TIBCO® MDM

Customization Guide

Version 9.3.2
May 2025

Document Updated: September 2025

Contents

Customization of User Interface

You can customize several functions within TIBCO MDM to suit your business environment without changing the application's source code.

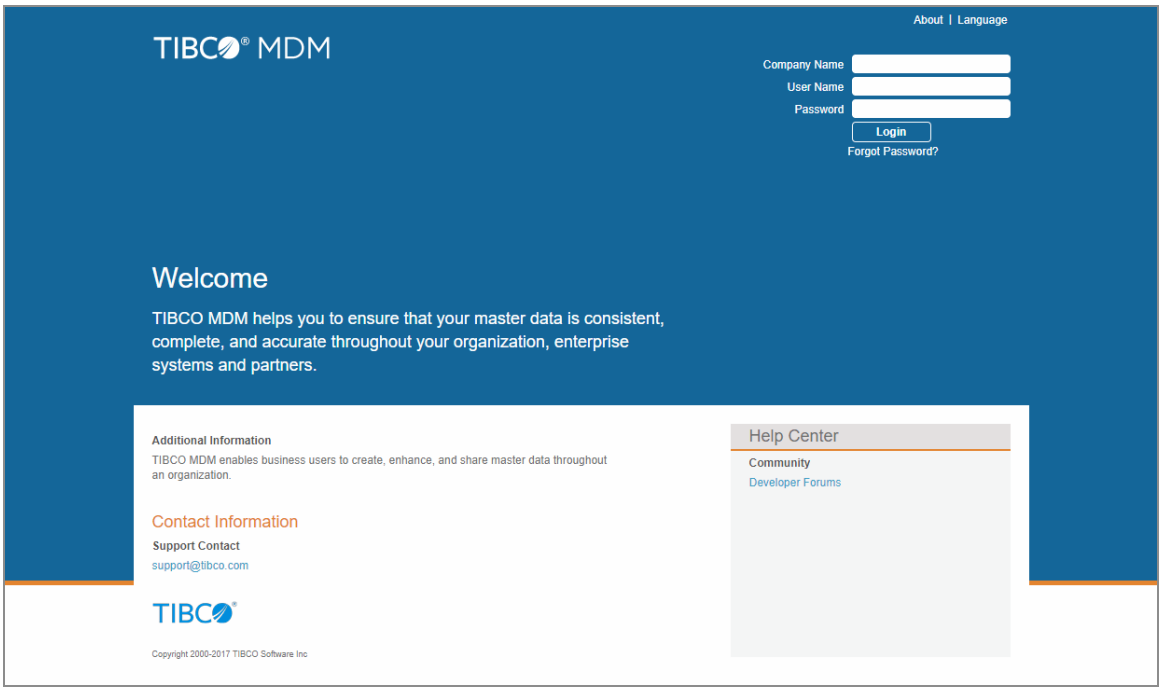
There are different environment customization options available in TIBCO MDM that are meant for advanced users and TIBCO's Professional Services team. Such customization does not need any changes to the application code.

To make the best use of this information, you must be familiar with:

- All basic features of TIBCO MDM
- Basic XML editing
- Basic SQL syntax

Customizing the Login Screen

You can customize images, colors, and text displayed on the Login page. Customization of the Login page is a static change and the changes are persisted within the period of application server run time.



You can customize the Login page using the following two ways:

Create a Custom Template

1. Contact TIBCO Support to get the html template for the Login page.
2. Make changes to the template as per your requirement.
3. Do not remove the IDs of an enterprise, username, and the password fields or the element itself from the template.
4. Compile the html template and generate xmlc class. see the section, Compiling and Deploying the Custom Form.

Use customLogin.html

1. Create the customLogin.html file in the $MQ_COMMON_DIR/htmlprops/ directory.

   Note: If the htmlprops folder is not in $MQ_COMMON_DIR, create it in $MQ_COMMON_DIR.

2. Create the custom Javascript to customize the login page. For example, the following script replaces the support contact text support@tibco.com to support@mycompany.com along with setting default values for the company name,

user name, and password fields.

```
<script>
jQuery("#enterpriseNameId").val("MYCOMPANY");
jQuery("#loginname").val("admin");
jQuery("#passwd").val("admin");
jQuery("#loginContactInfoSubContent").text("support@mycompany.com");
jQuery("#enterpriseNameId").focus();
</script>
```

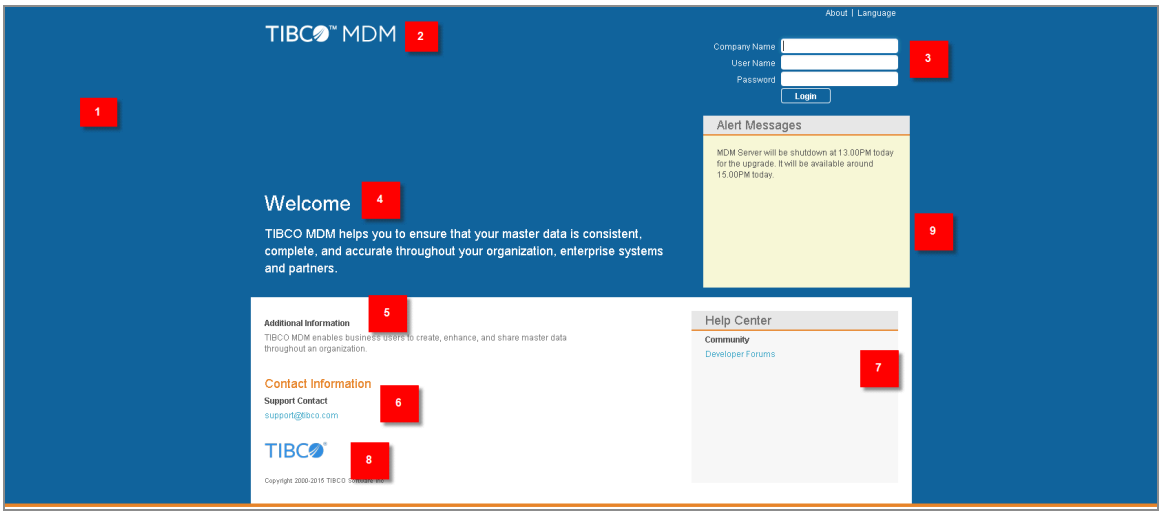
3. Save the file.

Note: You can customize the Login Screen extensively by using the customLogin.html file along with the use of external services.

Login Page Parts

The TIBCO MDM Login page consists of nine parts.

This is shown in the following figure:



1 - Main Page

This is a DIV HTML tag for which the css style class '.login-page' is defined in the styles.css file. Its background color can be modified. See Customizing the Login Page Background to customize the background color.

2 - Header Logo

This is the DIV HTML tag for which the image file is defined in '.login-page .header' css class in the styles.css file. See Customizing the Header Logo to customize the header logo.

3 - Login Information Box

See Customizing the Login Information Box to customize the text within the login information box.

4 - Welcome Message Box

See Customizing the Welcome Message Box to customize the welcome text.

5 - Additional Information Box

See Customizing the Additional Information Box to customize the additional information text.

6 - Contact Information Box

See Customizing the Contact Information Box to customize the contact information text.

7 - Help Center Box

See Customizing the Help Center Box to customize the help center text.

8 - Company Logo and Copyright Box

See Customizing the Company Logo and Copyright to customize the company logo and copyright information.

9 - Alert Messages Box

The alerts message box is, typically maintained by the system administrator to display any critical messages regarding the application to the end users. See Customizing the Alert Message Box to customize the alert message box.

Title Bar Display

The Title Bar of the application displays the following:

Application Name: *Screen Description - Screen ID*:*Application Provider*

For example, **MDM:Login-900:TIBCO Software Inc.**

where:

- *Application Name* refers to MDM
- *Screen Description* refers to Login
- *Screen ID* refers to 900
- *Application Provider* refers to TIBCO Software Inc.

See Customizing the Title Bar to customize the title bar display.

Customizing the Title Bar

Customize the title bar on the login information page.

Procedure

1. Start Configurator.
2. Go to **InitialConfig > Advanced > Miscellaneous**
3. Change the value for Application Name and Application Provider properties.

   However, the Application Name and Application Provider properties are optional. If you do not specify any of these properties, the corresponding section of the title bar is omitted.

   - If Application Provider is not specified, the title bar displays:

     Application name: Screen description - Screen ID
   - If Application Name is not specified, the title bar displays:

     Screen description - Screen ID: *Application Provider*

Customizing the Login Page Background

Customize the login page background.

- Modify the background color by editing the styles.css file in ECM.ear > EML.war > css > default location.

Customizing the Header Logo

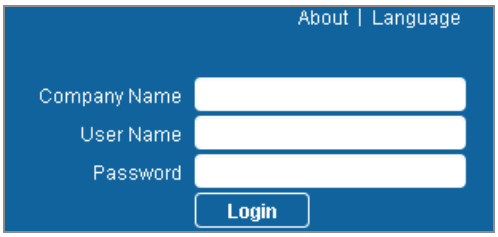
Customize the header logo.

Procedure

1. Create an image of the size 191 x 31 pixels
2. Package the image within the images folder. (ECM.ear > EML.war > images folder).
3. Modify the background property of the css class '.login-page .header' in the case the image file name is different from the default image name 'Logo_MDM_White_H.png'.

Customizing the Login Information Box

Customize the text within the login information box.



Procedure

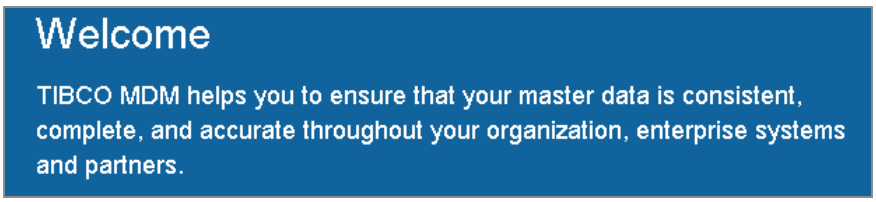
1. Extract the following resource keys from htmlresources.properties resource bundle into the Customhtmlresources.properties resource bundle:

- com.tibco.mdm.ui.directory.security.login.index.aboutLbl
- com.tibco.mdm.ui.directory.security.login.index.langLbl
- com.tibco.mdm.ui.directory.security.login.index.enterpriseName
- com.tibco.mdm.ui.directory.security.login.index.userName
- com.tibco.mdm.ui.directory.security.login.index.passwordName

2. To customize the text 'Login,' extract the 'Login' key from jsresources.properties resource bundle into the Customjsresources.properties resource bundle.

Customizing the Welcome Message Box

Customize the Welcome text on the login information page.

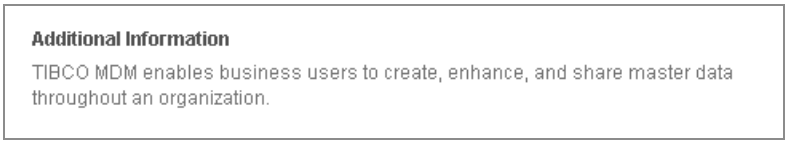


Procedure

1. Extract the following resource keys from the htmlresources.properties resource bundle into the Customhtmlresources.properties resource bundle:
   - com.tibco.mdm.ui.directory.security.login.index.loginWelcomeHeader
   - com.tibco.mdm.ui.directory.security.login.index.loginWelcomeContents

Customizing the Additional Information Box

Customize the Additional Information text on the login information page.



Procedure

1. Extract the following resource keys from the htmlresources.properties resource

bundle into the Customhtmlresources.properties resource bundle:

- com.tibco.mdm.ui.directory.security.login.index.additionalInfoHead
- com.tibco.mdm.ui.directory.security.login.index.additionalInfo

Customizing the Contact Information Box

Customize the Contact Information text on the login information page.

