOption = Parent, the summary is only for the spawned activity which started the children workflow.

GetAssocEventsSummary Parameters and Valid Execution Modes

The valid execution mode for GetAssocEventsSummary is SYNCHR.

The parameters of GetAssocEventsSummary are as follows:

GetAssocEventsSummary Parameters

Name	Semantics	Type	Valid Values	Multiplicity
Direction: In				
SummaryOption (Optional)	Get the summary of events	string	Parent and Children. Default is Parent.	1
LockOption	Whether a	Boolean	True and	1

Name	Semantics	Type	Valid Values	Multiplicity
	distributed lock be obtained before computing summary.		False Default is False.	
Direction: Out				
TotalEventsProcessed	How many events have been processed so far.	Long	Count	1
EventsToBeSpawned	How many events are to be spawned or actually spawned. Note that it is possible that all the events might not have been spawned yet.	Long	Count	1
EventsInError	How many events have been in error state.	Long	Count	1
EventsSuccessful	How many events have been successful.	Long	Count	1
EventsInProgress	How many events have been in progress state.	Long	Count	1
EventsCancelled	How many events have been canceled.	Long	Count	1

Name	Semantics	Type	Valid Values	Multiplicity
EventsYetToStart	How many events are yet to start.	Long	Count	1
EventID	The ID of the event for which summary was computed. If the activity is called from a children event with SummaryOption = Children, this is the ID of the parent event.	Long	Valid event ID	1

Example for GetAssocEventsSummary Activity

```
<Activity Name="GetEventSummary">
  <Action>GetAssocEventsSummary</Action>
  <Description lang="en">Get the summary of events</Description>
 <!--Parameter direction="in" type="boolean" eval="constant"
name="LockOption">true</Parameter-->
 <Parameter direction="out" name="EventsToBeSpawned" type="long"
eval="variable">totalEventsToBeSpawned</Parameter>
 <Parameter direction="out" name="TotalEventsProcessed" type="long"
eval="variable">totalEvents</Parameter>
 <Parameter direction="out" name="EventsSuccessful" type="long"
eval="variable">eventsSuccessful</Parameter>
 <Parameter direction="out" name="EventsInError" type="long"
eval="variable">eventsInError</Parameter>
 <Parameter direction="out" name="EventsCancelled" type="long"
eval="variable">eventsCancelled</Parameter>
 <Parameter direction="out" name="EventID" type="long"
eval="variable">parentEventID</Parameter>
</Activity>
```

GetHierarchy Activity

The GetHierarchy activity retrieves the hierarchy object from the input document in the workflow. The activity requires HIERARCHY_ID and (optionally) HIERARCHY_ VERSION parameters. If the HIERARCHY_VERSION parameter is not present, the activity loads the latest CONFIRMED or published hierarchy by using the specified HIERARCHY_ID. Additionally, the GetHierarchy activity retrieves HIERARCHY_ ACTION parameter from the input document and adds it to the workflow state.

GetHierarchy Parameters and Valid Execution Modes

The valid execution mode for GetHierarchy is SYNCHR.

The parameters of GetHierarchy are as follows:

GetHierarchy Parameters

Name	Semantics	Type	Valid Values	Multiplicity
Direction: In				
InDocument (Mandatory)	See Common Parameters.	Document	N/A	N/A
eventState	See Common Parameters.	N/A	GETHIERARCHY	0..1
HIERARCHY_ID (Mandatory)	A unique ID for the hierarchy	Long	An XPath expression to retrieve this value from the InDocument	1
HIERARCHY_ VERSION (Optional)	Refers to the version of the root node in hierarchy	Long	An XPath expression to retrieve this value from the InDocument	0..1
Direction: Out				

Name	Semantics	Type	Valid Values	Multiplicity
HIERARCHY	Display name of the hierarchy	String	A valid display name of the hierarchy	1
OutDocument	See Common Parameters.	Document	N/A	N/A

Example for GetHierarchy Activity

This example returns the hierarchy objects (HIERARCHY ID and VERSION) from the input document specified in the workflow.

```
<Activity Name="GetHierarchyData">
<Action>GetHierarchy</Action>
<Description lang="en">Get hierarchy data</Description>
<Parameter direction="in" name="InDocument" type="document"
eval="variable">workDoc</Parameter>
<Parameter direction="in" type="string" eval="constant"
name="eventState">GETHIERARCHY</Parameter>
<Parameter direction="in" eval="xpath" type="long" name="HIERARCHY_ID"
source="/Message/Body/Document/BusinessDocument/HierarchyAction[last
()]/HierarchyActionHeader/Hierarchy/RevisionID/DBID/text()">inDoc</Parameter>
<Parameter direction="in" eval="xpath" type="long" name="HIERARCHY_VERSION"
source="/Message/Body/Document/BusinessDocument/HierarchyAction[last
()]/HierarchyActionHeader/Hierarchy/RevisionID/Version/text()">inDoc</Parameter>          <!--
Hierarchy (display) Name -->
<Parameter direction="out" eval="variable" type="string"
name="HIERARCHY">HIERARCHY</Parameter>
<Parameter direction="out" eval="variable" type="document"
name="OutDocument">workDoc</Parameter>
</Activity>
```

GetSyncStatus Activity

The GetSyncStatus activity retrieves the sync status of records, and returns an arraylist of RecordKey, ChannelCredential, TradingPartnerCredential, SyncStatus, StatusGroup, ChannelID, and TradingPartnerID.

inRecordlist and statusGroup are required parameters; if any of the two are missing, the workflow terminates with an error.

If the trading partner is not found, in case the record is registered, a -1 is populated in out trading Partnerlist.

The activity can be used in a workflow to enquire about record status. If the record status cannot be found, the outrecordlist gets populated.

Typically, this activity is used to spawn workflows to multiple trading partners.

GetSyncStatus Parameters and Valid Execution Modes

The valid execution mode for GetSyncStatus is SYNCHR.

The parameters of GetSyncStatus are as follows:

GetSyncStatus Parameters

Name	Semantics	Type	Valid Values	Multi-plicity
Direction: In				
InRecordList (Mandatory)	If the parameter is missing, the workflow terminates with an error.	recordlist		1
ChannelID(Optional)	Contains the data pool ID.	string	Valid identification for the data pool.	0..1
ChannelName (Optional)	Contains the data pool name, the activity derives the ID of the data pool from the	string	Datapool name.	0..1

Name	Semantics	Type	Valid Values	Multi-plicity
	database.			
StatusGroup (Mandatory)	Record synchronization group.	string	REGISTRATION PUBLICATION CONFIRMATION	0..1
SyncStatus(Optional)	Status of the record.	string	PUBLISHED DISCONTINUED CANCELLED ACCEPTED REVIEWED REGISTERED.	0..1
OrganizationType (Optional)	Indicates the type of organization. The types are Retailer, Buyer, or Supplier.	string	RETAILER BUYER (Default) SUPPLIER	0..1
TradingPartnerID (Optional)	Trading Partner ID.	string		0..1
TradingPartnerName (Optional)	Trading Partner Name.	string		0..1
Direction: Out				
ChannelCredential (Optional)	Contains Channel (marketplace) credentials.	arraylist		1
ChannelID(Optional)	Contains marketplace ID.	arraylist		1
OutRecordList(Optional)	See Common Parameters.			

Name	Semantics	Type	Valid Values	Multi-plicity
RecordKey(Optional)	Contains records that are processed and have status in the productstatus table.	arraylist		1
StatusGroup(Optional)	Record synchronization group.	arraylist		1
SyncStatus(Optional)	Status of the record.	arraylist		1
TradingPartnerCredential (Optional)	Trading Partner Credential.	arraylist		1
TradingPartnerID (Optional)	Trading Partner ID.	arraylist		1

Example for GetSyncStatus Activity

The example returns the Channel, trading Partner IDs, sync status and trading partner credentials as well as the product key for a given status group, lynch status and organization type.

```
<Activity Name="GetSyncStatus">
  <Start/>
  <Action>GetSyncStatus</Action>
  <Description lang="en">Lookup function.</Description>
  <Parameter direction="in" type="string" eval="constant"
  name="eventState">GetSyncStatus</Parameter>
  <Parameter direction="in" name="InDocument" type="document"
  eval="variable">inDoc</Parameter>
  <Parameter direction="in" name="InRecordList" type="recordlist"
  eval="variable">workRecordList</Parameter>
```

```
  <Parameter direction="in" name="ChannelName" eval="xpath" type="String"
  source="/Message/Header/MessageHeader
[@origin='Receiver']  /Organization/PartyID/PartyName/text()">inDoc</Parameter>
  <Parameter direction="in" name="StatusGroup" eval="constant"
  type="String">PUBLICATION</Parameter>
  <Parameter direction="in" name="SyncStatus" eval="constant"
  type="String">PUBLISHED</Parameter>
  <!--RETAILER/TRADING PARTNER/MARKETPLACE/SUPPLIER-->
  <Parameter direction="in" name="OrganizationType" eval="constant"
  type="String">RETAILER</Parameter>
  <Parameter direction="out" name="RecordKey" type="arraylist"
  eval="variable">ProductKeyArray</Parameter>
  <Parameter direction="out" name="ChannelCredential" type="arraylist"
  eval="variable">SynchChannelArray</Parameter>
  <Parameter direction="out" name="TradingPartnerCredential" type="arraylist"
  eval="variable">SynchTradingPartnerArray</Parameter>
  <Parameter direction="out" name="SyncStatus" type="arraylist"
  eval="variable">SynchStatusArray</Parameter>
  <Parameter direction="out" name="StatusGroupOut" type="arraylist"
  eval="variable">SynchStatusGroupArray</Parameter>
  <Parameter direction="out" name="ChannelID" type="arraylist"
  eval="variable">SynchChannelIDArray</Parameter>
  <Parameter direction="out" name="TradingPartnerID" type="arraylist"
  eval="variable">SynchTradingPartnerIDArray</Parameter>
  <Parameter direction="out" name="OutRecordList" type="recordlist"
  eval="variable">outRecordList</Parameter>
 </Activity>
```

GetRecord Activity

The GetRecord activity retrieves record information from the database into an XML format. The XML format is defined in the mlXML DTD definition.

The GetRecord activity populates certain sections in the XML input document called the Catalog Action document. The activity populates the Catalog Item in the document, which contains the Classification, Item Data, and Relationship Data sections.

Classifications

Classifications represents classification schemes and classification codes of a record.

Note: If you do not specify value for the RelationshipDepth parameter, the classification data is not populated in the mlXML document.

The following partial XML message shows the format generated for classification data:

```
<ActionCode>
   <Code>
     <CodeType>CatalogAction</CodeType>
     <Value>AddProduct</Value>
     <Normal>AddProduct</Normal>
   </Code>
</ActionCode>
<SubLineItemInfo />
<UnitPrice />
<Classifications>
   <Classification>
     <ClassificationScheme>
       <RevisionID>
         <BaseName>FOOD</BaseName>
         <Version>1</Version>
         <DBID>336919</DBID>
       </RevisionID>
     </ClassificationScheme>
<ClassificationCodes>
   <ClassificationCode>
     <Code>Dairy</Code>
     <Value>DAIRY</Value>
    <ClassificationCode>
       <Code>Milk</Code>
       <Value>MILK</Value>
    </ClassificationCode>
   </ClassificationCode>
</ClassificationCodes>
   </Classification>
</Classifications>
```

Item Data

Item Data represents attributes of a record.

The following partial XML message shows the format generated for attributes:

```
<MasterCatalog>
 <RevisionID>
  <BaseName>PRODUCTS</BaseName>
  <Version>1</Version>
  <DBID>35961</DBID>
 </RevisionID>
</MasterCatalog>
<Contact />
<Reference />
<URL />
<Extension />
<ItemData>
 <Attribute name="PRODUCTID">
  <Value>1</Value>
 </Attribute>
 <Attribute name="PRODUCTIDEXT">
  <Value>1-ext</Value>
 </Attribute>
 <Attribute name="CONTAINS">
  <Value />
 </Attribute>
 <Attribute name="Product_Name">
  <Value>Chitale Milk</Value>
 </Attribute>
 <Attribute name="Product_Description">
  <Value>Milk</Value>
 </Attribute>
 <Attribute name="SKU">
  <Value>123</Value>
 </Attribute>
 <Attribute name="Price">
  <Value>30</Value>
 </Attribute>
 <Attribute name="Organic">
  <Value>False</Value>
 </Attribute>
 <Attribute name="Level_of_Fat_Claim">
  <Value>250</Value>
 </Attribute>
 <Attribute name="Type_of_Grain">
  <Value />
 </Attribute>
 <Attribute name="Type_of_Filling">
  <Value />
 </Attribute>
 <Attribute name="If_Sliced">
```

```
  <Value />
 </Attribute>
 <Attribute name="Firmness">
  <Value />
 </Attribute>
</ItemData>
<ActionLog>
 <Action>
 <ActionCode>
  <ActionCodeType>ActionLogCode</ActionCodeType>
  <Normal>RecordDiff</Normal>
 </ActionCode>
 <ReferenceItem>
  <ProdID>
   <IDNumber>1</IDNumber>
   <IDExtension>1-ext</IDExtension>
   <IDVersion>N/A</IDVersion>
   <DBID>13003</DBID>
  </ProdID>
 </ReferenceItem>
 <Attribute changed="true" name="PRODUCTID">
  <Value>1</Value>
  <OriginalValue />
   <GroupList>
    <Group name="Unassigned" />
   </GroupList>
 </Attribute>
 <Attribute changed="true" name="PRODUCTIDEXT">
  <Value>1-ext</Value>
  <OriginalValue />
   <GroupList>
    <Group name="Unassigned" />
   </GroupList>
 </Attribute>
  <Attribute changed="true" name="Product_Name">
   <Value>Chitale Milk</Value>
   <OriginalValue />
   <GroupList>
    <Group name="Unassigned" />
   </GroupList>
 </Attribute>
  <Attribute changed="true" name="Product_Description">
   <Value>Milk</Value>
   <OriginalValue />
   <GroupList>
    <Group name="Unassigned" />
```

```
    </GroupList>
  </Attribute>
   <Attribute changed="true" name="SKU">
     <Value>123</Value>
     <OriginalValue />
     <GroupList>
      <Group name="Unassigned" />
     </GroupList>
  </Attribute>
   <Attribute changed="true" name="Price">
     <Value>30</Value>
     <OriginalValue />
     <GroupList>
      <Group name="Unassigned" />
     </GroupList>
  </Attribute>
   <Attribute changed="true" name="Organic">
     <Value>False</Value>
     <OriginalValue />
     <GroupList>
      <Group name="AttributeGroup1" />
     </GroupList>
  </Attribute>
   <Attribute changed="true" name="Level_of_Fat_Claim">
     <Value>250</Value>
     <OriginalValue />
     <GroupList>
      <Group name="AttributeGroup1" />
     </GroupList>
  </Attribute>
   <Attribute changed="true" name="Type_of_Grain">
     <Value />
     <OriginalValue />
     <GroupList>
      <Group name="AttributeGroup1" />
     </GroupList>
  </Attribute>
   <Attribute changed="true" name="Type_of_Filling">
     <Value />
     <OriginalValue />
     <GroupList>
      <Group name="AttributeGroup1" />
     </GroupList>
   </Attribute>
   <Attribute changed="true" name="If_Sliced">
     <Value />
```

```
    <OriginalValue />
    <GroupList>
     <Group name="AttributeGroup1" />
    </GroupList>
  </Attribute>
   <Attribute changed="true" name="Firmness">
    <Value />
    <OriginalValue />
    <GroupList>
     <Group name="AttributeGroup1" />
    </GroupList>
  </Attribute>
  </Action>
 </ActionLog>
```

Relationship Data

The Relationship Data represents various relations of a record.

The following partial XML message shows the format generated for relationship data:

```
<RelationshipData>
  <Relationship>
   <RelationType>ContainedBy</RelationType>
   <RelatedItems count="1">
    <RelatedItem referenceKey="18361">
     <Attribute name="quantity">
      <Value>1</Value>
     </Attribute>
    </RelatedItem>
   </RelatedItems>
  </Relationship>
 </RelationshipData>
</CatalogItem>
<CatalogItem key="18361">
 <LineNumber>2</LineNumber>
 ...
 <ItemData>
 <Attribute name="PRODUCTIDEXT">
  <Value>2</Value>
 </Attribute>
 <Attribute name="PRODUCTID">
  <Value>BigB</Value>
 </Attribute>
```

```
 <Attribute name="SHORTDESC">
  <Value>prod B</Value>
 </Attribute>
</ItemData>
<ActionLog/>
<RelationshipData>
 <Relationship>
  <RelationType>ContainedBy</RelationType>
  <RelatedItems count="1">
   <RelatedItem referenceKey="18360">
    <Attribute name="quantity">
     <Value>1</Value>
    </Attribute>
   </RelatedItem>
  </RelatedItems>
 </Relationship>
 <Relationship>
  <RelationType>Contains</RelationType>
  <RelatedItems count="1">
   <RelatedItem referenceKey="18362">
    <Attribute name="quantity">
     <Value>6</Value>
    </Attribute>
   </RelatedItem>
  </RelatedItems>
 </Relationship>
</RelationshipData>
</CatalogItem>
<CatalogItem key="18360">
<LineNumber>3</LineNumber>
 ...
 <ItemData>
  <Attribute name="PRODUCTIDEXT">
   <Value>1</Value>
  </Attribute>
  <Attribute name="PRODUCTID">
   <Value>LargeA</Value>
  </Attribute>
  <Attribute name="SHORTDESC">
   <Value>prod A</Value>
  </Attribute>
  <Attribute name="IMAGE">
   <Value/>
  </Attribute>
  <Attribute name="RECORD_TYPE">
   <Value>TestBUNDLING</Value>
```

```
 </Attribute>
 <Attribute name="CONTAINS">
  <Value>BigB:2:5:1:39038:0</Value>
 </Attribute>
</ItemData>
<ActionLog/>
<RelationshipData>
 <Relationship>
  <RelationType>Contains</RelationType>
  <RelatedItems count="1">
   <RelatedItem referenceKey="18361">
    <Attribute name="quantity">
     <Value>5</Value>
    </Attribute>
   </RelatedItem>
  </RelatedItems>
 </Relationship>
</RelationshipData>
</CatalogItem>
</CatalogActionDetails>
```

GetRecord Parameters and Valid Execution Modes

The valid execution mode for GetRecord is SYNCHR.

The parameters of GetRecord are as follows:

GetRecord Parameters

Name	Semantics	Type	Valid Values	Multi-plicity
Direction: In				
Agency(Optional)	Specify the agency name.	string	SOURCE (default) SUPPLIER SELLER	0..1
CatalogOutputMap (Optional)	Output Catalog Map. Use of this parameter	string	Valid Output map for the repository	1

Name	Semantics	Type	Valid Values	Multi-plicity
	is discouraged. It will be deprecated in future releases.		specified.	
InDocument (Mandatory)	See Common Parameters.			
MasterCatalog (Mandatory)	See Common Parameters.			
RelationshipDepth (Optional)	Depth of Relationship to be retrieved. Note that the type for this parameter has been changed from string to long.	long	-1 (All levels) 0 (no level, only self). This is the default. 1 (first level) 2 to n (custom level)	0..1
RelationshipName# (Optional)	RelationshipName along with RelationshipDepth set in this activity determines the related records to be retrieved in the OutDocument. You can specify multiple RelationshipName parameters. Replace # with an integer, for example, RelationshipName1, RelationshipName2, and so on.	string	Any valid relationship name.	0..n

Name	Semantics	Type	Valid Values	Multi-plicity
VersionKeyword (Optional)	Record Version to be retrieved.	string	SUPPLIED CONFIRMED LATEST (default)	0..1
PerspectiveName (Optional)	See CreateWorkItem Parameters and Valid Execution Modes.			
Direction: Out				
OutDocument (Optional)	See Common Parameters.			

Example for GetRecord Activity

The following sample shows an example of a GetRecord activity:

```
<Activity Name="GetRecord">
  <Start/>
  <Action>GetRecord</Action>
  <Description lang="en">Get Item Data.</Description>
  <Parameter direction="in" name="InDocument" type="document"
  eval="variable">workDoc</Parameter>
  <Parameter direction="in" eval="constant" type="string"
  name="Agency">SOURCE</Parameter>
  <Parameter direction="in" eval="xpath" type="long" name="MasterCatalog"
  source="/Message/Body/Document/BusinessDocument/CatalogAction/C
  atalogActionHeader/MasterCatalog/RevisionID/DBID/text()">workDoc  </Parameter>
  <Parameter direction="out" eval="variable" type="document"
  name="OutDocument">workDoc</Parameter>
</Activity>
```

GenerateReportForSDD Activity

The GenerateReportForSDD activity generates a report for the scheduler duplicate detection process. The generated report is saved in the $MQ_COMMON_DIR/<enterprise_internal_name>/catalog/download folder.

During the report generation process, this activity performs the following:

- Retrieves data from the database for event details
- Generates report with data using the report template
- Saves generated report in the $MQ_COMMON_DIR/<enterprise_internal_name>/catalog/download folder

GenerateReportForSDD Parameters and Valid Execution Modes

The valid execution mode for GenerateReportForSDD is SYNCHR or ASYNCHR.

The parameters of GenerateReportForSDD are as follows:

GenerateReportForSDD Parameters

Name	Semantics	Type	Valid Values	Multi-plicity
Direction: In				
reportType	Specifies the type of report. • For the detailed type, the report includes duplicate record list in a form of bundles.	string	detailed Or simple	1

Name	Semantics	Type	Valid Values	Multi-plicity
	• For the simple type, the report includes the summary information.			
NumberOfRecordsToIncludeInReport (optional)	Maximum number of records that must be included in the duplicate detection report.	long	Any number	1

Example for GenerateReportForSDD Activity

Review the examples for the GenerateReportForSDD activity for a detailed report and a simple report.

This is the example for GenerateReportForSDD activity for a detailed report:

```
<Activity Name="CreateReport">
  <Start/>
  <Action>GenerateReportForSDD</Action>
  <Description>Generate report for Auto Duplicate Detection</Description>
  <Parameter direction="in" eval="constant" type="string"
  name="eventState">GENERATEFILE</Parameter>
  <Parameter direction="in" eval="constant" type="string"
  name="reportType">detailed</Parameter>
</Activity>
```

This is the example for GenerateReportForSDD activity for a simple report:

```
<Activity Name="CreateReport">
  <Start/>
  <Action>GenerateReportForSDD</Action>
  <Description>Generate report for Auto Duplicate Detection</Description>
  <Parameter direction="in" eval="constant" type="string"
```

```
  name="eventState">GENERATEFILE</Parameter>
  <Parameter direction="in" eval="constant" type="string"
  name="reportType">simple</Parameter>
</Activity>
```

HandleMessaging Activity

The HandleMessaging activity works in conjunction with the IdentifyProtocolOperations activity.

The activity generates the XML message per marketplace requirements based on operations identified by the IdentifyProtocolOperations activity. The generated message is in mlXML format.

The generated XML includes the following parts:

- **The message header:** This contains the credential information and is common to all marketplaces.
- **The message structure:** This section defines packaging of the outgoing message. It also defines transactional boundaries (where transactions are supported by channel). This section is marketplace specific.
- **The item data:** This contains the record data as required by the targeted marketplace.

The activity also takes parameters which indicate the number of operations/commands to be sent as part of a single message.

Typically, after this activity you would have a translate activity, followed by a SendProtocolMessage activity. After SendProtocolMessage, the HandleMessaging is called again to send any additional messages that must be sent.

Based on the ProtocolMessagingCommand given to the activity, the activity behaves differently.

- The START and CONTINUE commands are used for normal operations.
- The SUSPEND command is a request to suspend the process.
- The ABORT command is issued in cases where the entire process is to be aborted.
- The TIMEOUT command allows handling of cases where responses for a message sent are not received within a set amount of time.

During normal processing, the ProtocolMessagingNextCommand generated by the

activity as part of a ‘send’ loop is used as the ProtocolMessagingCommand in the next ‘send’ loop.

HandleMessaging Parameters and Valid Execution Modes

The valid execution mode for HandleMessaging is SYNCHR.

The parameters of HandleMessaging are as follows:

HandleMessaging Parameters

Name	Semantics	Type	Valid Values	Multi-plicity
Direction: In				
CommandsPerMessage(Optional)	The number of related records (bundle) to be sent in a message. This determines the size of the payload going out. In case of in-memory workflows, this parameter is ignored, and the entire record bundle is sent in one go. This does not impact performance in any way, since the size of the record bundle is small in case of in-memory workflows.	long	Defaults to 50.	0..1
ProtocolMessagingCommand (Optional)	The command for the activity to execute.	string	START (default) CONTINUE SUSPEND ABORT TIMEOUT	0..1
ReferenceStepID(Mandatory)	Points to the StepID of the IdentifyProtocolOperations step.	long		1
ResponseRequired(Optional)	This parameter decides whether the workflow must suspend and wait for responses. This parameter is ignored for Datapool-Marketplaces namely - WWRE and 1Sync.	boolean	True False (default) Defaults to false for non-Datapool marketplaces.	0..1
InDocument(Mandatory)	See Common Parameters.			
InRecordList(Mandatory)	See Common Parameters.			
FEDOption(Optional)	See Common Parameters.			
Direction: Out				

Name	Semantics	Type	Valid Values	Multi-plicity
OutDocument(Optional)	See Common Parameters.			
OutRecordList(Optional)	See Common Parameters.			
ProtocolMessageGenerated (Optional)	Flag to indicate if a message was generated.	boolean	True False	1
ProtocolMessagingNextCommand (Optional)	The next command that the activity has identified for it to do. This usually becomes the ProtocolMessagingCommand on the next send cycle.	string	CONTINUE SUSPEND DONE	1
RecordsProcessed(Optional)	Count of Number of Records processed by the activity.	long		1

Example for HandleMessaging Activity

```
<Activity Name="HandleMessagingDataPools">
 <Start/>
 <Action>HandleMessaging</Action>
 <Description lang="en">Create a message for a set of products, for specific data
pools</Description>
 <Parameter direction="in" name="CommandsPerMessage" type="long"
 eval="constant">50</Parameter>
 <Parameter direction="in" name="InDocument" type="document"
 eval="variable">inDoc</Parameter>
 <Parameter direction="in" name="ProtocolMessagingCommand" type="string"
 eval="variable">nextCmd</Parameter>
 <Parameter direction="in" name="ProtocolMessagingCommand" type="string"
 eval="constant">START</Parameter>
 <Parameter direction="in" name="ResponseRequired" type="boolean"
 eval="constant">true</Parameter>
 <Parameter direction="in" name="ReferenceStepID" eval="variable"
 type="long">prlog2</Parameter>
 <Parameter direction="in" name="InRecordList" type="recordlist"
 eval="variable">workRecordList</Parameter>
 <Parameter direction="out" name="OutRecordList" type="recordlist"
 eval="variable">workRecordList</Parameter>
 <Parameter direction="out" name="OutDocument" type="document"
 eval="variable">messageDoc</Parameter>
 <Parameter direction="out" name="ProtocolMessagingNextCommand"   type="string"
eval="variable">nextCmd</Parameter>
 <Parameter direction="out" name="ProtocolMessageGenerated" type="boolean"
 eval="variable">msgGenerated</Parameter>
</Activity>
```

IdentifyActionType Activity

The IdentifyActionType activity identifies the status of each record in the RecordCollection. Status is output in an array of string values (Exist, Unknown, Deleted) and two Record Lists.

OutRecordList contains those records that exist in the system, while OutRecordList2 contains all remaining records.

IdentifyActionType Parameters and Valid Execution Modes

The valid execution mode for IdentifyActionType is SYNCHR.

The parameters of IdentifyActionType are as follows:

IdentifyActionType Parameters

Name	Semantics	Type	Valid Values	Multiplicity
Direction: In				
InRecordList (Mandatory)	Collection of records to process.	recordlist	Valid variable.	1
Direction: Out				
RecordState (Optional)	The state of the record for each of the records in OutRecordList.	arraylist	Exist Unknown Deleted	1
OutRecordList (Optional)	See Common Parameters.			
OutRecordList2 (Optional)	List of Records which do not exist in the system.	recordlist		1

Example for IdentifyActionType Activity

```
<Activity Name="IdentifyActionType"><Start/>
  <Action>IdentifyActionType</Action>
  <Description lang="en">Identify the action and spawn appropriate   workflow</Description>
  <Parameter direction="in" name="InDocument" type="document"
  eval="variable">workDoc</Parameter>
  <Parameter direction="in" name="InRecordList" type="recordlist"
  eval="variable">workRecordList</Parameter>
```

```
  <Parameter direction="out" name="RecordState" type="arrayList"
  eval="variable">WorkRecordState</Parameter>
  <Parameter direction="out" name="OutRecordList" type="recordlist"
  eval="variable">outRecordList</Parameter>
  <Parameter direction="out" name="OutRecordList2" type="recordlist"
  eval="variable">outRecordList2</Parameter>
</Activity>
```

IdentifyProtocolOperations Activity

Use the IdentifyProtocolOperations activity in synchronization workflows to identify the type of operation to be performed on each record in a record list.

The type of operation identified for each record depends on the synchronization history and the selected synchronization options. The activity creates productlogs for records requiring an operation. Productlogs are typically used by the HandleMessaging Activity activity to generate protocol specific messages.

Note: The SkipIdentifyProtocolOps parameter is added to the following new transition from the ConverRecordsToOutput to the ApplySynchRulebase activity to improve the performance of synchronization process and to avoid the generation of product logs. For more information, see Changes to Out-of-the-Box Workflows and the chapter "Exporting Records Using Synchronization" in *User's Guide*.

IdentifyProtocolOperations Parameters and Valid Execution Modes

The valid execution mode for IdentifyProtocolOperations is SYNCHR.

The parameters of IdentifyProtocolOperations are as follows:

Parameters (default)

IdentifyProtocolOperations Parameters

Name	Semantics	Type	Valid Values	Multi-plicity
Direction: In				
ConnectorRole (Optional)	This parameter assigns a logical role which identifies the operations. If the parameter is not specified, a pre-configured rulebase, that is, rbconnectorrolederivation.xml is used to determine the role. The role is used to search for the plug-in that determines the operations.	string		0..1
BundlePerAsyncCall (Optional) *(applicable since version 7.1)*	See Common Parameters.			
AsynProcessIndicator (Optional) *(applicable since version 7.1)*	See Common Parameters.			
InDocument (Mandatory)	See Common Parameters.			
InRecordList (Mandatory)	Collection of records to process.	recordlist	N/A	1

Name	Semantics	Type	Valid Values	Multi-plicity
Rulebase(Optional)	See Common Parameters.			
FEDOption(Optional)	See Common Parameters.			
Direction: Out				
OutRecordList (Optional)	See Common Parameters.			
ConnectorProtocol (Optional)	This parameter can be used by other workflow activities (i.e. HandleMessaging).			
AsyncStepID(Optional) (applicable since version 7.1)	Process Log ID pointing to a unique number identifying an execution step. This can be passed as an optional input ReferenceStepID to other activities.	long		

Parameters (optional)

The following table lists the inputs to the rulebase that can be supplied as activity input parameters. Most flags map to user inputs on the Catalog synchronization page in the UI:

Note: These parameters are only applicable if the Rulebase parameter is specified.

Name	Semantics	Type	Valid Values	Multiplicity
Direction: In				
ADD_FLAG	Is Add requested?	boolean	true/false	1
RELOAD_FLAG	Is reload requested?	boolean	true/false	1
PUBLISH_ CORRECTION_FLAG	Is publication correction requested?	boolean	true/false	1
ADD_CORRECTION_ FLAG	Is adding the correct requested?	boolean	true/false	1
DELETE_FLAG	Is delete requested?	boolean	true/false	1
ADD_CANCEL_FLAG	Is add cancel requested?	boolean	true/false	1
ADD_DELETE_FLAG	Is add delete requested?	boolean	true/false	1
ADD_DISCONTINUE	Is add discontinue requested?	boolean	true/false	1
ADD_REINSTATE_ FLAG	Is add reinstate requested?	boolean	true/false	1
RFCIN_FLAG	Is RFCIN requested?	boolean	true/false	1

Name	Semantics	Type	Valid Values	Multiplicity
RFCIN_RELOAD_FLAG	Is RFCIN reload requested?	boolean	true/false	1
PUBLISH_NEW_FLAG	Are new publications to be sent?	boolean	true/false	1
INCLUDE_REJECTED_RECS_FLAG	Should previously rejected records be part of the publication?	boolean	true/false	1
INCREMENTAL_FLAG	Is incremental publication requested?	boolean	true/false	1
ACCEPT_FLAG	Is the publication Accepted?	boolean	true/false	1
REJECT_FLAG	Is the publication Rejected?	boolean	true/false	1
REVIEW_FLAG	Is the publication Reviewed?	boolean	true/false	1
SYNCHRONISE_FLAG	Is the publication Synchronized?	boolean	true/false	
MASTERCATALOG_NAME	Name of the repository whose data is to be synchronized.	string	A valid repository name.	1

Name	Semantics	Type	Valid Values	Multiplicity
DATAPOOL_NAME	Name of the data pool with which the data is being synchronized.	string	WWRE 1Sync Or any custom data pool configured in the system.	1

Example for IdentifyProtocolOperations Activity

```
<Activity Name="IdentifyProtocolOperations">
  <Start/>
  <Action>IdentifyProtocolOperations</Action>
  <Description lang="en">Identify Operations to be performed for each   Item.</Description>
  <Parameter direction="in" name="InDocument" type="document"
  eval="variable">inDoc</Parameter>
  <Parameter direction="in" name="ReferenceStepID" eval="variable"
  type="long">pl1</Parameter>
  <Parameter direction="in" name="InRecordList" type="RecordList"
  eval="variable">convertedRecordList</Parameter>
  <Parameter direction="out" name="OutRecordList" type="RecordList"
  eval="variable">workRecordList</Parameter>
</Activity>
```

IdentifyRecordVersions Activity

The IdentifyRecordVersions activity is required only if synchronization is initiated based on a catalog. If you are performing a lights-out synchronization without a catalog, this activity is not applicable.

This activity, if used, must be the first activity in the workflow and must be marked as Start. It is mandatory for catalog synchronization workflows.

Since release 5.8, this activity no longer processes records and does not return a record list of products that are processed in the workflow.

IdentifyRecordVersions Parameters and Valid Execution Modes

The valid execution mode for IdentifyRecordVersions is SYNCHR.

The parameters of IdentifyRecordVersions are as follows:

IdentifyRecordVersions Parameters

Name	Semantics	Type	Valid Values	Multiplicity
Direction: In				
InDocument (Mandatory)	See Common Parameters.			

Example for IdentifyRecordVersions Activity

```
<Activity Name="IdentifyRecordVersions">
 <Start/>
 <Action>IdentifyRecordVersions</Action>
 <Description lang="en">Prepare for trading partner catalog   synchronization.</Description>
 <Parameter direction="in" name="InDocument" type="document"
  eval="variable">inDoc</Parameter>
</Activity>
```

ImportCatalogRecords

The ImportCatalogRecords activity imports records into a repository.

This activity works only in conjunction with the PrepareForImportCatalog Activity activity.

ImportCatalogRecords Parameters and Valid Execution Modes

The valid execution mode for ImportCatalogRecords is SYNCHR.