Procedure

1. Extract the following resource keys from the htmlresources.properties resource bundle into the Customhtmlresources.properties resource bundle:
   - com.tibco.mdm.ui.directory.security.login.index.loginContactInfoHeader
   - com.tibco.mdm.ui.directory.security.login.index.loginContactInfoSubHeader

Customizing the Help Center Box

Customize the Help Center text on the login information page.

Procedure

1. Extract the following resource keys from the htmlresources.properties resource bundle into the Customhtmlresources.properties resource bundle:
   - com.tibco.mdm.ui.directory.security.login.index.loginCommunityHeaderTxt
   - com.tibco.mdm.ui.directory.security.login.index.commContentHeader2
   - com.tibco.mdm.ui.directory.security.login.index.commContent2

Customizing the Company Logo and Copyright

Customize the company logo and copyright information on the login information page.

Procedure

1. For the company logo, modify the TIBCO_Logo.png file in ECM.ear > EML.war/images folder.

   The default size of the image is 91 x 24 pixels.

2. Repackage it within EML.war.
3. For copyright text, extract the com.tibco.mdm.ui.directory.security.login.index.loginCopyright resource key from the htmlresources.properties into the Customhtmlresources.properties.
4. Modify the resource key.

Customizing the Alert Message Box

The alert message box is enabled or displayed only when the system administrator puts any messages in the $MQ_COMMON_DIR/htmlprops/alerts.txt file. It contains the text enclosed within HTML tags.

Procedure

1. Create the alerts.txt file in the $MQ_COMMON_DIR/htmlprops folder.

   Note: If the htmlprops folder is not in $MQ_COMMON_DIR, create it in $MQ_COMMON_DIR.

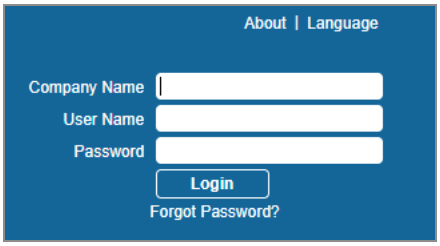
2. Save the file in the UTF-8 encoding format to have any encoded characters.
3. To customize the header text, extract the following resource key from the htmlresources.properties resource bundle into the Customhtmlresources.properties resource bundle:

com.tibco.mdm.ui.directory.security.login.index.loginAlertHeaderTxt

Customizing Forgot Password

Customize the forgot or reset password text and relevant fields.

For information about the configuration properties of password, see Configuration Properties for Password.



Procedure

1. Extract the following resource keys from htmlresources.properties resource bundle into the Customhtmlresources.properties resource bundle:

   ```
   com.tibco.mdm.ui.directory.member.ResetPassword.resetHeader=Reset Password
   com.tibco.mdm.ui.directory.member.ResetPassword.enterprise=Company Name
   com.tibco.mdm.ui.directory.member.ResetPassword.username=User Name
   com.tibco.mdm.ui.directory.member.ResetPassword.newPasswordText=New Password
   com.tibco.mdm.ui.directory.member.ResetPassword.confPasswordText=Confirm Password
   com.tibco.mdm.ui.directory.member.ResetPassword.resetBtn=Reset Password
   ```

2. To customize the text 'Forgot Password,' extract the 'Reset Password' key from jsresources.properties resource bundle into the Customjsresources.properties resource bundle.

Customization of Landing Page

The landing page is the first page which is displayed after logging in. By default, the Inbox is the landing page. After a session expires and you log in to the application again, the landing page changes.

Using a urlmap.prop file, the landing page is mapped to a user role and is externalized. It is similar to the rolemap.prop map file used in single sign on. Using a text editor, you can edit the mapping and specify the URL of the landing page.

The mapping of roles to the landing page is enterprise specific so that different mappings can be used for multi-tenancy. The urlmap.prop file is stored in an enterprise directory similar to the rolemap.prop file.

The urlmap.prop file is searched in the enterprise specific directory, followed by /standard directory. However, if the urlmap.prop file is not found in the specified directory, Inbox is selected as the landing page.

The sample urlmap.prop file.

```
rolename=URL
SUPERUSER=EnterpriseList?action=list&menulink=Enterprise Profiles
Repository\Manager=CatalogList?menulink=Master Catalogs
Admin=MemberList?menulink=User Accounts
Admin=RoleList
```

To customize the landing page:

- Specify the complete URL in rolename=URL

  The URL must be similar to the URL specified in allmenu.xml file. Do not specify the menulink parameter if the URLs do not correspond to the menus. The role name refers to the role as defined in the ROLE table. The role name appears in various UI pages. However, as the role names are also translated using the resource bundle, avoid using the translated role name in the mapping.
- If the role name contains a blank space, specify the URL which maps to an MDM pages in Repository\ Manager=URL.

If more than one role is assigned, each role can have different landing pages. The order of the list in the urlmap.prop file determines the landing page. For example, if you have Administrator and RepositoryManager roles and the order of the entries in the urlmap.prop file are as follows:

```
Administrator=url1
RepositoryManager=url2
```

URL1 is selected as the landing page.

```
RepositoryManager=url2
Administrator=url1
```

URL2 is selected as the landing page.

Enabling Hot Deployment

Enable Hot Deployment in the Configurator.

Procedure

1. Set the property com.tibco.cim.customize.firstpage to **true** in the Configurator.
   When hot deployment is initiated, the URL map is cleared and reloads the page.

Redirection of an Application Link to Custom Page

The following screens can be customized in TIBCO MDM:

- Record Add
- Record View
- Record Modify
- Inbox (work item list)
- Work Item detail

> **Note:** These changes are applied to all repositories in an enterprise.

Hidden Fields on UI

The following hidden fields available on each UI page. These fields can be passed to custom applications launched from the application.

Hidden Fields on UI

Hidden Field	ID	Name
Enterprise ID	ENTERPRISE_ID	enterpriseid
Enterprise Name	ENTERPRISE_NAME	enterprisename
User Name/ Login Name	ENTERPRISE_USER_NAME	enterpriseusername
User ID	ENTERPRISE_USER_ID	enterpriseuserid

Merging Custom Application

Merge custom TIBCO General Interface or GWT application to ECM.ear:

Procedure

1. Create a temporary directory, newECM.

```
md c:\newECM
```

2. Change directory to newECM.

```
cd c:\newECM
```

3. Extract %MQ_HOME%\ECM.ear:EML.war.

```
%JAVA_HOME%\bin\jar -xvf %MQ_HOME%\ECM.ear EML.war
```

4. Create a directory newEML.war.

```
md newEML.war
```

5. Change directory to newEML.war.

```
cd newEML.war
```

6. Extract ..\EML.war.

```
%JAVA_HOME%\bin\jar -xvf ..\EML.war
```

7. Change directory for GI or GWT
   - For GI: cd JSXAPPS
   - For GWT: cd GWTAPPS
8. Create and copy the GI or GWT project as follows:
   a. Create your GI or GWT project directory.
   - For GI: md CIMTestGIProject
   - For GWT: md CIMTestGWTProject
   a. Copy your CIMTestGIProject or CIMTestGWTProject project content folders manually to C:\CIMTestGIProject or C:\CIMTestGWTProject. Ensure the name of your project is the same as the folder name in the C drive.
   b. Copy your GI or GWT project.
   - For GI: xcopy /E /Y C:\CIMTestGIProject CIMTestGIProject
   - For GWT: xcopy /E /Y C:\CIMTestGWTProject CIMTestGWTProject
9. Change directory to newEML.war.

```
cd ..
```

10. Create a new EML.war.

```
%JAVA_HOME%\bin\jar -cvfM ..\EML.war *
```

11. Change directory to newECM.

```
cd\newECM
```

12. Copy %MQ_HOME%\ECM.ear file.

```
copy %MQ_HOME%\ECM.ear
```

13. Update new EML.war into ECM.ear.

```
%JAVA_HOME%\bin\jar -uvf ECM.ear EML.war
```

14. Set the relevant properties using the Configurator (**Advanced > UI Customization**). For example, for redirecting to custom Add Record from a custom GI-based Add Page, enter the value box next to the **Add Record External URL** property and press **Enter**:
    - For GI: GIClient?jsxapppath=JSXAPPS/CIMTestGIProject
    - For GWT: GWTClient?action=addRecord
15. Deploy new ECM.ear which is in folder newECM into the application server.

    Note: The default GI version is 3.7.1.

16. Login to the application by accessing the URL.
    - Navigate to the page which you have redirected to a custom page. For example, if you have redirected the **Add Record** page to a custom page, navigate to the **Add Record** page. The custom GI page must be displayed.

Parameters for Customizable Pages

You must append a different set of parameters to the specified custom page URL depending on the URL that is being customized. If no values are set in these properties, the application uses default pages.

The following table lists the parameters for each custom page.

Redirecting an Application Link to Custom Pages

Customizable Page	Configurator Option	Parameters	Description
Add Record	UI Customization > Add Record External URL	mastercatalogname	Name of the repository to which the record is being added.
Modify Record	UI	mastercatalogname	Name of the repository to which the record is being

Customizable Page	Configurator Option	Parameters	Description
	Customization > Modify Record External URL		added.
		productid	PRODUCTID attribute of the record.
		productidext	PRODUCTIDEXT attribute of the record (this can be null).
		modversion	The version of the record being modified. This parameter is optional. If this parameter is not specified, the latest version of the record is attempted for modification.
View Record	UI Customization > View Record External URL	mastercatalogname	Name of the repository to which the record is being added.
		productid	PRODUCTID attribute of the record.
		productidext	PRODUCTIDEXT attribute of the record. This can be null.
		modversion	The version of the record being modified. This parameter is optional.

Customizable Page	Configurator Option	Parameters	Description
			If this parameter is not specified, the latest version of record is attempted for view.
Work Item List (Inbox)	UI Customization > Work List External URL	username	Username (login ID) of the user whose work item list is needed. This can be only specified when the requesting user has WorkSupervisor permissions. By default, it retrieves a worklist for the user in the current login profile.
		organizationname	The company name to which the user belongs (same as specified in login page).
Work Item Detail	UI Customization > Work Item External URL	work itemid	A reference to the work item.
		username	Username (login ID) of the user whose worklist is needed.
		organizationname	The organization to which the user belongs.

Using Multiple Versions of General Interface

This release of the application ships with General Interface (GI) version 3.6.1. If you have custom pages created using another version of GI, complete these steps.

Note: You cannot overwrite version 3.6.1. The InputMap and OutputMap functionality runs on version 3.6.1. If version 3.6.1 is overwritten, the InputMap and OutputMap functionality might not work as expected.

Procedure

1. Create a new directory in any location. For example, C:\NewGI.
2. Copy your GI project to the newly created directory including JSXAPPS folder. For example, copy the $MQ_HOME/common/standard/samples/JSXAPPS folder to the NewGI directory.
3. Copy the required GI libraries, GI_HOME/JSX folder, to the newly created directory. For example, copy C:\tibco\gi\3.4\JSX to the C:\NewGI folder.
4. Expand the ECM.ear using the jar command as follows.
    a. Create a temporary directory called newECM and copy ECM.ear to this directory.
    b. Execute the following command to expand $MQ_HOME/ECM.ear into newECM/ECM.ear folder:
    c. $JAVA_HOME/bin/jar -xvf $MQ_HOME/ECM.ear
5. Expand EML.war using the jar command as follows.
    a. Create a temporary directory newEML and expand ECM.ear/EML.war to this directory.
    b. Execute the following command to expand EML.war into newECM/newEML/EML.war.