The parameters of ImportCatalogRecords are as follows:

ImportCatalogRecords Parameters

Name	Semantics	Type	Valid Values	Multi-plicity
Direction: In				
AsynProcessIndicator (Optional)	See Common Parameters.			
CatalogInputMap (Optional)	There can be multiple input maps associated with a catalog. So, when importing a catalog, the input map name must be specified. An mlXML document always contains an input map name, so even if this parameter is not present, the activity can use the one specified in mlXML.	string	Input Map Name.	0..1
ErrorThreshold(Optional)	If the error count reaches this threshold, the import is terminated. The value can be set using the Configurator (**Repository** > **Catalog Import Error Threshold**). The order of the configuration is: Value specified in workflow file > Value specified in the Configurator > Default value.	long	Default is 1000.	0..1
Incremental(Optional)	This parameter is provided to indicate if the given set of records is the ‘complete’ set for a given repository (Incremental=no) or is an addition to existing records in the repository. (Incremental=yes).	string	Yes (default) No	0..1
InDocument(Mandatory)	See Common Parameters.			
InRecordList(Optional)	Collection of records to process.	recordlist	Not applicable.	0..1
MasterCatalog (Mandatory)	See Common Parameters.			
MergeData(Optional)	This parameter is used to determine what the default value for an attribute in a record must be when nothing is specified in the input data source. When set to ‘yes’, the existing attribute value in the repository record is retained. When set to ‘no’, it is possible to over-write an attribute value as ‘null’. The ‘null if blank’ option in the input map definition in the UI determines the overall behavior.	string	Yes (default) No	1
ProcessOption(Optional)	Specifies the process option for import:	string	M	0..1

Name	Semantics	Type	Valid Values	Multi-plicity
	• MUTATION (M) – Indicates whether the record ID must be tested for mutations. Records are considered to have key mutation when PRODUCTID or EXT changes over the lifetime of the record. If the flag is not specified, the mutation test is not performed. (True in configurator means M in import option). False is default for Configurator (**Optimization** > **Import Mutation Test Performed**). • DUPLICATE (D) – Indicates that a test for duplicate rows in the import must be performed. If this flag is specified and the same record appears more than once, only the first occurrence is processed and other occurrences are rejected. True in the Configurator means D in Import options. True is default in the Configurator (**Optimization** > **Duplicate Row Check**).		D C Default is D.	
	• CYCLIC (C) – Performs a cyclic check on the new relationship being imported. A cyclic check is very time consuming and must be requested only when data is expected to have cyclic relationships. (True in the Configurator means C in import option). False is default in the Configurator (**Optimization** > **Cyclic Import Check**). The Workflow file value takes precedence over the Config file value.			
RecordPerAsyncCall (Optional)	See Common Parameters.			
Rulebase(Optional)	See Common Parameters. You can use ImportCatalog and EvaluateRuleBase in a single step, in which case, no additional record versions are created and import is faster. To configure this:Pass the rulebase through the ImportCatalog activity. For example: <Parameter direction="in" name="Rulebase" eval="constant" type="string">D:\common\ac\rulebase/catalogvalidation.xml</Parameter>	string	Existing Rulebase file.	1
ParameterNames (Optional)	This parameter is applicable only if using a Rulebase. The values to input variables defined in the rulebase can be passed though activity parameters. The names of the parameters must match the declarations in the rulebase. <declare usage=”input”>	string	Must be declared as input variables in the rulebase.	0..n
SplitImportBatch (Optional)	This value is stored as a Process Detail. It has no effect on the behavior of the import activity.	string	Split/Approval Required Split/No Approval Direct Load	1

Name	Semantics	Type	Valid Values	Multi-plicity
RecordState	Indicates the state of the saved record. You can specify the following states: • CONFIRMED: If you specify the CONFIRMED value, the imported records are confirmed immediately. When this state is specified, workflow does not require a separate UpdateRecordState activity. You must change the out-of-the-box workflow to skip updateRecordState activity. • UNCONFIRMED: If you specify the UNCONFIRMED or DRAFT value, workflow might require the UpdateRecordState activity. • DRAFT: If you do not specify any value, Draft is defaulted.	string	DRAFT (Default)CONFIRMED UNCONFIRMED	
MergeBeforeRulebase	You can merge existing records with new records before the rulebase is applied.	boolean	• True • False (Default)	1
VersionPolicy (Optional)	Indicates how the record versions are managed. The allowed VersionPolicy values are NEW, CORRECT, OPTIMIZE. • NEW: using this option, you can create a new version of a record. This is the default value. • CORRECT: using this option, you can update the existing record versions. New version is not created, only if the old record version state and new record version state match. • OPTIMIZE: using this option, you can optimize the best choice between the CORRECT and NEW options. TIBCO MDM automatically decides if a new version must be created. The following rules are used when VersionPolicy=OPTIMIZE: • A new record version is created, if the old and new record state do not match. • A new record version is created, if the old record state is REJECTED. • A new record version is created, if the old record version is DELETED. • The existing record versions are corrected, if both the old and new record state match. • For everything else, the existing record versions are corrected.	string	• NEW (Default): creates a new version • CORRECT: updates an existing record version, no new version. If policy rule allows correction. • OPTIMIZE: this is a combination of both NEW and CORRECT. TIBCO MDM automatically decides whether or not a record can be corrected; otherwise a new version is created.	0..1

Note: Record version correction is allowed in the ImportCatalogRecord activity only when the activity is running in the ConfirmImmediately mode. That is, VersionPolicy=Correct is applicable to ImportCatalogRecord Activity only if the activity is set to RecordState=CONFIRMED. For best results, use Optimize.

Direction: Out				
CatalogsWithErrors (Optional)	Total number of input catalogs with errors. If this parameter is not specified and RejectedRecords is not zero, the workflow aborts.	long		

Name	Semantics	Type	Valid Values	Multi-plicity
DeletedRecords (Optional)	Total number of records deleted.	long	Applicable only if Incremental=no.	1
DuplicateRecords (Optional)	Total number of records rejected as duplicates.	long	Any positive number.	1
EditedRecords(Optional)	Total number of modified records.	long	Any positive number.	1
NewRecords(Optional)	Total number of new records imported.	long		1
OutDocument (Deprecated)	See Common Parameters. For backward compatibility, outDocument is the same as InDocument.			1
RecordsAttempted (Optional)	Total number of records processed.	long	Any positive number.	1
RecordsProcessed (Optional)	Number of Records processed by the activity.	long	Any number.	1
RecordsSuccess (Optional)	Total number of records added, updated, or deleted.	long	Any positive number.	1
RejectedRecords (Optional)	Total number of records rejected due to validation errors.	long	Any positive number.	1

Example for ImportCatalogRecords Activity

```
<Activity Name="ImportCatalog">

    <Action>ImportCatalogRecords</Action>

    <Description lang="en">Import summary</Description>

    <Parameter direction="in" type="string" eval="constant" name="eventState">IMPORT</Parameter>

    <Parameter direction="in" eval="xpath" type="string" name="MasterCatalog" source
="/Message/Body/Document/BusinessDocument/CatalogAction/CatalogActionHeader/MasterCatalog/RevisionID/BaseNam
e/text()">inDoc</Parameter>

    <Parameter
direction="in" eval="xpath" type="string" name="Incremental" source
="/Message/Body/Document/BusinessDocument/CatalogAction/CatalogActionHeader/MasterCatalog/Extension
[@name='Incremental']/Value/text()">inDoc</Parameter>

    <Parameter
direction="in" eval="xpath" type="string" name="MergeData" source
="/Message/Body/Document/BusinessDocument/CatalogAction/CatalogActionHeader/MasterCatalog/Extension
[@name='MergeData']/Value/text()">inDoc</Parameter>

    <Parameter
direction="in" eval="xpath" type="string" name="SplitImportBatch" source
="/Message/Body/Document/BusinessDocument/CatalogAction/CatalogActionHeader/MasterCatalog/Extension
[@name='SplitImportBatch']/Value/text()">inDoc</Parameter>

    <Parameter direction="in" eval="xpath" type="string" name="CatalogInputMap" source
="/Message/Body/Document/BusinessDocument/CatalogAction/CatalogActionHeader/MasterCatalog/InputMap/RevisionID/
BaseName/text()">inDoc</Parameter>

    <Parameter direction="in" name="InDocument" type="document" eval="variable">inDoc</Parameter>

    <Parameter direction="out" eval="variable" type="document" name="OutDocument">workDoc</Parameter>

    <Parameter direction="out" eval="variable" type="long" name="RecordsSuccess">recordsSuccess</Parameter>
```

```
<!--Parameter direction="in" eval="constant" type="long" name="ErrorThreshold">100</Parameter-->
<!--Parameter direction="in" eval="constant" type="long" name="RecordPerAsyncCall">100</Parameter-->
<!--Parameter direction="in" name="AsynProcessIndicator" type="boolean" eval="constant">true</Parameter-->
<!-- ProcessOption: MUTATION - M, DUPLICATE - D, RELATIONSHIP - R, CYCLIC - C -->
<!--Parameter direction="in" name="ProcessOption" type="string" eval="constant">MD</Parameter-->
<!-- If this param is not defined, activity will throw an exception to abort the workflow. By defining -->
<!-- this param, you are taking responsibility to handle errors. -->
<Parameter direction="out" eval="variable" type="long" name="CatalogsWithErrors">catalogsWithError</Parameter>
</Activity>
```

ImportClassificationScheme Activity

The ImportClassificationScheme activity imports classification codes into an existing classification scheme. This activity can be triggered by FileWatcher.

ImportClassificationScheme Parameters and Valid Execution Modes

The valid execution mode for ImportClassificationScheme is SYNCHR.

The parameters of ImportClassificationScheme are as follows:

ImportClassificationScheme Parameters

Name	Semantics	Type	Valid Values	Multiplicity
Direction: In				
Incremental (Mandatory)	Indicates incremental or full load. True indicates incremental load. Currently, only incremental is supported.	string	True False	1
Direction: Out				
CSRowCount (Optional)	Number of rows processed.	long		
CSNewCount (Optional)	Number of new classification codes added.	long		
CSModCount (Optional)	Number of existing classification codes updated.	long		

Example for ImportClassificationScheme Activity

The following sample tries to resume a workflow using a MessageID.

```
<Activity Name="ImportClassificationScheme">
  <Start />
  <Action>ImportClassificationScheme</Action>
  <Description>Import Classification Scheme Summary</Description>
  <Parameter name="eventState" type="string" direction="in"
  eval="constant">CSIMPORT</Parameter>
  <Parameter name="Incremental" type="string" direction="in" eval="xpath"
  source="/Message/Body/Document/BusinessDocument/CatalogAction/C
  atalogActionHeader/ClassificationScheme/Extension[@name='Increm    ental']/Value/text
()">inDoc</Parameter>
  <Parameter name="InDocument" type="document" direction="in"
  eval="variable">inDoc</Parameter>
</Activity>
```

Note: The ImportClassificationScheme activity does not support failover.

InitiateDataTransfer Activity

Use the InitiateDataTransfer activity for the Data Transfer Export operation. The activity initiates export metadata and export data events and generates the metadata for the reimport operation.

On the Data Transfer Export page, when export is initiated by selecting **All** option for Repository, the InitiateDataTransfer activity finds all root repositories in the enterprise and initiates export metadata and data events for each one of them. In this case, the parent Data Transfer event initiates all subevents for export metadata and export data operations.

The next DataTransfer activity determines transition of the workflow. If any of the child event fails, then this workflow transitions to either TIMEOUT, ERROR, or SUCCESS.

The InitiateDataTransfer activity suspends after initiating all subevents and is revived to continue once all children events are completed. The export archive file is created when all subevents are completed successfully.

For each root repository, two separate export metadata and export data events are initiated. The export metadata is a data service query workflow and the export data is a DBdump workflow.

InitiateDataTransfer Parameters and Valid Execution Modes

The valid execution mode for InitiateDataTransfer is SYNCHR.

InitiateDataTransfer Parameters

Name	Semantics	Type	Valid Values	Multiplicity
Direction: In				
eventState	Determines the latest state of the workflow.	String	INITDATATRANSFER	0..1
InDocument(Mandatory)	See Common Parameters.	document		
Direction: Out				
DataTransferEventStatus	Indicates the final status of all Data Transfer sub-events propagated to the main parent event. If all sub-events are successful, then DataTransferEventStatus is SUCCESS. If any of the sub-event goes into the INPROGRESS, TIMEOUT, ERROR, or CANCELLED state, then the parent event is also considered to be in the same state and DataTransferEventStatus reflects the same.	string	DataTransferEventStatus	0..1

Example of InitiateDataTransfer Activity

```
<Activity Name="InitiateDataTransfer">
      <Start/>
      <Action>InitiateDataTransfer</Action>
      <Description>Initiate Data Transfer</Description>
      <Execution>SYNCHR</Execution>
      <Parameter name="eventState" direction="in" eval="constant"
type="string">INITDATATRANSFER</Parameter>
      <Parameter direction="in" name="InDocument" type="document"
eval="variable">inDoc</Parameter>
      <Parameter direction="in" name="MasterCatalog" eval="xpath" type="long"
source="/Message/Body/Document/BusinessDocument/CatalogAction/CatalogActionHeader/Mast
erCatalog/RevisionID/DBID/text()">inDoc</Parameter>
      <Parameter direction="out" eval="variable" type="document"
name="OutDocument">workDoc</Parameter>
      <Parameter direction="out" eval="variable" type="string"
name="DataTransferEventStatus">DataTransferEventStatus</Parameter>
   </Activity>
```

InitiateSubFlow Activity

The InitiateSubFlow activity calls another workflow in synchronous mode (function call style). When the synchronous mode is specified, the calling workflow suspends and waits for the called workflow to complete and potentially return outputs.

Using the InitiateSubFlow activity you can pass a number of input parameters to the called workflow. For information about how to use this activity to manage subflows, see Subflow Management.

InitiateSubFlow Parameters and Valid Execution Modes

The valid execution mode for InitiateSubFlow is SYNCHR.

The parameters of InitiateSubFlow are as follows:

InitiateSubFlow Parameters

Name	Semantics	Type	Valid Values	Multi-plicity
Direction: In				
Any parameter (Optional)	This activity accepts any parameter. All input parameters are passed to the called workflow.	Any of the supported parameter types		
InDocument (Mandatory)	See Common Parameters.			

InRecordList (Optional)	See Common Parameters.
	To pass the InRecordList parameter as an input parameter to the subflow, specify the similar external name for the recordlist in the subflow. For example,
	• In InitiateSubflow:
	`<Parameter direction="in" name="InRecordList" type="recordlist" eval="variable">WorkrecordList</Parameter>`
	• In Subflow:
	`<Parameter direction="in" name="InRecordList" type="recordlist" eval="variable">2</Parameter>`
			recordlist	

ProcessID	The name of the workflow to execute. If the workflow is specified, it is retrieved from the specified location. The location must be relative to MQ_COMMON_DIR. If workflow is not specified, the workflow is selected using the Process Definition Selection business rule. For details about selecting the workflow, see Workflow Process Selection.	string	The workflow file name along with the relative path of MQ_COMMON_DIR.	0..1
Direction: Out				
<Any parameter>	Any output parameters defined can be mapped from the called workflows. The output parameter names	Any of the supported		

Name	Semantics	Type	Valid Values	Multi-plicity
	must match the output parameters defined in the called workflow. First returned document is set in OUTDOCUMENT.	parameter types.		
OutDocument (Optional)	See Common Parameters.			
WorkflowPrority (Optional)	See Common Parameters. If the priority is not specified, event priority is inherited by the new message.			

Example for InitiateSubFlow Activity

The subflow call activity contains the following input parameters which are passed to the called workflow. Note that parameters received by the called workflow are controlled by the input parameters defined in the called workflow.

- **eventState:** defined as constant.
- **inDocument:** passed in as inDoc
- **inStatus:** defined as constant

This activity expects the following output parameters:

Output Parameters	Data Type	Description
outRecordList1	record List	Value is assigned to the workRecordList local variable .
statusFlag	boolean	Value is assigned to the finalStatusFlagx local variable.
status	String	Value is assigned to the finalStatusx local variable.
statusValue	long	Value is assigned to the statusValuex local variable.
Testunknown	String	Value is assigned to the unknownValue local variable.

The following example is a subflow call activity:

```
<Activity Name="TestSubflow">
  <Start/>
  <Action>InitiateSubFlow</Action>
  <Description>Spawn the subworkflow for generation of DBDump</Description>
```

```
  <Execution>SYNCHR</Execution>
  <Parameter direction="in" type="string" eval="constant"
  name="eventState">SPAWNWORKFLOW</Parameter>
  <Parameter direction="in" type="string" eval="constant"
  name="ProcessID">standard/workflow/wfin26Test1</Parameter>
  <Parameter direction="in" type="document" eval="variable"
  name="InDocument">inDoc</Parameter>
  <Parameter direction="in" type="String" eval="constant"
  name="InStatus">TestInStatus</Parameter>
  <Parameter direction="out" eval="variable" type="String"
  name="status">finalStatusx</Parameter>
  <Parameter direction="out" eval="variable" type="boolean"
  name="statusFlag">finalStatusFlagx</Parameter>
  <Parameter direction="out" eval="variable" type="long"
  name="statusValue">statusValuex</Parameter>
  <Parameter direction="out" eval="variable" type="recorList"
  name="outRecordList1">workRecordList</Parameter>
  <Parameter direction="out" eval="variable" type="string"
  name="Testunknown">unknownValue</Parameter>
 </Activity>
```

An example subflow follows. This workflow provides the following input parameters:

Input Parameters	Description
InDocument	This is always passed implicitly.
InStatus	String which is assigned the value passed in by the TestInStatus parameter of the calling workflow.

A predefined variable, isSubFlow, of type boolean is implicitly passed and set as true when this workflow is called as subflow.

This workflow provides the following output parameters:

Output Parameters	Data Type	Description
outRecordList1	record List	Value is assigned by the local variable workRecordList which is

Output Parameters	Data Type	Description
		output of ManageRecordCollection.
statusFlag	boolean	Value is assigned by local variable finalStatusFlag which is the output of the AddMsgInfoEvent activity.
status	String	Value is assigned by local variable finalStatus which is output of the AddMsgInfoEvent activity.
statusValue	Long	Assigned a value of 10.

```
<Workflow Version="0.1">
  <Owner>TIBCO</Owner>
  <Name>wfin26test1</Name>
  <Description lang="en">Test subflow activity</Description>
  <Parameter direction="in" eval="variable" type="document"   name="inDoc">1</Parameter>
  <Parameter direction="in" type="String" eval="variable"
  name="InStatus">TestInStatus</Parameter>
  <Parameter direction="out" eval="variable" type="document"
  name="OutDoc">workDoc</Parameter>
  <Parameter direction="out" eval="variable" type="String"
  name="status">finalStatus</Parameter>
  <Parameter direction="out" eval="variable" type="boolean"
  name="statusFlag">finalStatusFlag</Parameter>
  <Parameter direction="out" eval="constant" type="long"   name="statusValue">10</Parameter>
  <Parameter direction="out" eval="variable" type="recordList"
  name="outRecordList1">workRecordList</Parameter>
  <Activity Name="AddMsgInfoToEvent">
    <Start/>
    <Action>UpdateEvent</Action>
    <Start/>
    <Description lang="en">Initialize Event Info</Description>
    <Parameter direction="in" type="string" eval="constant"
    name="eventState">START</Parameter>
    <Parameter direction="in" name="eventDescriptor" type="string"     eval="xpath"
source="/Message/Body/Document/@subtype">inDoc</Parameter>
```

```
    <Parameter direction="in" eval="constant" type="String"
    name="finalStatus">finalStatus1</Parameter>
    <Parameter direction="in" eval="constant" type="boolean"
    name="finalStatusFlag">true</Parameter>
    <Parameter direction="in" name="deploymentMode" type="string"     eval="xpath"
source="/Message/@messageType">inDoc</Parameter>
    <Parameter direction="in" name="dbDirectLoadDump" type="string"
    eval="constant">N</Parameter>
    <Parameter direction="in" name="eventType" type="string"
    eval="constant">CAT</Parameter>
  </Activity>
  <Activity Name="TestSubflow">
    <Action>NoOperation</Action>
    <Description>Test Subflow</Description>
    <Execution>ASYNCHR</Execution>
    <Parameter direction="in" type="string" eval="constant"
    name="eventState">SPAWNWORKFLOW</Parameter>
    <Parameter direction="in" type="document" eval="variable"
    name="InDocument">inDoc</Parameter>
  </Activity>
</Workflow>
```

InitiateWorkflow Activity

The InitiateWorkflow activity allows initiation of a new workflow or restarting an existing workflow.

To resume a workflow, the activity allows resuming a workflow based on the processID. This is typically used when a response message is received and it must be correlated to the original workflow which sent the request to the external system. On receiving response messages, the CheckMessageStatus activity allows finding the processID, if any, associated with the correlation ID provided in the message.

While resuming an existing workflow, if ProcessIDType is specified as MessageID, there must be at least one suspended ApplicationTask work item for the specified process which can be restarted. Such work items would be closed before restarting the workflow. If there is more than one work item open, the work item is selected based on SelectionMethod. If ProcessIDType is specified as DefinitionID, the process could be suspended in any activity. It simply resumes from the activity which was suspended.

In case a new workflow must be started, the command is START. In this case, ProcessID and ProcessIDType are not required. An input document is submitted as an

asynchronous workflow request. No other workflow state is transferred. The workflow manager decides the workflow to be executed based on the data provided in the input document. The new workflow initiated is a separate event unlike the InitiateSubflow activity which continues execution of the subflow as part of the event which initiated the workflow. The new initiated event is not associated with the original event.

InitiateWorkflow Parameters and Valid Execution Modes

The valid execution mode for InitiateWorkflow is SYNCHR.

The parameters of InitiateWorkflow are as follows:

InitiateWorkflow Parameters

Name	Semantics	Type	Valid Values	Multi-plicity
Direction: In				
ProcessID (Optional)	Specifies the ID of the workflow process to restart.	string	Must be a valid ID of the suspended process.	0..1
ProcessIDType (Optional)	The type of process ID. Helps in interpreting the ProcessID.	string	MESSAGEID (default) DEFINITIONID	0..1
Command (Optional)	Start or restart a workflow.	string	START (default) RESUME	0..1
InDocument (Mandatory)	See Common Parameters. If the command is specified as START,			

Name	Semantics	Type	Valid Values	Multi-plicity
	this document is submitted as a workflow request.			
SelectionMethod (Optional)	Used to decide if the status of the first or last application task must be returned for the given message ID.	string	First Last (default)	1
WorkflowPriority (Optional)	See Common Parameters. If the priority is not specified, event priority is inherited by the new message.			

Example for InitiateWorkflow Activity

The following sample tries to resume a workflow using a MessageID.

```
<Activity Name="RestartWfMsg"><Start/>
  <Action>InitiateWorkflow</Action>
  <Execution>ASYNCHR</Execution>
  <Description lang="en">Create a new event to restart    workflow.</Description>
  <Parameter direction="in" eval="variable" type="document"
   name="InDocument">inDoc</Parameter>
  <Parameter direction="in" name="Command" type="string"
   eval="constant">RESUME</Parameter>
  <Parameter direction="in" name="ProcessIDType" type="string"
   eval="constant">MESSAGEID</Parameter>
  <Parameter direction="in" name="ProcessID" type="string"    eval="variable">msgID</Parameter>
</Activity>
```

InterpretCommand Activity

The InterpretCommand activity allows you to execute a script and retrieve results from the script. All “in” parameters are passed into the script, and all “out” parameters are assigned values equal to that of the variables inside the script.

Following the Parameters is a “Script” element which contains the Java code to execute. Currently, the allowed “format” for the script is “bsh” which means “Bean Shell” and is really a Java interpreter.

Example for InterpretCommand Activity

```
<Activity Name="InterpretCommand">
  <Start />
  <Action>InterpretCommand</Action>
  <Description lang="en">Compute the enterprise-specific directory to     store the export JAR
file.</Description>
  <Parameter direction="in" eval="xpath" type="string"    name="EnterpriseShortName"
  source="/Message/Header/MessageHeader[@origin='Receiver']/Enter
  prise/PartyID/ShortName/text()">inDoc</Parameter>
  <Parameter direction="out" name="ExportFileDirectory" type="string"
  eval="variable">ExportFileDirectory</Parameter>
<Script format="bsh">
<![CDATA[
ExportFileDirectory = EnterpriseShortName +
"/metadata/export/data";
</Script>
</Activity>
```

You can also use this activity to run a shell script. The following is an example:

```
<Activity Name="PurgeFilesThroughShellScript">
  <Action>InterpretCommand</Action>
  <Description lang="en">Delete the files associated with purged    data</Description>
  <Parameter direction="in" eval="variable" type="string"
  name="purgefilepath">relativepath</Parameter>
  <Parameter direction="in" eval="constant" type="string"
  name="shellscriptname">deletepurgefiles.sh</Parameter>
<Script format="bsh">
<![CDATA[
          java.lang.String envPath
          =com.tibco.mdm.infrastructure.EnvUtil.getDirEnvironmentVaria            ble
```

```
(com.tibco.mdm.infrastructure.IMqEnvVars.MQ_HOME);
            java.lang.String commondirpath =
            com.tibco.mdm.infrastructure.EnvUtil.getDirEnvironmentVariable
(com.tibco.mdm.infrastructure.IMqEnvVars.MQ_COMMON_DIR);
            envPath = envPath + "bin" ;
            java.lang.String absolutepathfile =   commondirpath +              purgefilepath ;
            java.lang.String executescriptstring =  " sh " +              shellscriptname + " " +
absolutepathfile ;
            java.lang.Runtime runtime = java.lang.Runtime.getRuntime() ;
          try {
              java.lang.Process child = runtime.exec(executescriptstring                , null,  new
java.io.File(envPath));
              // child.waitFor();
            } catch (java.io.IOException e) {
              com.tibco.mdm.infrastructure.error.MqException me = new
              com.tibco.mdm.infrastructure.error.MqException
(com.tibco.mdm.infrastructure.MqErrorCodes.IOERROR, e);
                          me.setArg
(com.tibco.mdm.infrastructure.MqErrorCodes.FILE              NAME, envPath );
              throw me;
            } catch (java.lang.InterruptedException e) {
              com.tibco.mdm.infrastructure.logging.MqLog.log("Interpr              etCommand",
              com.tibco.mdm.infrastructure.logging.MqLog.WARNING,              "Interupted purge
delete .");
            } finally {
          }
</Script>
</Activity>
```

JoinExistingWorkflow Activity

If a record in the InRecordList is in another workflow, this activity will “push” that record into the existing workflow.

If a conflict arises during the merge, the approvers in the target (existing) workflow have to resolve it.

The JoinExistingWorkflow activity fails if the parent event failed to complete before the Join action starts. Hence, join does not happen and workflow gets terminated.

JoinExistingWorkflow Parameters and Valid Execution Modes

The valid execution mode for JoinExistingWorkflow is SYNCHR.

The parameters of JoinExistingWorkflow are as follows:

JoinExistingWorkflow Parameters

Name	Semantics	Type	Valid Values	Multiplicity
Direction: In				
InRecordList (Mandatory)	See Common Parameters.			

Example for JoinExistingWorkflow Activity

```
<Activity Name="JoinExistingWorkflow"><Start/>
  <Action>JoinExistingWorkflow</Action>
  <Description lang="en">Join the existing workflow</Description>
  <Parameter direction="in" name="InRecordList" type="recordlist"
   eval="variable">workRecordList</Parameter>
</Activity>
```

ManageRecordCollection Activity

The ManageRecordCollection activity allows generation and manipulation of a record collection (“set” of records) for the records associated with an event.

This activity contains the following modes:

- If the setRecordListReadOnly function is passed as true: It takes the input record list and sets it to read-only. Setting a record list to read only adds all of the child records to the record list and does not allow any changes to the record list. This conversion is to be used to generate a snapshot of records and hierarchies so that

any report on the record list can include the child records. It does not affect how record collections are manipulated in workflow.

- If the setRecordListReadOnly is false: The activity reads the record information from the document/event and create an output RecordList object.

When record collections (RecordList object) are manipulated in workflow, each record in the record collection is considered the root of the record hierarchies and allows processing of the hierarchies.

The ManageRecordCollection activity also allows you to separate records for one repository from all the records associated with the event or InDocument. If more than one repository is associated with the workflow, the repository name is required to identify the root repository. The root repository defines the scope of all relationships included in the record hierarchies and is used to validate the relationships supplied for bundling.

If bundlingOption = true, this activity allows you to identify records which must be the root of the hierarchies. Bundling is often required when records must be grouped together along with relationships and processed as a unit. The activity reviews all records associated with the event/inDocument and identifies the ones which must be the root of the hierarchies. Such records are included in the record collection. The records which are not included in the record collection but are part of the record hierarchies are called “implicitly” included records. The activity does not allow one record to be included in more than one hierarchy. Such records are identified and output as another record collection in the OutRecordList2.

ManageRecordCollection Parameters and Valid Execution Modes

The valid execution mode for ManageRecordCollection is SYNCHR.

The parameters of ManageRecordCollection are as follows:

ManageRecordCollection Parameters

Name	Semantics	Type	Valid Values	Multi-plicity
Direction: In				
AsynProcessIndicator (Optional)	See Common Parameters.			
BundlingOption(Optional)	This parameter indicates if records must be converted to record hierarchies. If set to true, the output record collection have hierarchies of the record based on specified relationships. When record hierarchies are being created, a repository must be specified as input and would be used as a root repository. If bundlingOption is set to false, no hierarchies are created. This is typically used when you must create a record collection of all records in the input document or event so that this record collection can be processed by other workflow activities. Note that if the input document is not specified, a repository can be specified as input so that the resulting record collection contains records belonging to specified repository only.	boolean	True False	0..1
IncludeDraftRecords (Optional)	See Common Parameters.			
InDocument(Optional)	See Common Parameters.			
InRecordList(Optional)	See Common Parameters. Input RecordList. If none is specified, all records associated with the event are considered. This parameter is required only if SetRecordListReadOnly is true.	recordlist	Existing RecordList.	0..1
MasterCatalog(Optional)	When both MasterCatalog and Indoc are specified as input to ManageRecordCollection, the catalog specified in the MasterCatalog parameter is used for bundling. This parameter allows you to limit the record collection to a specific repository, if BundlingOption=false. • If the document is specified, MasterCatalog does not apply. • If the document is not specified, and BundlingOption=true, the repository is used to validate relationships specified. It does not limit the bundling to only one repository. All relationships starting from the repository are processed. If the repository is not	string		0..1

Name	Semantics	Type	Valid Values	Multi-plicity
	specified, it is taken from the first record. • If BundlingOption=false, and InDocument is not specified, only records of the specified repository are added to record collection. • If BundlingOption=true, this repository is used as root to select relationships across all related repositories.			
PerspectiveName (Optional)	See CreateWorkItem Parameters and Valid Execution Modes.			
RecordPerAsyncCall (Optional)	See Common Parameters.			
RelationshipName# (Optional)	See Common Parameters.			
setRecordListReadOnly (Optional)	A boolean to determine whether the activity must take the incoming record list object and set it to be read-only.	boolean	True False (default)	1
VersionOption(Optional)	See Common Parameters.			
Direction: Out				
OutRecordList	See Common Parameters.			
RecordsProcessed (Optional)	Number of records added to the newly created record list.	long		
BundlesCreated (applicable since version 7.1)	Number of bundles created using the specified relationship name.	long		

The following table describes how the activities process input records and valid input combinations.

Note: If InDoc is specified and it does not have any records, the following table must be read as if InDoc is not specified. Also, the record list is marked as read only if SetRecordListReadOnly is set to true.

Bundling	InDoc	Repository	Relation Name	Behavior
True	N/A	N/A	N	The following message is shown: None of the specified relationships are valid or none specified.
	Y	N	Y	Bundles for specified relation names and relationships are not validated.
	Y	Y	Y	Repository is used to validate relationships.
	N	N	Y	All records associated with the event are processed. Relationships are not validated. This is the slowest execution mode.
	N	Y	Y	All records associated with the event are processed. Relationships are validated. Processing happens in two passes. The first pass attempts to create bundles for a specified repository. The second pass processes only those records from other repositories which are not already associated with the roots identified so far. First, all records associated with the specified repository are processed. Next, all remaining records are processed if they are not already part of the bundle created in the first pass.
False	Y	N/A	N/A	Relationships specified are ignored. Repository name specified is ignored.
	N	Y	N/A	Relationship name is ignored. All records associated with the event are added to output record collection if they belong to the specified repository.
	N	N	N/A	Relationship name is ignored. All records associated with the event are added to output record collection.

Example for ManageRecordCollection Activity

This activity converts an existing record collection to read only.

```
<Activity Name="ManageRecordCollection">
 <Start/>
 <Action>ManageRecordCollection</Action>
 <Description lang="en">Picks the appropriate record versions from
 the document.</Description>
 <Parameter direction="in" name="InRecordList" type="recordlist"
 eval="variable">workRecordList</Parameter>
 <Parameter direction="out" name="OutRecordList" type="recordlist"
 eval="variable">workRecordList</Parameter>
 <Parameter direction="in" name="SetRecordListReadOnly" type="boolean"
 eval="constant">true</Parameter>
</Activity>
```

This example allows creation of record collection which includes root records for a specific catalog only.

```
<Activity Name="ManageRecordCollection">
 <Start/>
 <Description lang="en">Create Bundles in the Record   Collection</Description>
 <Action>ManageRecordCollection</Action>
 <Parameter direction="in" type="string" eval="constant"
 name="eventState">CREATEBUNDLE</Parameter>
 <Parameter direction="in" name="RelationshipName" type="string"
 eval="constant">CONTAINS</Parameter>
 <Parameter direction="in" name="BundlingOption" type="boolean"
 eval="constant">true</Parameter>
 <Parameter direction="in" name="MasterCatalog" type="string"
 eval="constant">XYZ</Parameter>
 <Parameter direction="in" name="VersionOption" type="string"
 eval="constant">LATEST</Parameter>
 <Parameter direction="out" name="OutRecordList" type="recordlist"
 eval="variable">workRecordList</Parameter>
 <Parameter direction="out" name="OutRecordList2" type="recordlist"
 eval="variable">rejectRecordList</Parameter>
 <Parameter direction="in" name="RecordPerAsyncCall" type="long"
 eval="constant">10</Parameter>
 <Parameter direction="in" name="AsynProcessIndicator" type="boolean"
 eval="constant">true</Parameter>
</Activity>
```

ManageWorkItem Activity

The ManageWorkItem activity performs the action of closing a work item or all work items associated with an event or a particular step.

The activity sets the status of work items to the specified status.

ManageWorkItem Parameters and Valid Execution Modes

The valid execution mode for ManageWorkItem is SYNCHR.

The parameters of ManageWorkItem are as follows:

ManageWorkItem Parameters

Name	Semantics	Type	Valid Values	Multiplicity
Direction: In				
WorkItemStatus (Mandatory)	Status of the work item.	string	CANCELLED CLOSED Any other valid status.	1
WorkItemAction (Mandatory)	Action to be performed.	string	CLOSE	1
ReferenceStepID (Optional)	Reference ID.	long		0..1

Example for ManageWorkItem Activity

```
<Activity Name="CloseAllWorkitems">
  <Action>ManageWorkItem</Action>
```

```
 <Start/>
 <Description lang="en">Closes all the open workitems and set the status   to
Cancelled.</Description>
 <Execution>SYNCHR</Execution>
 <Parameter direction="in" eval="constant" type="string"
  name="WorkItemAction">CLOSE</Parameter>
 <Parameter direction="in" eval="constant" type="string"
  name="WorkItemStatus">CANCELLED</Parameter>
 <Parameter direction="in" name="InDocument" type="document"
  eval="variable">inDoc</Parameter>
</Activity>
```

ManualClassification Activity

The ManualClassification activity allows you to manually classify records during the import process.

The ManualClassification activity is processed after the ExtractRelationship Activity.

ManualClassification Parameters and Valid Execution Modes

The valid execution mode for ManualClassification is SYNCHR.

The parameters of ManualClassification are as follows:

ManualClassification Parameters

Name	Semantics	Type	Valid Values	Multiplicity
Direction: In				
MasterCatalog	See Common Parameters.			
CatalogInputMap	Name of an input map.	string	Input map name.	1

Name	Semantics	Type	Valid Values	Multiplicity
	Multiple input maps can be associated with a catalog. Therefore, while importing a catalog, the input map name must be specified.			
ClassifyRecordsOnly	Classifies the records that already exist in a repository. Also, verifies whether or not the imported records, that is, new records, exist in a repository. If the imported records exist, they are classified, else they are not classified.	string	True False (Default)	0..1
ErrorThreshold	If the error count reaches the specified threshold value, the import is terminated. For more information about this parameter, see ImportCatalogRecords.	long	Any valid integer. Default value is 1000.	0..1
RecordPerAsyncCall (Optional)	See Common Parameters.			
AsynProcessIndicator	See Common Parameters.			
Direction: Out				

Name	Semantics	Type	Valid Values	Multiplicity
TotalRecords	Indicates the total number of records retrieved by the activity.	long	Any positive number	1
ClassifiedRecords	Indicates a number of records classified by the activity.	long	Any positive number	1
UnclassifiedRecords	Indicates a number of records that are not classified by the activity.	long	Any positive number	1
RecordsWithError	Indicates a number of records that are not processed by the activity.	long	Any positive number	1

Example for ManualClassification Activity

```
<Activity Name="ManualClassification">
 <Action>ManualClassification</Action>
        <Description lang="en">Manual Classification of Records</Description>
        <Parameter direction="in" type="string"
eval="constant"name="eventState">MANUALCLASSIFICATION</Parameter>
        <Parameter direction="in" eval="xpath" type="string"
name="MasterCatalog"source="/Message/Body/Document/BusinessDocument/CatalogAction/Cat
alogActionHeader/MasterCatalog/RevisionID/BaseName/text()">inDoc</Parameter>
 <Parameter direction="in" eval="xpath" type="string" name="CatalogInputMap"
source="/Message/Body/Document/BusinessDocument/CatalogAction/CatalogActionHeader/Mast
erCatalog/InputMap/RevisionID/BaseName/text()">inDoc</Parameter>
 <Parameter direction="in" eval="xpath" type="string" name="ClassifyRecordsOnly"
source="/Message/Body/Document/BusinessDocument/CatalogAction/CatalogActionHeader/Mast
erCatalog/Extension[@name='ClassifyRecordsOnly']/Value/text()">inDoc</Parameter>
        <!--Parameter direction="in" eval="constant" type="long"
name="ErrorThreshold">100</Parameter-->
        <!--Parameter direction="in" eval="constant" type="long"
name="RecordPerAsyncCall">100</Parameter-->
```

```
        <!--Parameter direction="in" name="AsynProcessIndicator" type="boolean"
eval="constant">true</Parameter-->
</Activity>
```

MatchRecord Activity

The MatchRecord activity allows you to search for matching records using selected record attributes as matching criteria. The activity takes a source record list as input and matches each source record against the existing repository data.