        $JAVA_HOME/bin/jar -xvf ../ECM.ear/EML.war

        cd EML.war.

        You must see two folders JSXAPPS and JSX. The JSXAPPS folder contains the application files of General Interface (GI) and the JSX folder contains the GI libraries.
6. Copy the newly created directory under EML.war.
7. Copy the custom GI project to the JSXAPPS folder under the EML.war folder.
8. Use the Jar command and re-create the EML.war and the ECM.ear as follows.
    a. From the EML.war folder, execute the following command to re-create

EML.war file in ECM.ear folder.

```
$JAVA_HOME/bin/jar -cvf ../../ECM.ear/EML.war ./*.*
```

b. Change to the ECM.ear folder:

cd ../../ECM.ear

c. From the ECM.ear folder, execute the following command to re-create the ECM.ear file:

```
$JAVA_HOME/bin/jar -cvf ../ECM.ear ./*.*
```

9. Verify that the NewGI/JSX and NewGI/JSXAPPS are in EAR (ECM.earEML.warNewGIJSX and ECM.earEML.warNewGIJSXAPPS).
10. Set the relevant properties using the Configurator (**Advanced > UI Customization**). Change the URL accordingly for JSX path. For example:

```
GIClient?jsxapppath=NewGI/JSXAPPS/CIMTestGIProject&jsxsrc=NewGI/JSX/js/JSX30.js
```

> **Note:** Path provided for jsxsrc is case sensitive. If jsxsrc is not specified in the URL, the default JSX libraries are used.

11. Redeploy and restart the application server.
12. Login to the application.
13. Navigate to the page which you have redirected to a custom page. For example, if you have redirected the **Add Record** page to a custom page, navigate to the **Add Record** page. The custom GI page must be displayed.

> **Note:** To verify the GI version used by the application, press Alt + Ctrl + Shift + J after loading the GI application. A popup with the details of the GI version is displayed.

Redirecting Application Link to Custom Page in Google Web Toolkit

Google Web Toolkit (GWT) allows you to develop a web-based application in Java. Using Google Web Toolkit, you can enhance the custom page as per your requirement. It also reduces the loading time required for general interface.

Procedure

1. Merge your web application war into EML.war inside ECM.ear. For more information, see Merging Custom Application.
   - If you use any of the TIBCO MDM classes such as IMqSessionProfile, you might have to build your application or servlet against TIBCO MDM classes.
   - If you use the session ID of the currently logged in TIBCO MDM user, for example, while requesting a web service, then the following code snippet is not required. You can get the session ID from the HTTP session.

     ```
     String sessionId = request.getSession(false).getId();
     ```

2. Extract information such as username, enterprisename, catalogname, and so on specific to the functionality, which you want to customize. Some of this information might be passed to the URL as request parameters; while the rest can be extracted from MqSessionProfile stored in the HTTP session.

   ```
   // Get User profile object stored in Http session
   IMqSessionProfile userProfile = (IMqSessionProfile)request.getSession().
           getAttribute(IMqSessionProfile.SESSION_PROFILE);
   //Get informtion like username, enterprisename, sessionid
   String entepriseName = userProfile.getAttr("ENTERPRISE_NAME");
   String userName = userProfile.getAttr("USER_NAME");
   ```

3. Add the relevant configuration for the new servlet into the web.xml in ECM.ear\EML.war\WEB-INF.

   ```
   <servlet id="Servlet_319">
       <servlet-name>GWTClient</servlet-name>
       <servlet-class>com.tibco.mdm.ui.infrastructure.CimToGwtServlet</servlet-class>
   </servlet>
   ```

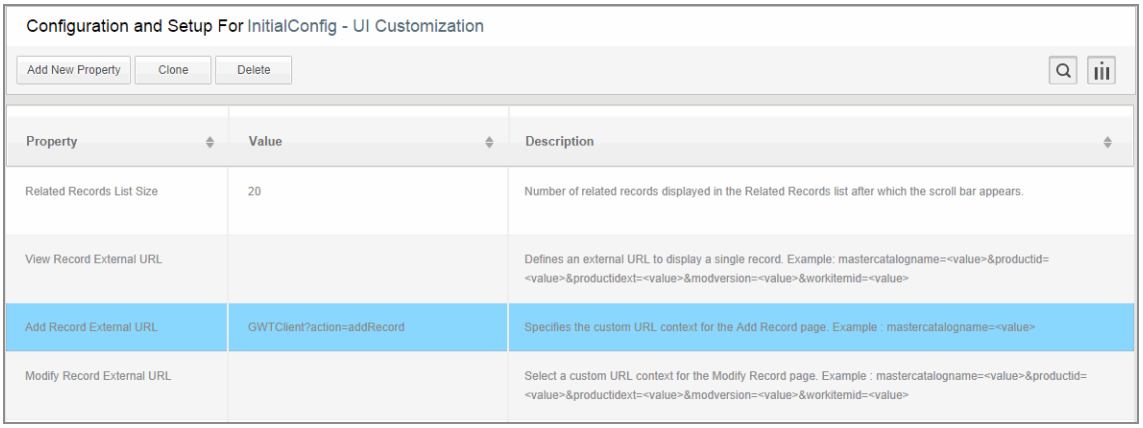
4. Update new EML.war into ECM.ear using the following parameter:

   ```
   %JAVA_HOME%\bin\jar -uvf ECM.ear EML.war
   ```

5. Set the relevant properties using the Configurator (**Advanced > UI Customization**).

   For example, for redirecting to a custom Add Record from a custom GWT based

Add Page, enter GWTClient?action=addRecord next to the Add Record External URL property and press **Enter**. You must append a different set of parameters to the specified custom page URL depending on the URL that is being customized. For more information, see Redirection from a Custom Page to the Application.



If no values are set in these properties, the application uses default pages.

6. Deploy the updated ECM.ear into the application server.
7. Log in to the application by accessing the URL.
8. Navigate to the page which you have redirected to a custom page. For example, if you have redirected the Add Record page to a custom page, navigate to the Add Record page. The custom GWT page must be displayed.

Redirection from a Custom Page to the Application

The following application screens can be hyperlinked from any other application or custom screens:

- Record Add
- Record View
- Record Modify
- Record History
- Record Compare
- Browse and Search

Redirection from a Custom Page to the Application

Page	URL	Parameter	Description	Required?
Record Add	http://host:port/eml/AddProduct?htmlaction=add& mastercatalogname=value& init=true&hiddenWFCheck=default&initrecordbundle=true& retURL=BrowseRepository ?&hiddenCriteriaPaneShown=true& hiddenViewAllClicked=true& currentPageNum=1& pageSize=50&go ToPageNum=1&pageRequest=0& Direction=NEXT&popup=y **Note:** • Specify a value for the mastercatalogname attribute. • For retURL attribute, specify the value of the custom page URL that you want to navigate back.	context	Must be in format:loginname=a%26enterprisename=test_ enterprise%26passwd=a%26correlationID1=1234. The values to substitute are: • **loginname** = username to access the application • **passwd**= password to access the application • **enterprisename**= company name	Yes
		mastercatalogname	Name of the repository to which the record is being added.	Yes
Record View	http://host:port/eml/GWTRecordUI?mastercatalogname=value& prodid=value& idext=value& productID=value& productExt=value&initrecordbundle=true& htmlaction=view&init=true&	context	Must be in format:loginname=a%26enterprisename=test_ enterprise%26passwd=a%26correlationID1=1234. The values to substitute are: • **loginname** = username to access the application • **passwd**= password to access the application	Yes

Page	URL	Parameter	Description	Required?
	hiddenWFCheck=Y&hasOpenWI=N& can_edit_in_wf=Y& canedit=Y&popup=y&Direction=NEXT& retURL=URL of the customized page **Note:** Specify value for the following attributes: • mastercatalogname • prodid • idext • productID • productExt • retURL: For retURL attribute, specify the value of the custom page URL that you want to navigate back.		• **enterprisename**= company name	
		mastercatalogname	Name of the repository to which the record is being added.	Yes
		productID	PRODUCTID attribute of the record.	Yes
		productExt	PRODUCTIDEXT attribute of the record.	Can be null
Record Modify	http://host:port/eml/GWTRecordUI? mastercatalogname=value& prodid=value& idext=value&productID=value& productExt=value&initrecordbundle=true& htmlaction=modify&init=true& hiddenWFCheck=Y&hasOpenWI=N& can_edit_in_wf=Y&canedit=Y& popup=y&Direction=NEXT& retURL=URL of the customized page **Note**: Specify value for followings attributes: • mastercatalogname • prodid • idext • productID • productExt	context	Must be in the following format:loginname=loginname=a%26enterprisename=test_enterprise%26passwd=a%26correlationID1=1234. The values to be substituted are: • **loginname** = username to access the application • **passwd**= password to access the application • **enterprisename**= company name	Yes

Page	URL	Parameter	Description	Required?
	• retURL: For retURL attribute, specify the value of the custom page URL that you want to navigate back.			
		mastercatalogname	Name of the repository to which the record is being added.	Yes
		productID	PRODUCTID attribute of the record.	Yes
		productExt	PRODUCTIDEXT attribute of the record.	Can be null
Workitem List	http://host:port/eml/Inbox?&menulink=Inbox	context	Must be in the following format:loginname=a%26enterprisename=test_enterprise%26passwd=a%26correlationID1=1234. The values to substitute are: • **loginname** = username to access the application • **passwd**= password to access the application • **enterprisename**= company name	Yes
Record Compare	http://host:port/eml/RecordCompare?&mastercatalogname=value&closeOnOK=true&htmlaction=showRecordCompare&Direction=NEXT&retURL=BrowseRepository	context	Must be in format:loginname=a%26passwd=a%26enterprisename=test_enterprise&externalURL=true&popup=y The values to substitute are: • **loginname** = username to access the application • **passwd**= password to access the application • **enterprisename**= company name	Yes
		mastercatalogname	Name of the repository from which records are being compared.	Yes
		recordcount	Number of records to be compared. Valid values are 2 and 3.	Yes
		productid<i>	The PRODUCTID attribute of the record.	Yes
		productidext<i>	The PRODUCTIDEXT attribute of the record.	Yes
		productkeyid<i>	The productkeyid value of the record. Either pass productkeyid or productid/productidext.	No
		modversion<i>	The version of the records being compared. If not passed, the latest version is used.	No

Page	URL	Parameter	Description	Required?
Browse and Search	http://host:port/eml/BrowseProduct?menulink=Browse&repoName=value **Note:** Specify value for the repoName attribute.	repoName	The 'repoName' refers to the repository to be defaulted for browse. This URL can be invoked from any application. The application might display a login error, if required. After logging on to the application, repository name is defaulted. The repository name is not sticky, it is used for this invocation only. If an invalid repository name is specified, the following error message is displayed: CAT-1030: Repository or output map with name or ID Account1111 does not exist; might have been already deleted by another user.	No

Redirecting from a Custom Page to the Application Page

Procedure

1. Type the custom page URL in the Address bar. For example, http://host:port/MDMToGwt.
2. Type the application page URL in the field. For example, Record Add URL. For a list of URLs, see Redirection from a Custom Page to the Application.

3. Click the **Send** button to open the application page. The application page is displayed.

> **Note:** You can create these buttons as per your requirement while developing the custom page.

Customizing GUI Colors

Colors of different sections of TIBCO MDM GUI as well as the font type, weight, and text alignment on different UI labels can be customized.

This is achieved by modifying the CSS style sheet file styleforwidgets6.css located in ECM.ear/EML.war/css. Changes made to the CSS modify the GUI for all users on that particular instance, irrespective of the enterprise.