The output of the activity is a record list containing possible matches along with the matching score. You can mandate a minimum matching percentage to ensure good quality matches.

You can choose to use the default Advanced Search Matching (TIBCO Patterns - Search), Basic Search Matching, or a Custom matching. You must set up the Indexing or Matching Engine properties using the Configurator.

You can specify attributes from different repositories as criteria for matching in the rulebase.

For example, consider a Person repository is related to the Address repository by ResidenceAddress relationship. You want to detect near-duplicate or duplicate records for a person John Doe, who has a ResidenceAddress Palo Alto, 94304. You can specify attributes across repositories as matching criteria. For example, FirstName and LastName from Person repository and City and ZIP from Address repository.

For more information about using basic search matching and advanced search matching, see the Search and Matching chapter in the *TIBCO MDM System Administrator's Guide*.

Note: Do not use attributes with the following attribute data types for matching attributes: Date, Integer, Decimal, File, Amount, Boolean, and Custom Decimal. The index treats them as text strings. Using such attributes does not return the expected match results.

MatchRecord Parameters and Valid Execution Modes

The valid execution mode for MatchRecord is SYNCHR

The parameters of MatchRecord are as follows:

MatchRecord Parameters

Name	Semantics	Type	Valid Values	Multi-plicity
Direction: In				
InDocument(Optional)	Contains the source records to be matched. The record version is derived from the InDocument (if specified) or else the record version from the event is used.	Document	Any valid mlXML document.	0..1
MatchingThreshold(Optional)	String representing matching score value. This value is treated as the minimum matching score. Records with a match score less than the specified score are not returned. If unspecified, the default value is “0.8”. Used as a literal value in case of deterministic index based search engines such as TIBCO Patterns - Search.	String	Any positive float value between 0 and 1. 0.8 (Default)	0..1
MatchRecordAttributeList(Optional)	List of record attributes to be used as matching criteria.	ArrayList	Any valid ArrayList of repository attribute names.	0..1
RecordAttributeName(Optional)	A record attribute name to be matched. You can also specify the weight of an attribute along with its name to provide more weightage for matching records. For example, UOM attribute with weight as 0.8: <Parameter direction="in" name="RecordAttributeName" eval="constant" type="string">uom^0.8</Parameter> You can specify either the MatchRecordAttributeList or the RecordAttributeName parameter. If a non-empty MatchRecordAttributeList parameter is specified, the RecordAttributeName parameter is ignored and the list is used.	String	Any valid repository attribute name.	1
SourceProductKey(Optional)	The key of the record to be matched. This parameter accepts only one record key as input.	String	Any valid record key. The value must parse as an integer. Typically, use an XPATH expression to get this value from the InDocument.	1

Name	Semantics	Type	Valid Values	Multi-plicity
SourceRecordKeyList (Optional)	A list of source product key IDs	ArrayList		0..1
IndexEntityAttributesList (Optional)	List of the record attributes mapped to the index table attributes. For more information about using index entity attributes, see the Cross-Repository Matching Against Index Data section of the *TIBCO MDM Customization*.	ArrayList	Any valid ArrayList of mapped attribute names.	0..1
PostProcessorClass (Optional)	Allows customized post-processing of the results returned by the matching engine. The matching engine returns both UNCONFIRMED and CONFIRMED matching records. To receive only CONFIRMED state matching records, you can specify com.tibco.dq.ConfirmedStatusFilterPostProcessorImpl post-processor class, which works as a status filter.	String	com.tibco.dq. ConfirmedStatusFilterPostProcessorImpl	0..1
IndividualMatchScoreForEntity (Optional)	Specifies the matching threshold on the individual join index entity. For example, in CustomerToBank and CustomerToAddress cross-repository matching, you can specify an individual matching threshold for each join index entity, that is, CustomerToBank (0.7) and CustomerToAddress (0.6).	String	Cross-repository join index entity name along with matching threshold. For example, CustomerToBank^0.7 and CustomerToAddress^0.6.	1
ConsiderNonMatchingRecords (Optional)	Adjusts the scoring of non-matching records. In cross-repository matching, although the matching records are not found for a child record, the child records are considered for the matching score. The matching score depends on the PenaltyForNonMatchingRecords parameter. For example, the following records are added in the cross-repository hierarchy: • CustomerToAddress: ◦ John_01 > California ◦ John_02 > California_02 • CustomerToBank: ◦ John _01 > ICICI In this case, the second record (John_02 > California_02) in the CustomerToAddress entity shows 100% (1) matching even though its respective bank record is not available. However, if you specify true, the matching score changes to	String	True False (Default)	1

Name	Semantics	Type	Valid Values	Multi-plicity
	0.5 instead of 1. Use this parameter in combination with the PenaltyForNonMatchingRecords parameter.			
PenaltyForNonMatchingRecords (Optional)	Specifies the penalty score for the non-matching entity or records. In cross-repository matching, although the matching records are not found for a child record, the child records are considered for the matching score. Specifying the penalty score adjusts the matching score of child records and displays the results. For example, the following records are added in the cross-repository hierarchy: • CustomerToAddress: ◦ John_01 > California ◦ John_02 > California_02 • CustomerToBank: ◦ John _01 > ICICI In this case, the second record (John_02 > California_02) in the CustomerToAddress entity shows 100% (1) matching even though its respective bank record is not available. However, if you specify the penalty score for non-availability of the second bank record as 0.4, the matching score for the second bank record is calculated as 1(matching record)-0.4 (penalty score)=0.6. The combined matching score of both the records is calculated as follows: Addition of the two matching scores (1+0.6=1.6) divided by a number of records (2)=0.8	String	Cross-repository join entity name that includes non-matching records along with penalty matching score. For example, CustomerToBank^0.4	1
UseCustomQueryForNetrics (Optional)	If specified as True, the custom Netrics query specified in the CustomNetricsQueryBuilderImpl parameter is used to get results from the TIBCO Patterns - Search indexes.	String	True False (Default)	1
CustomNetricsQueryBuilderImpl (Optional)	Use the custom implementation class to query the TIBCO Patterns - Search table. It must implement com.tibco.mdm.repository.search.INetricsQueryBuilder interface.	String	The Java class name that you want to load. • For custom Netrics query:	1

Name	Semantics	Type	Valid Values	Multi-plicity
			com.tibco.dq.customQuery.PatternsCustomQueryImpl • To call the Patterns-Search query builder API, specify the value as: com.tibco.dq.customQuery.queryBuilder.PatternsQueryBuilderImpl	
IndexEntityNameToSearch (Optional)	Allows to search the custom index entity. While searching the index entities, the specified index entity name is used for searching records instead of an auto detection.	String	Index entity name that is specified in the IndexerConfig.xml file. For example, Customer_IndexEntity.	1
Direction: Out				
MatchCount	Number of matching records found. If this parameter is 0, it indicates that no matching records were found for a given MatchingThreshold and a given RecordAttributeName.	Long	0 (default)	1
MatchingRecordKey	This is applicable in case of Single Record Introduction. When a single matching record is found, the record key of the matching record is returned.	String	Any valid record key.	0..1
MatchingRecordRelevanceScore	Relevance score of the matching record key. Applicable only in case of a single source record being matched and a single matching record found.	ArrayList		0..1
MatchRecordProcessLogID	MatchRecordProcessLogID is treated as StepID. It is equivalent to StepID in case of synchronous execution. Use the MatchRecordProcessLogID to link the MatchRecord execution output to the next activity.			0..1
MultipleMatchSourceRecordList	List of source records with more than one matching record. Applicable only in case of Bulk Import.	ArrayList		0..1
MultipleMatchTargetRecordList	List of target records corresponding to each source record in the MultipleMatchSourceRecordList list. Applicable only in case of Bulk Import.	ArrayList		0..1
MultipleMatchTargetRelevanceList	List of relevance scores for target records corresponding to each source record in the MultipleMatchSourceRecordList list. Each entry in this list is a list of target record relevance scores. For every entry in the MultipleMatchTargetRecordList object, there is a corresponding entry in the MultipleMatchTargetRelevanceList list.	ArrayList		0..1

Name	Semantics	Type	Valid Values	Multi-plicity
	Applicable only in case of Bulk Import.			
NoMatchSourceRecordList	List of source records with no matching records. Applicable only in case of Bulk Import.	ArrayList		0..1
SingleMatchSourceRecordList	List of source records with a single matching record.	ArrayList		0..1
SingleMatchTargetRecordList	List of target records corresponding to each source record in the SingleMatchSourceRecordList list	ArrayList		0..1
SingleMatchTargetRelevanceList	List of relevance scores corresponding to each target record in the SingleMatchTargetRecordList list Applicable only in case of Bulk Import.	ArrayList		0..1
CrossRepositoryMatch (Optional)	Indicates that the matching process has identified matching records that belong to a different repository than the input record's repository. In such case, Merge operation cannot be performed and the CrossRepositoryMatch output can be used by workflow designers to take appropriate actions. If IndexEntityAttributesList parameter is used in the MatchRecord activity, the activity output contains this optional output parameter.	Boolean	True (Default) False.	0..1

Example for MatchRecord Activity

```
<Activity Name="Matcher">
  <Action>MatchRecord</Action>
  <Description>Matcher record data</Description>
  <Execution>SYNCHR</Execution>
  <Parameter name="InDocument" direction="in" eval="variable"
  type="document">workDoc</Parameter>
  <Parameter name="SourceProductKey" direction="in" eval="variable"
  type="string">Key2</Parameter>
  <Parameter name="MatchRecordAttributeList" direction="in" eval="variable"
  type="arraylist">MatchAttributeArray</Parameter>
  <Parameter name="IndexEntityAttributesList" direction="in" eval="variable"
type="arraylist">indexEntityAttributeList</Parameter>
  <Parameter name="MatchingThreshold" direction="in" eval="constant"
  type="string">0.5</Parameter>
  <Parameter name="eventState" direction="in" eval="constant"
  type="string">MATCHRECORD</Parameter>
  <Parameter name="PostProcessorClass" direction="in" eval="constant"
type="string">com.tibco.dq. ConfirmedStatusFilterPostProcessorImpl</Parameter>
  <Parameter name="MatchCount" direction="out" eval="variable"
  type="long">numberOfMatches</Parameter>
  <Parameter name="MatchingRecordKey" direction="out" eval="variable"
  type="string">MatchingRecordKey</Parameter>
  <Parameter name="MatchingRecordRelevanceScore" direction="out"   eval="variable"
type="string">MatchingRecordRelevanceScore</Parameter>
  <Parameter name="SingleMatchSourceRecordList" direction="out"   eval="variable"
type="arraylist">singleMatchSourceList</Parameter>
  <Parameter name="SingleMatchTargetRecordList" direction="out"   eval="variable"
type="arraylist">singleMatchTargetList</Parameter>
  <Parameter name="SingleMatchTargetRelevanceList" direction="out"   eval="variable"
type="arraylist">singleMatchRelevanceList</Parameter>
  <Parameter name="MultipleMatchSourceRecordList" direction="out"   eval="variable"
type="arraylist">multipleMatchSourceList</Parameter>
  <Parameter name="MultipleMatchTargetRecordList" direction="out"   eval="variable"
type="arraylist">multipleMatchTargetList</Parameter>
  <Parameter name="MultipleMatchTargetRelevanceList" direction="out"   eval="variable"
type="arraylist">multipleMatchRelevanceList</Parameter>
  <Parameter name="NoMatchSourceRecordList" direction="out" eval="variable"
  type="arraylist">noMatchSourceList</Parameter>
  <Parameter name="OutDocument" direction="out" eval="variable"
  type="document">workDoc</Parameter>
  <Parameter name="MatchRecordProcessLogID" direction="out" eval="variable"
```

```
 type="long">update_step</Parameter>
 <Parameter name="CrossRepositoryMatch" direction="out" eval="variable"
type="boolean">isCrossRepoMatch</Parameter>
</Activity>
```

MergeRecord Activity

The MergeRecord activity allows you to merge two different records from the same repository.

You can merge data based on specific attributes, which can be specified using the SkipMergeAttributeList or AllowMergeAttributeList parameters. These attributes are evaluated using the EvaluateRulebase activity.

If more than one match target records are found, a work item is created. For multiple records merge, the CreateWorkitem activity is used to create a work item. To merge records using matcher work items, see the Work Item Page section of *TIBCO MDM Customization*.

You can also merge related records. However, related record's repository at each hierarchy level must match.

Prerequisite for this activity is the MatchRecord activity except for the Legacy mode. For more information about various modes of the MergeRecord activity and their usage, see the MergeRecord activity section of the*TIBCO MDMCustomization*.

MergeRecord Parameters and Valid Execution Modes

The valid execution mode for MergeRecord is SYNCHR.

The parameters of MergeRecord are as follows:

MergeRecord Parameters

Name	Semantics	Type	Valid Values	Multi-plicity
Direction: In				
AllowMergeAttributeList(Optional)	List of record attributes to be merged. You can specify either the SkipMergeAttributeList or the AllowMergeAttributeList parameter. If both the parameters are specified, an error message is displayed. Required to pass the matchRecordProcessLogID as ReferenceStepID.	ArrayList	Any valid attribute name.	0..1
SkipMergeAttributeList(Optional)	List of record attributes to be skipped from merging. You can specify either the SkipMergeAttributeList or the AllowMergeAttributeList parameter. If both the parameters are specified, an error message is displayed. Required to pass the matchRecordProcessLogID as ReferenceStepID.	ArrayList	Any valid attribute name.	0..1
InDocument(Optional)	The document that contains the source record to be merged. • If InDocument is not specified, the record from the current event is used for merging. • If the document contains relationships, the source relationships are not merged. However, target record relationships are always maintained.	Document	Any valid mIXML document.	0..1
IncludeDraftRecords(Optional)	See Common Parameters.	Boolean	TRUE (Default) FALSE	0..1
ReferenceStepID (Mandatory only in case of Bulk Import, otherwise Optional)	The ReferenceStepID is mapped to the MatchRecordProcessLogID out parameter of the MatchRecord activity.	Long	Any valid integer	0 or 1
RejectFlag(Optional)	• When you add a record and if this parameter is set to TRUE, the source record is rejected after merging the data.	Boolean	TRUE	0..1

Name	Semantics	Type	Valid Values	Multi-plicity
	When you modify a record and if the source record is in the CONFIRMED state, the state cannot be changed to REJECTED. • If this parameter is set to FALSE, the source record is merged with the target record, but it is not rejected. However, in the case of a Modify record, its version is changed.		(Default) FALSE	
SaveFlag (Optional)	Applicable for all modes, such as, Bulk, InDocument, and Legacy. • If this parameter is set to SAVE, the target record is saved after it is merged. • If this parameter is set to NOSAVE, ◦ Target record is not saved after it is merged. ◦ By default, the OutDocument is generated. That is, merged data is returned in the mlXML format. The OutDocument is generated only for a single record merge single record hierarchy merge. ◦ The FILE attributes are not merged in the resultant OutDocument.	String	SAVE (Default) NOSAVE	0..1
SourceRecordKey(Optional)	The record key of the source record to be merged. • This parameter accepts only one record key as input. • If InDocument is specified, the SourceRecordKey must match one of the records from the InDocument. • If the source record key is not found in the InDocument or the current event, an exception is not thrown and the activity skips processing.	String	Any valid record key.	1
TargetRecordKey (Optional)	The record key of the target record with which the source record must be merged. • This parameter accepts only one record key as the input. • If the target record key is not found, an exception is not thrown and the activity skips processing.	String	Any valid record key.	1
VersionOption(Optional)	See Common Parameters. Only applicable in case of a single record add.	String	CONFIRMED (Default) LATEST	0..1
ImplicitRelationshipMerge (Optional)	Indicates if any non-matching relationships must be merged between source and target record(s). This works only for direct merge. For more information about using the Implicit Relationship Merge, see *TIBCO MDM Customization*.	Boolean	True (Default) False	0..1
MergeRecordUsingLegacyProcess	If this parameter is set to TRUE, you can merge only one level of data. You cannot merge hierarchy data. For relationship	Boolean	True	0..1

Name	Semantics	Type	Valid Values	Multi-plicity
(Optional)	merge, this feature is backward compatible with 8.0. Using this parameter, you can specify target and source records directly without invoking MatchRecord activity.		False (Default)	
MergeTargetInOutdoc (Optional)	Use to replace the source record values with the target record values. By default, the Outdoc consists of the source record values.	Boolean	True False (Default)	1
Direction: Out				
OutDocument	Merged data is returned in a format specified by the user.	Document	N/A	0..1
RecordKey	Contains record key IDs of the matching records merged. Not applicable in case of a single record.	Arraylist		0..1

Example for MergeRecord Activity

Examples for MergeRecord activity include:

- Single record merge using direct merge
- Single record merge from work item
- Bulk record merge using direct merge
- Bulk record merge using direct merge

Single Record Merge Using Direct Merge

```
<Activity Name="DirectMerge">
<Action>MergeRecord</Action>
<Description>Merge record data</Description>
<Execution>SYNCHR</Execution>
<!--Parameter name="ReferenceStepID" direction="in" eval="variable" type="long">update_
step</Parameter-->
<Parameter name="InDocument" direction="in" eval="variable"
type="document">workDoc</Parameter>
<Parameter name="SourceRecordKey" direction="in" eval="xpath"
source="/Message/Body/Document/BusinessDocument/CatalogAction/CatalogActionDetails/Catal
ogItem[LineNumber=1]/PartNumber/GlobalPartNumber/ProdID/DBID/text()"
type="string">inDoc</Parameter>
<Parameter name="TargetRecordKey" direction="in" eval="variable"
type="string">MatchingRecordKey</Parameter>
<Parameter name="VersionOption" direction="in" eval="constant"
type="string">LATEST</Parameter>
<Parameter name="SaveFlag" direction="in" eval="constant" type="string">SAVE</Parameter>
<Parameter name="SkipMergeAttributeList" direction="in" eval="variable"
type="arraylist">skipMergeAttributeList</Parameter>
<Parameter name="eventState" direction="in" eval="constant"
type="string">MERGERECORD</Parameter>
<Parameter name="OutDocument" direction="out" eval="variable"
type="document">workDoc</Parameter>
<Parameter direction="in" name="RejectFlag" type="boolean"
eval="constant">TRUE</Parameter>
<Parameter name="RecordKey" direction="out" eval="variable"
type="arraylist">RecordKey1</Parameter>
</Activity>
```

Single Record Merge from Work Item

```
<Activity Name="MergeFromWI">
<Action>MergeRecord</Action>
<Description>Merge record from workitem</Description>
<Execution>SYNCHR</Execution>
<Parameter name="ReferenceStepID" direction="in" eval="variable" type="long">update_
step</Parameter>
<Parameter name="InDocument" direction="in" eval="variable"
type="document">inDoc</Parameter>
<Parameter name="VersionOption" direction="in" eval="constant"
type="string">LATEST</Parameter>
<Parameter name="SaveFlag" direction="in" eval="constant" type="string">SAVE</Parameter>
<Parameter name="eventState" direction="in" eval="constant"
type="string">MERGERECORD</Parameter>
<Parameter name="OutDocument" direction="out" eval="variable"
type="document">workDoc</Parameter>
<Parameter name="RecordKey" direction="out" eval="variable"
type="arraylist">RecordKey1</Parameter>
</Activity>
```

Bulk Record Merge Using Direct Merge

```
<Activity Name="MergeRecordBulk">
<Action>MergeRecord</Action>
<Description>Merge record data</Description>
<Execution>SYNCHR</Execution>
<Parameter name="eventState" direction="in" eval="constant"
type="string">MERGERECORD</Parameter>
<Parameter name="ReferenceStepID" direction="in" eval="variable" type="long">update_
step</Parameter>
<Parameter name="RecordKey" direction="out" eval="variable"
type="arraylist">RecordKey1</Parameter>
</Activity>
```

Bulk Record Merge from Work Item

```
<Activity Name="MergeRecordBulkFromWI">
<Action>MergeRecord</Action>
<Description>Merge record data</Description>
<Execution>SYNCHR</Execution>
```

```
<Parameter name="eventState" direction="in" eval="constant"
type="string">MERGERECORD</Parameter>
<Parameter name="ReferenceStepID" direction="in" eval="variable"
type="long">MatcherProcessLogID</Parameter>
<Parameter name="InDocument" direction="in" eval="variable"
type="document">inDoc</Parameter>
<Parameter name="SourceRecordKey" direction="in" eval="constant"
type="string">123</Parameter>
<Parameter name="TargetRecordKey" direction="in" eval="constant"
type="string">456</Parameter>
<Parameter name="VersionOption" direction="in" eval="constant"
type="string">LATEST</Parameter>
<Parameter name="SaveFlag" direction="in" eval="constant" type="string">SAVE</Parameter>
<Parameter name="OutDocument" direction="out" eval="variable"
type="document">workDoc</Parameter>
<Parameter name="RecordKey" direction="out" eval="variable"
type="arraylist">RecordKey1</Parameter>
</Activity>
```

Examples

Consider a case where the source record is:

Product ID	Extn	UOM	Record Type	Enterprise
X	X	PL	CUST	ACME

Consider a case where the target record is:

Product ID	Extn	UOM	Record Type	Enterprise	Record Version
Y	Y	CS	ADDR	MYACME	1

Case 1: The SkipMergeAttributeList and AllowMergeAttributeList parameters are not specified

The source record is merged with productid-extn and a new version is created.

Product ID	Extn	UOM	Record Type	Enterprise	Record Version
Y	Y	PL	CUST	ACME	2

Case 2: The SkipMergeAttributeList and AllowMergeAttributeList parameters are specified but no attributes in result

This is the same as Case 1, that is, the source record is merged with productid-extn and a new version is created.

Product ID	Extn	UOM	Record Type	Enterprise	Record Version
Y	Y	PL	CUST	ACME	2

Case 3: AllowMergeAttributeList parameter set for UOM

	Product ID	Extn	UOM	Record Type	Enterprise	Record Version
Source Record	X	X	PL	CUST	ACME	
Target Record	Y	Y	CS	ADDR	MYACME	1

The resultant record contains the source UOM:

Product ID	Extn	UOM	Record Type	Enterprise	Record Version
Y	Y	PL	ADDR	MYACME	2

Case 4: SkipMergeAttributeList parameter set for RECORD_TYPE

	Product ID	Extn	UOM	Record Type	Enterprise	Record Version
Source Record	X	X	PL	CUST	ACME	
Target Record	Y	Y	CS	ADDR	MYACME	1

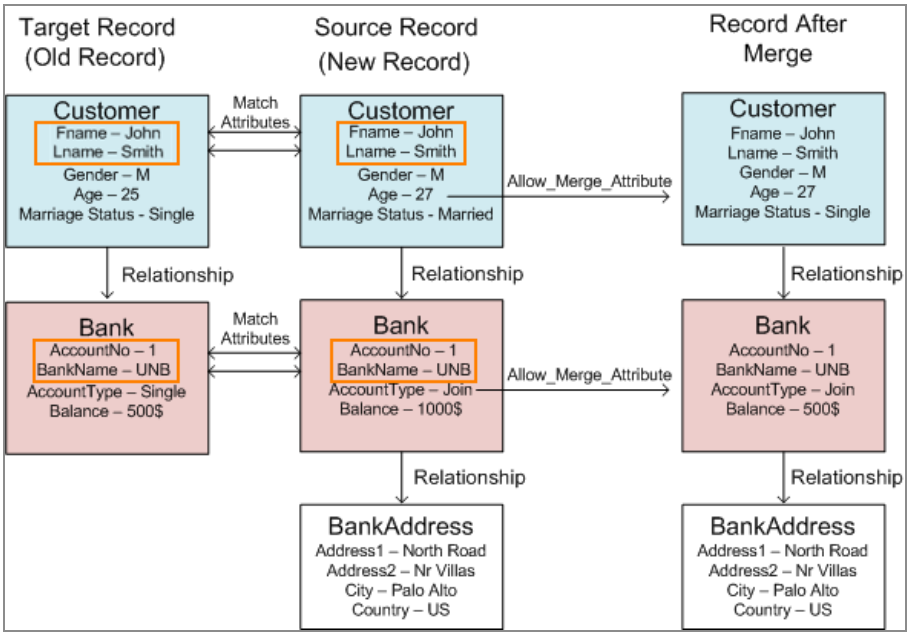
The resultant record carries the source Record_Type into the target.

Product ID	Extn	UOM	Record Type	Enterprise	Record Version
Y	Y	PL	ADDR	ACME	2

Case 5: Relationship Merge

Relationship Merge

Consider the following record hierarchy:



The resultant record contains the matching attributes of the source and target records, specified to allow merge attributes of the source record, and the relationship merge.

MergeForm

The MergeForm activity allows a results document (a work item output document format) to be merged into another document, and used within the workflow to control the process flow.

The MergeForm activity can also be used to merge any arbitrary form data into inDocument.

MergeForm Parameters and Valid Execution Modes

The valid execution mode for MergeForm is SYNCHR.

The parameters of MergeForm are as follows:

MergeForm Parameters

Name	Semantics	Type	Valid Values	Multiplicity
Direction: In				
InDocument (Mandatory)	See Common Parameters.			
InWiDocument (Mandatory)	A work item form result document which is generated when a work item is submitted.	document		1
Direction: Out				
OutDocument	See Common Parameters.			

Example for MergeForm Activity

```
<Activity Name="MergeResults">
  <Start/>
  <Action>MergeForm</Action>
  <Description lang="en">Merge results from    Backend.</Description>
  <Parameter direction="in" name="InDocument" type="document"
   eval="variable">inDoc</Parameter>
  <Parameter direction="in" name="InWiDocument" type="document"
  eval="variable">wiDoc</Parameter>
  <Parameter direction="out" eval="constant" type="document"
  name="OutDocument">outDoc</Parameter>
</Activity>
```

NoOperation Activity

The NoOperation activity does not perform any task, it is provided to create nodes which can be used to branch off workflows or to join multiple branches to a single point.

The NoOperation activity allows you to create transitions into or out of this activity from/to various points in the workflow.

For example, to execute a set of workflow activities based on document type, you can create a NoOperation activity to create a decision node in the workflow.

Another use of this activity is to add a new state attribute to the workflow by defining a new variable as input to this activity or to remove a state attribute by nullifying it.

NoOperation Parameters and Valid Execution Modes

The valid execution mode for NoOperation is SYNCHR.

The parameters of NoOperation are as follows:

NoOperation Parameters

Name	Semantics	Type	Valid Values	Multiplicity
Direction: In				
InDocument(Optional) (Deprecated since version 7.1)	See Common Parameters.			
Direction: Out				
OutDocument(Optional) (Deprecated since version 7.1)	See Common Parameters.			

Example for NoOperation Activity

```
<Activity Name="NoOperation">
<Action>NoOperation</Action>
<Execution>SYNCHR</Execution>
</Activity>
```

PrepareForImportCatalog Activity

Use the PrepareForImportCatalog activity to import records.

It works in conjunction with the following activities to complete the import process:

- ImportCatalogRecords
- EvaluateRuleBase Activity
- SaveRecord Activity

When the PrepareForImportCatalog activity is processed, the following actions are performed:

- Data is copied to the repository and classification staging tables.
- An identity is assigned to records, if you have specified the <assignidentity> parameter in rulebase.
- Basic data is validated. To perform additional data validations, specify the TRUE value for the ValidateOnly parameter.

For more information about these actions, see *TIBCO MDM Studio Rulebase Designer User's Guide*.

On completion, this activity calls the ImportCatalogRecords activity.

PrepareForImportCatalog Parameters and Valid Execution Modes

The valid execution mode for PrepareForImportCatalog is SYNCHR.

The parameters of PrepareForImportCatalog are as follows:

PrepareForImportCatalog Parameters

Name	Semantics	Type	Valid Values	Multi-plicity
Direction: In				
AsynProcessIndicator (Optional)	See Common Parameters.			
CatalogInputMap (Optional)	There can be multiple input maps associated with a catalog. Therefore, when importing a catalog, the input map name must be specified. An mlXML document always contains an input map name. So, even if this parameter is not present, the activity can use the one specified in mlXML.	string	Input Map Name.	0..1
ErrorThreshold (Optional)	This parameter is applicable only if a rulebase is applied. If the error count reaches this threshold, the import is terminated. The value can be set using the Configurator (**Repository** > **Catalog Import Error Threshold**). The order of the configuration is: Value specified in workflow file > Value specified in the Configurator > Default value.	long	Default is 1000.	0..1
InDocument(Mandatory)	See Common Parameters.			
MasterCatalog (Mandatory)	See Common Parameters.			
RecordPerAsyncCall (Optional)	See Common Parameters.			
Rulebase(Optional)	See Common Parameters. Use to update the PRODUCTID and PRODUCTIDEXT the staging tables, that is, to assign an identity. If the identity of a record is generated using a sequence number specified in the <assignidentity> action, only PRODUCTID is updated. To configure this, pass the rulebase. For example:	string	Existing Rulebase file.	1

Name	Semantics	Type	Valid Values	Multi-plicity
	<Parameter direction="in" name="Rulebase" eval="constant" type="string">D:\common\ac\rulebase/catalogvalidation.xml</Parameter> Using this rulebase, the activity performs the followings: • Allows assignments to PRODUCTID and PRODUCTIDEXT using the <assignidentity> action • Executes all actions defined in rulebase • Records the results in a error log file as rulebase messages However, the activity does not record the changes to data other than PRODUCTID and PRODUCTIDEXT.			
<ParameterNames> (Optional)	This parameter is applicable only if using a Rulebase. The values to input variables defined in the rulebase can be passed through activity parameters. The names of the parameters must match the declarations in the rulebase. <declare usage=”input”>	string	Must be declared as input variables in the rulebase.	0..n
ValidateOnly	Validates data and generates an error report and a log file. When you use the ValidateOnly parameter, it is expected that workflow does not continue to proceed for the ImportCatalogRecords activity. You must change the out-of-the-box workflow to terminate immediately after PrepareForImportCatalog activity. • If you specify TRUE for this parameter, you must specify the rulebase. • When you use this parameter, the activity does not write any data to the staging tables, it is only validated. • As this activity does not insert any rows to the database, it is expected to run fast. • As the staging data is not available to purge, the workflow must end after this activity.	Boolean	TRUE FALSE (Default)	

Example for PrepareForImportCatalog Activity

```
<Activity Name="PrepareForImportCatalog">
   <Action>PrepareForImportCatalog</Action>
   <Description lang="en">Import summary</Description>
   <Parameter direction="in" type="string" eval="constant"
   name="eventState">PREPAREFORIMPORT</Parameter>
   <Parameter direction="in" eval="xpath" type="string"     name="MasterCatalog"
   source="/Message/Body/Document/BusinessDocument/CatalogAction/CatalogAc    tionHeader/
MasterCatalog/RevisionID/BaseName/text()">inDoc</Parameter>
   <Parameter direction="in" eval="xpath" type="string"     name="CatalogInputMap"
   source="/Message/Body/Document/BusinessDocument/CatalogAction/CatalogAc    tionHeader/
MasterCatalog/InputMap/RevisionID/BaseName/text()">inDoc</Pa    rameter>
   <Parameter direction="in" name="InDocument" type="document"
   eval="variable">inDoc</Parameter>
   <!--Parameter direction="in" eval="constant" type="long"
   name="ErrorThreshold">100</Parameter-->
   <!--Parameter direction="in" eval="constant" type="long"
   name="RecordPerAsyncCall">100</Parameter-->
   <!--Parameter direction="in" name="AsynProcessIndicator" type="boolean"
   eval="constant">true</Parameter-->
   <!--Parameter direction="in" name="Rulebase" eval="constant"
   type="string">D:\common\a\rulebase\catalogvalidation.xml</Parameter-->
   <!--Parameter direction="in" name="ValidateOnly" eval="constant"
   type="boolean">true</Parameter-->
</Activity>
```

ProcessServiceMessage Activity

The ProcessServiceMessage activity works with the DataService messages. The DataService messages are primarily used to query or update system meta-data. DataService messages are also used for querying repository record data.

The type of entity (a business object) to be worked on (for example, repository, BusinessProcessRules, and so on) and the type of operation to be performed (for example, Query or Update) are determined by the DataService message.

ProcessServiceMessage Parameters and Valid Execution Modes

The valid execution mode for ProcessServiceMessage is SYNCHR.

The parameters of ProcessServiceMessage are as follows:

ProcessServiceMessage Parameters

Name	Semantics	Type	Valid Values	Multiplicity
Direction: In				
InDocument(Mandatory)	See Common Parameters.			
MetadataVersionCompatability	When exporting metadata, you can either use the default 8.0 format, or use the 7.2 format which is done by turning on this flag.	string		
Direction: Out				
LogFile(Optional)	A file path relative to MQ_COMMON_DIR to a log file, that contains information about how the DataService message was processed.	string		0..1

Name	Semantics	Type	Valid Values	Multiplicity
OutDocument(Optional)	See Common Parameters.			

Example for ProcessServiceMessage Activity

```
<Activity Name="ProcessServiceMessage"><Start/>
  <Action>ProcessServiceMessage</Action>
  <Description lang="en">Process the service message</Description>
  <Parameter direction="in" type="string" eval="constant"
   name="eventState">SERVICEMESSAGEPROCESS</Parameter>
  <Parameter direction="in" name="InDocument" type="document"
   eval="variable">inDoc</Parameter>
  <Parameter direction="out" eval="variable" type="document"
   name="OutDocument">workDoc</Parameter>
</Activity>
```

Purge Activity

The Purge activity allows you to delete historical data and record versions. This activity is a long-running activity and consumes significant resources in the database and Application Server.

It purges historical data based on a specific retention criteria, which is specified using RetentionUOM and RetentionUnits. Only redundant data is removed. Any records not associated with the history and corresponding synchronization records are also deleted.

Note: Version V1 of the repository record is never deleted. Records in the Master Catalog Table (MCT) match the History except that version V1 is always retained. A specific version for synchronization data is not retained.

A purge log file is created which includes details such as the enterprise (specific or all), catalog (specific or all), credentials that are used, objects that are deleted, name of the FILELIST file and so on.

Purge Parameters and Valid Execution Modes

The valid execution mode for Purge is SYNCHR.

The parameters of Purge are as follows:

Purge Parameters

Name	Semantics	Type	Multi-plicity
Direction: In			
DeleteRecordVersions (Optional)	Specifies whether records must be deleted. If this parameter is set to Yes, old record versions are deleted. To delete record versions, you must specify DeleteRecordVersions as Y in File Watcher and pass the DeleteRecordVersions parameter in the workflow. **Valid Values** • Y • Yes • N • No (Default)	string	0..1
MasterCatalog(Optional)	Specifies the repository for which record versions and historical data is to be purged. You can specify either a single repository name or ALL for all repositories. • Purge is limited to history and data only for the specified repository or all repositories of the specified enterprise when ALL is specified.	string	0..1

Name	Semantics	Type	Multi-plicity
	• You must specify an enterprise name if the repository name is specified. **Valid Values** Any valid repository name. Default is ALL.		
RetentionUnits(Optional)	Specifies the number of days or months from the current date beyond which data is purged. The cutoff date can also be the current date, that is, RetentionUnits can be 0. **Valid Values** Number greater than or equal to 0. Default is 6.	long	0..1
RetentionUOM(Optional)	Specifies the unit of measure for RetentionUnits. **Valid Values** MONTH (default) DAY	string	0..1
VersionsToRetain (Option)	Gets the counter of versions which qualify for delete. This is done to ensure that versions are retained.	long	0..1
StartDate (Optional)	Specifies the start date from which data is purged.	Time	0..1

Name	Semantics	Type	Multi-plicity
	Note: StartDate and EndDate always take precedence over the RetentionUnits, RetentionUOM parameters. **Valid Values** yyyy-MM-dd HH:mm:ss		
EndDate (Optional)	Specifies the end date till which the data is purged. **Valid Values** yyyy-MM-dd HH:mm:ss	Time	0..1
PurgeExecMode (Optional)	Specifies the execution mode of the purge. **Valid Values** history or historyForce The values are case sensitive.	string	0..1
PurgeEnterpriseOption (Optional)	Specifies that history of a specific enterprise must be purged. The credential specified must belong to the enterprise specified. If ALL is specified, the user who executes the workflow must be a valid user in TIBCOCIM enterprise. If this parameter is not specified, purge is performed for the enterprise of the user who executes the workflow. **Valid Values** ALL (Default)	string	0..1

Name	Semantics	Type	Multi-plicity
	Any enterprise name.		
Hints	Indicates how to optimize the purge if partitioning is already done. You can specify a certain data set or a table to be purged. For example, to skip data quality or match and merge feature, you can purge data quality tables or to skip master data, you can purge MCT tables. You must specify a hint character along with the -h option. For more information about hints and its characters, see Using Hints section in *TIBCO MDM System Administration*. **Valid Values** Any combination of hints	string	1
Interval	Instructs that purge must consider only those records which have changed in the specified interval (number of days). The parameter applies to only RecordVersions mode. For more information about using intervals, see Using Interval in the Record Versions section in *TIBCO MDM System Administration*. **Valid Values** Any valid integer. No default value. Value less than 1 is ignored.	long	1
InDocument(Optional)	See Common Parameters.		

Rules for the Purge Activity

The following rules are applied for purging:

- Only events with the SUCCESS, ERROR, and CANCELLED statuses are considered for purge.
- All logs for records with the SUCCESS, ERROR, and CANCELLED statuses and ERROR, INPROGRESS, and IDENTIFIED statuses or NOOPERATION sub-operation are purged.
- If two or more records exist with the SUCCESS status having the same operation in the repository and no channel or partner is involved, then the earlier of the two is purged and the latter is retained. Note that records in ERROR or CANCELLED states are purged automatically.
- All records with the status SUCCESS, ERROR, and CANCELLED and for Sync validate and preview are deleted. If two or more records exist with the SUCCESS status having the same operation in the repository with the same channel and partner combination, then the earlier of the two is purged and the later is retained. Note that records in ERROR or CANCELLED status are automatically purged.
- All process logs that do not have any product logs associated are purged. Before deleting, any associated documents are collected and stored in the *purgedDocument* table. Same is true for any associated record collections.
- All processes that do not have process logs are purged. ProcessDetail and ProcessState are automatically purged while Process is purged.
- All process logs which do not have any associated product logs are purged.
- All attribute logs that refer to any process log are purged.
- All events that do not have any processes associated are purged. EventDetail is automatically purged when Event is purged.
- All work items for which no processes exist are purged.
- All work items which do not have process logs are deleted. WorkItemDetail is automatically purged while WorkItem is purged.
- All associated documents and record collections are purged.
- All associated tables (eventdetail, workitemdetail, processdetail, processstate, conversationkey, recordList, recordCollectionDetail) are purged when the parent table is purged.

- For all product logs in the ERROR state older than cutoff date, error logs are purged.
- For all product logs earlier than the cutoff date, if the same operation is repeated after the cutoff date, the product logs are purged.
- All record collections which do not have a process log are purged. RecordsCollectionDetail and RecordList are automatically purged when RecordCollection is purged.
- General documents whose references do not exist in the process log are purged.
- When records are deleted, any associated relationships (Relationship table) and principal key entries (PrincipalKey table) are also deleted.
- Historical Data (Repository Records):
    - All records from MCT tables which do not have any product logs associated with them are deleted.
    - All records from multi-value tables and shared multi-value tables which do not have ProductKey and ModVersion in the respective repositories (MCT) tables are deleted.
    - For completed events (status as success, error, or cancelled) - error, in-progress, and redundant productlogs are deleted.
    - For in-progress events and deleted records, productlogs are not deleted.
    - If the repositories are specified in the FileWatcher, the records are deleted from only those MCT tables which do have any product logs associated with them.
    - Version V1 of any record is not deleted from the MCT.
    - All records from BCT tables which do not have a corresponding synchronization event or product log are deleted.
    - If a repository is specified, MCT and BCT records are deleted for the specified catalogs only. If no repository is specified, MCT and BCT records are deleted from all repositories across the enterprise.
    - To delete record versions, you must specify DeleteRecordVersions as Y in File Watcher and pass the DeleteRecordVersions parameter in the workflow.
    - To delete both history and data, you must define two activities: one for deleting history and one for deleting data. In one activity, do not include the

DeleteRecordVersions parameter. Transitions must always be from delete history to delete record versions and it must be conditional.

Note: Purge does not delete product logs for the inprogress events or deleted records.

Examples for Purge Activity

Example1

```
<Activity Name="Purge">
 <Start/>
 <Action>Purge</Action>
 <Description lang="en">Purge Historical Data</Description>
 <Parameter direction="in" type="string" eval="constant" name="eventState">Purge</Parameter>
 <Parameter
direction="in" eval="xpath" type="string" name="RetentionUOM" source
="/Message/Body/Document/BusinessDocument/CatalogAction/CatalogActionHeader/MasterCatalog/Extension
[@name='RetentionUOM']/Value/text()">inDoc</Parameter>
 <Parameter
direction="in" eval="xpath" type="string" name="RetentionUnits" source
="/Message/Body/Document/BusinessDocument/CatalogAction/CatalogActionHeader/MasterCatalog/Extension
[@name='RetentionUnits'/Value/text()">inDoc</Parameter>
 <Parameter direction="in" name="InDocument" type="document" eval="variable">inDoc</Parameter>
 <Parameter
direction="in" eval="xpath" type="string" name="DeleteRecordVersions" source
="/Message/Body/Document/BusinessDocument/CatalogAction/CatalogActionHeader/MasterCatalog/Extension
[@name='IncludeRecords']/Value/text()">inDoc</Parameter>
```

```
 <Parameter
direction="in" eval="xpath" type="string" name="EnterpriseName" source
="/Message/Body/Document/BusinessDocument/CatalogAction/text()">inDoc</Parameter>

</Activity>
```

Example 2

This example illustrates how to delete both history and data.

```
Activity 1:

<Activity Name="PurgeHistory">

 <Start/>

 <Action>Purge</Action>

 <Description lang="en">Purge Historical Data</Description>

 <Parameter direction="in" type="string" eval="constant"   name="eventState">PURGE</Parameter>

 <Parameter
direction="in" eval="xpath" type="string" name="RetentionUOM"   source
="/Message/Body/Document/BusinessDocument/CatalogAction/  CatalogActionHeader/MasterCatalog/Extension
[@name='RetentionUOM']/Value/  text()">inDoc</Parameter>

 <Parameter
direction="in" eval="xpath" type="long" name="RetentionUnits"   source
="/Message/Body/Document/BusinessDocument/CatalogAction/  CatalogActionHeader/MasterCatalog/Extension
[@name='RetentionUnits']/Value/  text()">inDoc</Parameter>

 <Parameter
direction="in" eval="xpath" type="string" name="EnterpriseName"   source
="/Message/Body/Document/BusinessDocument/CatalogAction/  CatalogActionHeader/MasterCatalog/Extension
[@name='EnterpriseName']/Value/  text()">inDoc</Parameter>
```

```
 <Parameter direction="in" name="InDocument" type="document"   eval="variable">inDoc</Parameter>

</Activity>

Activity 2:

<Activity Name="InitiatePurgeRecordVersions">

    <Action>Purge</Action>

    <Description lang="en">Initiate purge for record versions</Description>

     <Parameter direction="in" type="string" eval="constant" name="eventState">PURGERECORDVERSION</Parameter>

        <Parameter direction="in" type="string" eval="constant" name="PurgeExecMode">history</Parameter>

        <Parameter
direction="in" eval="xpath" type="string" name="RetentionUOM" source
="/Message/Body/Document/BusinessDocument/CatalogAction/CatalogActionHeader/MasterCatalog/Extension
[@name='RetentionUOM']/Value/text()">inDoc</Parameter>

        <Parameter
direction="in" eval="xpath" type="long" name="RetentionUnits" source
="/Message/Body/Document/BusinessDocument/CatalogAction/CatalogActionHeader/MasterCatalog/Extension
[@name='RetentionUnits']/Value/text()">inDoc</Parameter>

        <Parameter
direction="in" eval="xpath" type="string" name="DeleteRecordVersions" source
="/Message/Body/Document/BusinessDocument/CatalogAction/CatalogActionHeader/MasterCatalog/Extension
[@name='IncludeRecords']/Value/text()">inDoc</Parameter>

        <Parameter
direction="in" eval="xpath" type="string" name="VersionsToRetain" source
="/Message/Body/Document/BusinessDocument/CatalogAction/CatalogActionHeader/MasterCatalog/Extension
[@name='VersionsToRetain']/Value/text()">inDoc</Parameter>

     <Parameter direction="in" eval="xpath" type="string" name="MasterCatalog" source
="/Message/Body/Document/BusinessDocument/CatalogAction/CatalogActionHeader/MasterCatalog/RevisionID/BaseNam
e/text()">inDoc</Parameter>
```

```
        <Parameter direction="in" name="InDocument" type="document" eval="variable">inDoc</Parameter>

        <Parameter
direction="in" eval="xpath" type="long" name="Interval" source
="/Message/Body/Document/BusinessDocument/CatalogAction/CatalogActionHeader/MasterCatalog/Extension
[@name='Interval']/Value/text()">inDoc</Parameter>

    <!-- <Parameter direction="in" name="PurgeEnterpriseOption" type="string" eval="constant">ALL</Parameter> -->

</Activity>
```

Transition:

```
<Transition FromActivity="UpdatePurgeEvent" ToActivity="InitiatePurgeRecordVersions">

        <Description>Use Delete Forms if action is Delete.</Description>

        <Rule>

            <Parameter
name="IncludeRecords" type="string" eval="xpath" source
="/Message/Body/Document/BusinessDocument/CatalogAction/CatalogActionHeader/MasterCatalog/Extension
[@name='IncludeRecords']/Value/text()" direction="in">inDoc</Parameter>

            <Parameter name="result" type="boolean" direction="out"/>

            <Condition format="java"><![CDATA[com.tibco.mdm.workflow.engine.transition.WfSharedConditionTransition.isYes
(IncludeRecords);

            </Condition>

        </Rule>

    </Transition>
```

PurgeStaging Activity

The PurgeStaging activity deletes all the records from the staging table for each input map associated with the import for the event.

The activity can purge the data in a synchronous or asynchronous call to delete query. This Activity is executed after ExtractRelationshipActivity.

> **Note:** This is done based on the event ID so that all the events which terminate at ImportRecords either due to user action or by any other means are not cleaned up.

PurgeStaging Parameters and Valid Execution Modes

The valid execution mode for PurgeStaging is SYNCHR.

The parameters of PurgeStaging are as follows:

PurgeStaging Parameters

Name	Semantics	Type	Valid Values	Multi-plicity
Direction: In				
MasterCatalog (Mandatory)	See Common Parameters.			
CatalogInputMap (Optional)	There can be multiple input maps associated with a catalog. So, when importing a catalog, the input map name must be specified. An mIXML document	string	Input Map Name	0..1

Name	Semantics	Type	Valid Values	Multi-plicity
	always contains an input map name, so even if this parameter is not present, the activity can use the one specified in mIXML.			
InDocument(Mandatory)	See Common Parameters.			
AsynProcessIndicator (Optional)	See Common Parameters.			
RecordPerAsyncCall (Optional)	See Common Parameters.			
eventState(Optional)	The latest state of the workflow.	string	Predefined state: DONE. Custom states can be defined in the DOMAINENTRY table	0..1

Example for PurgeStaging Activity

```
<Activity Name="PurgeStaging">
  <Action>PurgeStaging</Action>
  <Description lang="en">Purge Staging Data</Description>
  <Parameter direction="in" type="string" eval="constant"
   name="eventState">PURGE</Parameter>
  <Parameter direction="in" eval="xpath" type="string" name="MasterCatalog"
  source="/Message/Body/Document/BusinessDocument/CatalogAction/CatalogActio  nHeader/M
asterCatalog/RevisionID/BaseName/text()">inDoc</Parameter>
```

```
  <Parameter direction="in" eval="xpath" type="string" name="CatalogInputMap"
  source="/Message/Body/Document/BusinessDocument/CatalogAction/CatalogActio  nHeader/M
asterCatalog/InputMap/RevisionID/BaseName/text()">inDoc
  </Parameter>
  <Parameter direction="in" name="InDocument" type="document"
  eval="variable">inDoc</Parameter>
  <Parameter direction="in" name="AsynProcessIndicator" type="boolean"
  eval="constant">false</Parameter>
</Activity>
```

ReclassifyRecord Activity

The ReclassifyRecord activity reclassifies all records for a classification scheme of a repository. This activity is to be used when a classification scheme is uploaded.

If the classification scheme is a pre-defined classification scheme, all catalogs associated with the classification scheme are reclassified.

This activity is also invoked when importing records to extract classification schemes for all catalogs in the import. It is invoked by the ExtractRelationship Activity if the ImportClassificationScheme parameter is set to true.

ReclassifyRecord Parameters and Valid Execution Modes

The valid execution mode for ReclassifyRecord is SYNCHR.

The parameters of ReclassifyRecord are as follows:

ReclassifyRecord Parameters

Name	Semantics	Type	Multiplicity
Direction: In			
ImportClassificationScheme	This parameter extracts classification schemes for records being imported.	boolean	0..1

Name	Semantics	Type	Multiplicity
	This parameter is only used when the ReclassifyRecord activity is invoked as part of the import process. Valid values are True and False.		

Example for ReclassifyRecord Activity

```
<Activity Name="ReclassifyRecordsForImport">
   <Action>ReclassifyRecord</Action>
   <Description lang="en">Extract all Classification Schemas for Import</Description>
   <Parameter direction="in" type="string" eval="constant"
   name="eventState">IMPORTEXTRACTCLASSIFICATION</Parameter>
   <Parameter direction="in" name="ImportClassificationScheme" type="boolean"
eval="constant">true</Parameter>
   <Parameter direction="in" eval="xpath" type="string"     name="MasterCatalog"
   source="/Message/Body/Document/BusinessDocument/CatalogAction/CatalogActionHeader/
MasterCatalog/RevisionID/BaseName/text()">inDoc</Parameter>
</Activity>
```

Note: The Reclassify activity does not support failover.

RefreshSubset Activity

The RefreshSubset activity refreshes the subset definition. The activity only applies to subsets of type “Subset by list”.

A subset based on a list is defined on a data source. The RefreshSubset activity reads the data from data sources and changes the subset definition to use this new data. Subset by list is a *statically* evaluated subset, that is, after the subset is defined, the records included in the subset do not change. Subset by list includes records by matching the record attributes with the data loaded through the data source. However, this data is *copied* into the subset and any change to data source does not change the

subset. RefreshSubset activity provides a way to update such definitions. Any subsequent EvaluateSubset activity uses the updated subset definition. The activity is atomic and can be included in part of in-memory workflows. The activity does not spawn additional threads. The activity identifies the subset that is to be refreshed using the following parameters:

RefreshSubset Parameters and Valid Execution Modes

The valid execution mode for RefreshSubset is SYNCHR.

RefreshSubset Parameters

Name	Semantics	Type	Valid Values	Multiplicity
Direction: In				
eventState	Determines the latest state of the workflow.	String	SUBSET	0..1
Subset (Optional)	Subset rule name. If the Subset parameter is specified in the EvaluateSubset activity, the activity identifies the subset ID. Do not specify the subset name. In this case, InDocument is not required.	String	Name of the subset rule.	0..1
InDocument	If the Subset parameter is specified, InDocument is not required. See Common Parameters.			
MasterCatalog (Optional)	See Common Parameters			

Example for RefreshSubset Activity

The sample workflows which are used in RefreshSubset activity are available in *$MQ_COMMON_DIR*/standard/samples/workflow/ProcessRecordsService.

RefreshSubset Workflow Samples

Workflow File Name	Description
wfin26catprocessrecordsamplesv1.xml	This is a sample workflow to demonstrate how ProcessRecordService can be used to validate a set of records using a subset. The data source specifies the records to be validated. Subset must be defined as "subset by list" so that it can accept the data source.
wfin26catprocessrecordsample2v1.xml	This sub process deletes records specified through the data source. The data loaded from data source is matched against subset definition. Subset must be defined as "subset by list" so that it can accept the data source. The records matched by subset are deleted and confirmed.
wfin26catprocessrecordsample3v1.xml	This is a sample process to show how ProcessRecordService can invoke the workflow. It performs no operation.

RestartEvent Activity

The RestartEvent activity allows restart of any workflow if the workflow was previously suspended. Attempts to restart a workflow which was not suspended is illegal though not prohibited.

Additionally, if workflow is suspended in HandleProtocolMessaging activity, restart is not allowed as the restart would lead to incorrect synchronization.

The activity accepts an EventID as input to provide flexibility of determining which event to restart. Best practice for a child workflow to execute is to use GetAssocEventsSummary activity to find the parent event of the event to restart. You

can also determine the event ID using customizations or by explicitly passing the event ID as input to the workflow.

The restart activity can initiate the event restart only if

1. The event is a valid event ID.
2. The event is not complete, that is event status is not ERROR, CANCELLED or SUCCESS.
3. The status of the process associated with the event is not CANCELLED, COMPLETED or SUCCESS.
4. The process is not already executing.
5. The process is not executing and lock can be obtained. Normally, if a process is not executing, it can be locked. However, certain application failures (complete cluster failure) might leave the process locked. In this case, a manual intervention is required to correct the process state and then restart it from UI. In case of such an abnormal failure, the locked processes can be checked by looking up table PROCESS (STATUS=WAIT_TASK).

The activity handles exceptions as follows:

- If the event (or more precisely, the associated process) is still running, the process can be not locked and restarted. Process must be locked before it can be restarted. The restart is attempted again for a specified retry count, if retry count is more than 1. Activity sleeps for a specified interval between restart attempts. Default retry count is 3 and sleep interval is 10000 milliseconds.
- If there is an exception during restart, activity throws an exception. If there is any error transition specified in the workflow, error transition is invoked.
- If a) and c) is the case, the output InitiatedEventID is null. Otherwise InitiatedEventIDis set as the event which was restarted.

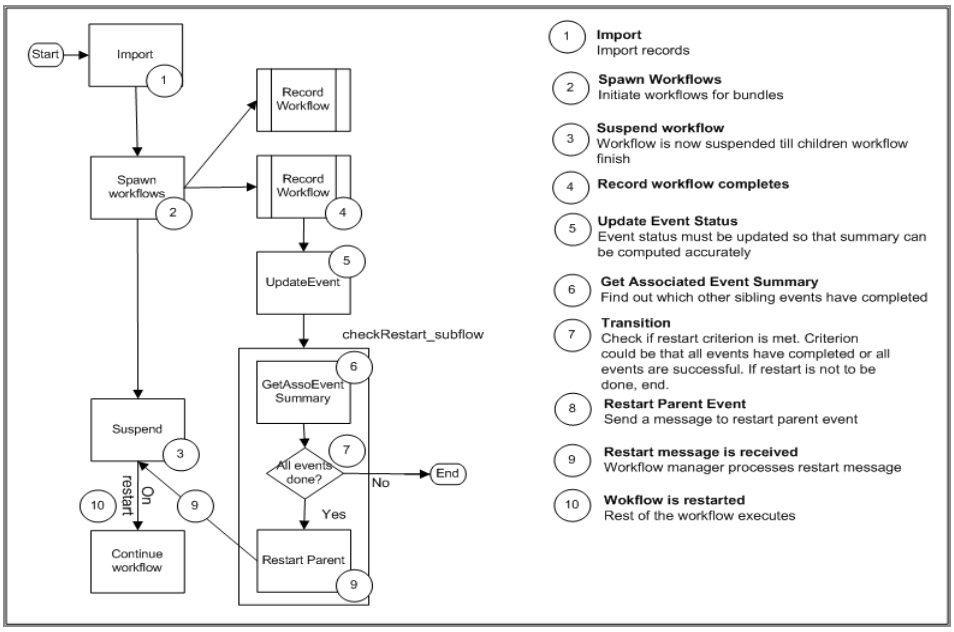
The RestartEvent activity works in tandem with the Suspend activity. See the supplied samples wfin26catsourcev5_suspend_sample.xml and wfin26producteditapprovalv3_restart_sample.xml under $MQ_HOME/ common/standard/samples/workflow. Following is an example of the use case which includes this activity.

- Import is initiated with "Split/ Approval Required".
- Import batch is saved and depending on the action to be taken record add, edit or delete workflows are initiated using SpawnWorkflow activity. Immediately after import the workflows are initiated, import workflow suspends using Suspend

activity.

- Each of the workflow goes through processing and reaches an end state. When the end state is reached, event status is updated to SUCCESS or ERROR, and a subflow is initiated to check for restart.
- The subflow first obtains the summary of all events spawned. Based on the summary, restart can be initiated.

RestartEvent Activity Process



RestartEvent Parameters and Valid Execution Modes

The valid execution mode for RestartEvent is SYNCHR.

The parameters of RestartEvent are as follows:

RestartEvent Parameters

Name	Semantics	Type	Valid Values	Multiplicity
Direction: In				

Name	Semantics	Type	Valid Values	Multiplicity
EventID (Mandatory)	Event which is to be restarted.	long	Valid event ID	1
RetryCount (Optional)	The restart is attempted as many times as the retryCount.	long	Any number more than 0	1
Interval(Optional)	It is specified in milliseconds. Indicates the interval between each attempt to restart.	long	>0, recommended value 10 seconds or more. Default 10000ms	1
Direction: Out				
InitiatedEventID	If event is restarted, event ID is output otherwise null	long	Valid event ID	1

Example for RestartEvent Activity

```
<Activity Name="RestartParentEvent">
                <Action>RestartEvent</Action>
          <Description lang="en">Restart parent event </Description>
          <Parameter direction="in" name="EventID" type="long"
eval="variable">parentEventID</Parameter>
     </Activity>>
```

SaveRecord Activity

The SaveRecord activity reads the record information from the document or the workflow, and updates or adds the corresponding records into the database to the specified repository.

The records information includes:

- Record Attributes
- Relationships
- Relationship Attributes
- Relationship Target Records
- Classifications – The SaveRecord activity extracts the classification data for each record from the mlXML document and adds to the created record item.

If the document is not provided, a name-value pair can be provided as a parameter, which is used to update that record in the database. If both document and name-value pair are provided, the name-value pair information overrides the information in the document.

SaveRecord Parameters and Valid Execution Modes

The valid execution mode for SaveRecord is SYNCHR or ASYNCHR.

The parameters of SaveRecord are as follows:

SaveRecord Parameters

Name	Semantics	Type	Valid Values	Multi-plicity
Direction: In				
ApplyValidation (Optional)	Indicates whether or not to run the standard rulebase (catalogvalidation.xml) for a repository when the Rulebase parameter is not specified. The parameter is ignored if the Rulebase parameter is specified.	Boolean	True False (Default)	0..1
AsynProcessIndicator (Optional)	See Common Parameters.			
Comment(Optional)	Comment to be stored in the productlog message. This message is shown on the product history page.	string		0..1
ErrorSeverity(Optional)	See Common Parameters.			
InDocument(Optional)	See Common Parameters.			
MasterCatalog(Optional)	This parameter must be specified if InDocument is specified. The activity tries to get the repository name in the following order: From InParameter From InDocument An exception is thrown and the workflow quits if the: • repository name is not found in both InParameter and InDocument. • repository name is found but the catalog does not exist. If InDocument is not passed, it is mandatory to pass the repository name as a parameter to get the MCT_ID.	string		0..1
OverrideConflict (Optional)	If true, overrides the conflicting attributes else, conflict is fired.	boolean	True False (Default)	0..1

Name	Semantics	Type	Valid Values	Multi-plicity
Overwrite(Optional)	Overwrite flag indicates that data being saved overwrites any concurrent updates being done by other processes if updates are attempted at the same time.	boolean	True False (Default)	0..1
PerspectiveName (Optional)	See CreateWorkItem Parameters and Valid Execution Modes.			
ProductExtns(Optional)	Used when the ProductIds parameter is used. Indicates the extensions of the records to be processed.	arraylist		0..1
ProductIds(Optional)	Indicates the product IDs of the records to be processed. This is a required parameter only when you want to process the record from the input. The attribute value can also be provided as an arraylist. However, the attribute value arraylist size must be equal to the product ID list size.	ArrayList		0..1
Reason(Optional)	The reason to be stored in the productlog message. This message is shown in the product history page.	string		0..1
RecordPerAsyncCall (Optional)	See Common Parameters.			
Rulebase(Optional)	See Common Parameters. If this parameter is not specified, the activity executes the standard validations.			
RetainExistingRelati onships	If InDocument is used as an input parameter, all the old relationships are removed if they are not part of the document. However, if you specify RetainExistingRelati onships to true, the existing relationships are retained, which are not part of the current document.	boolean	True False (Default)	0..1
StandardValidation (Optional)	Whether to do Standard Validations in addition to those specified in Rulebase.	boolean	True False (Default)	0..1
RecordState(Optional)	Specifies the state of a record. Valid values are DRAFT, CONFIRMED, or UNCONFIRMED. This parameter replaces the Status parameter. Since 8.3.0 release, the Status parameter has been deprecated. For backward compatibility, if you do not specify the RecordState parameter, the Status parameter is searched and used. However, for best results, specify the RecordState parameter.	string	DRAFTCONFIRMEDUNCONFIRMED (Default)	0..1
VersionOption(Optional)	See Common Parameters.			

Name	Semantics	Type	Valid Values	Multi-plicity
VersionPolicy (Optional)	See ImportCatalogRecords Parameters and Valid Execution Modes.			
Direction: Out				
OutDocument (Deprecated)	See Common Parameters. For backward compatibility, outDocument is the same as InDocument.			
RecordsAttempted (Optional)	Number of records attempted to update.	long		
RecordsWithErrors (Optional)	Number of records that had an error during save. Error could occur during save, relationship save, or classification extraction.	long		
RecordsWithWarnings (Optional)	Number of records that had a warning during save. A warning could occur during save, relationship save, or classification extraction.	long		
StaleDataError(Optional)	The StaleDataError is thrown from concurrent record updates by multiple users/workflow threads. If this parameter is specified and if the StaleDataError occurs, this parameter is set to true and the workflow continues. If this parameter is not specified and there is an error, an exception is thrown. To resolve the StaleDataError, select true for the Enable Save Lock (com.tibco.cim.record.useSaveLock) property in the Configurator. The property enables locks to avoid concurrent modification of parent and its child record. Additionally, you can specify the time period for which the user must wait to save the record that is concurrently being modified by another user. Specify the time period value for the Specify Timeout property in the Configurator. By default, the time period is 300.	boolean	True False (default)	

Example for SaveRecord Activity

```
<Activity Name="SaveRecord">
 <Start/>
 <Description lang="en">Process Record and generate an output   document.</Description>
 <Action>SaveRecord</Action>
 <Parameter direction="in" name="InDocument" type="document"
 eval="variable">work1Doc</Parameter>
 <Parameter direction="in" name="Rulebase" type="string"
 eval="constant">standard/rulebase/rbtest.xml</Parameter>
 <Parameter direction="in" name="Severity" type="long"   eval="constant">9</Parameter>
 <Parameter direction="out" eval="variable" type="long"
 name="RecordsAttempted">workDoc</Parameter>
 <Parameter direction="out" eval="variable" type="long"
 name="RecordsWithErrors">errorsFound</Parameter>
 <Parameter direction="out" eval="variable" type="document"
 name="OutDocument">workDoc</Parameter>
 <Parameter direction="in" name="RecordPerAsyncCall" type="long"
 eval="constant">10</Parameter>
 <Parameter direction="in" name="AsynProcessIndicator" type="boolean"
 eval="constant">true</Parameter>
</Activity>
```

Send Activity

The Send activity is used to send a document/message to an email recipient or to an HTTP server.

In the case of an email, the send activity lets you specify the look of the message as a compiled HTML page, and allows you to set parameters that can be substituted into the message body at runtime (for example, sender name). For e-mails, it is not necessary to specify a document object to send along, but if you do so, the document is attached to the email. For HTTP, you can only specify a document, which is then posted to the specified server.

Execution mode is ignored if the protocol used is JMS. In ASYNCHR mode the send is done in a separate execution thread. In asynchronous mode, if any errors happen during the send they cannot be handled in the workflow and step might show up as successful even if actual send failed.

Send Parameters and Valid Execution Modes

The valid execution mode for the Send activity is SYNCHR or ASYNCHR.

The parameters of Send are as follows:

Send Parameters

Name	Semantics	Type	Valid Values	Multi-plicity
Direction: In				
Protocol (Mandatory)	The protocol by which to send the message.	string	SMTP HTTP FTP	1..n
Address(Optional)	The address to send the message to. It can be either for a catalog or a file. For Catalog FTP on demand side two delivery targets are possible: **None**: The catalog is ftp-ed to ftp address provided in the company profile.	string	Email or FTP address	1..n
UserAddress (Optional)	The numeric ID of the user who is to receive email.	long	Any ID that maps to a user's database ID. Only valid for Protocol = SMTP.	0..n
Sender(Optional)	The email address of the sender for	string		0..1

Name	Semantics	Type	Valid Values	Multi-plicity
	protocol=SMTP. Unused for HTTP.			
Subject(Optional)	The subject of the email message for protocol=SMTP. Unused for HTTP.	string		0..1
Presentation (Optional)	The class name of an XMLC compiled HTML page to use as the email body when protocol=SMTP. Unused for HTTP.	string	Any fully qualified XMLC class name.	0..1
AttachDocument (Optional)	Flag indicating that InDocument is attached to the email message. InAttachments are always attached, regardless of the value of this flag.	boolean	True (Default) False	0..1
InDocument (Optional)	See Common Parameters. When protocol = SMTP and this parameter is specified, the corresponding document is attached to the email. The parameter is ignored when protocol=HTTP.	document		0..1
InAttachments	Attachments to show for	document or	The list of	0..n

Name	Semantics	Type	Valid Values	Multi-plicity
(Optional)	this work item.	string	attachments to include in the work item.	
WorkflowPrority (Optional)	See Common Parameters. If the priority is not specified, event priority is inherited by the new message.			

In addition to the mentioned parameters, when the protocol is SMTP you can specify any number of string parameters. The send activity looks for a tag inside the presentation page with an ID corresponding to the name of each of these parameters and set the value of that tag to the value of the parameter.

The following example sends an import completion notification email to users, evaluated using the ‘Message or Workflow Completion’ rule. The presentation document com.tibco.mdm.ui.workflow.engine.emailtemplates.EmailSourceMessage expects input for the element ID named OperationStatus. You must specify this element value as a parameter to the send activity.

If you supply an anchor tag called “inboxUrl” in the presentation document, the send activity additionally replaces the href attribute of the link with the address of the Inbox, as specified in the Configurator. This behavior is similar to what happens when a work item notification is created in the CreateWorkItem activity.

Example for Send Activity

```
<Activity Name="SendImportCompletionNotification">
  <Action>Send</Action>
  <Description lang="en">Send import completion notification</Description>
  <Execution>ASYNCHR</Execution>
  <Parameter direction="in" type="string" eval="constant"
  name="eventState">SENDSUCCESSEMAIL</Parameter>
```

```
  <Parameter direction="in" eval="rule" source="Message or Workflow    Completion" type="string"
name="Protocol">inDoc</Parameter>
  <Parameter direction="in" eval="rule" source="Message or Workflow    Completion" type="string"
name="Address">inDoc</Parameter>
  <Parameter direction="in" name="Sender" type="string"
  eval="constant">support@tibco.com</Parameter>
  <Parameter direction="in" name="Presentation" type="string"
  eval="constant">com.tibco.mdm.ui.workflow.engine.emailtemplates.EmailSource   Message</Pa
rameter>
  <Parameter direction="in" eval="constant" type="string"
  name="Form">standard/forms/fm26ca.xml</Parameter>
  <Parameter direction="in" type="document" eval="variable"
  name="InDocument">inDoc</Parameter>
  <Parameter direction="in" type="string" eval="constant"
  name="OperationStatus">succeeded</Parameter>
  <Parameter direction="in" eval="constant" type="string" name="Message">The    master catalog
upload/import operation succeeded.</Parameter>
</Activity>
```

SendProtocolMessage Activity

The SendProtocolMessage activity performs a protocol specific send.

The SendProtocolMessage activity can use AS2 or JMS protocols to do a send. In case of JMS send, if a messageID is supplied, the application suspends after this step and waits for a response. A timeout interval can be introduced on the wait by adding the ExpiryDate parameter.

SendProtocolMessage Parameters and Valid Execution Modes

The valid execution mode for SendProtocolMessage is SYNCHR or ASYNCHR.

The parameters of SendProtocolMessage are as follows:

SendProtocolMessage Parameters

Name	Semantics	Type	Valid Values	Multi-plicity
Direction: In				
Address(Optional)	Receiver GLN.	arraylist		0..1
BizProtocol(Mandatory)	The business protocol to be used for communication.	string	JMS: for communicating with backend systems, EAI products and between multiple TIBCO MDM instances. INTERNAL_TRANSPORT: For communicating between two enterprises in the same instance of TIBCO MDM.	1
ChannelCredential(Optional)	Receiver GLN.	string		0..1
ChannelOrganizationName (Optional)	Name of the Receiver Organization.	string		0..1
ConversationKey(Optional)	Conversation ID.	string		0..1
ExpiryDate(Optional)	Relative or absolute time within which the response to this message must be received. If not received, this step causes a timeout event which can be handled in the workflow as in the outgoing transitions of this activity to do any exception processing.	string	See ExpiryType.	0..1
ExpiryType(Optional)	The type of expiry date.	string	RELATIVE or ABSOLUTE. In the first case, time is specified as DD:hh:mm:ss and in the second as MM:DD:hh:mm:ss where MM is month, DD is day, hh is hour, mm is minute, and ss is second.	0..1
InDocument	See Common Parameters.			
InDocument2	The mlXML document on which the InDocument was formed.	document		0..1
MessageID#(Optional)	MessageID for the outgoing message. MessageID is required if you want the	string		0..1

Name	Semantics	Type	Valid Values	Multi-plicity
	application to suspend after the send. Replace # with an integer, for example, MessageID1, MessageID2.			
PayloadPackagingScheme (Optional)	Determines the type of packaging to be used around the payload. The value is not used by code, but maps to a marshaling pipeline that packages the payload using the configured marshallers.	string	String. Currently only one value is supported: STANDARD_INTEGRATIONThis packages the payload into an ebXML/SOAP envelope.	0..1
ReceiverCredential(Optional)	The receiver GLN the activity uses while sending the document.	string	Valid credentials.	0..1
ReceiverOrganizationName (Optional)	Name of the Receiver organization as given in the 'marketplace profiles'. For example, '1SyncTest'. Required if Channel Credential is not supplied.	string		0..1
SenderCredential(Optional)	The sender GLN the activity uses while sending the document.	string	Valid credentials.	0..1
TrackMessageResponse (Optional)	Indicates if the activity must track the outgoing message. By using this parameter, you can set time-outs on message processing or receiving responses. This parameter works in conjunction with the execution mode. • If you specify True and the execution mode is SYNCHR, then workflow suspends. • If you specify True and the execution mode is ASYNCHR, then workflow does not suspend. • If you specify False and the execution mode is SYNCHR, then workflow does not suspend. • If you specify False and the execution mode is ASYNCHR, an incorrect input warning message is displayed and the workflow is proceeded considering SYNCHR the execution mode is specified.	boolean	• True (Default) • False	0..1
GenerateCommEvent (Optional) (applicable since version 7.1)	Indicates whether to generate an internal communication event. If not specified (default is false), no communication event is generated. If specified as True, a communication event is generated.	boolean	True False (Default)	0..1
DefaultDomain(Optional)	Specifies the domain the credentials belong to.	string	Valid domain	
SaveCachedDocument (Optional)	Useful only when document caching is enabled. Persists documents to disk by default (when set to true).	boolean	True (default) False	

Name	Semantics	Type	Valid Values	Multi-plicity
Direction: Out				
OutDocument(Optional)	See Common Parameters. The response document could be protocol specific.			