This is achieved by making a copy of ECM.ear/EML.war/css/styleforwidgets6.css and placing it in the $MQ_COMMON_DIR/htmlprops/css directory. Modify the required CSS style class. To better understand different CSS styles, see the section Style Sheet for GUI.

A custom CSS style sheet can also be placed by creating a custom.css file and placing it in the $MQ_COMMON_DIR/<enterprisename>/htmlprops/css directory.

Procedure

1. Open {MQ_HOME}\ECM.ear\EML.war and extract the styleforwidgets6.css file.
2. Edit the CSS file for any GUI-related changes required and save the file. You can edit the CSS file in Notepad or use any CSS editing software available.

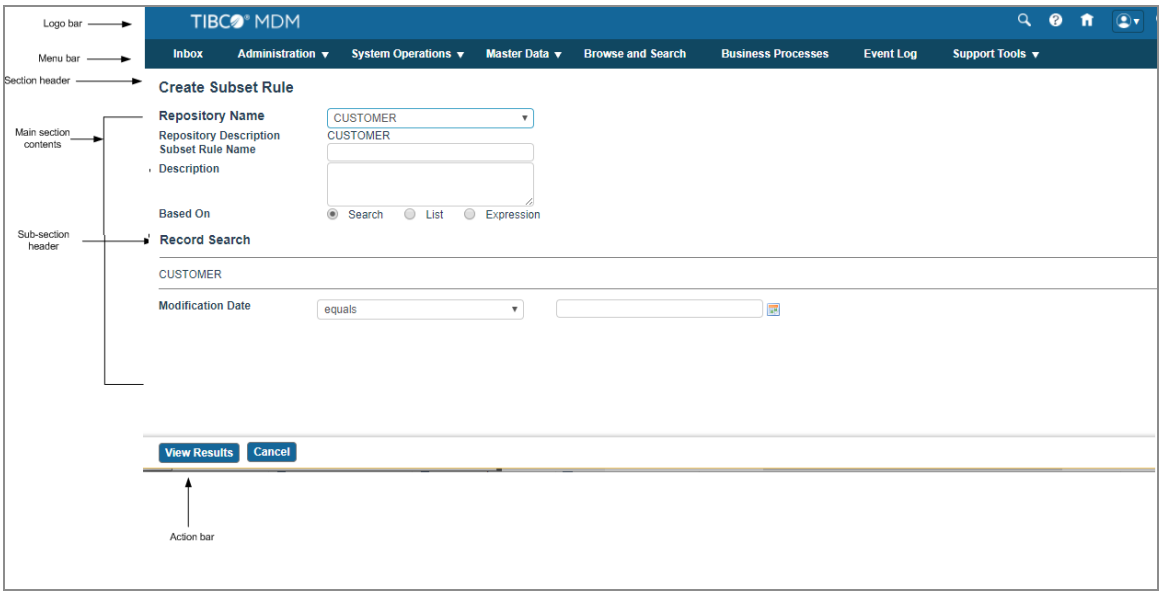
   To modify a particular part of the GUI, edit the relevant class in the CSS file. For a list of classes that must be modified for different parts of the UI, see Style Sheet for GUI.
3. Replace the updated CSS file in the same {MQ_HOME}\ECM.ear\EML.war\.
4. Deploy the updated ECM.ear on the application server.
5. Restart the application.

Style Sheet for GUI

The basic terminology used for GUI of TIBCO MDM and the CSS classes used for them are described in the following sections:

- Logo Bar
- Navigation Bar
- Contents Frame
- Page or Section Header
- Page or Section Contents
- Sub Section
- Submit Section or Action Bar
- Name Value Pair Contents
- Grid Table Contents
- Display Normal Messages

- Display Important Notes or Messages
- Display Error Messages
- Control Image Buttons
- Control Button Links
- General CSS Classes
- Attribute Details Edit Form
- Multi-value Select
- Miscellaneous CSS Classes



Logo Bar

This contains the company logo, Login Profile information, About and Logout action.

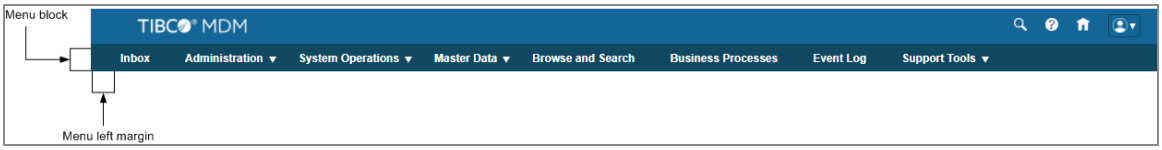
To control the visual style of contents within the legal bar, following CSS style definitions are used.

CSS Classes to Control Visual Style if Contents within Legal Bar

CSS Class Name	HTML elements on which it is applied	Properties defined
logoFrame	TABLE	Border=0, width=100%, height=38px, font-size=10px, font-weight=bold
A.logoA:link/active/visited	A (Anchor)	Color= #CCEEFF
TD	TD	font-family= geneva,arial,helvetica, font-size= 11px

Navigation Bar

The navigation bar is a container that has navigation items.

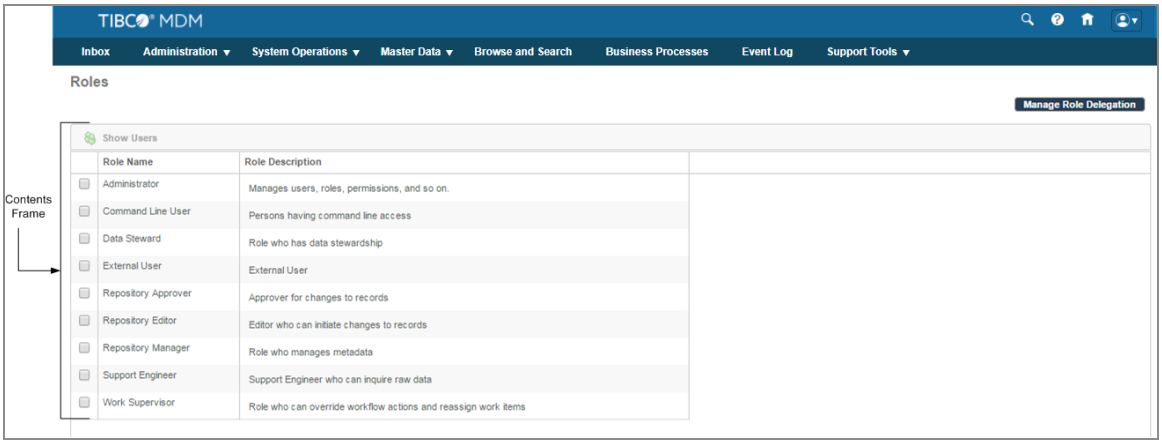


CSS Classes Used to Control the Elements within the Menu Bar

CSS Class Name	HTML elements on which it is applied	Properties defined
menuFrame	TABLE	Border=0, width=100%, height=22px, padding-top=0px
menuBlock	TD	Background=#F9DB6D, width: 7px, height=22px
menuLeftMargin	TD	padding-left=20px
clBar, clLevel0, clLevel1, clLevel2	TD	background-color = #396570, layer-background-color=#396570

Contents Frame

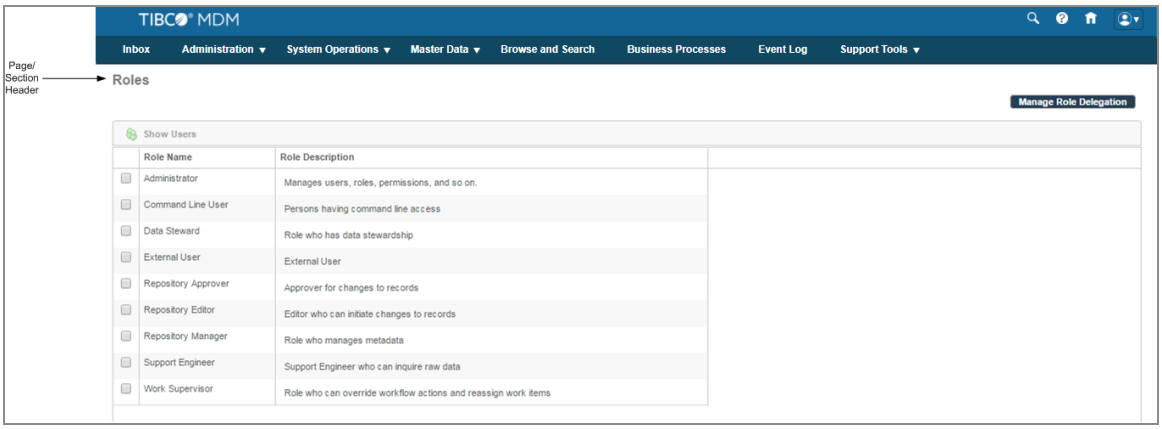
The contents frame contains the actual page contents for the selected menu item. All the contents are displayed within the contentFrame table.



CSS Classes Used for the Table

CSS Class Name	HTML elements on which it is applied	Properties defined
contentFrame	TABLE	Border=0, width=100%, height=82%, padding-top/left/right=3px

Page or Section Header

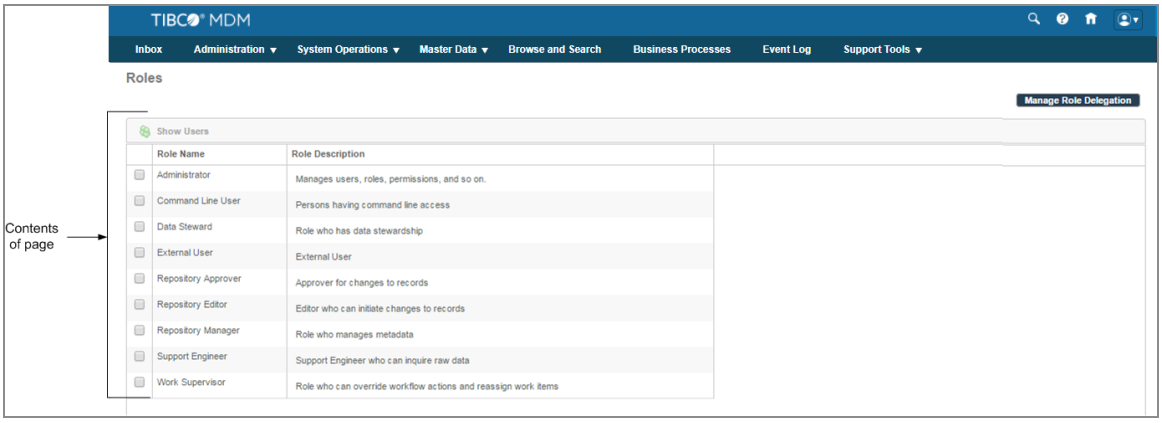


CSS Classes to Control the Page or Section Header

CSS Class Name	HTML elements on which it is applied	Properties defined
headerTable	TABLE	Border=0, width=100%, padding-top=20px
cHeader	TD	Font-size=16px, font-weight=bold, color=#000000, background-color=#ffffff, width=100%, text-align=left
headerHR	HR	border-top: 0px gray; background-color: gray; height: 1px;display: none;

Page or Section Contents

All the contents of the page will be defined within the DIV and TABLE element.

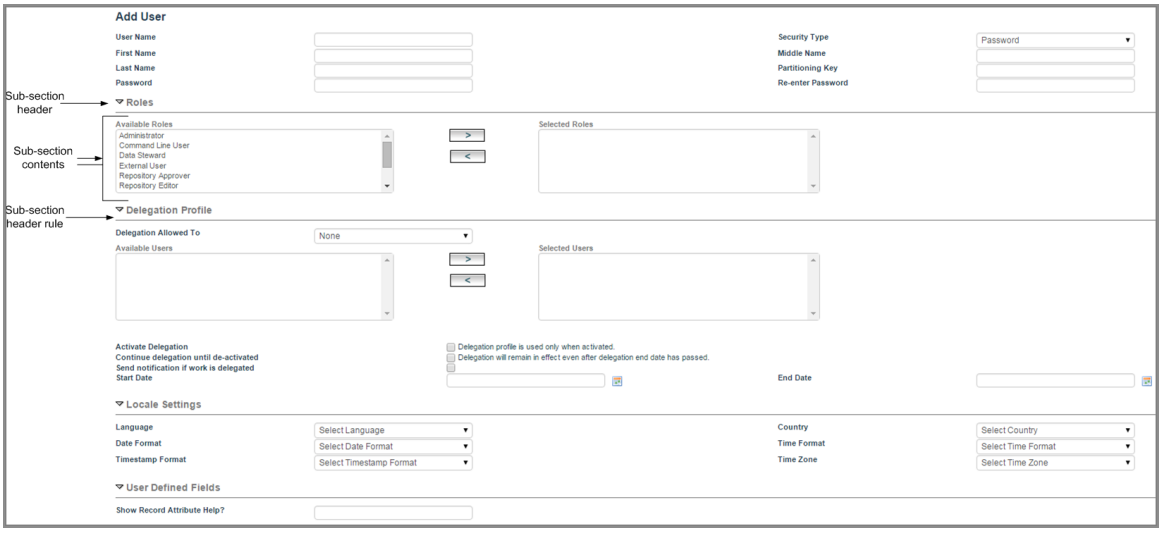


All the contents of the page will be defined with in DIV and TABLE element.

CSS Classes to Control These Elements

CSS Class Name	HTML elements on which it is applied	Properties defined
headerContentsDiv	DIV	padding-bottom: 0px; padding-left: 20px; width: 99%; float: left; position: relative;
headerContentsTable	TABLE	width: 100%; border: 0;

Sub Section

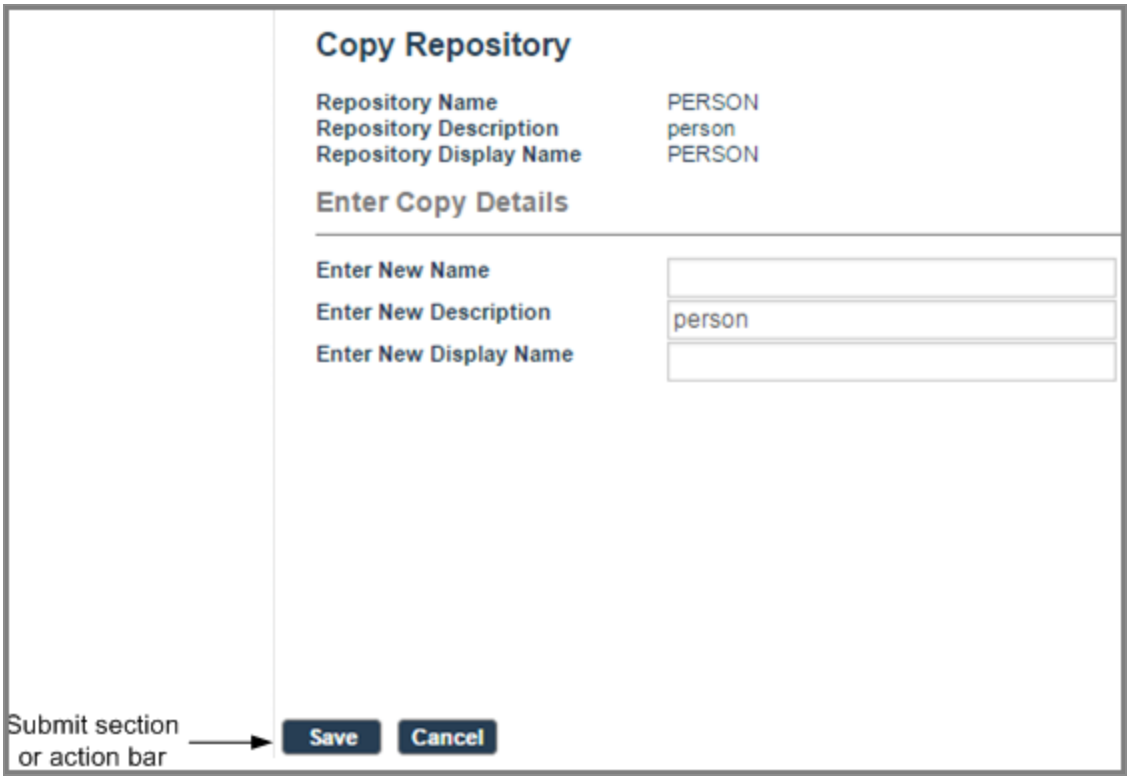


CSS Classes Used for Sub-Sections

CSS Class Name	HTML elements on which is applied	Properties defined
subheaderContentsDiv	DIV	padding-bottom: 5px; width: 99%; float: left; position: relative;

CSS Class Name	HTML elements on which is applied	Properties defined
subheaderTable	TABLE	width: 100%;border: 0;
subheaderHR	HR	border-top: 0px gray; background-color: gray; height: 1px; display: visible;
cSubheader	TD	font-size: 14px; text-align: left; font-weight: bold; color: #000000; background-color: #ffffff width: 100%
subheaderContentsTable	TABLE	width: 100%; border: 0;

Submit Section or Action Bar



CSS Classes Used for Subsection or Action Bar

CSS Class Name	HTML elements on which it is applied	Properties defined
submitContainerDiv	DIV	padding-top: 20px; padding-left:16px; width: 100%; float: left; position: relative;
submitTable	TABLE	width: 100%;border: 0;
submitHR	HR	border-top: 0px gray; background-color: gray; height: 1px; display: none;
submitBtns	TD	font-weight : bold; background-color : #ffffff; color : #116699; text-align : left; vertical-align: middle; height : 20; width : 100%

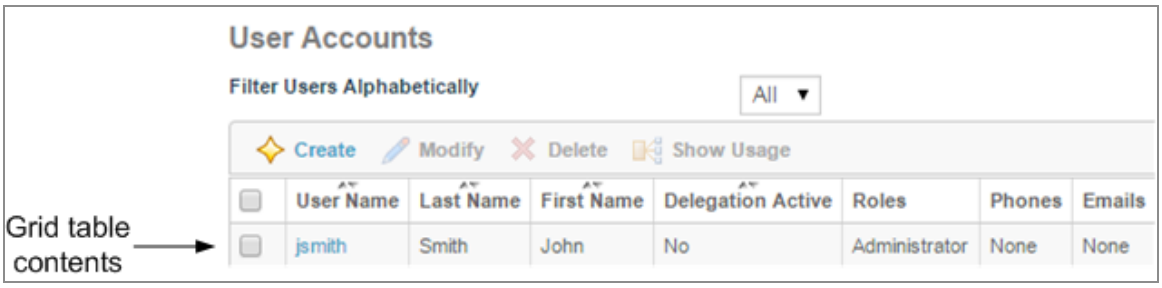
Name Value Pair Contents

CSS Classes Used for Name Value Pair

CSS Class Name	HTML elements on which it is applied	Properties defined
cRightTDA	TD	font-weight : bold; color : #000000; text-align : right; vertical-align: top; padding-top: 1px;
cRightTD	TD	font-weight : normal; color : #000000; text-align : right; vertical-align: top; padding-top: 1px;
cLeftTDA	TD	font-weight : bold; color : #000000; text-align : left; padding-right : 5; vertical-align: top; padding-top: 1px;

CSS Class Name	HTML elements on which it is applied	Properties defined
cLeftTD	TD	font-weight : normal; color : #000000; text-align : left; padding-right : 5; vertical-align: top; padding-top: 1px
cCenterTDA	TD	font-weight : bold; color : #000000; text-align : center; vertical-align: top; padding-top: 1px;
cCenterTD	TD	font-weight : normal; color : #000000; text-align : center; vertical-align: middle; padding-top: 1px;
cCenterTD	TD	font-weight : normal; color : #000000; text-align : center; vertical-align: middle; padding-top: 1px;
cCenterTD	TD	font-weight : normal; color : #000000; text-align : center; vertical-align: middle; padding-top: 1px;

Grid Table Contents



CSS Classes Used to Control Grid Table

CSS Class Name	HTML elements on which it is applied	Properties defined
contentsTableWithGrid	TABLE	width: 100%; border-width: 1px; border-color: #555555; border-style: solid; border-collapse: collapse; padding-left: 5px;
table.contentsTableWithGrid TR	TR	height:23px;
TD.gridTD, TD.msgGridTD	TD	text-align : left; font-weight : normal; color : #000000; border-width: 1px; border-color: #EAEAEA; border-style: outset; padding-left: 3px;
TD.gridTDb	TD	text-align : left; font-weight : normal; color : #000000; border-width: 1px; border-color: #EAEAEA;

CSS Class Name	HTML elements on which it is applied	Properties defined
		border-style: outset; padding-left: 3px;
contentsTableWithGridHeader	TD	text-align : left; font-weight : bold; color : #000000; background-color : #CCCCCC; border-width: 1px; border-color: #E4E4E4; border-style: outset; padding-left: 3px; background-image: url (../images/tableHeader2rowsBkgrd.jpg)
cContentsTableWithGridHeader	TD	text-align : center; font-weight : bold; color : #000000; background-color : #CCCCCC; border-width: 1px; border-color: #E4E4E4; border-style: outset; padding-left: 3px; background-image: url

CSS Class Name	HTML elements on which it is applied	Properties defined
		(../images/tableHeader2rowsBkgrd.jpg)
contentsTableWithGridHeaderCursor	TD	text-align : left; font-weight : bold; color : #000000; background-color : #CCCCCC; border-width: 1px; border-color: #E4E4E4; border-style: outset; padding-left: 3px; background-image: url (../images/tableHeader2rowsBkgrd.jpg); cursor: pointer

Display Normal Messages

CSS Classes Used for Displaying Normal Messages

CSS Class Name	HTML elements on which it is applied	Properties defined
msgLeftTD	TD	font-family : geneva,arial,helvetica; font-size : 11px; font-weight : normal; color : #000000; text-align : left; vertical-align: top; padding-top: 1px;
msgRightTD	TD	font-family : geneva,arial,helvetica; font-size : 11px; font-weight : normal; color : #000000; text-align : Right; vertical-align: top; padding-top: 1px;
resultRightTD	TD	font-family : geneva,arial,helvetica; font-size : 11px; font-weight : normal; color : #000000; text-align : Right; vertical-align: top;

CSS Class Name	HTML elements on which it is applied	Properties defined
		padding-top: 1px;
resultLeftTD	TD	font-family : geneva,arial,helvetica; font-size : 11px; font-weight : normal; color : #000000; text-align : Left; vertical-align: top; padding-top: 1px;

Display Important Notes or Messages

CSS Classes for Displaying Important Notes or Messages

CSS Class Name	HTML elements on which it is applied	Properties defined
highlightFontLeft	TD	font-family : geneva,arial,helvetica; font-size : 11px;

CSS Class Name	HTML elements on which it is applied	Properties defined
		font-weight : bold; color : #116699; text-align : left; vertical-align: middle; height : 20; width : 100%;
normalFont	TD	font-family : geneva,arial,helvetica; font-size : 11px; font-weight : normal; color : #116699; text-align : center; vertical-align: middle; height : 20; width : 100%;
highlightFont	TD	font-family : geneva,arial,helvetica; font-size : 11px; font-weight : bold; color : #116699; text-align : center; vertical-align: middle; height : 20; width : 100%;