Example for SendProtocolMessage Activity

```
<Activity Name="PublishToEAI">
  <Action>SendProtocolMessage</Action>
  <Description>Publish the confirmed record to JMS for EAI to    pickup</Description>
  <Execution>ASYNCH</Execution>
  <Parameter direction="in" eval="constant" type="string"    name="BizProtocol">JMS</Parameter>
  <Parameter direction="in" name="InDocument" type="document"
  eval="variable">workDoc</Parameter>
  <Parameter direction="in" name="InDocument" type="document"
  eval="variable">inDoc</Parameter>
  <Parameter direction="in" name="DefaultDomain" type="string"
  eval="constant">DUNS</Parameter>
  <Parameter direction="in" name="SenderCredential" type="string"
  eval="constant">0065064183213</Parameter>
  <Parameter direction="in" name="ChannelCredential" type="string"
  eval="constant">0065064183213</Parameter>
  <Parameter direction="in" name="TradingPartnerCredential" type="string"
  eval="constant">0065068483364</Parameter>
  <Parameter direction="in" name="PayloadPackagingScheme" eval="constant"
  type="string">STANDARD_INTEGRATION</Parameter>
  <Parameter direction="in" name="GenerateCommEvent " eval="constant"
  type="boolean">false</Parameter>
  <Parameter direction="in" name="MessageID1"
  source="/Message/Body/Document/BusinessDocument/CatalogAction/CatalogActi   onDetails/C
atalogItem[1]/@key" type="string"    eval="xpath">inDoc</Parameter>
  <Parameter direction="in" eval="constant" type="string"
  name="ExpiryType">RELATIVE</Parameter>
  <Parameter direction="in" eval="constant" type="string"
  name="ExpiryDate">0:5:5:0</Parameter>
  <Parameter direction="out" name="OutDocument" eval="variable"
  type="document">EAIresponse</Parameter>
  <Parameter direction="in" type="string" eval="constant"
  name="eventState">SENDCATALOG</Parameter>
</Activity>
```

Sleep Activity

The Sleep activity sleeps for a specified interval and then wakes to complete.

Sleep Parameters and Valid Execution Modes

The valid execution mode for Sleep is SYNCHR.

The parameters of Sleep are as follows:

Sleep Parameters

Name	Semantics	Type	Valid Values	Multi-plicity
Direction: In				
Interval (Mandatory)	It is specified in milliseconds. It must be a whole number. Indicates the interval between each attempt to restart.	long	>0, for best results, 10 seconds or more.	1

Example for Sleep Activity

```
<Activity Name="SleepForAWhile">
   <Action>Sleep</Action>
   <Execution>SYNCHR</Execution>
   <Parameter name="Interval" direction="in" eval="constant"      type="long">3000</Parameter>
</Activity>
```

SpawnWorkflow Activity

The SpawnWorkflow activity initiates one or more new events. The events are routed through the WorkflowManagerConfig and follow all workflow routing rules.

The SpawnWorkflow activity accepts a template to generate Workflow request documents and a new workflow can be configured using this template.

This activity accepts a list of relationships as input parameters. This input parameter is optional and must be specified only to change the relationships included in the spawned workflow. If not specified, relationships specified in the input record collection are used.

SpawnWorkflow Parameters and Valid Execution Modes

The valid execution mode for SpawnWorkflow is ASYNCHR.

The parameters of SpawnWorkflow are as follows:

SpawnWorkflow Parameters

Name	Semantics	Type	Valid Values	Multiplicity
Direction: In				
AsynProcessIndicator(Optional)	See Common Parameters.			
ChannelCredential(Optional)	Channel Credential.	string		0..N
DocumentTemplate(Mandatory)	Template for new Event.	string	file name relative to $MQ_COMMON_DIR	1
DefaultDomain (Optional)	Default Credential Domain	string	Any valid Domain. If not supplied, then the value is “GLN”.	0..1
Form(Mandatory)	Form file used for merge	string	file name relative to $MQ_COMMON_DIR	1
InDocument (Optional)	See Common Parameters.			
InRecordList (Optional)	RecordList	recordlist	RecordList	0..1
	See CreateWorkItem			

Name	Semantics	Type	Valid Values	Multiplicity
PerspectiveName (Optional)	Parameters and Valid Execution Modes.			
ReceiverCredential(Optional)	Receiver Credential	string		0..N
RecordKey (Optional)	List of Record Key IDs	string		0..N
RecordPerAsyncCall(Optional)	See Common Parameters.			
RelationshipName(Optional)	See Common Parameters. This input must be specified only to change the relationships included in the spawned workflow. If not specified, relationships specified in the input record collection are used.			
SenderCredential(Optional)	Sender Credential	string		0..N
TradingPartnerCredential (Optional)	Trading Partner Credential	string		0..N
VersionOption (Optional)	See Common Parameters.	string	If there are any draft records in the workflow, they are included automatically. This parameter is applicable	1

Name	Semantics	Type	Valid Values	Multiplicity
			only if in record collection is null or event is for synchronization of a catalog. If specified, it overrides the VersionOption already in effect in input record collection.	
WorkflowPriority (Optional)	See Common Parameters. If the priority is not specified, event priority is inherited by the new message.			
RecordSequencing (Optional)	Controls the record sequencing. If set to true, PRODUCTKEYID of each record from the bundle is used as REGISTRATIONKEY. The parameter is applicable only when the activity is specified in the RecordCollection mode. If the record sequencing is not required and to improve the performance, do not enable the RecordSequencing parameter. This parameter overrides the com.tibco.cim.optimization.record.sequencing configuration property.	String	True (Default) False	0..1
RecordSequencingAttribute	Controls the record sequencing of specific records in case of	String	Attribute name in the following	1

Name	Semantics	Type	Valid Values	Multiplicity
(Optional)	huge record bundles. Specify a repository name along with its attributes as follows: <Parameter direction="in" name="RecordSequencingAttribute" type="string" eval="constant">ITEM/GTIN</Parameter> You can specify multiple attributes as follows: <Parameter direction="in" name="RecordSequencingAttribute" type="string" eval="constant">ITEM/PRODUCTID</Parameter> <Parameter direction="in" name="RecordSequencingAttribute1" type="string" eval="constant">ITEM/DIVISIONNUMBER</Parameter> The specified value of an attribute from the record bundle is used as REGISTRATIONKEY for events. If multiple attributes are specified, they are used to create a single REGISTRATIONKEY for each record bundle. A single REGISTRATIONKEY follows sorting based on the attribute name.		format: *RepositoryName/Attribute Name* • If you specify an incorrect repository or attribute name, the parameter is ignored. • In case of all invalid parameters, the record sequencing happens based on the default PRODUCTKEYID.	

- Specify the RecordSequencing and RecordSequencingAttribute parameters for the following two main reasons:

Name	Semantics	Type	Valid Values	Multiplicity

- to control the concurrency of spawned events
- to define sequential processing of bundles at workflow level rather than global level using the com.tibco.cim.optimization.record.sequencing configuration property.

- While working with huge record bundles, control REGISTRATIONKEYS using the RecordSequencingAttribute parameter. The parameter defines only specific records and attributes from the bundle that are involved in sequencing decisions. Additionally, the parameter restricts a number of entries made into the EVENTDETAIL table for each REGISTRATIONKEY, which helps to improve the performance.

DocumentTemplate

The template document is the mlXML document that is used for the new Event.

The program looks for the specified file in (until it finds a file):

```
$MQ_COMMON_DIR$MQ_COMMON_DIR/<enterprise internal name>/template$MQ_COMMON_
DIR/standard/template
```

The activity looks for a catalog reference and use it to look up passed in records:

```
<CatalogReference type="MasterCatalog">
 <RevisionID>
  <BaseName>JTEST1</BaseName>
  <Version>1</Version>
  <DBID/>
 </RevisionID>
</CatalogReference>
```

Form

The form file is used to merge activity input parameters into the Document Template file. It acts much the same as the form file that is used to access data for work items.

The “Value” in the form can come from two places:

- from the input parameters in the activity.

- hard-coded in the Form file.

Hard-coding the value in the form file is a degenerate case since we skip getting the value from the workflow activity. Getting the value from the workflow activity works by having the same name in both files. For example, if we have an input parameter to the SpawnWorkflow activity as follows:

```
<Parameter direction="in" name="OutputFormat" type="string"
eval="constant">MyOutputFormat</Parameter>
```

and the Form file contains the following entry:

```
<Field>
  <ID>OutputFormat</ID>
  <InputFrom>/Message/Body/Document/BusinessDocument/CatalogAction   /CatalogActionHead
er/Extension[@name='OutputFormat']/Value/text   ()</InputFrom>   <OutputTo>(//Form/Field/ID
[text()='OutputFormat'])/following-sib   ling::Value</OutputTo>
  <Value/>
</Field>
```

Since the name of the input parameter (name="OutputFormat") matches the ID of the Form File field (<ID>OutputFormat</ID>) the value of the input parameter (MyOutputFormat) is substituted into the corresponding xpath location in the Document Template File:

```
<Extension name="OutputFormat">
  <Value>MyOutputFormat</Value>
</Extension>
```

Credentials

You must specify either the sender or the receiver credential. In addition, you can specify the Channel and Trading Partner Credentials.

Credentials can be in the form of “Value” in which case the domain is defaulted to the default domain passed in as a “Domain“ parameter.

You can pass in either a single credential or an array of credentials. If you pass in a single credential, the same credential is used for all spawned events. If you pass in an array of credentials, an event is created for each entry in the array. Note that all array sizes either must match or be equal to 1. For example, you cannot pass in 5 channels and 8 trading partners. The lengths of the arrays must match.

Records

A record can be passed in the following ways:

- By specifying an “InRecordList” + Credential array: In this case, each bundle in the record collection is sent to all trading partners in the specified trading partner array.
- By specifying a RecordKey array/Credential array + RelationshipNameIn this case, each entry in the RecordKey array is sent to the corresponding trading partner in the trading partner array. This mode provides high selectivity – you can pick and choose which records to publish to any trading partner.

The number of records in the RecordKey must match the credential array size.

Each entry in the RecordKey must be of the form “CatalogID+++RecordKeyID”. This can be generated by rulebase by using the built in RECORD_KEY attribute. If only one value is passed, it is assumed to be the RecordKeyID and the catalog is defaulted to the catalog associated with the specified RecordKeyID.

The “RelationshipName” parameter can be defined to specify the relationships to use when constructing the record collection for the spawned event.

One way to call SpawnWorkflow is to first call the “EvaluateRuleBase” activity which generates the following four parallel arrays:

- RecordKey
- Receiver
- Channel Credential
- Trading Partner Credential

All resulting arrays must be of the same size. Each “Entry” generates a new event with one record list in it. The record list contains exactly one RecordBundle which is derived by parsing the RecordKey.

Example for SpawnWorkflow Activity

Workflow with preceding EvaluateRuleBase which generates the entries:

```
<Workflow Version="1.3" xmlns:cim="http://www.tibco.com/xpd/cimService1.0.0"
xmlns:xpdExt="http://www.tibco.com/XPD/xpdExtension1.0.0"
```

```
xmlns:xpdl2="http://www.wfmc.org/2004/XPDL2.0alpha" xmlns:exsl="http://exslt.org/common">
 <Owner>Standard</Owner>
 <Name>SpawnWorkflowExample</Name>
 <Activity Name="ApplySynchRuleBase">
  <Description lang="en">Lookup function.</Description>
  <Start/>
  <Action>EvaluateRuleBase</Action>
  <Parameter direction="in" name="Rulebase" type="string"
  eval="constant">testEnt/rulebase/rbsynchtradingpartner.xml</Parameter>
  <Parameter direction="in" name="InDocument" type="document"
  eval="variable">inDoc</Parameter>
  <!-- LogOption: A - AttributeLog, L - Execution Log, F - Log File-->
  <Parameter direction="in" name="LogOption" type="string"    eval="constant">F</Parameter>
  <!-- Number of FATAL errors (see Severity) -->
  <Parameter direction="out" name="Validationerrors" type="long"
  eval="variable">fatalerrors</Parameter>
  <!-- Number of Warnings (see Severity) -->
  <Parameter direction="out" name="Validationerrors1" type="long"
  eval="variable">warningerrors</Parameter>
  <Parameter direction="out" name="ProductKeyArray" type="arraylist"
  eval="variable">SynchProductKeyArray</Parameter>
  <Parameter direction="out" name="SenderArray" type="arraylist"
  eval="variable">SynchSenderArray</Parameter>
  <Parameter direction="out" name="ChannelArray" type="arraylist"
  eval="variable">SynchChannelArray</Parameter>
  <Parameter direction="out" name="TradingPartnerArray" type="arraylist"
  eval="variable">SynchTradingPartnerArray</Parameter>
 </Activity>
 <Activity Name="InitiateSynchProducts">
  <Action>SpawnWorkflow</Action>
  <Description lang="en">Start publish workflows.</Description>
  <Parameter direction="in" name="InDocument" type="document"
  eval="variable">workDoc</Parameter>
  <Parameter direction="in" name="DocumentTemplate" type="string"
  eval="constant">standard/template/tm26catwcatv1.xml</Parameter>
  <Parameter direction="in" name="RecordKey" type="arraylist"
  eval="variable">SynchProductKeyArray</Parameter>
  <Parameter direction="in" name="SenderCredential" type="arraylist"
  eval="variable">SynchSenderArray</Parameter>
  <Parameter direction="in" name="ChannelCredential" type="arraylist"
  eval="variable">SynchChannelArray</Parameter>
  <Parameter direction="in" name="TradingPartnerCredential"    type="arraylist"
eval="variable">SynchTradingPartnerArray</Parameter>
  <Parameter direction="in" eval="constant" name="Form"
  type="string">standard/forms/fm26catpubtemplatev1.xml</Parameter>
  <Parameter direction="in" name="RelationshipName" type="string"
```

```
  eval="constant">CONTAINS</Parameter>
  <Parameter direction="in" name="TransformRulebase" type="string"
  eval="constant">testEnt/rulebase/rbworkflowtransform.xml</Parameter>
  <Parameter direction="in" name="OutputFormat" type="string"
  eval="constant">MyOutputFormat</Parameter>
  <Parameter direction="in" name="RecordPerAsyncCall" type="long"
  eval="constant">10</Parameter>
  <Parameter direction="in" name="AsynProcessIndicator" type="boolean"
  eval="constant">true</Parameter>
 </Activity>
 <Transition FromActivity="ApplySynchRuleBase"   ToActivity="InitiateSynchProducts">
  <Description>Initiate Product synchronization by applying synchronization     rules</Description>
 </Transition>
</Workflow>
```

StateTransition Activity

The StateTransition activity defines a simple state machine that allows the workflow designer to build the business state of an enterprise across workflow and document types.

The states and transitions are defined in two separate tables, SupplierState and SupplierStateTransition, which must be fed with seed data, when the supplier is first connected to the service. The state machine operates on the implicit assumption that documents that logically belong together have one or more IDs in common, which identify the conversation they take part in. When a document that is already part of a conversation comes in, the workflow can use xpath expressions to get to the keys and lookup the current state of the conversation to which the document belongs. It can then move to a new state if appropriate or handle errors, if documents arrive out of order. Conversations are stored in the Conversation table, in which there is one and only one record per conversation.

The state transition activity has two primary actions, ‘new’ and ‘join’. When the former is specified, the activity creates a new entry in the Conversation table based on a unique key and an initial conversation state, and thus initialize the supplier specific state machine for a new set of documents. The latter action allows the workflow designer to specify that a document must ‘join’ an existing conversation, and move to a new state. The new state to move to is specified in the ‘MoveTo’ parameter in the workflow, but the set of legal moves are defined in the SupplierStateTransition table. The key used to lookup the conversation must be unique for a set of documents across marketplaces, which means that at least some sort of DocumentID (for example, POID) and the

marketplace ID or a name is needed to ensure uniqueness. Since the exact key needed is not known, the workflow designer can specify any number of key-parts (Key1, Key2, and so on), which are then concatenated into one big key and used for lookup in the Conversation table. When joining a conversation, you can specify a “NewKey” which is added to the list of keys for this conversation. This is useful for bridging keys for long conversations.

If the state machine fails to move the transition, it throws an exception if the out parameter ‘CouldMove’ is not specified. If the parameter is specified and an error occurs, ‘CouldMove’ is false, otherwise it is true. Another optional out parameter, ‘ConversationState’ contains the current state of the current conversation after the state machine has attempted a move.

StateTransition Parameters and Valid Execution Modes

The valid execution mode for StateTransition is SYNCHR.

The parameters of StateTransition are as follows:

StateTransition Parameters

Name	Semantics	Type	Valid Values	Multi-plicity
Direction: In				
InDocument (Mandatory)	See Common Parameters.			
ConversationAction (Mandatory)	Specifies whether to start a new conversation (new) or to participate in an existing conversation (join) or to join an existing conversation or create it if it does not exist (connect).	string	new, join, connect	1

Name	Semantics	Type	Valid Values	Multi-plicity
ConversationType (Optional)	Type of conversation. GLOBAL updates the Event with the resulting conversation id. LOCAL (the default) does not.	string	GLOBAL LOCAL	0..1
MoveTo(Mandatory)	Specifies the new state to move the conversation.	string	The states defined in the SupplierState table for the current enterprise	1
Key#(Mandatory)	A set of keys used to uniquely identify this conversation. Replace # with an integer, for example, Key1, Key2.	string		1..n
NewKey#(Optional)	Add this key to the list of keys for this conversation. Replace # with an integer, for example, NewKey1, NewKey2, and so on.	string		0..n
Direction: Out				
CouldMove(Optional)	True if the state machine moved to a new state, false otherwise. If not specified and no	boolean	True, False	N/A

Name	Semantics	Type	Valid Values	Multi-plicity
	move was made, an exception is thrown.			
ConversationState (Optional)	The state of the conversation after a move has been attempted	string	The states defined in the SupplierState table for the current enterprise	N/A

Example for StateTransition Activity

In the following example, the activity tries to create a new conversation assuming that the message was never received before.

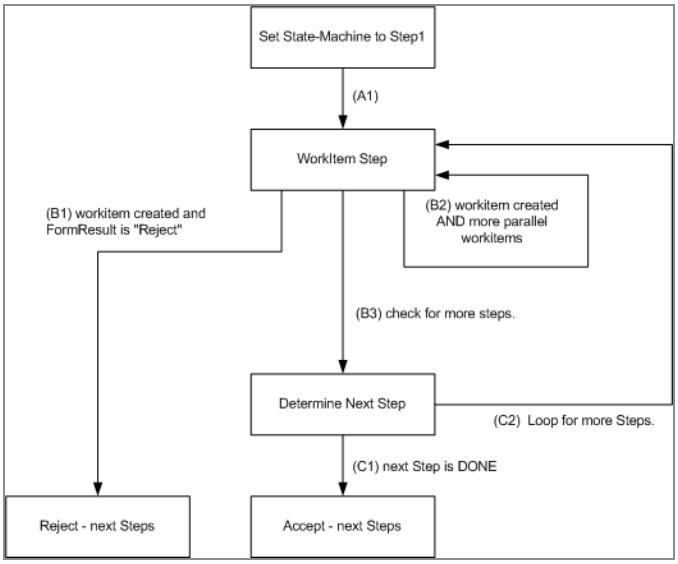
It builds keys as "Message" + "InternalID + PublishDate" – if this key is already found in the system, then the activity sets ‘couldMoveToNew’ as ‘false’. The 'couldMoveToNew' can then be used to handle where a duplicate message is received.

```
<Activity Name="MessageReceived">
  <Start/>
  <Action>StateTransition</Action>
  <Description lang="en">Set state to new message received</Description>
  <Parameter direction="in" type="string" eval="constant"
  name="eventState">NEWMESSAGE</Parameter>
  <Parameter direction="in" name="InDocument" type="document"
  eval="variable">inDoc</Parameter>
  <Parameter direction="in" name="ConversationAction" eval="constant"
  type="string">new</Parameter>
  <Parameter direction="in" name="CheckTransition" eval="constant"
  type="string">true</Parameter>
  <Parameter direction="in" name="MoveTo" eval="constant"   type="string">NEW</Parameter>
  <!-- Key is made up of "Message" + "InternalID + PublishDate" -->
  <Parameter direction="in" name="Key1" eval="constant"   type="string">Message</Parameter>
  <Parameter direction="in" name="Key2" eval="xpath" type="string"
  source="/Message/@externalControlNumber">inDoc</Parameter>
  <Parameter direction="in" name="Key3" eval="xpath" type="string"   source="//CatalogReference
[@type='Catalog']/RevisionID/DBID/text()">inDoc  </Parameter>
  <Parameter direction="in" name="Key3" eval="xpath" type="string"   source="//CatalogReference
```

```
[@type='MasterCatalog']/RevisionID/DBID/text()">  inDoc</Parameter>
  <Parameter direction="out" eval="variable" type="document"
  name="OutDocument">workDoc</Parameter>
  <Parameter direction="out" name="ConversationState" eval="variable"
  type="string">newState</Parameter>
  <Parameter direction="out" name="CouldMove" eval="variable"
  type="boolean">couldMoveToNew</Parameter>
</Activity>
```

When defining work item chains in generic workflows, you can use the StateTransition activity to set up a "mini state machine" to control how many times the workflow repeats through that work item.

Setting up Mini State Machine



SurvivingRecordSelection Activity

The SurvivingRecordSelection activity selects one target record from the detected duplicate records and saves all data in the merge tables. The selection of target record is based on the auto selection rule defined in the AutoRecordSelectionConfig.xml file.

For information about the auto selection rules and auto record selection configuration file, see *TIBCO MDM Customization*.

The source and target records are passed to the MergeRecord Activity activity for the merging process. The MergeRecord Activity activity identifies the records from the

merge table and merges the records. The activity selects only one record as the survivor. The survivor record is merged with the incoming record.

SurvivingRecordSelection Parameters and Valid Execution Modes

The valid execution mode for SurvivingRecordSelection is SYNCHR.

The parameters of SurvivingRecordSelection are as follows:

Name	Semantics	Type	Valid Values	Multi-plicity
Direction: In				
eventState	Determines the latest state of the workflow.	string	AutoMerge	0..1
AutoSelectionConfigFileName	Name of the auto selection configuration file.	string	Any valid file name. AutoRecordSelectionConfig.xml	1
ReferenceStepID (Mandatory)	The ReferenceStepID is mapped to the AutoSelectionProcessLogID out parameter.	long	Any valid integer	0 or 1
Direction: Out				
AutoSelectionProcessLogID	The Message ID for which status is retrieved. It is a concatenation of all MessageID# values passed to the activity.	long	Any valid integer	1
MergeCount	Number of records that are merged in the surviving record selection process.	long	Any valid integer • If a single record is added or modified, the number is one. • If a record is imported, the number is more than one.	1
AutoSelection (Mandatory)	Name of the auto selection process rule. You can verify the event log to check which auto selection process rule is selected.	string	HighScore	1
ConflictResolver (Mandatory)	Name of the conflict resolution process rule specified in the AutoRecordSelectionConfig.xml file. You can verify the event log to check which conflict resolver is used.	string	MostlyNew	1
RecordKey (Mandatory)	Contains the source record and its target record key IDs in the matching selection. For example, Record_101 has three duplicates Record_102, Record_103, and Record_104. If the auto selection finds 103 as surviving record, then the map is ([Record_101,Record_103],....)	arraylist	List of record keys	0..1

Example for SurvivingRecordSelection Activity

```
<Activity Name="AutoSelection">
<Action>SurvivingRecordSelection</Action>
<Description>Surviving Record Selection</Description>
<Execution>SYNCHR</Execution>
<Parameter name="eventState" direction="in" eval="constant" type="string">AutoMerge</Parameter>
<Parameter
name="AutoSelectionConfigFileName" direction="in" eval="constant" type="string">AutoRecordSelectionConfig.xml
</Parameter>
<Parameter name="ReferenceStepID" direction="in" eval="variable" type="long">update_step</Parameter>
<Parameter name="AutoSelectionProcessLogID" direction="out" eval="variable" type="long">update_step1</Parameter>
</Activity>
```

SuspendWorkflow Activity

The SuspendWorkflow activity suspends the workflow.

The SuspendWorkflow activity allows a sub-workflow to indicate the parent workflow that the sub workflow has not completed. There are no input or output parameters.

If the Suspend activity is used to suspend the parent event, join for the record fails to join the parent event, and workflow is terminated.

SuspendWorkflow Valid Execution Modes

The valid execution mode for SuspendWorkflow is SYNCHR.

Example for SuspendWorkflow Activity

```
<Activity Name="SuspendWorkflow">
  <Action>SuspendWorkflow</Action>
  <Execution>SYNCHR</Execution>
</Activity>
```

Translate Activity

The Translate activity translates a document from one format into another or performs a replacement of characters within a source document.

The Translate activity currently supports translating from any XML format to any XML format. The replacement functionality can replace any #PCDATA character string with any other #PCDATA character string inside a document.

Translate Parameters and Valid Execution Modes

The valid execution mode for Translate is SYNCHR.

The parameters of Translate are as follows:

Translate Parameters

Name	Semantics	Type	Valid Values	Multiplicity
Direction: In				
InDocument (Mandatory)	See Common Parameters.			
Translator (Optional)	Used to distinguish between the three operation modes.	string	XSLT (Default) REPLACE XML2PDF	0..1

Name	Semantics	Type	Valid Values	Multiplicity
Direction: Out				
OutDocument	See Common Parameters.			

Translator XSLT Activity

The Translator: XSLT activity translates one XML format to another, using an XSL style sheet.

Translator XSLT Additional Parameters

Name	Semantics	Type	Valid Values	Multi-plicity
Direction: In				
DefaultTimeZone (Optional)	Default Time Zone	string		
Derived(Optional)	Indicates whether the output of this translation is the same type of document as the input, or whether it is a different type of document. This sets the Derived flag in the GeneralDocument table.	boolean	'True' - this is a derived document. 'False' - this is NOT a derived document. Default is "True"	0..1

Name	Semantics	Type	Valid Values	Multi-plicity
Normalize(Optional)	Indicates whether or not the normalization is performed.	string	“None”: This is the default value. Does not perform any normalization. “Pre”: Normalization is done prior to the translation (In case of translations from mlXML to other protocol). “Post”: Normalization is done after the translation. (Cases where the incoming document is converted to mlXML format.)	0..1
OutputFormat (Optional)	The output format produced by the style sheet. **Note:** This parameter is mandatory if the Translator mode is XSLT.	string	XML, mlXML, cXML, and FILE.	0..1
OutputFormatVersion (Optional)	Version of the output format	string		0..1

Name	Semantics	Type	Valid Values	Multi-plicity
	produced by the style sheet. **Note:** This parameter is mandatory if the Translator mode is XSLT.			
Map(Mandatory)	Specify an XSLT stylesheet to use for translation relative to $MQ_COMMON_DIR.	string	Any filename that resolves to an XSLT stylesheet, but without the '.xsl' extension.	1
Xsl_Param_*	Using parameters that have names of the pattern 'Xsl_Param_*', users can pass parameters to the XSL. They can then be accessed in the XSL as a parameter with the same name.	long, string, boolean.	Any valid non-null value.	0..N

Example for Translator XSLT Activity

```
<Activity Name="TranslateDocSubTypeForAdd">
  <Action>Translate</Action>
  <Description lang="en">Translate import document to 'Add   Message'</Description>
  <Parameter direction="in" name="Derived" type="string"   eval="constant">false</Parameter>
  <Parameter direction="in" eval="constant" type="string"   name="Translator">XSLT</Parameter>
  <Parameter direction="in" eval="constant" type="string"
```

```
 name="Map">standard/maps/mp26importto26cateditadd</Parameter>
 <Parameter direction="in" eval="constant" type="string"
 name="OutputFormat">mlXML</Parameter>
 <Parameter direction="in" eval="constant" type="string"   name="Normalize">None</Parameter>
 <Parameter direction="in" eval="variable" type="document"
 name="InDocument">inDoc</Parameter>
 <Parameter direction="out" eval="variable" type="document"
 name="OutDocument">workDoc</Parameter>
 <Parameter direction="in" type="string" eval="constant"
 name="eventState">TRANSLATE</Parameter>
</Activity>
```

Translator REPLACE

Replace a set of character strings inside a document with a specified set of character strings.

This translation type can be used to convert from non-printable characters to printable ones.

Translator XML2PDF

Translate from XML to PDF format.

Additional Parameters

Map (mandatory) Specify an XSLT stylesheet to use for translation

Example

```
<Activity Name="CreatePDFDataSheet">
 <Action>Translate</Action>
 <Description>Translate the record data to to PDF datasheet</Description>
 <Parameter direction="in" type="string" eval="constant"
 name="eventState">TRANSLATE</Parameter>
 <Parameter direction="in" name="Derived" type="string"   eval="constant">true</Parameter>
 <Parameter direction="in" eval="constant" type="string"
 name="Translator">XML2PDF</Parameter>
 <Parameter direction="in" eval="constant" type="string"
 name="Map">standard/maps/mpfrom26topdfprodspecv1</Parameter>
```

```
  <Parameter direction="in" eval="constant" type="string"
  name="OutputFormat">PDF</Parameter>
  <Parameter direction="in" eval="variable" type="document"
  name="InDocument">workDoc</Parameter>
  <Parameter direction="out" eval="variable" type="document"
  name="OutDocument">pdfDataDoc</Parameter>
</Activity>
```

Additional Tags

Tag	Semantics	Signature	Required
Replacements	A grouping tag that contains any number of <Replacement>.	<Replacements> (Replacement+) </Replacements>	Y
Replacement	A single replacement replacing the string in the ‘from’ attribute with the string in the ‘to’ attribute in the input document.	<Replacement from=” to=”/>	Y (at least 1)

Additional Parameters

For InDocument and OutDocument parameters, see Common Parameters.

Example

```
<Activity Name="ReplaceEDIChars"><Start/>
  <Action>Translate</Action>
  <Description>Replace special EDI bytes</Description>
  <Execution>SYNCHR</Execution>
  <Parameter direction="in" type="string" eval="constant"
   name="Translator">REPLACE</Parameter>
  <Replacements>
  <Replacement from="0x1F" to="~"/>
  <Replacement from="0x0A" to="0x5C"/>
  <Replacement from="0x1D" to="*"/>
  </Replacements>
  <Parameter direction="in" type="document" eval="constant"
   name="InDocument">Doc0</Parameter>
  <Parameter direction="out" type="document" eval="constant"
```

```
  name="OutDocument">Doc1</Parameter>
</Activity>
```

UpdateAutomaticHierarchies Activity

Use the UpdateAutomaticHierarchies activity to update the automatic hierarchies. The records are automatically linked to hierarchies. The HierarchiesProcessed output parameter indicates the names of the automatic hierarchies and updates them as a part of activity execution.

When you create automatic hierarchies, the following workflows are updated to include the UpdateAutomaticHierarchies activity:

- Import Records (wfin26catsourcev7.xml and wfin26catsourceimportv2.xml)
- Mass Update (wfin26catmassupdate2v1.xml)
- Record Add (wfin26productaddapprovalv3.xml)
- Record Modify and Record Delete (wfin26producteditapprovalv3.xml)
- Data Quality (wfin26dqproductaddapprovalv1.xml)

When you confirm a record (state is Confirmed), the activity is invoked and related hierarchies are updated. However, if you delete or reject a record (state is Deleted or Rejected), hierarchies are not updated.

UpdateAutomaticHierarchies Parameters and Valid Execution Modes

The valid execution mode for UpdateAutomaticHierarchies is SYNCHR.

UpdateAutomaticHierarchiesParameters

Parameter	Description
Direction: In	
eventState	• **Semantics:** See Common Parameters.

Parameter	Description
	• **Type:** String • **Valid Values:** UPDATEHIERARCHY • **Multiplicity:** 0..1
MasterCatalog	See Common Parameters.
InDocument	**Note:** All three parameters are mandatory for the UpdateAutomaticHierarchies activity.
InRecordList	
Direction: Out	
OutDocument	See Common Parameters.
HierarchiesProcessed	• **Semantics:** A comma separated names of automatic hierarchies. Hierarchies to be processed are identified based on the repositories in which the records are being imported or updated. For example, DeptVsDesignationView, EmployeeVsDesignationView. • **Type:** String • **Valid Values:** Names of automatic hierarchies processed. • **Multiplicity:** 0..*n*

Example for UpdateAutomaticHierarchies Activity

The following example shows how the UpdateAutomaticHierarchies activity is used when initiating a workflow:

```
<Activity Name="UpdateAutomaticHierarchies">
<Action>UpdateAutomaticHierarchies</Action>
<Description lang="en">Update Automatic Hierarchies and link records</Description>
```

```
<Parameter direction="in" type="string" eval="constant"
name="eventState">UPDATEHIERARCHY</Parameter>
<Parameter direction="in" eval="xpath" type="string" name="MasterCatalog"
source="/Message/Body/Document/BusinessDocument/CatalogAction/CatalogActionHeader/Mast
erCatalog/RevisionID/BaseName/text()">inDoc</Parameter>
<Parameter direction="in" name="InDocument"
type="document"eval="variable">inDoc</Parameter>
<Parameter direction="out" eval="variable" type="document"
name="OutDocument">workDoc</Parameter>
<Parameter direction="in" name="InRecordList" type="recordlist"
eval="variable">workRecordList</Parameter>
<Parameter direction="out" eval="variable" type="string"
name="HierarchiesProcessed">DeptVsDesignationView,EmployeeVsDesignationView</Paramet
er>
</Activity>
```

UpdateEvent Activity

Every major action in the system (Add a Product, Receive Message from DataPool, Upload Datasource, and so on) creates an EVENT. The steps of the event can be viewed from EventLog.

In most cases, you update an event in the following cases:

- When you receive a document, you might now have more information about the purpose of the Event than when the event was first created. For example, the receiver receives a document from a trading partner, and until you translate the document into mIXML you do not know what the exact content of the document is. After you have translated it, then you know that it is a “Publish Notification”.
- When the Event is complete, you want to indicate that the event is now “Done” and whether it failed or succeeded. This is usually the last step done by the workflow.

UpdateEvent Parameters and Valid Execution Modes

The valid execution mode for UpdateEvent is SYNCHR.

The parameters of UpdateEventare as follows:

UpdateEvent Parameters

Name	Semantics	Type	Valid Values	Multi-plicity
Direction: In				
eventState(Optional)	The latest state of the workflow.	string	Predefined state: DONE. Custom states can be defined in the domain entry table.	0..1
eventStatus(Optional)	Set to show completion status of the workflow.	string	SUCCESS ERROR CANCELLED	0..1
GlobalConversationID (Optional)	Set Event.ConversationID to this conversation.	long	The workflow variable “ConversationID” holds conversation ID from the last state transition. Usually that is the value you want to input.	0..1
eventType(Optional)	The type of the event. Appended to the existing eventType.	string	CAT – if catalog event. To be set only if workflow is initiated by a document of unknown type. Followed by “-“ and other descriptors.	0..1

Name	Semantics	Type	Valid Values	Multi-plicity
eventDescriptor (Optional)	The descriptor for the event.	string	select * from domainentry where domaintype such as 'APPFUNCTION' for valid value list. This is the same as the subtype attribute in the mlXML document. To be set only if the workflow is initiated by a document of unknown type.	0..1
deploymentMode (Optional)	Whether this is a test or production event.	string	Production Test	0..1
ALL OTHER VALUES	Event Details	any	All other values are inserted into the EventDetail table.	

You can pass any value to this activity for storage into EVENTDETAIL table. This trick can help you capture custom information for debugging and reporting. (for example, to capture the GLN).

```
<Parameter direction="in" type="string" eval="catalog" name="senderGLN"
source="SenderGLN">inDoc</Parameter>
```

Example for UpdateEvent Activity

The following is an example for UpdateEvent activity when initiating a workflow:

```
<Activity Name="AddMsgInfoToEvent">
 <Start/>
 <Action>UpdateEvent</Action>
 <Description lang="en">Set the Event State to Receive</Description>
 <Parameter direction="in" name="eventDescriptor" type="string" eval="xpath"
 source="/Message/Body/Document/@subtype">inDoc</Parameter>
 <Parameter direction="in" name="deploymentMode" type="string" eval="xpath"
 source="/Message/@messageType">inDoc</Parameter>
 <Parameter direction="in" type="string" eval="constant"
 name="eventState">RECEIVE</Parameter>
 <Parameter direction="in" name="eventType" type="string"   eval="constant">CAT-
CIN</Parameter>
</Activity>
>
```

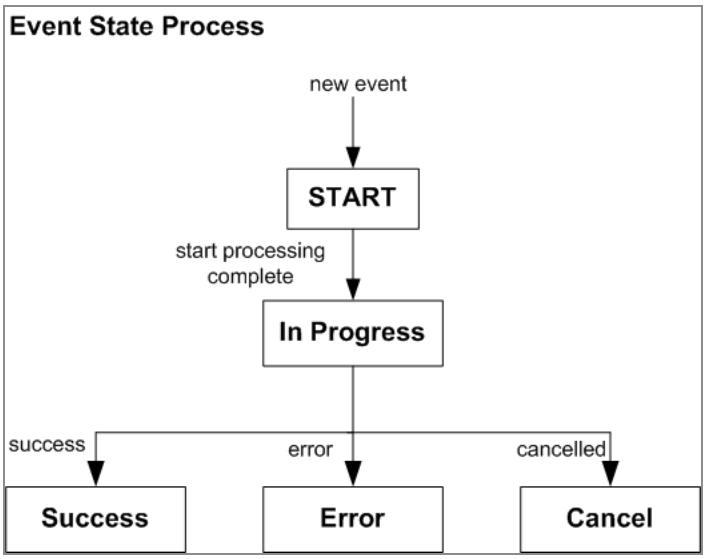
Here, “inDoc” is the mIXML document which contains the subtype, which is usually equivalent to the Application Function.

When a workflow is done:

```
<Activity Name="SetStatusToSuccess">
 <Action>UpdateEvent</Action>
 <Description lang="en">Set the event state to Success/Done.</Description>
 <Parameter direction="in" type="string" eval="constant"
 name="eventStatus">SUCCESS</Parameter>
 <Parameter direction="in" type="string" eval="constant"
 name="eventState">DONE</Parameter>
 <Parameter direction="in" type="string" eval="catalog" name="senderGLN"
 source="SenderGLN">inDoc</Parameter>
</Activity>
```

Here, the status is “SUCCESS” and the final state is “DONE”. Also, you can set a “senderGLN” EventDetail, which contains the GLN from the catalog object. “senderGLN” could be any other name, but the result is that it creates an entry in the EventDetail table which can be used in subsequent processing.

Event State Process



UpdateHierarchyLink Activity

You can use the UpdateHierarchyLink activity to approve or reject the record links by specifying appropriate values for the Status parameter. The UpdateHierarchyLink activity is used in the Hierarchy Edit Approval (wfin26hierarchyeditapprovalv1.xml) and that Hierarchy Link Approval (wfin26hierarchylinkapprovalv1.xml) workflows. In the Hierarchy Edit Approval workflow, this activity updates the links modified by split or merge operation.

In Hierarchy Link Approval, the work item has the **Accept** and **Reject** options. You can view or browse the details of record links for different repositories within the work item. If the associated hierarchy link approval work item is accepted, this activity runs in UpdateHierarchyLinkAsApproved mode. If the approval work item is rejected, the activity runs in UpdateHierarchyLinkAsRejected mode.

UpdateHierarchyLink Parameters and Valid Execution Modes

The valid execution mode for UpdateHierarchyLink is SYNCHR. The parameters of UpdateHierarchyLink are as follows:

UpdateHierarchyLink Parameters

Name	Semantics	Type	Valid Values	Multiplicity
Direction: In				
eventState	See Common Parameters.	N/A	UPDATEHIERARCHYLINK	0..1
Status (Mandatory)	See UpdateHierarchyState Activity.	String	The valid values are CONFIRMED and REJECTED. The default value is CONFIRMED.	0..1
Direction: Out				
Links Added	Total number of links (records) linked to different hierarchy nodes	Long	Number of links (records) linked to different nodes in this event.	1
Links Removed	Total number of links (records) removed or unlinked from different hierarchy nodes	Long	Number of links (records) removed from different nodes in this event.	1
OutDocument	See Common Parameters.	Document	N/A	N/A

Example for UpdateHierarchyLink Activity

The following example updates the hierarchy link as CONFIRMED.

```
<Activity Name="UpdateHierarchyLinkAsApproved">
<Action>UpdateHierarchyLink</Action>
<Description lang="en">Set the hierarchy linking result as confirmed</Description>
<Parameter direction="in" type="string" eval="constant"
name="eventState">UPDATEHIERARCHYLINK</Parameter>
<Parameter direction="in" name="Status" type="string"
eval="constant">CONFIRMED</Parameter>
<Parameter direction="in" name="InDocument" type="document"
eval="variable">workDoc</Parameter>
<Parameter direction="out" eval="variable" type="document"
name="OutDocument">workDoc</Parameter>
</Activity>
```

The following example updates the hierarchy link as REJECTED.

```
<Activity Name="UpdateHierarchyLinkAsRejected">
<Action>UpdateHierarchyLink</Action>
<Description lang="en">Set the hierarchy linking result as rejected</Description>
<Parameter direction="in" type="string" eval="constant"
name="eventState">UPDATEHIERARCHYLINK</Parameter>
<Parameter direction="in" name="Status" type="string" eval="constant">REJECTED</Parameter>

<Parameter direction="in" name="InDocument" type="document"
eval="variable">workDoc</Parameter>
<Parameter direction="out" eval="variable" type="document"
name="OutDocument">workDoc</Parameter>
</Activity>
```

UpdateHierarchyState Activity

You can use the UpdateHierarchyState activity to update the state of a hierarchy by specifying appropriate values for the Status parameter. The activity works according to the actions being performed on hierarchy. If the associated hierarchy approval work item is accepted, this activity runs in UpdateHierarchyStateAsApproved mode. If the associated hierarchy approval work item is rejected, the activity runs in UpdateHierarchyStateAsRejected mode.

How the UpdateHierarchyState Activity Works

- When the hierarchy is saved without any node rejections (from hierarchy approval work item), the UpdateHierarchyState activity updates its state to CONFIRMED.
- When the hierarchy is modified and any of the nodes are rejected from the work item, the RejectedNodes parameter contains the details of rejections. In this case, the UpdateHierarchyState activity updates the hierarchy log details with rejections in the database.

  However, if the hierarchy is sent back for correction (to RejectionParticipantID) from the work item, the UpdateHierarchyState activity continues to lock the hierarchy and does not release the lock.
- When a hierarchy is deleted and the delete operation is rejected from an associated work item, the UpdateHierarchyState activity restores the deleted hierarchy and updates the deleted version as REJECTED.

  However, if the delete operation is confirmed from an associated work item, the UpdateHierarchyState activity updates the deleted version as CONFIRMED.
- Finally, the UpdateHierarchyState activity unlocks or releases the lock acquired on the hierarchy by the current user or member and current event.

UpdateHierarchyState Parameters and Valid Execution Modes

The valid execution mode for UpdateHierarchyState is SYNCHR.

The parameters of UpdateHierarchyState are as follows:

UpdateHierarchyState Parameters

Name	Semantics	Type	Valid Values	Multiplicity
Direction: In				
Name	**Semantics**	**Type**	**Valid Values**	**Multi-plicity**
eventState	See Common Parameters.		UPDATEHIERARCHY	0..1

Name	Semantics	Type	Valid Values	Multiplicity
Status (Mandatory)	Refers to the state of hierarchy.	String	The valid values are CONFIRMED and REJECTED. The default value is CONFIRMED.	0..1
RejectedNodes (Optional)	A pair of node ID and node rejection comments.	String	An XPath expression to retrieve this value from the InDocument.	1
Direction: Out				
OutDocument	See Common Parameters.	Document	N/A	N/A

Example for UpdateHierarchyState Activity

The following example updates the hierarchy state as CONFIRMED.

```
<Activity Name="UpdateHierarchyStateAsApproved">
<Action>UpdateHierarchyState</Action>
<Description lang="en">Set the hierarchy status as confirmed</Description>
<Parameter
direction="in" type="string" eval="constant" name="eventState">UPDATEHIERARCHY</Parameter>
<Parameter direction="in" name="Status" type="string" eval="constant">CONFIRMED</Parameter>
<Parameter direction="in" name="InDocument" type="document" eval="variable">workDoc</Parameter>
```

```
<Parameter
direction="out" eval="variable" type="document" name="OutDocument">workDoc</Parameter>

</Activity>
```

The following example updates the hierarchy state as REJECTED.

```
<Activity Name="UpdateHierarchyStateAsRejected">

<Action>UpdateHierarchyState</Action>      <Description lang="en">Set the hierarchy status as REJECTED</Description>

<Parameter direction="in" type="string" eval="constant" name="eventState">UPDATEHIERARCHY</Parameter>

<Parameter
name="RejectedNodes" type="string" eval="xpath" source="/Message/Body/Document/BusinessDocument/HierarchyAction
[last()]/HierarchyActionHeader/HierarchyActionHeaderAck/NodeRejections/text()" direction="in">workDoc</Parameter>

<Parameter direction="in" name="Status" type="string" eval="constant">REJECTED</Parameter>

<Parameter direction="in" name="InDocument" type="document" eval="variable">workDoc</Parameter>

<Parameter direction="out" eval="variable" type="document" name="OutDocument">workDoc</Parameter>

</Activity>
```

UpdateRecordState Activity

The products processed in the UpdateRecordState activity come from the ProductLog table.

For all the products passed for processing, the UpdateRecordState activity updates the product status in the MCT table to whatever status is passed to the activity. If the status passed is CONFIRMED, the latest version of the product is set to CONFIRMED. If the

status is REJECTED, every version in that event is set to REJECTED. The output of this activity is a RecordCollection object.

UpdateRecordState Parameters and Valid Execution Modes

The valid execution mode for UpdateRecordState is SYNCHR.

The parameters of UpdateRecordState are as follows:

UpdateRecordState Parameters

Name	Semantics	Type	Valid Values	Multi-plicity
Direction: In				
AsynProcessIndicator (Optional)	See Common Parameters.			
InDocument(Optional)	See Common Parameters.			
InRecordList(Optional)	The list of records for which status change is requested.	recordlist		0..1
MasterCatalog(Optional)	This parameter must be specified if InDocument is specified. The activity tries to get the repository name in the following order: From InParameter From InDocument An exception is thrown and the workflow quits if the: • repository name is not found in both InParameter and InDocument. • repository name is found but the catalog cannot be built.	string		0..1
OverrideConflict (Optional)	Use this to ignore any conflicts.	boolean	True False (Default)	0..1
ProductExtns(Optional)	Used when the ProductIds parameter is used. Indicates the product extensions of the records to be processed.	ArrayList		0..1
ProductIds(Optional)	Indicates the product IDs of the records to be processed. This is a required parameter only when you want to process the record from the input. The attribute value can also be provided as an arraylist. However, the attribute value arraylist size must be	ArrayList		0..1

Name	Semantics	Type	Valid Values	Multi-plicity
	equal to the product ID list size.			
RecordKey(Optional)	If this parameter is specified as input and its size is greater than 0, this activity updates the product status of the specified records in the input. If the repository name is specified in the input parameter, it is considered. If it is not found, the activity tries to get it from InDocument.	ArrayList		0..1
RecordPerAsyncCall (Optional)	See Common Parameters.			
RelationshipMerge	By default, when a draft record is updated to UNCONFIRMED or CONFIRMED, this parameter merges the relationships as follows: • UNCONFIRMED– DRAFT record with the latest UNCONFIRMED record. • CONFIRMED– DRAFT record with the latest CONFIRMED record. • If Float happens, the floated version has all the relationships of the latest CONFIRMED version. When this parameter is set to false, no relationship merges are done.	boolean	True (Default) False	1
Status(Mandatory)	The status the products of the event must be set to. If status is not specified, CONFIRMED is assumed.	string	CONFIRMED REJECTED UNCONFIRMED	0..1
Direction: Out				
CommandStatusout	Indicates the result of activity.	string	Merged, Overridden, Normal, ConflictResolutionRequired	1

Example for UpdateRecordState Activity

```
<Activity Name="UpdateRecordState">
 <Start/>
 <Description lang="en">Set status as approved.</Description>
 <Action>UpdateRecordState</Action>
 <Parameter direction="in" name="Status" type="string"
 eval="constant">CONFIRMED</Parameter>
 <Parameter direction="in" name="InDocument" type="document"
 eval="variable">inDoc</Parameter>
 <Parameter direction="in" name="RecordPerAsyncCall" type="long"
 eval="constant">10</Parameter>
 <Parameter direction="in" name="AsynProcessIndicator" type="boolean"
 eval="constant">true</Parameter>
</Activity>
```

For all records which are already confirmed or Rejected, any change requested is ignored. Status change request to UNCONFIRMED is ignored for records which are not DRAFT.

If the record List is specified, and it is not null, records are selected using the record list. From the record list, only those records which have product logs are selected for status update. If the record list is not specified or is null, all records associated with the event (based on product logs) are selected for status update. Note that only product logs with operations IMPORT, CREATE PRODUCT, EDIT PRODUCT, DELETE PRODUCT, MASSUPDATE are considered.

If new status requested is REJECTED, all unconfirmed versions associated with the event are rejected.

If new status requested is CONFIRMED or UNCONFIRMED, only the last version associated with the event is updated.

If the new status requested is REJECTED or CONFIRMED, record count for the catalog is updated as well.

Record status is not updated to CONFIRMED or UNCONFIRMED if the record version being updated generates a conflict and OverrideConflict flag is false.

CommandStatus indicates if there was any conflict, override of data, merge done for any one record. Conflict takes precedence over override and override takes precedence over merge.

If a conflict is detected, the record status update for ALL records is ignored.

UploadDataSource Activity

The UploadDataSource activity is used to upload a data source.

The data source is used to import data into classification schemes, repositories, to create Subset Rules, build valid values lists, and so on.

UploadDataSource Parameters and Valid Execution Modes

The valid execution mode for UploadDataSource is SYNCHR.

The parameters of UploadDataSource are as follows:

UploadDataSource Parameters

Name	Semantics	Type	Valid Values	Multi-plicity
Direction: In				
DataSource(Mandatory)	The name of the data source to be uploaded.	string	Any existing data source name.	1...N
DataSourceFileName (Optional)	Name of the data source file to be uploaded. DataSourceURI, DataSourceFileName OR InDocument must be specified.	string	Any valid filename relative to DataSourceURI.	0..1
DataSourceURI (Optional)	URI of the data source to be uploaded. The	string		0..1

Name	Semantics	Type	Valid Values	Multi-plicity
	location is relative to MQ_COMMON_DIR. DataSourceURI, DataSourceFileName OR InDocument must be specified.			
InDocument(Optional)	See Common Parameters. DataSourceURI, DataSourceFileName OR InDocument must be specified.			0..1
Direction: Out				
DataSourcesWithErrors (Optional)	Number of data sources errored out. Though it is possible to upload the same file to more than one data source, it is discouraged. If this parameter is not specified, and if any data source upload results in an error, an exception is thrown.	long		0..1
ErrorFile	If this parameter is set, an error file is created if there is an error when uploading the data source. This file is created in the	string		0..1

Name	Semantics	Type	Valid Values	Multi-plicity
	directory specified by MQ_COMMON_DIR.			
TotalRecordsAttempted (Optional)	Number of rows from the input file considered for upload.	long		0..1
TotalRecordsWithErrors (Optional)	Number of rows which could not be uploaded.	long		0..1

Example for UploadDataSource Activity

```
<Activity Name="UploadDataSource">
 <Start/>
 <Action>UploadDataSource</Action>
 <Description lang="en">Upload Data Source Summary</Description>
 <Parameter direction="in" type="string" eval="constant"
 name="eventState">UPLOAD</Parameter>
 <Parameter direction="in" eval="xpath" type="string" name="DataSource"
 source="/Message/Body/Document/BusinessDocument/CatalogAction/  CatalogActionHeader/Da
taSource/RevisionID/BaseName/text()">inDoc  </Parameter>
 <Parameter direction="in" name="InDocument" type="document"
 eval="variable">inDoc</Parameter>
 <Parameter direction="out" eval="variable" type="long"
 name="TotalRecordsAttempted">totalRowsAttempted</Parameter>
 <Parameter direction="out" eval="variable" type="long"
 name="TotalRecordsWithErrors">totalRowsWithErrors</Parameter>
 <Parameter direction="in" eval="xpath" type="string"   name="DataSourceFileName"
 source="/Message/Body/Document/BusinessDocument/CatalogAction/  CatalogActionHeader/Da
taSource/URIInfo/FileName/text()">inDoc</Parameter>
 <Parameter direction="in" eval="xpath" type="string" name="DataSourceURI"
 source="/Message/Body/Document/BusinessDocument/CatalogAction/  CatalogActionHeader/Da
taSource/URIInfo/URI/text()">inDoc</Parameter>
 <Parameter direction="out" eval="variable" type="string"
 name="ErrorFile">ErrorFile</Parameter>
</Activity>
```

UnZipFile Activity

Using the UnZipFile activity, you can extract the compressed files from a ZIP archive.

UnZipFile Parameters and Valid Execution Modes

The valid execution mode for UnZipFile is SYNCHR.

The parameters of UnZipFile are as follows:

UnZipFile Parameters

Name	Semantics	Type	Valid Values	Multiplicity
Direction: In				
FileName (Mandatory)	Indicates the name of the ZIP file.	document	Any valid file name.	1
Todir (Optional)	Refers to the target directory, which is relative to MQ_COMMON_DIR/Temp. **Note:** If Todir is not specified, CreateDateDir must be specified as true.	string	Any valid directory name.	0..1
CreateDateDir (Optional)	Creates the date for a directory. If true is specified, the files are placed in a directory structure yyyy/mm/dd/k. k indicates hours 1 - 24.	boolean	True and False Default is False.	0..1

Name	Semantics	Type	Valid Values	Multiplicity
DeleteSource (Optional)	Indicates whether or not you want to delete the source file.	boolean	True and False Default is True.	0..1
RetainPaths (Optional)	Indicates whether or not the paths of the unzipped files are retained.	boolean	True and False Default is False.	0..1
eventState (Optional)	Indicates the latest state of the event.	string	Predefined state:DONE. Custom states can be defined in the DOMAINENTRY table.	0..1
Direction: Out				
FileName	List of the extracted file names.	arraylist		

Example for UnZipFile Activity

```
<Activity Name="Unzip">
<Action>UnZipFile</Action>
<Description>Unzip the provided file</Description>
<Execution>SYNCHR</Execution>
<Parameter name="FileName" direction="in" type="document" eval="variable">File</Parameter>
<Parameter name="CreateDateDir" direction="in" type="boolean"
eval="constant">false</Parameter>
<Parameter name="DeleteSource" direction="in" type="boolean"
eval="constant">true</Parameter>
<Parameter name="RetainPaths" direction="in" type="boolean" eval="constant">true</Parameter>
<Parameter name="Todir" direction="in" type="string" eval="constant">/home/apps</Parameter>
<Parameter name="eventState" direction="in" type="string" eval="constant">Unzip
```

```
Files</Parameter>
</Activity>
```

ValidateDocument Activity

The ValidateDocument activity is used to check the validity of a document.

ValidateDocument Parameters and Valid Execution Modes

The valid execution mode for ValidateDocument is SYNCHR or ASYNCHR.

The parameters of ValidateDocument are as follows:

ValidateDocument Parameters

Name	Semantics	Type	Valid Values	Multiplicity
Direction: In				
Validator (Optional)	Which validation to do.	string	Schema (Default)	0..1
InDocument (Optional)	See Common Parameters.			
Direction: Out				
IsValid	Returns whether the document is valid or not.	boolean	True, False	1

Example for ValidateDocument Activity

Validator: Schema

Checks if a document is validated against the supplied XML Schema.

```
<Activity Name="ValidateDocument1"><Start/>
  <Action>ValidateDocument</Action>
  <Description lang="en">Validate Document.</Description>
  <Parameter direction="in" name="InDocument" type="document"
   eval="variable">inDoc</Parameter>
  <Parameter direction="out" eval="variable" type="boolean"
   name="IsValid">DocIsValid</Parameter>
</Activity>
```

VerifyPartner Activity

The VerifyPartner activity validates the trading partner credentials specified in the incoming document or verifies if a default trading partner was present for the supplier.

The VerifyPartner activity uses credentials in the mlXML header along with the BuyerKeys specified to validate and identify the trading partner sending the message. If the activity succeeds in verifying the trading partner, it updates the mlXML header with details of the validated credentials. If none of the credentials can be validated, the default trading partner is verified.

If the default Trading Partner also does not exist, the user is asked to create a Trading Partner before processing of the document continues.

In pseudo code:

```
If (TradingPartnerCredentials were specified) {
Extract TradingPartnerCredentials from the incoming document;
Add/Update the Credentials found in the message header of the MLXML Document;
}
For every credential specified in the document {
Validate every Credentials specified in the Incoming document;
}
If (atleast one Credentials match)
   Update Mlxml Document;
   TradingPartnerFound = True;
```

```
Else // No Credentials found or validated
{
   if (Default TradingPartner specified exists)
  Update OrganisationID and Organization Name from Lookup in the MIxml document
BuyerFound=True;
   }
else {
BuyerFound=False;
}
}
```

The BuyerFound output variable of the activity can be used in the workflow to prompt a user to manually verify and add the trading partner into the system.

VerifyPartner Parameters and Valid Execution Modes

The valid execution mode for VerifyPartner is SYNCHR.

The parameters of VerifyPartnerare as follows:

VerifyPartner Parameters

Name	Semantics	Type	Valid Values	Multi-plicity
Direction: In				
BuyerKey (Optional)	An email address that could be looked up in the EMAIL domain or a duns number to look up in the DUNS domain.	string	Any key that would resolve to a Trading Partner when looked up in the specified domain.	0..1
Domain (Optional)	Specify the domain in which to look up the Trading Partner information relative to the current supplier's	string	'ZZ': Mutually defined. 'DUNS': Duns number.	0..1

Name	Semantics	Type	Valid Values	Multi-plicity
	enterprise.		'EMAIL': Email address. 'AribaNetworkID' 'NetworkID' ... and any additional domains that might be added in the future.	
InDocument (Mandatory)	See Common Parameters.			
Direction: Out				
BuyerFound (Mandatory)	Output parameter that is set to 'true' if Trading Partner is found.	boolean	True, False.	1
OutDocument (Mandatory)	See Common Parameters.			

Example for VerifyPartner Activity

```
<Activity Name="VerifyPartner"><Start/>
 <Action>VerifyPartner</Action>
 <Execution>SYNCHR</Execution>
 <Parameter direction="in" name="InDocument" type="document"
  eval="variable">inDoc</Parameter>
 <Parameter direction="in" name="Domain" type="string"    eval="xpath"
  source="//Message/Header/MessageHeader[@role='Trading
    Partner']/Credentia   l/@domain">inDoc</Parameter>
 <Parameter direction="in" name="BuyerKey" type="string"    eval="xpath"
  source="//Message/Header/MessageHeader[@role='Trading
```

```
    Partner']/Credentia   l/Identity/text()">inDoc</Parameter>
  <Parameter direction="out" name="BuyerFound" type="boolean"
   eval="variable">BuyerFound</Parameter>
  <Parameter direction="out" name="OutDocument" type="document"
   eval="variable">buyerDoc</Parameter>
 </Activity>
```

VerifyRecordStatus Activity

The VerifyRecordStatus activity retrieves the record information from the database and fills it into the mlXML document.

To look up the record, you can specify the “Agency” to use. If Agency = ‘SOURCE’, the system looks up the information using the TIBCO MDM record key. Otherwise, the system checks the GlobalPartNumber with the corresponding Agency. All information is filled into the “Supplier” record location, since the system assumes that is what we are looking up.

However, if a repository name is given, the system looks up information in GlobalPartNumber and populate that information. The record is looked up in the repository specified by the workflow, and uses the External Keys supplied in the document.

VerifyRecordStatus Parameters and Valid Execution Modes

The valid execution mode for VerifyRecordStatus is SYNCHR.

The parameters of VerifyRecordStatus are as follows:

VerifyRecordStatus Parameters

Name	Semantics	Type	Valid Values	Multi-plicity
Direction: In				

Name	Semantics	Type	Valid Values	Multi-plicity
Agency(Optional)	Where in the document to get information from.	string	'SOURCE' or 'SUPPLIER' or 'SELLER' for Supplier. Any other that matches GlobalPartNumber /Agency.	1
InDocument (Mandatory)	See Common Parameters.			
MasterCatalog (Optional)	See Common Parameters.			
OriginalDocID (Optional)	CIN Reference.	string		0..1
ProductOperation (Optional)	Applicable only for incoming messages. The operation to be used in the ProductLog Operation. If none supplied, no logging is done.	string		1
RelationshipName# (Optional)	See Common Parameters.			
Status(Optional)	Status of the incoming document.	string	Any string	0..1

Name	Semantics	Type	Valid Values	Multi-plicity
Direction: Out				
AllConfirmed (Optional)	Set to true if all products found in the given repository were confirmed.	boolean	True, False.	0..1
AllFound(Optional)	Set to true if all products were found.	boolean	True, False.	0..1
NoneInWorkflow (Optional)	Set to true if all products found in the given repository were not in workflow.	boolean	True, False.	0..1
OutDocument (Optional)	See Common Parameters.			

Example for VerifyRecordStatus Activity

```
<Activity Name="VerifyRecordStatus"><Start/>
  <Action>VerifyRecordStatus</Action>
  <Parameter direction="in" name="InDocument" type="document"
   eval="variable">inDoc</Parameter>
  <Parameter direction="in" name="Agency" type="string"    eval="constant">GTIN</Parameter>
  <Parameter direction="in" name="RelationshipName" type="string"
   eval="constant">Contains</Parameter>
  <Parameter direction="in" name="MasterCatalog" type="string"
   eval="variable">SomeName</Parameter>
  <Parameter direction="out" name="AllFound" type="boolean"
   eval="variable">AllProductsFound</Parameter>
  <Parameter direction="out" name="AllConfirmed" type="boolean"
   eval="variable">AllProductsConfirmed</Parameter>
```

```
  <Parameter direction="out" name="OutDocument" type="document"
   eval="variable">prodDoc</Parameter>
</Activity>
```

WaitForResponse Activity

The WaitForResponse activity provides a mechanism to track messages sent out to external systems. Each task is associated with a MessageID.

WaitForResponse activity also provides a mechanism to suspend the workflow and wait for a response.

Application tasks can also timeout, allowing for handling of cases where the response is not received in a set amount of time.

ApplicationTask is a specialization of the CreateWorkItem activity.

Work items created by application tasks are not visible to any user.

WaitForResponse Parameters and Valid Execution Modes

The valid execution mode for WaitForResponse is SYNCHR.

The parameters of WaitForResponse are as follows:

WaitForResponse Parameters

Name	Semantics	Type	Valid Values	Multi-plicity
Direction: In				
ExpiryDate(Optional)	Relative or absolute time when this application task	string	See ExpiryType.	0..1

Name	Semantics	Type	Valid Values	Multi-plicity
	is no longer valid. If such a time occurs without the application task being closed, the workflow is revived with a TIMEDOUT event which can be polled in the activity's outgoing transitions. If the date is not specified, no expiry date is set.			
ExpiryType(Optional)	The type of expiry date. If not specified, no expiry date is set.	string	RELATIVE or ABSOLUTE. In the first case, time is specified as 'DD:hh:mm:ss', and in the second as 'MM:DD:hh:mm:ss' where MM is month, DD is day, hh is hour, mm is minute, and ss is second.	0..1
GlobalConversationID (Optional)	Unique conversationID set for the message.	long		0..1

Name	Semantics	Type	Valid Values	Multi-plicity
InDocument (Mandatory)	See Common Parameters.			
InDocument2(Optional)	The mlXML document on which the InDocument was formed.	document		0..1
MailPresentation (Optional)	The name of an XMLC file used as the body of an email to send to a work item participant for notification. The XMLC page can have an anchor element called inboxUrl, which when specified, has its href attribute set to the address of the Inbox as specified in the Configurator. If not specified, no email is sent.	string	A class file generated using XMLC.	0..1
MailType(Optional)	The type of mail to send. If LINK, a link to the inbox is sent. If not specified, LINK is	string	LINK.	0..1

Name	Semantics	Type	Valid Values	Multi-plicity
	assumed.			
MessageID# (Mandatory)	A sequence of strings that are concatenated to derive the MessageID. A MessageID is a unique identification of the Message that was sent to Application. Replace # with an integer, for example, MessageID1, MessageID2.	string	It is an Xpath into the message generated by the Translate Step.	0..1
ParticipatID (Mandatory)	Specifies the application that must participate in this work item.	long	Any MemberID or RoleID as appropriate.	1
Direction: Out				
TaskCreated	Tells you if a work item was created.	boolean	If TaskCreated is set, and the ParticipantID or MessageID parameter evaluates to null, TaskCreated is set to False.	1

Name	Semantics	Type	Valid Values	Multi-plicity
			If no TaskCreated is specified in the activity, it throws an exception.	
OutDocument (Optional)	See Common Parameters.			

Example for WaitForResponse Activity

```
<Activity Name="WaitForResponse">
 <Start/>
 <Action>WaitForResponse</Action>
 <Description lang="en">Create a application    task.</Description>
 <Execution>SYNCHR</Execution>
 <Parameter direction="in" eval="constant" type="long"
 name="ParticipantID">34487</Parameter>
 <Parameter direction="in" eval="constant" type="string"
 name="MailPresentation">com.tibco.eml.html.  EmailProductAddE   ditWorkItem</Parameter>
 <Parameter direction="in" eval="constant" type="string"
 name="Form">standard/forms/fm23ca.xml</Parameter>
 <Parameter direction="in" eval="variable" type="document"
 name="InDocument">inDoc</Parameter>
 <Parameter direction="out" eval="variable" type="document"
 name="OutDocument">wiDoc</Parameter>
 <Parameter direction="in" eval="constant" type="string"
 name="ExpiryType">RELATIVE</Parameter>
 <Parameter direction="in" eval="constant" type="string"
 name="ExpiryDate">0:0:0:1</Parameter>
</Activity>
```

Note: The following activities are not backward compatible:

```
APPLYRULEBASE
UPDATEPRODUCTSTATUS
BIZSEND
WORKITEM
APPLICATIONTASK
BUYERVERIFY
CATALOGCONTENT
CHECKSTATUS
COMPAREPRODUCT
CONVERTRECORDSTOOUTPUTFORMANT
DUPLICATECHECK
EXECSCRIPT
GETSUBSETRECORDS
ITEMMASTER
NOOP
PROCESSRECORD
PROCESSRECORDCOLLECTION
SUBFLOW
```

A migration script has been provided for these activities (replaceactivities.sh/bat). For more details, see the *TIBCO MDM Installation and Configuration*.

ZipFiles Activity

Using the ZipFiles activity, you can compress the files located in a directory.

ZipFiles Parameters and Valid Execution Modes

The valid execution mode for ZipFiles is SYNCHR.

The parameters of ZipFiles are as follows:

ZipFiles Parameters

Name	Semantics	Type	Valid Values	Multi-plicity
Direction: In				
SourceDir (Mandatory)	Refers to the directory which contains files to be zipped.	string	Any valid directory name.	1
FileName (Mandatory)	Indicates the name of the target zip file.	string	Any valid file name.	1
Todir (Optional)	Refers to the target directory, which is relative to MQ_COMMON_DIR/Temp. **Note:** If Todir is not specified, CreateDateDir must be specified as true.	string	Any valid directory name.	0..1
CreateDateDir (Optional)	Creates the date for a directory. If true is specified, the files are placed in a directory structure yyyy/mm/dd/k. k indicates hours 1 - 24.	boolean	True and False Default is False.	0..1
TofileExt (Optional)	Indicates an extension of the target file.	string	Default is zip.	0..1
DeleteSource (Optional)	Indicates whether or not you want to delete the source directory after compressing it.	boolean	True and False Default is True.	0..1
Recurse (Optional)	Indicates whether or not you want to compress the directory recursively.	boolean	True and False Default is True.	0..1
RelativePath (Optional)	Refers to the relative path. • If you specified the path, the path of the files or directories to be zipped are retained as per the specified path. For example, if the path is set to /user/local/common/Work and if SourceDir is set to /user/local/common/Work/2003/Jan, then all the sub directories and the files under this directory are zipped with the path starting with 2003/Jan.	string	Default is null.	0..1

Name	Semantics	Type	Valid Values	Multi-plicity
	• If the path is not specified, the path of the files and sub directories are removed.			
eventState (Optional)	Indicates the latest state of the event.	string	Predefined state:DONE. Custom states can be defined in the DOMAINENTRY table.	0..1
Direction: Out				
FileName	Name of the compressed file.			

Example for ZipFiles Activity

```
<Activity Name="ZipFile">
<Action>ZipFiles</Action>
<Description>Zip the provided file</Description>
<Execution>SYNCHR</Execution>
<Parameter name="SourceDir" direction="in" type="string"
eval="constant">/home/apps/common900jboss_sql/Temp/2016/Mar/20/10</Parameter>
<Parameter name="FileName" direction="in" type="string" eval="constant">File</Parameter>
<Parameter name="Todir" direction="in" type="string" eval="constant">/A/B</Parameter>
<Parameter name="CreateDateDir" direction="in" type="boolean"
eval="constant">false</Parameter>
<Parameter name="TofileExt" direction="in" type="string" eval="constant">tar</Parameter>
<Parameter name="DeleteSource" direction="in" type="boolean"
eval="constant">true</Parameter>
<Parameter name="Recurse" direction="in" type="boolean" eval="constant">true</Parameter>
<Parameter name="RelativePath" direction="in" type="string" eval="constant"></Parameter>
<Parameter name="eventState" direction="in" type="string" eval="constant">Zip
Files</Parameter>
</Activity>
```

Transitions

Transitions define the next course of action once an activity is executed. Transitions might have a condition (guard) associated with them.

Used to “glue” activities together, transitions are located at the very end of the workflow definition. When an activity has finished executing, the Process engine goes through all transitions that have the attribute FromActivity set to the name of the activity that just finished. There could be more than one transition from one activity to another. All transitions are evaluated in the order in which they appear in the workflow file. The first one that is applicable (either has no guard or whose guard evaluates to True) is picked, and the WorkflowManager then executes the activity specified by that transition’s ‘ToActivity’ attribute.

There two types of transitions: simple transitions and conditional or guarded transitions. In addition, there are 3 special transitions: timeout, cancel and error.

Note: Do not execute any code which is not for transition evaluation. Instead, use custom activities. Do not do any transaction management.

By default, transitions are not transactional. To make transitions transactional, contact TIBCO Support.

Transition Definition

Attribute Name	Semantics	Signature
Description	A textual description. **Valid Values** Description of the transition, for documentation only.	<Name> #PCDATA </Name>

Attribute Name	Semantics	Signature
FromActivity	The 'FromActivity' attribute specifies the completed activity. **Valid Values** Valid Activity Name.	<Name> #PCDATA </Name>
ToActivity	The 'ToActivity' attribute specifies the activity which must be executed next. **Valid Values** Valid Activity Name.	<Name> #PCDATA </Name>
type	These are pre-defined transitions types. **Valid Values** • timeout • error • join • split • cancel	<Name> #PCDATA </Name>

Simple Transitions

A simple transition specifies unconditional transfer from one activity to another.

Example

```
<Transition FromActivity="ImportCatalog"    ToActivity="ManageRecordCollection">
  <Description> Transition from ImportCatalog activity to ManageRecordCollection
activity</Description>
</Transition>
```

Conditional Transitions

A conditional (guarded) transition follows the same syntax as a simple transition, but allows a Rule tag inside the Transition tag.

The content of the Condition tag is evaluated using the Bean Shell Java interpreter, and the result is used to determine if the transition is valid or not.

Tag	Semantics	Signature	Valid Values
Rule	The Rule tag encloses a set of in parameter tags, a single out parameter tag, and a condition tag.	<Rule>…</Rule>	Parameter and Condition tags.
Condition	Specifies a condition which must assign a boolean object to the special internal variable ‘result’, indicating whether or not this transition can be fired.	<Condition> #CDATA </Condition>	The condition expression must be a valid Java statement or a valid JAVA method call, which has a net result of assigning a boolean object to the special variable ‘result’.
Format	This is an attribute of the Condition tag.	<Condition format=<format>> #CDATA </Condition>	bsh java (version 7.1 onwards)
Parameter	Input arguments passed to the condition.		

The parameters of a guarded transition specify input variables that can be used inside the bean shell expression. The WorkFlowManager uses the parameter attribute ‘name’ as the name of the variable in the scope of the bean shell. A guarded transition can have any number of input variables, but is required to have only one output variable, which must be named result.

bsh and Java formats

bsh and java formats are also supported for workflow transitions. The transition format bsh is compiled every time it is executed, and this is not very optimized. Using the java format (which executes transition code using reflection) is faster.

The java condition transition must be within the activity and specified as:

```
<condition format=java>
classname.method name (pass argumnts)
```

Using Java transitions

Convert an existing transition to a Java transition. Create a custom class using Java condition code specified in the workflow.

- Method signature must be public static boolean <methodname>(<argument list>).
- Method overloading is *not* supported.
- Methods can be reused – the same method can be called for multiple transitions if condition code is the same.
- Method argument must be an Object. For primitives, use wrapper classes.

Commonly Used Parameter Types in Workflows

The following is a list of commonly used parameter types in the workflow and corresponding method argument types.

Parameter type	Method argument
string	java.lang.String
long	java.lang.Long

Parameter type	Method argument
boolean	java.lang.Boolean
recordlist	com.tibco.mdm.repository.core.IRecordCollection
arraylist	java.util.List

recordlist is, typically, used for null check. So, the argument type can also be java.lang.Object.

Next, place the JAR file in the application server's CLASSPATH.

Replace Transition Code

Replace transition code in the workflow with a class.method call.