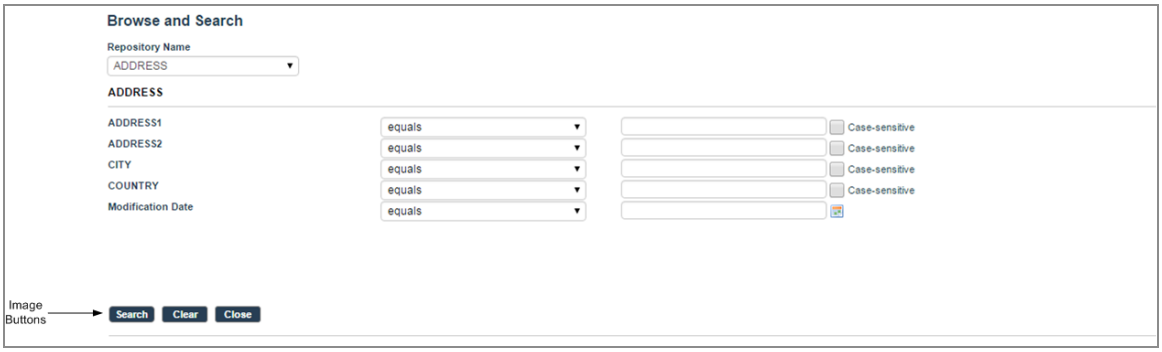
CSS Class Name	HTML elements on which it is applied	Properties defined
auditFont	TD	font-family : geneva,arial,helvetica; font-size : 11px; font-weight : normal; color : #0000FF; text-align : left;

Display Error Messages

CSS Classes for Displaying Error Message or Text

CSS Class Name	HTML elements on which it is applied	Properties defined
errorTD	TD	font-family : geneva,arial,helvetica; font-size : 11px; font-weight : normal; color : red; text-align : left; vertical-align: top;

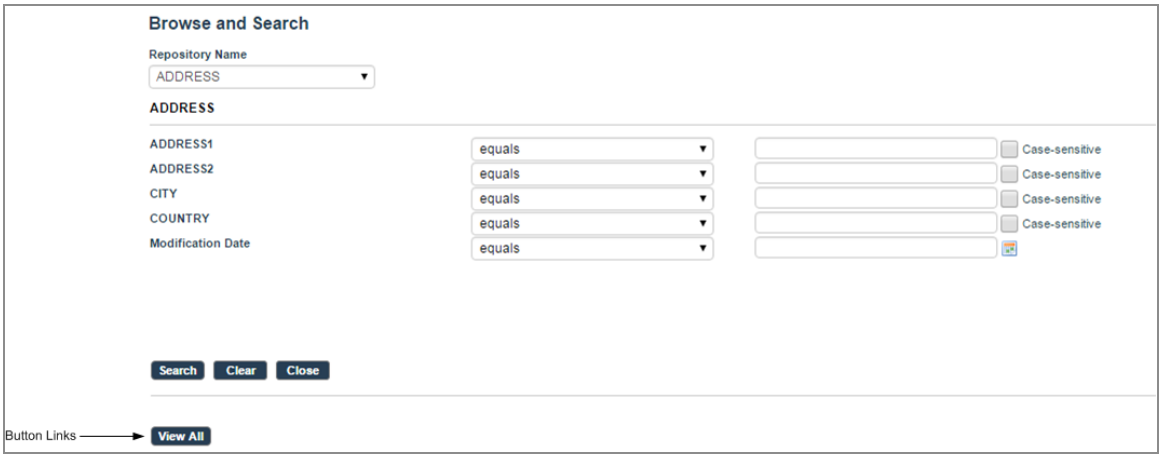
Control Image Buttons



CSS Classes to Control Image Buttons

CSS Class Name	HTML elements on which it is applied	Properties defined
cBtns	TD	font-weight : bold; color : #116699; text-align : center; vertical-align: middle; height : 20; width : 100%; padding-top: 1px;

Control Button Links



CSS Classes to Control Button Links

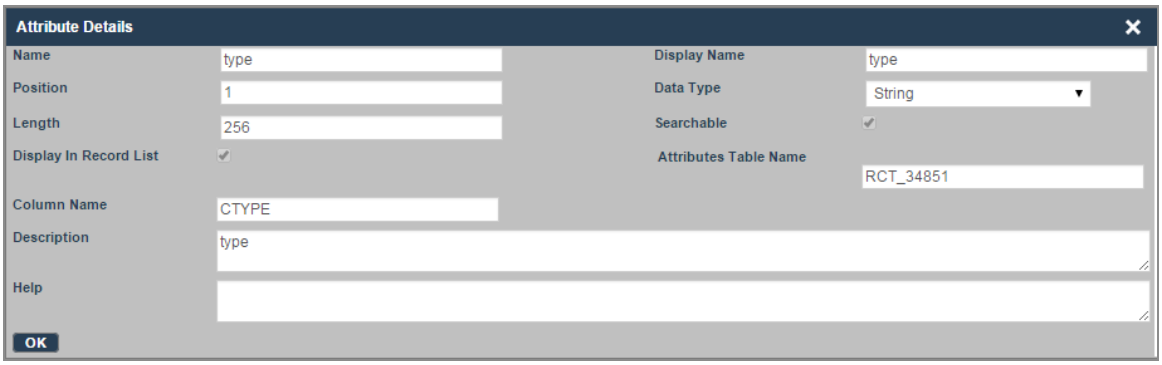
CSS Class Name	HTML elements on which it is applied	Properties defined
A:link, A:visited	A	color : #396099;
A:active	A	color : #396099;
A: hover	A	color : #396099;
cBtnLink	TD	font-weight : normal; text-align : right;
btnLinkLeft	TD	font-weight : normal; text-align : left;

General CSS Classes

Some general styles are:

CSS Class Name	HTML elements on which it is applied	Properties defined
Body, A, Select, Option, Textarea, Input, TD, Font	Body, A, Select, Option, Textarea, Input, TD, Font	font-family : geneva,arial,helvetica; font-size : 11px
Body	Body	margin-top : 0; margin-left : 0; margin-right : 0;
defaultTable	TABLE	border: 0; width: 100%;
inputCls	INPUT	width: 225px;
selectCls	SELECT	width: 225px;

Attribute Details Edit Form

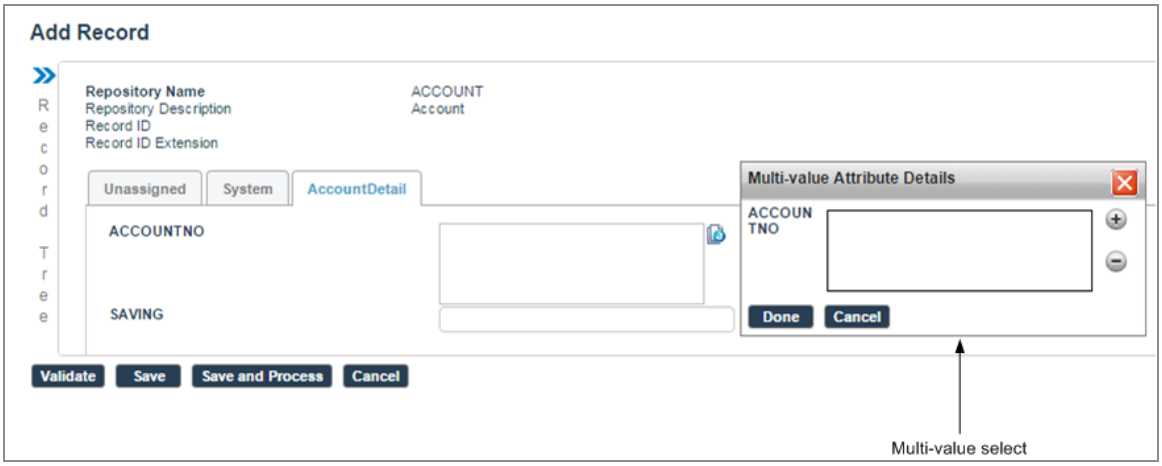


CSS Classes Used for Attributes Details Edit Form

CSS Class Name	HTML elements on which it is applied	Properties defined
popupTableBgColor	TABLE	background-color : #C0C0C0;
editFormTitleTD	TD	font-size : 12px; text-align : left; font-weight : bold; color : #ffffff; background-color : #396099; width : 100%; cursor: move; vertical-align: middle;
cLeftTD_popup	TD	font-weight : normal; color : #111111; background-color : #C0C0C0; text-align : left; padding-right : 5; vertical-align: top; padding-top: 1px;
cLeftTDA_popup	TD	font-weight : normal; color : #111111; background-color :

CSS Class Name	HTML elements on which it is applied	Properties defined
		#C0C0C0; text-align : left; padding-right : 5; vertical-align: top; padding-top: 1px; background-color : #C0C0C0; text-align : left; padding-right : 5; vertical-align: top; padding-top: 1px;

Multi-value Select



CSS Classes to Control Multi-Value Select

CSS Class Name	HTML elements on which it is applied	Properties defined
mvpdselectiondiv	DIV	width: 100%; height: 50%; overflow: auto;
mvpdselectionlist	TABLE	background-color: #CCCCCC; border: 0; width: 100%; height: 100%;
mvpdHeader	TD	font-family : geneva,arial,helvetica; font-size : 8pt; font-weight : bold; color : #ffffff; background-color : #116699
mvpdTD	TD	font-family : geneva,arial,helvetica; font-size : 8pt; font-weight : normal; color : #000000; text-align : left; padding-right : 3
mvpdselectiontable	SPAN	visibility : hidden;

CSS Class Name	HTML elements on which it is applied	Properties defined
		height : 180; overflow : auto

Miscellaneous CSS Classes

CSS Class Name	HTML elements on which it is applied	Properties defined
cWarningFont	FONT	color: red;
cErrorFont	FONT	color: red;
cListPopupFont	FONT	color: #FFFFFF;
calenderCls	IMG	height: 21px; vertical-align: top; border: 0;
#linkhovermenu	TR	position: absolute; background-color: #BDE0EA; line-height: 18px; z-index: 100; visibility: hidden; font-family : arial,comic sans ms,technical; color : #FFFFFF; background-color : #396570; text-align:center;

CSS Class Name	HTML elements on which it is applied	Properties defined
#linkhovermenu a	A	text-decoration: none; text-align:center; font-family : arial,comic sans ms,technical; font-size : 9pt; font-weight : normal; color : #FFFFFF; display: block; width:100%;
#linkhovermenu a:hover	A	background-color: #DCE2ED; color: #396570;

Subclasses in styleforwidgets6.css

To Change	Assign Value To	Possible Values	Comments
Font type	font-family	Font name and font family (serif, sans-serif, cursive, fantasy, monospace).	When you specify a list of fonts, each value is attempted in turn, from left to right, until one is found that the system can display. That is, the first font in the list is used by default. If it is not available on the user's computer, the next one is used and so on. If none of the listed fonts are available, a default font for the listed family is used.
Font size	font-size	Font size in pt.	Pt.= points. The points used by CSS2 are equal to 1/72th of an inch.

To Change	Assign Value To	Possible Values	Comments
Font weight	font-weight	Normal or **Bold**.	
Font color	color	Name or hexadecimal value of the required color.	To select the correct value for the required color, you might have to download one of many free color picker programs from the web.
Background color	background-color	Name or hexadecimal value of the required color.	To select the correct value for the required color, you might have to download one of many free color picker programs from the web.
Text alignment	text-align	Right, Left, or Centre.	
Minimum space between right aligned text and right border of cell	padding-right	Size in pt.	
visibility	visibility	Yes or No.	

For more images that can be customized via the resource bundles, see the next section.