Some points to note:

- Condition format must be java - <Condition format="java">
- No statement other than classname.methodname call must be present in the CDATA section of the condition.
- Specify fully qualified class name (class name along with the package name).
- Parameter name and the argument name specified in the classname.methodname call must match.

bsh Example

```
<Transition FromActivity="CheckMasterCatalog" ToActivity="GetMasterCatalogId">
  <Description>If mastercatalog is not equal to 'ITEM_VENDOR_NUMBER', then    go to
'GetMasterCatalogId' activity</Description>
  <Rule>
    <Parameter name="MasterCatalog" direction="in"
```

```
type="string"      eval="xpath"source="/Message/Body/Document/OriginalDocument/Message/
Body/Document/BusinessDocument/CatalogAction/CatalogActionHeader/      MasterCatalog/Revi
sionID/BaseName/text()">inDoc</Parameter>
      <Parameter name="result" type="boolean" direction="out" />
     <Condition format="bsh">
       <![CDATA[
                                result = !(MasterCatalog.equalsIgnoreCase("ITEM_VENDOR_NUMBER") ||
MasterCatalog.equalsIgnoreCase("STATUS_TICKLER"));
     </Condition>
  </Rule>
</Transition>
```

Java Example

```
<Transition FromActivity="CheckMasterCatalog" ToActivity="GetMasterCatalogId">
  <Description>If mastercatalog is not equal to 'ITEM_VENDOR_NUMBER', then    go to
'GetMasterCatalogId' activity</Description>
  <Rule>
     <Parameter name="MasterCatalog" direction="in" type="string"      eval="xpath"
     source="/Message/Body/Document/OriginalDocument/Message/Body/Document/       Busines
sDocument/CatalogAction/CatalogActionHeader/MasterCatalog/Re       visionID/BaseName/text
()">inDoc</Parameter>
     <Parameter name="result" type="boolean" direction="out" />
     <Condition format="java">
       <![CDATA[
com.tibco.ConditionEval.addMsgInfoToEventToCheckMasterCatalog(MasterCatalog);
     </Condition>
  </Rule>
</Transition>
```

Java Example where data source upload must be done first.

```
<Transition FromActivity="AddMsgInfoToEvent" ToActivity="ProcessExportArchive">
  <Description lang="en">If ExportArchive param is present, this is special import case for Data
Transfer/Roundtrip which is handled by ProcessExportArchive activity</Description>
  <Rule>
     <Parameter direction="in" name="exportArchive" type="string" eval="xpath"

source="/Message/Body/Document/BusinessDocument/CatalogAction/CatalogActionHeader/Mast
erCatalog/Extension[@name='ExportArchive']/Value/text()">inDoc</Parameter>
     <Parameter name="result" type="boolean" direction="out"/>
     <Condition format="java"><![CDATA[
```

```
com.tibco.mdm.workflow.engine.transition.WfSharedConditionTransition.isNotNull(exportArchive);
    ]]></Condition>
  </Rule>
</Transition>
<!--Data source upload needs to be done first -->
<Transition FromActivity="AddMsgInfoToEvent" ToActivity="UploadDataSource"/>
<Transition FromActivity="ProcessExportArchive" ToActivity="UploadDataSource"/>
```

Special Transitions

To process exceptions, the following predefined transitions are supported.

For each of these transitions, the form activity can be specified as “any” to indicate this transition is applicable for any activity.

Timeout Transitions

A timeout transition is executed when an activity times out. This allows interception of timeout processing, that is, sending an email notification.

To specify this transition, specify type = “timeout”. The timeout transition is optional.

```
<Transition type="timeout" FromActivity="Any"
  ToActivity="SendTimeOutEmailForTargettedRejectionWorkItem"/>
<Transition type="timeout" FromActivity="TargettedRejectionWorkItem"
ToActivity="SendTimeOutEmailForTargettedRejectionWorkItem"/>
```

Error Transition

An error transition is executed whenever an activity fails with a technical exception.

If no error transition is specified, the workflow stops. If an error transition is specified, the workflow starts execution from this transition.

Though optional, it is a good practice to specify an error transition. To specify this transition, specify type = “error”.

```
<Transition type="error" FromActivity="Any"
ToActivity="SendTimeOutEmailForTargettedRejectionWorkItem"/>
<Transition type="error" FromActivity="TargettedRejectionWorkItem"
ToActivity="SendTimeOutEmailForTargettedRejectionWorkItem"/>
```

Cancel Transition

A cancel transition is executed whenever the user attempts to cancel the workflow using GUI.

The transition executes only if workflow is not currently running and if it is semantically possible to cancel the workflow.

A cancel transition is optional and is normally not required. To specify this transition, specify type = “cancel”.

```
<Transition type="cancel" FromActivity="Any"
ToActivity="SendTimeOutEmailForTargettedRejectionWorkItem"/>
```

Split and Join

Parallel transitions refer to transitions that execute simultaneously. From a given activity or depending on the outcome of a given activity, you can have more than one activity executing in parallel.

For example, once the status of a record is set to "approved", suppose work item notifications must get sent to three different users with different roles. Each of these notifications is created in parallel, and after the necessary action is taken by all three, the workflow proceeds to the next task.

To define these transitions following predefined transition types, split and join can be used.

split

When the transition type is defined as split, it forks execution for each transition that evaluates to true from the specified fromActivity.

```
<Transition FromActivity="UpdateProductStatusAsApproved" ToActivity="NotifyWorkItemPricing"
type="split"/>
<Transition FromActivity="UpdateProductStatusAsApproved"
ToActivity="NotifyWorkItemSchematics" type="split"/>
```

join

This transition type specifies the point where split or parallel transitions can join and proceed with the activity execution specified by the ToActivity.

```
<Transition FromActivity="NotifyWorkItemPricing" ToActivity="SetStatusToSuccess" type="join"/>
<Transition FromActivity="NotifyWorkItemSchematics" ToActivity="SetStatusToSuccess"
type="join"/>
```

group

The join transitions can be grouped together by specifying a group attribute. If a group attribute is not specified, all join transitions belong to the default group.

grouplimit

Used with the group attribute to indicate the number of transitions to be completed so join can proceed.

For example, if you have five parallel transitions and the grouplimit specified as three, the join takes place after completing three transitions. If not specified, all activities in the join group are required to complete for join to proceed.

```
<Transition FromActivity="NotifyWorkItemSchematics" ToActivity="SetStatusToSuccess"
type="join" group="notify" grouplimit="3"/>
```

Subflow Management

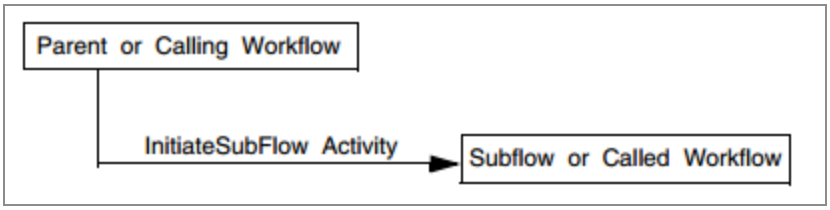
You must manage small workflow segments which then can be used together to compose bigger workflows. This makes management of configuration easy.

In any Development environment, there might be a need for various teams to develop workflow components independently and then merge them together.

For example, different teams can focus on configurations for different catalogs and then put them together.

You can call one workflow from another using a workflow activity called InitiateSubFlow. You can pass the required parameters and get the required output without any change to the local state of the calling workflow, except for the state transferred back using output parameters.

Managing Subflows



For more information about the syntax of the InitiateSubFlow activity, see InitiateSubFlow Activity.

Sub workflows can be nested, that is, a subflow can invoke another workflow as a subflow.

Example

Here is an example of where subflows could be used. Let us say there are two repositories - LegalEntity and Account.

When a record is added to the LegalEntity repository, the business process is defined in workflow L1. When a record is added to the Account repository, the business process is defined in workflow A1.

When a record is added to both the LegalEntity and Account catalogs, another workflow LA1 must be executed. This workflow LA1 performs some tasks, then calls workflow L1, followed by workflow A1. After return from A1, records are confirmed in workflow LA1.

This can be implemented using subflows. LA1 can call L1 and A1 as subflows and receive status from each of the workflows to decide if the process must continue.

Calling a Workflow

The workflow to be called can be selected in one of the following ways:

- Specify the exact workflow to be executed.
- Do not specify a workflow to be executed; let the workflow selection be done using the configuration file. The workflow engine parses the workflow configuration file and selects the applicable workflow.

In either case, you can specify whether the initiation of the new workflow must be done in the synchronous or asynchronous mode.

Input to Subflow

The input that must be passed from the parent workflow to the subflow, must be passed as input to the InitiateSubFlow activity.

When the subflow is initiated, all input parameters passed to the InitiateSubFlow activity are passed on to the subflow. This is applicable for all subflow initiations.

A workflow can have any number of input parameters. When a workflow is called directly (not as a subflow), the only parameter usually passed to it is the input document. In some cases, when a workflow is called using the SpawnWorkflow activity, a record collection might also be passed as input. Any other input parameters, if specified, are uninitialized (null).

When the same workflow is called as a subflow, all input parameters might be initialized based on inputs specified in the calling workflow.

The input parameters of the InitiateSubFlow activity are matched with input parameters specified for the subflow. Even though the InitiateSubflow activity can have any number of input parameters, the subflow only receives those parameters which are defined as input parameters to the workflow.

Output

A workflow can have any number of output parameters. Even though output parameters are specified, they make sense in some situations only.

When a workflow is called directly or through the SpawnWorkflow activity, no outputs are returned to the caller, all outputs are discarded. However, when the workflow executes as subflow, the output parameters are passed to the caller workflow (in SYNCHR mode only).

The output parameters are also evaluated similar to input parameters. Output parameters of the subflow are matched by name with the output parameters specified in the calling InitiateSubFlow activity. Only matched parameters are transferred.

- **Asynchronous mode:** When asynchronous mode of initiation is specified, the parent or calling workflow does not receive any output back from the subflow.
- **Synchronous mode:** When synchronous mode of initiation is specified, all output parameters can be returned. The first document output parameter is also set as OutDocument.

All parameters specified as output parameters in the InitiateSubflow activity can be received from the subflow.

Identifying the State of a Workflow

When a subflow is initiated, a predefined variable isSubFlow is set to true. Using this variable, you can find out whether a workflow is running as a subflow.

This applies even when a workflow is initiated in the asynchronous mode.

The state of the subflow is independent of the calling workflow even though both workflows are part of the same event.

Error Handling in Subflow

If an exception occurs during execution, it is propagated to the calling workflow when global error handling is not defined in the subflow.

If a global error transition is defined in the subflow, it is executed on exception, and control is returned to the parent workflow to resume normal execution. To handle the subflow exceptions in a parent workflow, you can set an error flag in the subflow that can

be returned to the parent workflow. The parent workflow can execute an error transition based on this flag.

Example of Error Handling in Subflow

Subflow snippet:

```
<?xml version="1.0"?>
<Workflow Version="0.1">
  <Owner>TIBCO</Owner>
  <Name>wfin26test1</Name>
  <Description lang="en">Test subflow activity</Description>
  <Parameter direction="in" eval="variable" type="document" name="inDoc">1</Parameter>
  <Parameter direction="out" name="errorFlag" type="boolean"
eval="variable">errorFlag</Parameter>
  <Activity Name="SetErrorFlag">
    <Action>NoOperation</Action>
    <Description lang="en">Set the error flag</Description>
    <Parameter direction="in" name="errorFlag" type="boolean" eval="constant">true</Parameter>
  </Activity>
  <Transition type="error" FromActivity="Any" ToActivity=" SetErrorFlag" />
</Workflow>
```

Parent workflow snippet:

```
<Activity Name=" CallTestSubflow">
  <Action>InitiateSubFlow</Action>
  <Description>Calling Test Subflow</Description>
  <Execution>SYNCHR</Execution>
  <Parameter direction="in" type="string" eval="constant"
  name="eventState">SPAWNWORKFLOW</Parameter>
  <Parameter direction="in" type="string" eval="constant"
  name="ProcessID">standard/workflow/wfin26Test1</Parameter>
  <Parameter direction="in" type="document" eval="variable"
  name="InDocument">inDoc</Parameter>
  <Parameter name="errorFlag" type="boolean" eval="variable"
  direction="out">errorFlag</Parameter>
</Activity>
<Transition FromActivity="CallTestSubflow" ToActivity="RejectDueToWorkflowError">
  <Description>If error in subflow</Description>
  <Rule>
    <Parameter name="errorFlag" type="boolean" eval="variable"
    direction="in">errorFlag</Parameter>
    <Parameter name="result" type="boolean" direction="out" />
```

```
    <Condition format="bsh">
      <![CDATA[
                                    result =  (errorFlag);
                                    System.out.println("Error Flag in Parent workflow - " + errorFlag);
    </Condition>
  </Rule>
</Transition>
```

Example of Parent or Calling Workflow

An example of a parent workflow is as follows:

```
<Activity Name="TestSubflow">
<Action>InitiateSubFlow</Action>
<Description>Spawn the subworkflow for generation of DBDump</Description>
<Execution>SYNCHR</Execution>
<Parameter direction="in" type="string" eval="constant"
name="eventState">SPAWNWORKFLOW</Parameter>
<Parameter direction="in" type="string" eval="constant"
name="ProcessID">standard/workflow/wfin26Test1</Parameter>
<Parameter direction="in" type="document" eval="variable"
name="InDocument">inDoc</Parameter>
<Parameter direction="in" type="String" eval="constant"
name="InStatus">TestInStatus</Parameter>
<Parameter direction="out" eval="variable" type="String" name="status">finalStatusx</Parameter>
<Parameter direction="out" eval="variable" type="boolean"
name="statusFlag">finalStatusFlagx</Parameter>
<Parameter direction="out" eval="variable" type="long"
name="statusValue">statusValuex</Parameter>
<Parameter direction="out" eval="variable" type="recorList"
name="outRecordList1">workRecordList</Parameter>
<Parameter direction="out" eval="variable" type="string"
name="Testunknown">unknownValue</Parameter>
</Activity>
```

Input Parameters

This activity has the following input parameters which are passed to the subflow. Note that the parameters received by the subflow are controlled by the input parameters defined in the subflow.

- eventState – defined as constant

- inDocument – passed as inDoc
- InStatus – defined as constant

Output Parameters

This activity expects the following output parameters:

- outRecordList1 – Record List. The value is assigned to the workRecordList local variable.
- statusFlag – Boolean. Value is assigned to the finalStatusFlagx local variable.
- status – String. Value is assigned to the finalStatusx local variable.
- statusValue – Long. Value is assigned to the statusValuex local variable.
- Testunknown – String. Value is assigned to the unknownValue local variable.

Subflow or Called Workflow Example

The workflow described in the previous section calls the following workflow using the InitiateSubFlow activity:

```
<?xml version="1.0"?>
<Workflow Version="0.1">
<Owner>TIBCO</Owner>
<Name>wfin26test1</Name>
<Description lang="en">Test subflow activity</Description>
<Parameter direction="in" eval="variable" type="document" name="inDoc">1</Parameter>
<Parameter direction="in" type="String" eval="variable" name="InStatus">TestInStatus</Parameter>
```

```
<Parameter direction="out" eval="variable" type="document" name="OutDoc">workDoc</Parameter>
<Parameter direction="out" eval="variable" type="String" name="status">finalStatus</Parameter>
<Parameter direction="out" eval="variable" type="boolean" name="statusFlag">finalStatusFlag</Parameter>
<Parameter direction="out" eval="constant" type="long" name="statusValue">10</Parameter>
<Parameter direction="out" eval="variable" type="recordList" name="outRecordList1">workRecordList</Parameter>
<Activity Name="AddMsgInfoToEvent">
<Action>UpdateEvent</Action>
<Start/>
<Description lang="en">Initialize Event Info</Description>
<Parameter direction="in" type="string" eval="constant" name="eventState">START</Parameter>
<Parameter direction="in" name="eventDescriptor" type="string" eval="xpath"
source="/Message/Body/Document/@subtype">inDoc</Parameter>
<Parameter direction="in" eval="constant" type="String" name="finalStatus">finalStatus1</Parameter>
<Parameter direction="in" eval="constant" type="boolean" name="finalStatusFlag">true</Parameter>
<Parameter direction="in" name="deploymentMode" type="string" eval="xpath"
source="/Message/@messageType">inDoc</Parameter>
<Parameter direction="in" name="dbDirectLoadDump" type="string" eval="constant">N</Parameter>
<Parameter direction="in" name="eventType" type="string" eval="constant">CAT</Parameter>
</Activity>
```

```
<Activity Name="TestSubflow">
<Action>NoOperation</Action>
<Description>Test Subflow</Description>
<Execution>ASYNCHR</Execution>
<Parameter direction="in" type="string" eval="constant" name="eventState">SPAWNWORKFLOW</Parameter>
<Parameter direction="in" type="document" eval="variable" name="InDocument">inDoc</Parameter>
</Activity>
<Activity Name="ManageRecordCollection">
<Description lang="en">Create Bundles in the Record Collection</Description>
<Action>ManageRecordCollection</Action>
<Parameter direction="in" type="string" eval="constant" name="eventState">CREATEBUNDLE</Parameter>
<Parameter direction="in" name="BundlingOption" type="boolean" eval="constant">false</Parameter>
<Parameter direction="in" name="VersionOption" type="string" eval="constant">LATEST</Parameter>
<Parameter direction="out" name="OutRecordList" type="recordlist" eval="variable">workRecordList</Parameter>
</Activity>
<Transition FromActivity="AddMsgInfoToEvent" ToActivity="TestSubflow"/>
<Transition FromActivity="TestSubflow" ToActivity="ManageRecordCollection"/>
</Workflow>
```

Input Parameters

This workflow has the following input parameters:

- InDocument – This parameter is always passed implicitly.
- InStatus – String. Assigned the value passed by the TestInStatus parameter of the calling workflow.
- isSubFlow – Boolean. Implicitly passed and set as true when this workflow is called as a subflow. Otherwise, this value is false.

Note: When the subflow executes, it does not receive the eventState parameter even though it is passed by the parent workflow. This is because it is not defined as an IN parameter in the subflow.

Output Parameters

This workflow has following output parameters:

- outRecordList1 – Record list. Value is assigned by local variable workRecordList which is the output of the ManageRecordCollection activity.
- statusFlag – Boolean. Value is assigned by local variable finalStatusFlag which is the output of AddMsgInfoEvent activity.
- status – String. Value is assigned by local variable finalStatus which is the output of the AddMsgInfoEvent activity.
- statusValue – Long. Assigned a constant value of 10.

Note: When the subflow completes, the output parameter Testunknown is null as it is not returned by the subflow.

Workflow Customization

Each workflow activity is controlled by a rule - an encapsulated piece of business logic that specifies parameters for a record.

For example, the Notification activity might use a rule such as "All notifications from Retailer1 must be routed to the VP of Sales."

The SpawnWorkflow activity can start a new workflow based on a given workflow template. You can modify one of the existing standard workflows, using it as a template, rather than building a new workflow from scratch.

For example, in the New Record Introduction Edit (NRIE) workflow, all new product introductions and modifications are routed to the responsible parties for data entry and approval. You can use the SpawnWorkflow activity to customize the standard NRIE workflow. The SpawnWorkflow activity initiates and routes one or more new events; you can configure the type of event and determine who is notified about it. You could structure the workflow so that, in addition to notifying the responsible parties about new products, when a new product is confirmed, a "publish" workflow is started and the record is sent to the data pool.

Spawning the Catalog Synchronization Workflow at the End of Another Workflow

New and changed items can be synchronized to occur as part of any workflow in the system.

For example, after completing an NRIE workflow, the NRIE workflow can automatically be synchronized with data pools and trading partners that must receive this item.

Note: This optional capability requires configuration by TIBCO Professional Services.

A *subscription catalog* is created. It stores custom attributes of the following form:

Custom Attributes of Subscription Catalog

Brand	UDEX Code	Data Pool	Trading Partner
Drills	05.0577.05*	WWRE	Retailer1
Drills	05.0577.05*	WWRE	Retailer2

Using the rulebase, any workflow can be configured to evaluate the subscription catalog against matching product attributes. The output is a list of matching data pools and trading partners. This output list is used to synchronize the new or changed item with the resulting data pools or trading partners.

All workflow actions and statuses are recorded in the Event Log.

Bundling Records

You can simplify the workflows by bundling the related records together, and hence avoid duplication of notifications and activities.

Writing Custom Workflows

Each workflow is in XML format and stored in a file. The main components are Activities and Transitions.

Activities define what is to be done, and Transitions tie together Activities. You can write custom workflows using the correct combination of Activities and Transitions.

Customizing wfin26catsynchv7.xml

Customize the wfin26catsynchv7.xml workflow to pass relationships other than Contains while exporting through synchronization profile (local publish).

To process synchronization or local publish, a workflow must be modified to include the cross-repository relationships.

By default, “Contains” is used in the IdentifyRecordVersions activity as follows:

```
<Parameter direction="in" name="relName" type="string" eval="constant">Contains</Parameter>
```

To allow synchronization of records with other relationships:

Procedure

1. Enter the relationship name in the IdentifyRecordVersions activity as mentioned for Contains. Ensure that the name is unique. For example,

```
<Parameter direction="in" name="relName" type="string"
eval="constant">Contains</Parameter>
<Parameter direction="in" name="relName1" type="string"
eval="constant">Billing</Parameter>
<Parameter direction="in" name="relName2" type="string"
eval="constant">Shipping</Parameter>
```

   For new relationships, the relationship names are passed to the next activity, that is, EvaluateSubset activity. For example,

```
<Parameter direction="in" name="RelationshipName" type="string"
eval="variable">relName</Parameter>
<Parameter direction="in" name="RelationshipName1"  type="string"
eval="variable">relName1</Parameter>
<Parameter direction="in" name="RelationshipName2"  type="string"
eval="variable">relName2</Parameter>
```

2. Rename relationship name Contains to ALL. This passes all relationships.

Workflow Modes

TIBCO MDM supports two workflow modes: Standard Workflow Processing and In-memory Workflow Processing.

Standard Workflow Processing (Transactional)

Any new process is initiated through a workflow message. These messages are posted to queues for processing. After a message is received on the JMS queue, an event (associated with the message) and documents are created from it and stored in the distributed cache. The process to be executed is decided based on the event and document type and subtype (defined in the workflow manager configuration file).

If the process is to be executed in the transactional mode, the event is persisted to the database and the document to the file system. The name of the file is associated with the event in the database. The event is then posted to a workflow queue for processing.

The event type and subtype determine workflows triggered for an event, and the workflow reads the message associated with the event from the file system and starts a new process. This file is the main input parameter to the workflow. Each workflow has a set of activities which in turn have input and output parameters.

Activity-state changes are stored to the database. Activities can generate documents (written to the file system) that act as output parameters for that activity and input parameters to a subsequent activity in the flow.

In-Memory Workflow Processing

TIBCO MDM supports an in-memory workflow execution mode where workflow process states are managed only in-memory. The following are the features of in-memory workflows:

- Provides a way to get higher throughput
- Reduces the required space as you might record much less data
- Reduces the need to run purge more often

Except for the batch activities, such as import and mass update all other activities can run in-memory. You can control when to commit the CheckpointWorkflow activity and to switch over to a regular workflow by firing a subflow.

For more information about how in-memory workflow works, see In-Memory Workflows - How it Works.

Regular Workflows versus In-Memory Workflows

In-memory workflows provide almost double the throughput as compared to regular workflows. Moreover, this gap widens as more and more data accumulates, regular workflows get slower unless you purge history regularly.

Regular Workflows versus In-Memory Workflows

Regular Workflows	In-Memory Workflows
During each activity execution, process states are persisted to database and distributed cache.	During each activity execution, process states are written to the local cache.
Input, output, and intermediate documents are persisted to the file system.	Input, output, and intermediate documents are maintained in the local cache.
Each activity is considered a transaction.	The entire workflow (consisting of multiple activities) is considered a transaction.
When a transaction is committed, all workflow changes (process states, documents, record related data) in that activity are committed.	When a transaction is committed, record related data in the workflow is committed. If the saveonsuccess parameter is on, data from the local cache (process states, documents) is committed.

In-Memory Workflows - How it Works

When workflows are set to in-memory, all the workflow data is retrieved from the cache and you can run workflows without persisting workflow state information.

The following object types are put in memory:

- Event
- EventDetails
- Process
- ProcessLog
- ProcessDetails
- ProcessState
- AttributeLog
- MLXMLDoc

Workflow requests that are written and read from a file are now part of the JMS payload. Input and output parameters that are XML files, are stored in memory and are not written to a disk. When a workflow is executed, the workflow states and documents are managed in the local cache.

You can enable or disable the execution of in-memory workflows (in which case, workflows are executed in the transactional mode) using the Configurator. For more information, see Configuring In-memory Execution through the Configurator.

After a message is received, and the associated event and document created and stored in the distributed cache, if the workflow execution mode is set to **in-memory**, the event and document are copied from the distributed cache to the local cache. Activities in the processes are executed sequentially in a single thread on a node in the cluster.

Note: If there are any asynchronous activities in a workflow set to process in-memory, such activities are executed synchronously.

Persisting Workflow States

The persisting workflow states are suspended workflow, failed workflow, checkpointworkflow activity, and subprocesses.

Suspended Workflows

A workflow might suspend on account of a single activity, such as one awaiting user action before the workflow can resume. In case of such suspended workflows, the entire

state of the workflow is persisted to the database and file system. This persisted information is used to show the necessary information on the UI so that you can take the required action on the work item and the workflow can resume.

Failed Workflows

If a workflow fails, it cannot proceed further and the entire workflow state is written to the database and file system. You can locate the exact reason for the workflow failure on the Event Log page.

CheckPointWorkflow Activity

If a workflow contains the special CheckPointWorkflow activity, it forces the workflow to commit the transaction and persist (to the database and file system) the workflow state and all documents till that point in the workflow. For more details, see CheckpointWorkflow Activity.

Subprocesses

The following rules apply to subprocesses:

- If the invoking activity (InitiateSubflow, SpawnWorkflow, or InitiateWorkflow) is asynchronous, the activity prior to the asynchronous activity is persisted.
- If the invoking activity is synchronous, the subprocess must be run in-memory, and the subflow cannot contain another InitiateSubflow, SpawnWorkflow activity, or CheckPointWorkflow activity.

Transaction Scopes

In regular (transactional) workflows, each activity in the workflow is executed in a separate transaction and the workflow state changes and records data committed at the end of a transaction.

For in-memory workflows (designed for performance), the workflow states are not persisted to the database, but are maintained in the local cache. Typical in-memory workflows have sub-second responses. By persisting record changes at the end of the workflow instead of for each activity, the entire workflow becomes the scope of the transaction.

If there are any special activities or if any special conditions occur during the workflow execution, some variations in transaction scope exist:

Transaction Scopes

In-Memory Workflow Condition	Transaction Scope
Has synchronous activities	From the first activity to the last activity of the workflow.
Has asynchronous activities	From the first activity to the last activity of the workflow.
Has InitiateSubflow activity executed in synchronous mode.(Subflow can only be an in-memory workflow).	From the first activity to the last activity of the parent workflow. No new transaction is initiated for the child subflow.
Has InitiateSubflow activity executed in asynchronous mode. (Subflow can be in-memory or regular workflow).	The four transactions for this workflow are: Transaction from the first activity of the parent workflow to the activity before the InitiateSubflow activity. Transaction for the subflow. Transaction from the InitiateSubflow activity to the last activity in the workflow.
Has SpawnWorkflow activity executed in asynchronous mode. (Spawned workflow can be in-memory or regular)	The four transactions for this parent workflow are: Transaction from the first activity of the parent workflow to the activity before the SpawnWorkflow activity. Transaction for the spawned workflow. Transaction from the SpawnWorkflow activity to the last activity in the parent workflow.
Workflow suspends	The two transactions for this workflow are:

In-Memory Workflow Condition	Transaction Scope
	Transaction from the first activity of the parent workflow to the activity that suspends. Transaction from the suspended activity to the last activity in the workflow.
Workflow errors out	From the first activity of the workflow to the last activity where the workflow errors out.
Has CheckPointWorkflow activity.	The two transactions for this workflow are: Transaction from the first activity of the parent workflow to the CheckPointWorkflow activity. Transaction from the activity after the CheckPointWorkflow activity to the last activity in the workflow.

Recovery and Failure

Recovery for in-memory workflows works the same as transactional workflows for the most part; a few special cases are handled differently.

The necessary information is persisted to the cache, database, or filesystem for recovery. In-memory workflows can recover from the following failure conditions:

- **Node Crash** If the application server on which the workflow is running crashes.
- **Subsystem Failure** If one or more of the subsystems (database, filesystem, JMS, or cache) are not operational.

Based on the transaction scopes (see Transaction Scopes), the required information for a process to be newly created or restarted from the last persisted state is stored.

Node Crash

If a node crashes, messages that initiated the process are not consumed, but redelivered to another node in the cluster or the same node if it becomes operational.

When a message is redelivered, the recovery mechanism starts up.

- If there is a process associated with the incoming message in the system, a new process is created and the workflow executed from the first activity.
- If a process was already created, the process resumes from the last point the state of the process was persisted.

Subsystem Failure

In case a subsystem fails, the in-memory workflow waits and retries (based on configuration parameters) for the subsystem(s) to become operational. This results in incoming message consumption without action.

The Wait and Retry processes differ for transactional workflows and in-memory workflows.

- For transactional workflows, the retry mechanism is handled at the activity level, as the scope of the transaction is the activity.
- For in-memory workflows, the retry mechanism is handled at the process level, as the scope of the transaction is the entire workflow.

When a subsystem fails, the following steps are executed:

The transaction is rolled back. All changes to the record data are rolled back.

Record data updated to the distributed cache also must be rolled back, but as the cache system does not support transactional semantics, the rollback action is simulated. All record data modifications associated with this process are removed from the cache.

A retry checks if all the subsystems are up and running. A utility checks if the major subsystems (database, file system, JMS and cache) are running. Only when all the subsystems are running is the process re-executed.

When the subsystems are running, a new message from the event, the document and process related information of the current process is generated and published to the workflow queue. This message has a special flag re-delivered, set to true.

The event, document, and process related information is cleared from the local cache.

On receiving the message with the redelivered flag set to true, the regular workflow recovery mechanism kicks in.

- If no process is associated with the incoming message in the system, a new process is created and executed from the first activity.
- If there is a process associated with the incoming messages, the process is resumed from the last activity where the process related information has been persisted.

Increasing Workflow Throughput

In the initial phase of workflow message processing (for both in-memory and regular workflows), important information such as event, eventdetail, document, process, and processlog are stored in the cache.

A period of vulnerability exists where if the cache server goes down before the workflow listener picks up the message, important information related to that message is lost.

A new Save state before sending workflow message (Advanced View, Workflow Settings) property set to true by default, controls whether all initial information related to the workflow message must be persisted to the database and file system.

Warning:

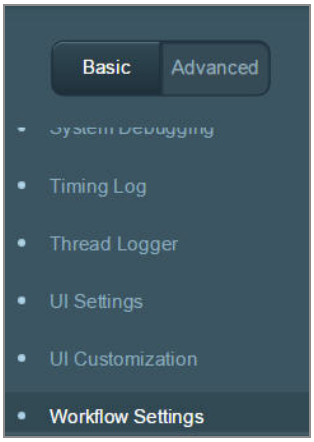
- If you are using only regular workflows, do not change any configuration value.
- In high throughput (in-memory) installations, the value for this Save state before sending workflow message (Advanced View, Workflow Settings) property must be changed to false.

Note: When Save state before sending workflow message is set to true, a single process log is visible on the event log UI page for an event.

If additionally, Save status on success is false, the process log is not persisted.

Configuring In-memory Execution through the Configurator

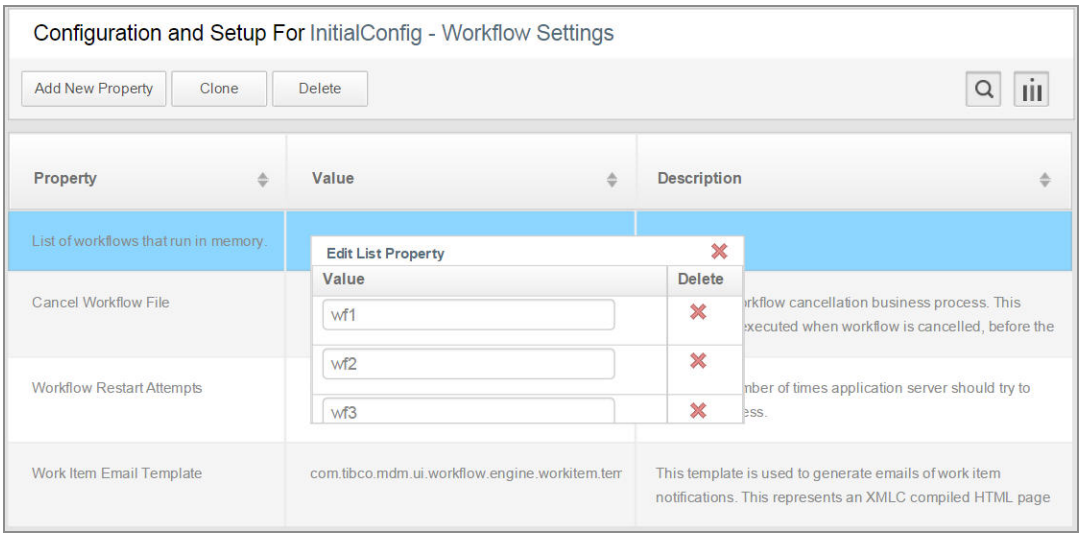
You can enable in-memory workflows using the Configurator. Go to the **Advanced** > **Workflow Settings**.



Configure the following two properties:

- **List of Workflows that run in memory** In the Value column of this property, you can enter the names of the workflows that you want to run in memory. Click the cross icon to remove a specified workflow from in memory execution.
- **List of workflows that run in-memory and their state need to persist on success** In the Value column of this property, you can enter the names of the workflows that you want to run in memory and persist to the database on success. Click the cross icon to remove a specified workflow from in memory execution and database persistence.

Note: These properties are not hot deployable. You must exit the application, add or remove the workflow name from the list of in-memory workflows, and restart the server for the execution mode to take effect.



In-Memory Workflows Impact on UI

In-memory workflows have the following impact on the User Interface:

Checking Progress in the Event Log

When a workflow is run as in-memory, no event is generated. As a result, you cannot check the event details for operations triggered by an in-memory workflow. If you have configured in-memory workflows that must be persisted on success (saveonsucess flag has been set to true), you can see the result in the Event Log after the workflow has completed executing.

Intermediate event logs are available if the workflow contains a CheckpointWorkflow activity, async subflow, spawn workflow activity, or if the activity suspends. If a workflow errors out, the entire Event Log becomes available for debugging.

If a data source upload or an import operation triggers an in-memory workflow, the **checkprogress** link (which can be used to track progress in the case of regular workflows) redirects you to an appropriate page with a message that the operation is in-memory.

Checking Progress of Records for Workflows Run In-Memory

If you add a record in-memory, you can check the success of the workflow by searching for a record in the repository or checking the record count in the repository.

Record history does not contain any reference to events processed in-memory. The link to the event is disabled, and the event column shows a message "in- memory”.

CheckpointWorkflow Activity

For in-memory workflows, since the workflow state is not saved to the database during the workflow execution, in case a failure or recovery scenario occurs, workflow execution starts from the first activity in the workflow.

To avoid such re-execution of workflows post failure, using the CheckpointWorkflow activity, the process state can be saved to the database at workflow milestones. CheckpointWorkflow saves minimal data required for recovery, and in case of failure, workflow execution starts with the next activity.

Note: The CheckpointWorkflow activity has no impact on regular (transactional) workflows, and it behaves like a NoOperation activity. For regular workflows with Cache documents enabled, CheckpointWorkflow persists all documents in the workflow in that activity.

How It Works

If a CheckpointWorkflow activity is present in a workflow, it persists minimal process-related objects required for recovery, such as the event, event details, process, process details, the last process log, all non-eliminated documents, and the process state.

No process data is persisted before the CheckpointWorkflow activity in an in-memory workflow. After CheckpointWorkflow is executed, the data required for recovery is saved for that process, and the required data is committed to the database.

If multiple CheckpointWorkflow activities are present in a single in-memory workflow, the first time CheckpointWorkflow runs, it saves the required process data, and for subsequent times, the process data (but not the process log and documents) is updated. After that, the last process log and non-eliminated documents are saved. The transaction initiated after the last CheckpointWorkflow is committed and a new global transaction is started for the next activity.

CheckpointWorkflow in a Subflow

For in-memory workflows, when a checkpoint (implicit or explicit) occurs, the transaction is committed and the event processing resumes from the activity following the checkpointed activity.