Note: The users of the system might have to remove cached content from their computer to get the new look and feel.

To preserve customization during migration to future releases, ensure you back up these images before applying the new version of the product.

Customizing Logos and Images

Using the resource bundles you can customize the logos and images displayed on the user interface.

For example, the TIBCO MDM GUI displays:

- Company logo at the top left of the main UI page (tibcologo2.gif)
- Legal bar at the bottom of the user interface (TibcoLegalbar.gif)

You can replace these images with images of your company's logo. You can also have a separate set of images for each enterprise.

Procedure

1. Extract and create customized resource bundle file, that is, SharedStringResources.properties. Refer to Localize Text Strings.
2. Edit the customized enterprise specific resource bundle, that is, SharedStringResources.properties as required according to the following table. For example, change neutralized_Logo=VeloselLogowBorder52.gif to neutralized_Logo=<name of the new logo image file> like neutralized_Logo=MyCompany.gif.

   SharedStringResources.properties

To customize this image	Set this property in SharedStringResources.properties
Company Logo	neutralized_logo
Legal bar at the bottom of the page	neutralized_productversion

3. To change any logos or images:
   a. create a new directory called "images" under $MQ_COMMON_DIR/<<Enterprise name>>/.
   b. Copy the desired logo image file and other images (For example, MyCompany.gif) in this images directory.
4. Save the files.
5. Restart the application server.

Note:

- The TIBCO MDM login page does not display the custom images. This procedure only changes the images on all the GUI pages except the login page.
- The logo image size must be 1600 X 67.

Customizing Logo Bar

Procedure

1. Create a header.html in $MQ_COMMON_DIR/<enterprisename>/htmlprops.
2. Check whether the contents of header.html are similar to the following.

```
<TABLE class="logoFrame" id="logoFrame" summary="" cellpadding="0" cellspacing="0">
<tr>
<TD background="images/tibcologo2.gif" valign="top" id="loginfo" align="right" nowrap width="100%">
<TABLE cellpadding="0" cellspacing="0" class="logoFrame">
<TR><TD> </TD></TR>
<TR>
<TD align="right">
<font color="#ffffff" size="2" >
<span> Logged in as <A  class="logoA" alt="" href="MemberProfileView?action=myinfo&menulink=My Account Profile" ><SPAN id="whois"></SPAN></A>  </span>
```

```
</font>
</TD>
</TR>
<TR>
<TD align="right">
<font color="#ffffff" size="2" >
<span>
<A class="logoA" alt="About" href="javascript:popupAboutPage('help/About.html');">About</A> | 
<A class="logoA" alt="Logout" href="javascript:doLogOut();">Logout-test</A>  
</span>
</font><!-- End Top Information -->
</TD>
</TR>
<TR><TD> </TD></TR>
</TABLE>
</TD>
```

```
</tr>
</TABLE>
```

3. Ensure you have enclosed the contents within the table.

Workitem Forms and Email Templates Customization

The work item forms and email templates customization includes:

- How Forms Work
- XML Form Definition
- Write a Custom Work Item Form
- Compiling and Deploying the Custom Form
- Gathering Inputs from a User Using the Work Item Form
- Add Repeating Elements in the Form
- Use Reserved XMLC IDs
- Email Template Customization
- Email Subject Line Configuration
- HTML Labels Using Resource Bundles Customization

How Forms Work

TIBCO MDM forms work using three basic components:

- A HTML template which is the work item form.
- An XML document: This is the source of data to be shown in the HTML form.
- An XML form: This is the XML file that maps data from the source XML document

to a Workitem or Email HTML template using an XPATH.

HTML form is used to write a custom work item. The HTML form is used as input to the XMLC tool. XMLC generates Java code and automatically compiles it to generate class files (these files are referred as XMLC components). Each HTML ID in HTML form is converted to Java variable and can be accessed using XMLC APIs. Application uses XMLC APIs to get HTML IDs, set values, do HTML manipulation and finally generate HTML, which is ready to be displayed as a work item.

XML Form Definition

An XML form is used by the application to map data from source XML to the work item display. When the application displays a work item form or an Email template, it uses an XML form to get an HTML ID, and populates it with the value pointed to by the XPATH.

For example an entry in the XML form looks like:

```
<Field>
<ID>HeaderFirstShipDate</ID>
<InputFrom>/Message/Body/Document/BusinessDocument/CatalogAction/C
atalogActionHeader/Date/Code/Normal[text()='FirstShipDate']/../../
DateValue/Value/text()</InputFrom>
<OutputTo>(//Form/Field/ID[text()='HeaderFirstShipDate'])/followin
g-sibling::Value</OutputTo>
<Value />
</Field>
```

The XML form consists of fields defining specific data to be shown for an ID. Each field contains an 'ID' and an 'InputFrom' tag. The form can, optionally, have 'OutputTo' and 'Value' tags.

- ID: This tag maps to the ID in the HTML template.
- InputFrom: This contains the XPATH from which the data is extracted and is shown at the given location identified by ID.
- OutputTo: This contains an XPATH to which user input is to be written. In most cases, it points to the 'Value' node in the XML form.
- Value: This tag is a placeholder for any user input that must be written out to the XML form.

A form definition looks like:

```
<?xml version="1.0" encoding="UTF-8"?>
<Form>
<!-- Field elements can be added here for each HTML id -->
<!-- example -->
<Field>
<ID>HeaderFirstShipDate</ID>
<InputFrom>/Message/Body/Document/BusinessDocument/CatalogAction/CatalogActionHeader/Date/Code/Normal[text
()='FirstShipDate']/../../DateValue/Value/text()</InputFrom>
<OutputTo>(//Form/Field/ID[text()='HeaderFirstShipDate'])/following-sibling::Value</OutputTo>
<Value/>
</Field>
</Form>
```

Write a Custom Work Item Form

HTML form is used to write a work item. The HTML form is used as input to XMLC. XMLC generates a compiled Java class (XMLC component). Each ID in the HTML page is converted to an access method in XMLC component. The XMLC component is used by the application to access HTML element using XMLC APIs, set values and generate HTML.

The compiled Java class implements application interfaces such as IMqContent and IWorkItemDisplayDoc. These interfaces are used by the application to access the HTML elements. That is why your HTML page must have the following elements with a predefined ID. The IDs are:

- hiddenRetURL
- hiddenURLCtr
- divHeadId
- divBodyId
- command
- work itemid
- closedgroup
- opengroup
- numberOfRows
- status
- endday
- endmonth
- endyear
- startday
- startmonth
- startyear
- buyer
- marketplace
- recordcount
- pagebeginid
- pageendid
- dir
- doctype
- setbeginid
- numberOfRows
- trAttachments
- aAttachments

- trPrintableVersion
- aPrintablePO

You can use these IDs appropriately in your page.

Apart from these elements you might want to display some additional information from the source XML which is an mlXML document for the work item.

Use an HTML editor to draw the layout and appearance of the form. Assign unique IDs to each HTML section that must be populated with a value.

Sample HTML Page

```
<html>
  <head>
    <title id="titleId">Page Title</title>
    <div id="divHeadId">
    <script language="JavaScript" src="javascripts/cal.js" type="text/javascript"></script>
    <script language="JavaScript" src="javascripts/DateFormat.js" type="text/javascript"></script>
    <script language="JavaScript" type="text/javascript"></script>
    <script>
      // boolean flag to check if the page has already been submitted yet or not
      // fixes the problem with double clicking on the save button
      var alreadySubmitted = false;
```

```
function gotoInbox() {
document.WorkItemForm.submit();
}
function setAlreadySubmitted (submitted) {
alreadySubmitted = submitted;
}
function isAlreadySubmitted () {
return alreadySubmitted;
}
function doSubmit (command) {
setCommand(command);
if (!isAlreadySubmitted()) {
   setAlreadySubmitted(true);
   return true;
}
return false;
}
```

```
        function setCommand(str){
        document.WorkItemForm.command.value=str;
        return;
        }
      </script>
      </div>
    </head>
    <body topmargin="0" leftmargin="0">
      <div id="divBodyId">
        <form id="formTag" method="POST" action="Action" name="Name">
          <!-- Mandatory id elements -->
      <input id="hiddenRetURL" type="hidden" name="retURL" value="">
      <input id="hiddenURLCtr" type="hidden" name="URLCtr" value="">
      <input id="numberOfRows" type="hidden" name="numberOfRows" value="">
      <input id="workitemid" type="hidden" NAME="workitemid" value="" >
      <input id="command" type="hidden" name="command" value="">
      <input id="status" type="hidden" name="status" value="">
      <input id="endday" type="hidden" name="endday" value="">
```

```
<input id="endmonth" type="hidden" name="endmonth" value="">
<input id="endyear" type="hidden" name="endyear" value="">
<input id="startday" type="hidden" name="startday" value="">
<input id="startmonth" type="hidden" name="startmonth" value="">
<input id="startyear" type="hidden" name="startyear" value="">
<input id="buyer" type="hidden" name="buyer" value="">
<input id="marketplace" type="hidden" name="marketplace" value="">
<input id="recordcount" type="hidden" name="recordcount" value="">
<input id="pagebeginid" type="hidden" name="pagebeginid" value="">
<input id="pageendid" type="hidden" name="pageendid" value="">
<input id="dir" type="hidden" name="dir" value="">
<input id="doctype" type="hidden" name="doctype" value="">
<input id="setbeginid" type="hidden" name="setbeginid" value="">
<input id="tasktype" type="hidden" name="tasktype" value="">
<input id="FormResult" type="hidden" name="" value="Publish">
<input id="DocumentEffectiveDate"type="hidden" name="effectiveDate"-->
<br />
<table width="100%" border="0" cellspacing="1" cellpadding="1">
```

```
        <tr>
          <td class="header" align="left" colSpan="2" width="100%"><span id="PageTitle" NAME=" ">Page Title</span></td>
        </tr>
      </table>
      <table width="100%" border="0" cellspacing="1" cellpadding="1">
        <tr>
          <td class="header" align="left" colSpan="2" width="100%"><span id="FormTitle" NAME=" ">Form Title</span></td>
        </tr>
        <tr id="trPrintableVersion">
          <td class="subheader" colspan=2 align=center><div ><a
id="aPrintablePO" href="DisplayPrintableDoc"></a></div></td>
        </tr>
        <tr id="trAttachments">
          <td class="subheader" colspan=2 align="center"><div><a
id="aAttachments" href="DisplayWorkItem"></a></div></td>
        </tr>
        <tr id="trLatestPO">
          <td align="center" class="subheader" colspan="2">
          <div>There is a <a id="aLatestPO" href="DisplayWorkItem"><font color="white">newer</font></a> workitem for this
Order.</div>
```

```
            </td>
          </tr>
        </table>
        <!-- Add customization here -->
        <table width="100%" border="0" cellspacing="1" cellpadding="2">
          <tr>
            <td width="15%" align="left" class="leftTD">Master Cataog Name</td>
            <td width="85%" align="left" valign="middle" class="rightTD">
            <span id="MessageRetailerMasterCatalogName" align="left"> </span>
            </td>
          </tr>
          <tr>
            <td align="left" class="leftTD">Trading Partner</td>
            <td align="left" valign="middle" class="rightTD">
            <span id="MessageChannelCredentialDomain" NAME=" "> </span> - <span
id="MessageChannelCredentialIdentity" NAME=" "></span>
            </td>
          </tr>
```

```
        <tr>
          <td align="left" class="leftTD">Message Source</td>
          <td align="left" valign="middle" class="rightTD">
          <span id="MessageChannelOrganizationName" align="left"> </span>
          </td>
        </tr>
        <tr>
          <td align="left" class="leftTD">Date</td>
          <td align="left" valign="middle" class="rightTD">
          <span id="HeaderTimeStampDate" NAME=" "> </span> <span id="HeaderTimeStampTime" NAME="
"> </span> <span id="HeaderTimeStampZone" NAME=" "> </span>
          </td>
        </tr>
      </table>
      <table width="100%" border="0" cellspacing="1" cellpadding="2">
        <tr>
          <td width="100%" colspan="2" align="left" class="subheader">
          <div id="elementTableTitle">Product Details</div>
```

```
            </td>
          </tr>
          <tr id="Line">
            <td width="15%" class="leftTD">
            <span id="LineProductID" name=" "> </span>
            </td>
            <td width="85%" class="rightTD">
            <span id="LineProductName" name=" "> </span>
            </td>
          </tr>
        </table>
        <!-- Add customization here -->
        <table border="0" cellpadding="2" cellspacing="1" id="closedgroup" width="100%">
          <tr class="btns">
          <td align="center" width="100%">
          <A href="javascript:document.WorkItemForm.submit();" onclick="return doSubmit
('PROCESS');" onmouseout="javascript:submitbtn.src='images/submit_btn.gif'; return
true;" onmouseover="javascript:submitbtn.src='images/submit_btn.gif'; return true;">
            <IMG alt="Submit" border="0" name="submitbtn" src="images/submit_btn.gif" />
```

```
        </A>
         
        <A href="javascript:document.WorkItemForm.submit();" onclick="return doSubmit
('CANCEL');" onmouseout="javascript:cancelbtn.src='images/cancel_btn.gif'; return
true;" onmouseover="javascript:cancelbtn.src='images/cancel_btn.gif';return true;">
          <IMG alt="Cancel" border="0" name="cancelbtn" src="images/cancel_btn.gif" />
        </A>
        </td>
        </tr>
      </table>
      <table border="0" cellpadding="2" cellspacing="1" id="opengroup" width="100%">
        <tr class="btns">
        <td align="center" colspan="2">
          <a href="javascript:gotoInbox();" onclick="return doSubmit
('CANCEL');" onmouseout="javascript:cancelbtn.src='images/cancel_btn.gif'; return
true;" onmouseover="javascript:cancelbtn.src='images/cancel_btn.gif';return true;">
          <IMG alt="Cancel" border="0" name="cancelbtn" src="images/cancel_btn.gif" />
          </a>
        </td>
        </tr>
```

```
        </table>
          </form>
        </div>
      </body>
    </html>
```