Limitations for In-memory Workflows

The following are limitations of in-memory workflows:

- Do not use the following activities for in-memory workflows are as follows:
    - UploadDataSource
    - Purge
    - ProcessServiceMessage
    - ImportCatalog
    - ImportClassificationScheme
    - ReclassifyRecord
- Do not use in-memory workflows in case of CIM2CIM.
- Mass update is not supported for in-memory workflows.
- Synchronization Export is not supported for in-memory workflows.

For best practices of in-memory workflows, see *TIBCO MDM Best Practices*.

Workflow Optimizations

Workflow optimizations are done to improve the performance of the workflow.

Heavy file access occurs during workflow execution. For each activity, there are input and output parameters and most activities work with documents stored on the file system. XML documents that are used as inputs and outputs of activities in workflows are frequently persisted and retrieved.

Documents Created During Workflow Execution

Documents are used to initiate workflows, or are created intermediately in between workflow execution. These create bottlenecks in performance due to the constant access.

To address the bottlenecks in performance due to the constant access, the following steps are performed:

- An alternate persisting scheme is used - files are stored to the cache instead of the file system. See Caching and Persisting mlXML Documents for Regular Workflows.
- Also, documents are persisted whenever they are created. To provide control over that, settings are provided via the Configurator. See Flags to Control Persistence of Documents.

Caching and Persisting mlXML Documents for Regular Workflows

For regular workflows, documents are persisted to the database instead of the filesystem.

This applies when the **Cache Intermediate MLXML Documents** flag is set to **true.**

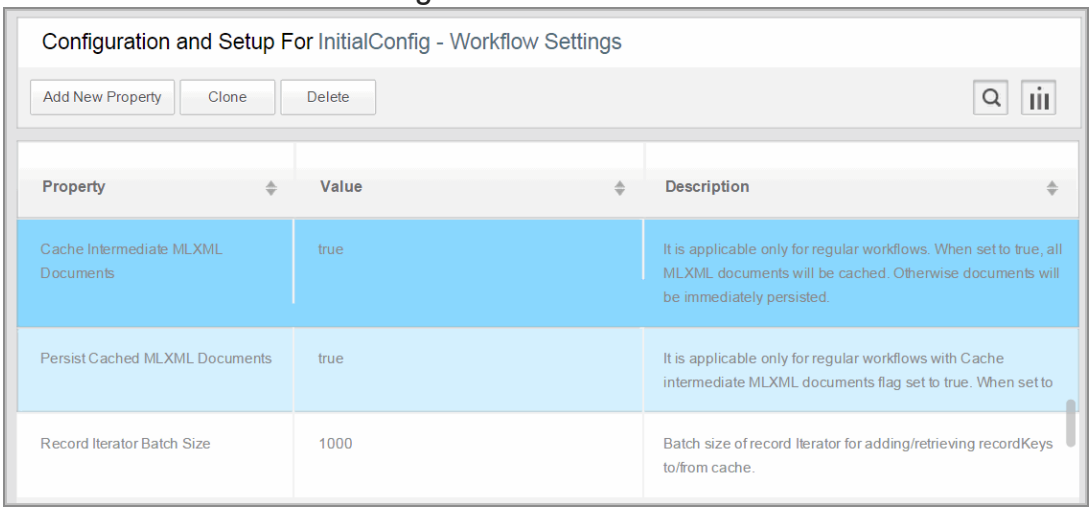
Caching of mlXML documents is done using Fast Infoset provided by Sun - this is a binary representation for xml and an efficient alternative to it. The major advantage is the size, since the size of Fast infoset binary xml documents are smaller than standard xml documents. Secondly, serialization is faster with Fast Infoset. While other compression tools might reduce size, both size and serialization are achieved with Fast Infoset.

Flags to Control Persistence of Documents

There are two flags in the Configurator that can be used to control the persistence of documents (accessible from the **Advanced** view, **Workflow Settings**).

The flags are as follows:

- **Cache Intermediate MLXML Documents**: this flag controls caching of mlXML documents created during execution of regular workflows. The default is false (none of the documents are cached and are immediately persisted). It is a good practice to set this to true for better performance.
- **Persist Cached MLXML Documents to Database**: this flag controls persistence of cached mlXML documents on successful completion of workflows. When set to true, all non-eliminated documents are persisted. When set to false, none of the documents are persisted. This flag is only considered if the **Cache Intermediate MLXML Documents** flag is set to true.



Note: If the inmemory flag is set for a particular workflow, in-memory execution always takes precedence.

When the workflow completes, depending on the flag set, either none of the documents persists or only non-eliminated documents persist. If the workflow suspends or errors out, all documents persist.

Nullification or Elimination of Documents

Intermediate documents created during the regular workflow execution and no longer required can be nullified.

You can eliminate such unnecessary documents from the workflow state and they do not persist on successful workflow completion. You can nullify a document by specifying a non-existing variable to a document variable. By doing this, the document is no longer referred by any state variable, and this removes the document or variable from the workflow state.

In this case, on workflow completion, all cached documents except nullified ones persist, if the persistDocumentsOnDone flag is set. If the workflow suspends or errors out, irrespective of the persistDocumentsOnDone flag, all cached documents persist.

Utility for Migration of Files to Database

You can use utilities for migration of files to the database.

For details, see the "Upgrading TIBCO MDM" chapter in *TIBCO MDM Installation and Configuration* Guide.

Enabling Process State for Large Workflows

The large workflows often create a large process state. Out-of-the-box 8000 character process states are supported.

If the workflow use case generates more than 8000 characters, enable the Enable Huge process state property.

Procedure

1. Add the Enable Huge process state property in the Configurator (**Tools > Add Configuration Value**). Enter the following details in each field:

- Configuration value name: Enable Huge process state
- Internal name: com.tibco.mdm.processstate.hugeoverflow
- Version: 8.3
- Visibility: Advanced
- Description: This property enables huge overflow on PROCESSSTATE.STATEPARAM.
- Current Value: **True**
- Default Value: **False**
- Category: Optimization

2. Run the AlterProcessState.sql script to change the data type of STATEPARAM2 column from the PROCESSSTATE table. The script is located in the $MQ_HOME/db/*databasename*/migration/Migrate82_83 folder

> **Note:**
> - The script might mark indexes on the PROCESSSTATE table unusable. To resolve this problem, you can rebuild the indexes.
> - After the support for the huge process state is enabled, reverting to the out-of-the-box support is difficult and results in the loss of information. Therefore, switching back and forth is not a good practice.
> - Enabling the huge process state support results in storing the huge data in the PROCESSSTATE table. This impacts the workflow throughput.

Workflow Sequencing

Workflow sequencing deals with how it works, why it is required, and the criteria for picking up messages.

Workflow sequencing also deals with workflows that are influenced by sequencing, and configuration of sequencing.

The application allows for execution of multiple workflows in parallel, which might work on overlapping sets of records. As records and hierarchies are modified concurrently, sequencing of changes is required to ensure that changes are applied in order and without data corruption.

Enforcement of sequencing of workflows ensures that there is no loss of data or state due to concurrent modification to common data, if any, shared by the workflows.

How Sequencing Works

Sequencing of workflows is implemented to ensure:

- If there are dependencies between workflows, workflows are run one after the other.
- Sequenced workflows are in an order - in most cases, based on the date/time of the received message, web service or UI operation.

If workflows do not have overlapping data, sequencing does not apply. Workflows which have already completed, suspended waiting for user input, or system events (process is in WAIT status) are not considered and do not influence sequencing.

Sequencing is done before a workflow starts execution and workflows which are either executing or about to execute are checked for dependency. Workflows that have already started execution are not stopped or re-sequenced.

When a dependency between workflows is detected, one of the workflows (survivor) is selected to run. All other dependent workflows are queued up. When the survivor workflow completes, suspends, or its status is changed to indicate that dependent workflows are allowed to run, queued up workflows are evaluated and another workflow

is selected to run. This process continues till all the queued workflows are allowed to run.

Dependency between workflows is detected based on one or more registration keys. Each workflow might have any number of keys. A workflow is considered dependent as long as it has one key common with other workflows.

Workflows Influenced by Sequencing

The following workflows implement sequencing:

Upload, Upload and Import, or Import

Data source uploads initiated by FileWatcher or the UI and data source uploads and import workflows initiated by FileWatcher are dependent on each other and implement sequencing. The dependency is due to data being uploaded in a data source upload.

Import or upload and import workflows are dependent on each other as data is being imported in the same repository.

Record workflows (add, modify, and delete)

Record workflows have dependencies with other record workflows. Record workflows could be started from the UI, web services, JMS (coming in from external sources, such as CIM2CIM synchronization or data pool operations) or spawned.

Configuration of Sequencing

By default, sequencing is enabled. Sequencing of each type of workflow can be disabled through properties in the Configurator under **Optimization switches**.

com.tibco.cim.optimization.import.sequencing – Enables or disables import sequencing

com.tibco.cim.optimization.record.sequencing – Enables or disables record message sequencing

com.tibco.cim.optimization.upload.sequencing – Enables or disables data source upload sequencing

Sequence Determining - Registration Order

To order workflows, a registration order is established. For incoming messages, one of the fields in the message can be designated as registration order.

In such cases, registration order must typically be a date so that chronological order can be maintained. However, if needed, other ordering data can be used (for example, unique sequence numbers).

For all workflows initiated by other channels, the application generates the registration order as a unique, sequenced long number. The application ensures that no two events executing concurrently share the same registration order value.

The event with the earlier registration order is taken up for execution by the workflow engine and other dependent events with a later registration order must wait in the queue till the currently executing event completes or gives up control.

Registration Keys

When a workflow event is created, some of the data associated with the message is identified as its **registration key**.

Registration keys allow the workflow engine to detect any data dependency between events for execution.

You can configure the data used for registration keys for incoming JMS messages. For all other channels and workflows spawned by the application and to pre-configure the registration keys, add the following configuration properties in the ConfigValues.xml file under the Messaging Settings category:

- XPATH_CREATIONDATE: A unique and sequential identifier xpath used for the incoming message.
- com.tibco.XPATH_REGISTRATIONKEY:A list of xpaths used for the registration keys.

The following is a sample ConfigValues.xml entries:

```
<ConfValue name="Registration order XPATH" description="XPATH for unique message identifier used for sequencing" propname="XPATH_CREATIONDATE" sinceVersion="7.0" visibility="All">
<ConfString value="//@externalControlNumber" default="//@externalControlNumber"/>
</ConfValue>
<ConfValue description="Registration keys used for sequencing” listDefault=""
```

```
propname="com.tibco.XPATH_REGISTRATIONKEY" sinceVersion="7.0" visibility="Advanced">
<ConfList>
<ConfListString value="//CatalogItem/ItemData/Attribute[@name='PRODUCTID']/Value/text()" />
<ConfListString value="//CatalogItem/ItemData/Attribute[@name='PRODUCTIDEXT']/Value/text()"
/>
</ConfList>
</ConfValue>
```

After the dependencies are established, the workflow engine uses registration order as another event identifier to sequence multiple events. For some events, such as Record Add notification events or spawned events, the Product IDs of the records form the event registration keys in the event. For events generated by FileWatcher, the input map name, catalog name, and data source name form the registration keys.

The QUEUEENTRY Table

All events and processes that are queued up as a result of sequencing, are stored in the QUEUEENTRY table. The event or process ID is stored in the IDVALUE column.

- For queued events, the queuename is EVENT
- For queued processes, the queuename is PROCESS.

Workflows waiting in this table are retrieved when the running workflow process completes. The event with the lowest registration order is retrieved from the QUEUEENTRY table.

Workflow Troubleshooting

Troubleshooting deals with provision of help in debugging and enhanced error reporting.

Troubleshooting also deals with workflow troubleshooting guidelines and tips.

Trace File

While debugging workflows, whenever a workflow is executed, a trace file is created in the *$MQ_COMMON_DIR*/Temp directory. Using this file, you can gain a good understanding of what happened in the workflow.

This trace file logs the process selection and transition details. It includes details such as:

- All process selections details. For example, which workflow is selected, what were the selection inputs, and so on.
- The workflow definition.
- All transition details. For example, which transition is selected, input and output parameters of the transition, and so on.
- The In and Out states of each activity executed in the workflow.
- All actions taken. For example, abort, timeout, suspend, and so on.
- Important object IDs. For example, event ID, process ID, process log ID, and so on.
- Exceptions, if any.

Disable Tracing

A trace file is generated by default. To disable the generation, use the Configurator option **Workflow Settings** > **Workflow Trace Enabled**.

Download Trace File

The trace file can be downloaded from the event header division of the Event Detail page.

The **Download** link is present only if a trace file is present, as it might not be present for older events and is deleted from the temp folder.

Error Reporting

Enhanced error reporting functionality has been provided. When viewing event details, you can click Error Report to get a detailed dump of all objects associated with the event.

This is very powerful and includes all files generated and most of the data from the workflow state and records.

The error report can be downloaded from the event details itself. The report includes most of the configuration files, workflow definition, all in/out docs for each of the process logs, data for all associated repositories, catalog editions, process, event, product logs, records and so on.

To get a description of the process detail entry, run the following SQL in your database:

```
INSERT INTO DOMAINENTRY (domaintype,value, description) VALUES (
'PROCESSDETAIL','ERRORREPORT','Error Report');
```

Workflow Failovers Handling

Handling workflow failovers deals with backend fault tolerance and failover for the workflow engine.

TIBCO MDM comprises the following components:

- The Application Server (primary system)
- Four external components
    - JMS
    - Database
    - File system
    - Distributed cache

These external components could fail simultaneously in case of network issues in connecting to the application server.

Fault tolerance usually refers to ensuring that users are not affected by system failure. In the context of TIBCO MDM, this means that no data corruption must occur and that workflows must restart from the point they executed last. TIBCO MDM provides a workflow failover mechanism, where in case of a failure, workflows do not terminate but restart at the point of last execution.

The following sections explain the various components that could encounter failure, the reasons for failure, and the failover process.

To directly see the failover configuration, see Failover Configuration.

Application Server Failure

The application server failure occurs due to a fatal crash of the hosting operating system, failed hardware components, or a failure of the java process containing the application context.

In case of an abnormal shutdown (as opposed to a graceful or controlled shutdown) it is not informed of a shutdown request and is not allowed to stop all running processes. In

such cases, the currently running processes are generally not in a consistent state. Java process termination might occur by administrative intervention; on UNIX it is typically using **kill -9** while on Windows it is the **End Process** button on the Task Manager.

Application Server Failover

If the application server terminates abruptly, external systems are not affected due to the use of messaging and once the system is reachable again, any workflows that were running at the time of crashing, restart from the last known state.

In case of a graceful shutdown of the application server, the current activity of the running workflow is completed and the application then shuts down. When the application server is restarted, the workflow resumes processing normally from the point they were suspended at the shutdown time.

Note: In case of the WebSphere application server, it is a good practice to first shut down the application and then shut down the application server (to successfully save the state of currently running processes in the database).

Single Application Server Instance

If the application has a single application server instance, failover is provided - after the application server instance is restarted or is available again, workflows continue normal processing from the point they were suspended at the time of shutdown.

Multiple Application Server instances

If the application is set up with multiple application server instances acting as a cluster, in case of failure of a single application server instance, execution of a workflow proceeds due to the presence of other application server instances.

Note: All synchronous activities are repeated. Asynchronous activities are not repeatable except for the asynchronous Subflow activity.

Messaging System Failure

The messaging system is integral in the workflow system. Any new process is initiated with a ‘workflow’ message.

You can use multiple messaging server instances and set them up in a fault tolerant fashion. If a workflow is suspended and resumed later, the resumption is triggered by a workflow message on the workflow queue. A messaging system failure is deemed to have occurred when the server cannot connect to the JMS to read or write a message from or in the queue.

Between workflow messages, the workflow typically executes on a single thread in a single instance. Asynchronous activities are an exception to this; the activity workload is split into multiple threads.

The supported messaging solution is EMS. It allows a client aware fault tolerant setup. You can configure a messaging cluster with two members (a primary and a backup server). The addresses of both these servers must be part of the configuration. If the primary server fails, fallback to the backup server occurs.

Messaging System Failover

If the messaging server becomes unavailable during workflow processing, the application server automatically attempts to reconnect to the messaging server repeatedly after configured time intervals, for a configurable number of times (the retry period cannot exceed the transaction timeout limit, which is 2 hours by default).

If a reconnection attempt succeeds before the number of retries has passed, the workflows re-execute the same activity.

Saving Messages After the Retry Period

You can save messages by shutting down the application server. Messages are available in the queue for processing after the application server restarts.

Saving messages after the retry period makes message processing more reliable and messages are not lost in case of system errors.

Preconditions for Saving Messages and Graceful Shutdown

Preconditions for saving messages and graceful shutdown are as follows:

- It is only applicable in case of messages already in the messaging server queue for workflow execution. Such messages are not lost and when the messaging server is available again, the application automatically connects to the messaging server and processes the messages.
- If the messaging server becomes unavailable before a message is sent to the queue, message recovery and graceful shutdown is not possible, and the message is lost. In such cases, you can follow the process to recover failed messages. For more information, see the TIBCO MDM System Administrator's Guide.
- If the messaging server becomes unavailable after a message is delivered to the queue but before receiving the message acknowledgement, the message remains in the messaging server. After the messaging server is available again, the message is redelivered. In spite of this, some activities might show an error in the process log.
- If workflow execution is in process and tries to send a message to the messaging server, and if the messaging server is unavailable at that time, the retry logic applies, failing which, graceful shutdown takes place.
- Non-repeatable activities (in workflow execution) are not recovered even though the message is redelivered.

Shutdown Initiation

In case of any resource failure, workflow execution retries for the specified period.

(See Configuring Workflow Retry Attempts). After this period:

- All related workflow receivers are closed and no further messages are taken into the queue.
- Shutdown is initiated in a separate thread; the related JMS session closes, terminates processing, and retains the message(s) in the queue.

At the time of shutdown, any attempts to use application links direct you to a shutdown page.

For details about how to configure graceful application server shutdown, see Configuring Application Server Shutdown.

Database Failure

Database connectivity is vital for the execution of any workflow. The database contains the persistent state of the workflow system.

It contains the state of the process execution and indicates if a process instance is currently executing an activity. The appropriate state is the process state which contains the last fully executed activity. Each restart attempt must execute from that activity onwards.

Typically, a database failure is deemed to have occurred if:

- The server cannot connect to the database to read or write data.
- The Database runs out of resources, that is, connections, memory, or archive log space and tablespace.
- Database connectivity is broken due to network failure.

Database Failover

During workflow processing if the database shuts down, the application server automatically attempts to reconnect to the database repeatedly after configured time intervals, for a configurable number of times.

The conditions for database failover are as follows:

- If the reconnect attempts still fail when the last retry has been executed, the processed workflows are in an inconsistent state. The retry period cannot exceed the transaction timeout limit, which is by default, set to 2 hours. When the application server starts the next time, the workflow is marked as a failed workflow and an error notification email is sent to the user.
- If one of the reconnection attempts succeeds before the number of retries has passed, workflows re-executes the last executed activity at the time of failure.

Shared File System Failure

For best results, use an external file system to store files created by the application, for instance mIXML documents, log files, and so on.

The external file system must be placed onto a fast, reliable disk array and must ideally be on a different machine from the database for better performance.

Conditions such as a full file system and wrong permissions could lead to an effective component failure. The impact of temporary network outages is particularly severe for the external file system. In most cases, a DNS mapping must be performed before the external file systems can be accessed, adding one more component which can fail.

Failure can occur if the Application server cannot access the file system anymore on which the cluster shared information is located. This typically happens during any of the file operations:

- File system existence check
- File Read/Write operations
- Getting the file name from the file system

Shared File System Failover

If the network connection between the application server and the shared file system fails when a workflow is in process, processing waits and retries the current activity.

If the network connection fails while writing data to the file system, the application attempts to reconnect a specified number of times. If the network connection is re-established, workflows resume processing. If the file system is still not available after the specified retries, messages are retained in the queue and the application is shut down. When the filesystem is up and connected to the network and the application is up, this message is redelivered for execution.

If the network connection fails while submitting a work item through the UI, the user interface is down, but when connectivity is restored, the work item is in the same state prior to failure. If a submitted work item must be merged with the previous copy, and if the file system fails during merge, it is repeated based on the configured timeout and retries.

Distributed Cache Failure

Distributed cache failure can occur due to network failure or if the cache servers are down.

- **Network failure**

  This refers to a sustained loss of network connectivity between all cluster nodes. Distributed cache architecture provides a robust mechanism to handle such network failure for nodes leaving and rejoining the cluster. However, in case of a complete network failure, cache consistency cannot be established.

- **Cache servers going down**

  If all the cache servers go down simultaneously or one by one, the application must respond to cache failure in a graceful manner.

Distributed Cache Failover

If a single cache server goes down or gets disconnected from the network, you get an error until the cache server comes back up, and the application does not function as expected.

If a cache server in a fault tolerant cache setup (presuming at least two backup servers) goes down, the failure does not impact the system except for a temporary slowdown. In case of a clustered distributed cache architecture, the following mechanism is provided for:

Wait and Retry

When a “fetch” operation on the cache fails with an exception, a Wait and Retry approach is used to access the cache. If the cache is not available after a predefined number of times, it is deemed a sustained cache failure and the application shuts down.

Backup Policy

Using the distributed cache service, you can configure the number of backups. If any of the cluster nodes fail, as long as you have one or more backups, there is no data loss.

Death Detection

When the death or departure of a cluster member is automatically and quickly detected, failover occurs very quickly and transparently and the responsibilities of the failed cluster member can be assumed by the cluster.

Failover Configuration

You can configure the following fault tolerance properties though the Configurator:

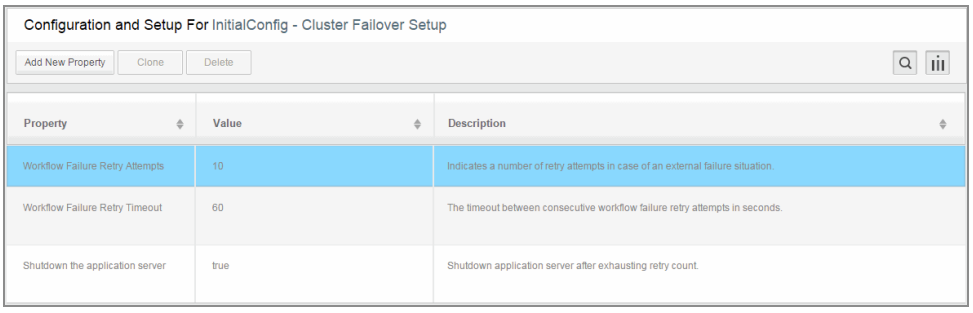
- Configuring Workflow Retry Attempts
- Configuring Application Server Shutdown
- Configuring Error Messages

Configuring Workflow Retry Attempts

In the **Cluster Failover Setup** (**All** Configuration Outline)

- **Workflow Failure Retry Attempts**The number of retry attempts in case of external failure.
- **Workflow Failure Retry Timeout** The timeout (in seconds) between consecutive workflow failure retry attempts.

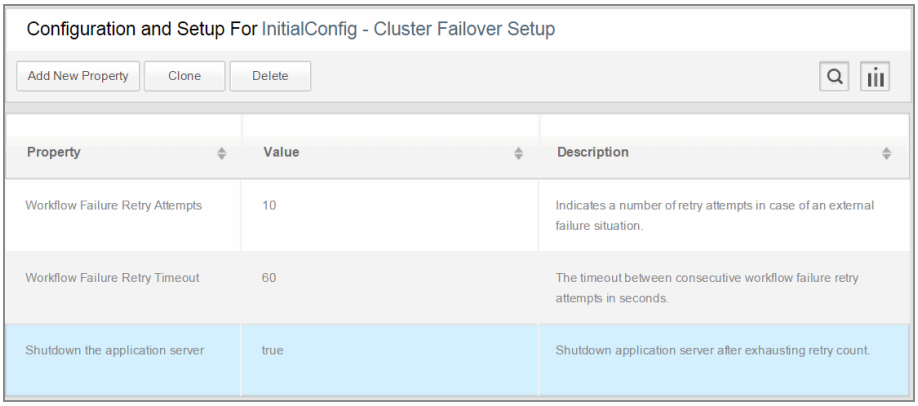
> **Note:** If processing is suspended during a database transaction, a transaction timeout might occur. The total retry time (retry interval * number of retries) must be smaller than the transaction timeout.



Configuring Application Server Shutdown

The **Shutdown the application server (com.tibco.cim.faulttolerant.shutdownAppserver)** property is set to true by default. It enables configuration of automatic shutdown of the Application server.

Through this property, automatic graceful shutdown of the server is initiated on completion of the specified retry count.



Configuring Error Messages

In the **Member Failover Setup:**

- **Transient Database Error Messages**These are a list of database error messages for temporary errors. If any of these errors are encountered, it is presumed that a database error has occurred and the relevant failover process is followed. The messages provided here are vendor specific; the following have been provided for Oracle:
    - Closed Connection
    - The Network Adapter could not establish the connection
    - Listener refused the connection
    - Could not create pool connection
    - Connection reset by peer
    - Illegal Connect
    - ORACLE initialization or shutdown in progress

    - Connection refused
    - Could not create connection
    - immediate shutdown in progress
    - The mandatory key
    - TNS:no listener
- Transient Shared File System Error Messages

  These are shared file system error messages. The messages provided here are OS specific. The following have been provided here:

    - The network path was not found
    - The network location cannot be reached
    - Windows cannot find the network path
    - The system cannot find the path specified
    - Access is denied
- Transient Messaging Server Error Messages

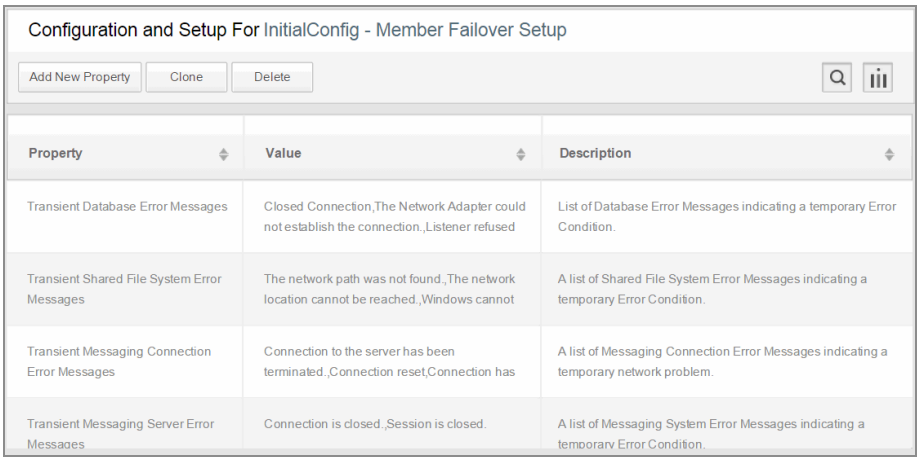
  These are messaging server related exceptions. The messages provided here are messaging system specific. The following have been provided here:

    - Connection is closed
    - Session is closed
- Transient Messaging Connection Error Messages

  These are network connection related exceptions. The following have been provided here:

    - Connection to the server has been terminated
    - Connection reset
    - Connection has been terminated by the server
- Transient Cache Server Error Messages
    - SafeNamedCache was explicitly released
    - Storage is not configured
    - Cache is not active

Note: You can add messages by clicking the **Value** column and entering messages for your specific messaging system, database, and so on.



Failover Log File

To monitor failover situations, a special log file is configured. The default name and location for this file is failover.log located in the main logging directory ($MQ_LOG).

The file is only written to when the system is in wait-retry mode, that is, the system has problems with one of the external components. The file itself can be monitored to find out if the system is in wait-retry mode and needs administrative intervention.

Entries in Failover Log File

The log entries in the failover log file are similar to the following example:

```
2009-12-21 17:20:37,897, [TIBCO EMS Session Dispatcher (121)], DEBUG,
MqProcessInst.execActivityInLoop, - Retrying evaluation for event : 168003 with message : JMS-
8401 [Session is closed] on action : SendProtocolMessage
2009-12-21 17:20:37,897, [TIBCO EMS Session Dispatcher (121)], DEBUG,
MqProcessInst.execActivityInLoop, - Error while executing activity
(PublishToEAI,SendProtocolMessage) below. Sleep and Retry for the 0 th time
2009-12-21 17:20:58,335, [TIBCO EMS Session Dispatcher (121)], DEBUG,
```

```
MqProcessInst.execActivityInLoop, - Retrying evaluation for event : 168003 with message : JMS-
8401 [Session is closed] on action : SendProtocolMessage
2009-12-21 17:20:58,335, [TIBCO EMS Session Dispatcher (121)], DEBUG,
MqProcessInst.execActivityInLoop, - Error while executing activity
(PublishToEAI,SendProtocolMessage) below. Sleep and Retry for the 1 th time
2009-12-21 17:21:18,678, [TIBCO EMS Session Dispatcher (121)], DEBUG,
MqProcessInst.execActivityInLoop, - Retrying evaluation for event : 168003 with message : JMS-
8401 [Session is closed] on action : SendProtocolMessage
2009-12-21 17:21:18,678, [TIBCO EMS Session Dispatcher (121)], DEBUG,
MqProcessInst.execActivityInLoop, - Error while executing activity
(PublishToEAI,SendProtocolMessage) below. Sleep and Retry for the 2 th time
2009-12-21 17:21:38,960, [TIBCO EMS Session Dispatcher (121)], DEBUG,
MqProcessInst.execActivityInLoop, - Retrying evaluation for event : 168003 with message : JMS-
8401 [Session is closed] on action : SendProtocolMessage
2009-12-21 17:21:38,960, [TIBCO EMS Session Dispatcher (121)], DEBUG,
MqProcessInst.execActivityInLoop, - Error while executing activity
(PublishToEAI,SendProtocolMessage) below. Sleep and Retry for the 3 th time
2009-12-21 17:21:59,210, [TIBCO EMS Session Dispatcher (121)], DEBUG,
MqProcessInst.execActivityInLoop, - Retrying evaluation for event : 168003 with message : JMS-
8401 [Session is closed] on action : SendProtocolMessage
2009-12-21 17:21:59,210, [TIBCO EMS Session Dispatcher (121)], DEBUG,
MqProcessInst.execActivityInLoop, - Error while executing activity
(PublishToEAI,SendProtocolMessage) below. Sleep and Retry for the 4 th time
2009-12-21 17:22:19,506, [TIBCO EMS Session Dispatcher (121)], DEBUG,
MqProcessInst.execActivityInLoop, - Retrying evaluation for event : 168003 with message : JMS-
8401 [Session is closed] on action : SendProtocolMessage
2009-12-21 17:22:19,506, [TIBCO EMS Session Dispatcher (121)], DEBUG,
MqProcessInst.execActivityInLoop, - Error while executing activity
(PublishToEAI,SendProtocolMessage) below. Sleep and Retry for the 5 th time
2009-12-21 17:22:39,506, [TIBCO EMS Session Dispatcher (121)], ERROR,
MqProcessInst.execActivityInLoop, - Retry count(5) exhausted for event id:168003 Activity failed :
PublishToEAI - Workflow is set to error or no longer processed
```

The failover log can be configured through **Logging->Failover Log** in each cluster member category in the Configurator. It has the same parameters as all the other logging categories, but very few entries so that the file can be watched and easily understood.

Failover Limitations

All activities support failover and re-execution after restart, with the following exceptions:

The activities that do not support failover are:

- ImportClassificationScheme Activity
- ReclassifyRecord Activity

- PrepareForImportCatalog Activity

  Activities run in the async execution mode are not supported for failover. For a list of activities that can be run in async execution mode, see Workflow Activities. One exception to this is the InitiateSubFlow activity which can be run in both sync and async execution mode and both modes are supported for failover.

  Multiple process selections do not have a fully functional failover mechanism. If failover is required, ensure that the workflow utilizes subflows.

TIBCO Documentation and Support Services

For information about this product, you can read the documentation, contact TIBCO Support, and join TIBCO Community.

How to Access TIBCO Documentation

Documentation for TIBCO products is available on the Product Documentation website, mainly in HTML and PDF formats.

The Product Documentation website is updated frequently and is more current than any other documentation included with the product.

Product-Specific Documentation

Documentation for TIBCO® MDM is available on the TIBCO® MDM Product Documentation page.

How to Contact Support for TIBCO Products

You can contact the Support team in the following ways:

- To access the Support Knowledge Base and getting personalized content about products you are interested in, visit our product Support website.
- To create a Support case, you must have a valid maintenance or support contract with a Cloud Software Group entity. You also need a username and password to log in to the product Support website. If you do not have a username, you can request one by clicking **Register** on the website.

How to Join TIBCO Community

TIBCO Community is the official channel for TIBCO customers, partners, and employee subject matter experts to share and access their collective experience. TIBCO Community offers access to Q&A forums, product wikis, and best practices. It also offers access to extensions, adapters, solution accelerators, and tools that extend and enable

customers to gain full value from TIBCO products. In addition, users can submit and vote on feature requests from within the TIBCO Ideas Portal. For a free registration, go to TIBCO Community.

Legal and Third-Party Notices

SOME CLOUD SOFTWARE GROUP, INC. (“CLOUD SG”) SOFTWARE AND CLOUD SERVICES EMBED, BUNDLE, OR OTHERWISE INCLUDE OTHER SOFTWARE, INCLUDING OTHER CLOUD SG SOFTWARE (COLLECTIVELY, “INCLUDED SOFTWARE”). USE OF INCLUDED SOFTWARE IS SOLELY TO ENABLE THE FUNCTIONALITY (OR PROVIDE LIMITED ADD-ON FUNCTIONALITY) OF THE LICENSED CLOUD SG SOFTWARE AND/OR CLOUD SERVICES. THE INCLUDED SOFTWARE IS NOT LICENSED TO BE USED OR ACCESSED BY ANY OTHER CLOUD SG SOFTWARE AND/OR CLOUD SERVICES OR FOR ANY OTHER PURPOSE.

USE OF CLOUD SG SOFTWARE AND CLOUD SERVICES IS SUBJECT TO THE TERMS AND CONDITIONS OF AN AGREEMENT FOUND IN EITHER A SEPARATELY EXECUTED AGREEMENT, OR, IF THERE IS NO SUCH SEPARATE AGREEMENT, THE CLICKWRAP END USER AGREEMENT WHICH IS DISPLAYED WHEN ACCESSING, DOWNLOADING, OR INSTALLING THE SOFTWARE OR CLOUD SERVICES (AND WHICH IS DUPLICATED IN THE LICENSE FILE) OR IF THERE IS NO SUCH LICENSE AGREEMENT OR CLICKWRAP END USER AGREEMENT, THE LICENSE(S) LOCATED IN THE “LICENSE” FILE(S) OF THE SOFTWARE. USE OF THIS DOCUMENT IS SUBJECT TO THOSE SAME TERMS AND CONDITIONS, AND YOUR USE HEREOF SHALL CONSTITUTE ACCEPTANCE OF AND AN AGREEMENT TO BE BOUND BY THE SAME.

TIBCO, the TIBCO logo, the TIBCO O logo, FTL, eFTL, and Rendezvous are either registered trademarks or trademarks of Cloud Software Group, Inc. in the United States and/or other countries.

All other product and company names and marks mentioned in this document are the property of their respective owners and are mentioned for identification purposes only. You acknowledge that all rights to these third party marks are the exclusive property of their respective owners. Please refer to Cloud SG’s Third Party Trademark Notices (https://www.cloud.com/legal) for more information.

This document includes fonts that are licensed under the SIL Open Font License, Version 1.1, which is available at: https://scripts.sil.org/OFL

Copyright (c) Paul D. Hunt, with Reserved Font Name Source Sans Pro and Source Code Pro.

Cloud SG software may be available on multiple operating systems. However, not all operating system platforms for a specific software version are released at the same time. See the “readme” file for the availability of a specific version of Cloud SG software on a specific operating system platform.

THIS DOCUMENT IS PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS DOCUMENT COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THIS DOCUMENT. CLOUD SG MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S), THE PROGRAM(S), AND/OR THE SERVICES DESCRIBED IN THIS DOCUMENT AT ANY TIME WITHOUT NOTICE.

THE CONTENTS OF THIS DOCUMENT MAY BE MODIFIED AND/OR QUALIFIED, DIRECTLY OR INDIRECTLY, BY OTHER DOCUMENTATION WHICH ACCOMPANIES THIS SOFTWARE, INCLUDING BUT NOT LIMITED TO ANY RELEASE NOTES AND "README" FILES.

This and other products of Cloud SG may be covered by registered patents. For details, please refer to the Virtual Patent Marking document located at https://www.cloud.com/legal.