Use an HTML editor to draw the layout and appearance of the form. Assign unique IDs to each HTML section that must be populated with a value. For example, if you have an HTML element like:

```
<table border="1">
  <tr>
    <td>Name</td>
    <td id="Name">Name goes here.</td>
  </tr>
</table>
```

Compiling and Deploying the Custom Form

Procedure

1. Set CUST_DEVROOT to your preferred location.
2. Change to the %MQ_HOME%\build\custom.
3. Execute the customutil script as follows:

   ```
   customutil.sh -createDirStructure
   ```

   This command creates a directory structure in the directory specified by the CUST_DEVROOT variable. You might notice some errors. However, as long as the directory structure is created, you can ignore the errors.

4. Set the ANT_HOME variable as follows:

```
set ANT_HOME=%MQ_HOME%\bin\buildTool
```

5. Set the PATH variable as follows:

```
set PATH=%PATH%;%MQ_HOME%\bin\buildTool\bin
```

6. Create a %MQ_HOME%\thirdparty directory and copy the XMLC JAR files, gnu-regexp JAR files, and the jsse JAR files to the %MQ_HOME%\thirdparty directory.
7. Create the templates folder in the %CUST_DEVROOT%\src\com\tibco\mdm\ui\workflow\engine\workitem directory.
8. Copy Javascript, images, css from ECM.ear\EML.war to %CUST_DEVROOT%\src\com\tibco\mdm\ui\infrastructure.
9. Copy the custom HTML to the following directories:
   - Workitem:

     ```
     %CUST_DEVROOT%\src\com\tibco\mdm\ui\workflow\engine\workitem \templates
     ```

   - Email form:

     ```
     %CUST_DEVROOT%\src\com\tibco\mdm\ui\directory\contact
     ```

10. Save all the HTML files in the html folder and other source files under relevant folders. For example, images in the images folder. After copying the HTML files, remember to change the links to images, css, and Javascript.
11. Edit the $MQ_HOME\build\resources\html_interfaces.txt file to add a line for the HTML form that implements IWorkItemDisplayDoc:
    - Workitem:

      ```
      com.tibco.mdm.ui.workflow.engine.workitem.templates.MyCustomization=com.tibco.
      mdm.ui.infrastructure.IMqContent,com.tibco.mdm.ui.workflow.engine.workitem.workit
      em.IWorkItemDisplayDoc
      ```

    - Email form:

```
com.tibco.mdm.ui.directory.contact.MyEmailCustomization=com.tibco.mdm.ui.infrastr
ucture.IMqContent
```

12. To compile HTML into xMLC classes, open a command prompt, change directory to %MQ_HOME%\build\custom, and execute the following command:

    ```
    ant compile
    ```

    The class files are created in the

    - Workitem:

      $CUST_DEVROOT\classes\servletclasses\WEB-INF\classes\com\tibco\mdm\ui\workflow\engine\workitem\templates directory

    - Form:

      ```
      $CUST_DEVROOT\classes\servletclasses\WEB-
      INF\classes\com\tibco\mdm\directory\contact
      ```

13. To deploy the custom forms, open a command prompt, change directory to $MQ_HOME/build/custom and execute the following command:

    ant deployCustom

14. Add the class file to the following:

    - For work item:

      ECM.ear\ECMClasses.jar\com\tibco\mdm\ui\workflow\engine\workitem\templates folder.

    - For Form:

      ECM.ear\ECMClasses.jar\com\tibco\mdm\directory\contact folder.

15. To create a custom TaskType to work on the HTML:

    a. Create a com.tibco.mdm.ui.workflow.engine.workitem.MyCustomization.java class that implements ICustomizer. Compile and deploy the class as described in the previous steps.

    b. In the createWorkItem activity of the workflow, modify the TaskType parameter with the new value as follows:

       <Parameter direction="in" name="TaskType" eval="constant"

type="string">MATCHRECORD</Parameter>

c. Associate the TaskType and the class by adding the following property in the ConfigValues.xml file:

```
com.tibco.workflow.workitem.MATCHRECORD.class=com.tibco.mdm.ui.workflow.engine.workitem.MyCustomization
```

16. Add an entry in WORKFLOWFORM table in application database. The attributes of this table and the value they take are:
    - ID: Primary Key, provide a unique number
    - VERSION: Default value. For example, 1.
    - OWNERORGANIZATIONID: Default value is 1 for all organizations. Otherwise, provide a specific organization ID.
    - NAME: Custom form name.
    - DEFINITION: XMLC generated class name.
    - ACTIVE: Set to 'Y'.
    - MODMEMBERID: Default value is 101.
    - MODDATE: Provide current date.
    - MODVERSION: Default value is 1.
    - TYPE: EMAIL/TASK, depending on your form type. Use TASK for work item and EMAIL for Email.
    - ACTIONABLE: If form has some action, set to 'Y', otherwise 'N' for notifications.

      For example:

```
INSERT INTO WORKFLOWFORM (ID, VERSION, OWNERORGANIZATIONID, NAME, DEFINITION, ACTIVE, MODMEMBERID, MODDATE, MODVERSION, TYPE, ACTIONABLE) VALUES (MQ_SEQUENCE_1.nextval, 1, 1, 'My Custom Form Name', 'com.tibco.mdm.ui.workflow.engine.workitem.template.MyCustomization', 'Y', 101, CURRENT_TIMESTAMP, 1, 'TASK', 'N');
```

Gathering Inputs from a User Using the Work Item Form

Procedure

1. Define an HTML input element. You can use any type for the input element.
2. Assign the input element an 'ID'. For example, 'ID=myuserinput'.
3. Set an 'OutputTo' XPATH in the XML form being used by the work item. For example:

```
<OutputTo>(//Form/Field/ID[text()='myuserinput'])/following-sibling::Value</OutputTo>
```

Result

The work item generates two outputs, one is the result XML, and the other is a merged document.

- Result XML: This is a copy of the XML form. The difference being, any user inputs given are part of it.
- Merged Document: Apart from the above configurations, if the user had also provided a valid 'InputFrom' XPATH into the input work item document, the work item activity merges the user input into the input document (which in most cases is a mlXML document).

Note:

- The form that you write only displays the body of the page. It does not control either the left hand menu or the top banner.
- The IDs used in the form must NOT be substrings of one another. That is, you cannot have two IDs like 'MasterCatalogHeader' and 'MasterCatalogHeader2'. IDs of this kind are used for cloning of nodes (see 'Defining Repeating sections in a form'). Instead, in this example, you could use 'MasterCatalogHeader' and '2MasterCatalogHeader'.
- For merging the result, it is mandatory that all XML nodes (except text nodes) given in the 'InputFrom' XPATH are present in the input document.

Add Repeating Elements in the Form

It is possible to repeat HTML elements based on collection of multiple XML nodes resulting from an xpath in the source XML.

In most cases, these repeating elements would form 'rows' in a 'table' and each node in collection could be mapped to a HTML row, for example, one 'CatalogItem' in the source XML could be mapped to one tr element.

- An HTML element that would be repeated must be added to the HTML form. This element is not displayed to the user but is used as a reference node that would be cloned at runtime. In the current example, it is simply a HTML row element.
- Each column in the HTML row must have an 'id' attribute that starts with the 'id' of the reference node. Here the reference node is id 'Line' and hence all child elements must have id like 'LineProductID', 'LineProductExtension' and so on.
- The reference node and its child element must have a corresponding entry in XML form.
    - The entry in XML form corresponding to reference node must have:

      <ID>: set to id of reference node

      <InputForm>: set to XPath of repeating XML nodes in source XML document. For example, if <CatalogItem> results in collection, XPath to CatalogItem.
    - The entry in XML form corresponding to child element must have,

      <ID>: set to id of child element

      <InputForm>: set to XPath, which starts with XPath of reference node.
- The following example illustrates the procedure.
    - In the HTML code, add a sample row:

```
<table border="1">
<tr id="tableHeaderRow">
<td>ProductID</td>
<td>Product Extension</td>
<td>Long Desc</td>
</tr>
<tr id="Line">
<td id="LineProductID">&amp;nbsp;</td>
<td id="LineProductExtension">&amp;nbsp;</td>
```

```
<td id="LineProductDescription"> </td>
</tr>
</table>
```

- In the XML form, add entries having XPath pointing to location of data for each of the row elements.

```
<Field>
<ID>Line</ID>
<InputFrom>
/Message/Body/Document/BusinessDocument/CatalogAction/CatalogActionDetails/CatalogItem
</InputFrom>
<OutputTo></OutputTo>
<Value/>
</Field>
<Field>
<ID>LineProductID</ID>
<InputFrom>
/Message/Body/Document/BusinessDocument/CatalogAction/CatalogActionDetails/CatalogItem/PartNum
ber/GlobalPartNumber/ProdID/IDNumber</InputFrom>
<OutputTo></OutputTo>
<Value/>
</Field>
```

```
<Field>
<ID>LineProductExtension</ID>
<InputFrom>
/Message/Body/Document/BusinessDocument/CatalogAction/CatalogActionDetails/CatalogItem/PartNum
ber/GlobalPartNumber/ProdID/IDExtension</InputFrom>
<OutputTo></OutputTo>
<Value/>
</Field>
<Field>
<ID>LineProductDescription</ID>
<InputFrom>
/Message/Body/Document/BusinessDocument/CatalogAction/CatalogActionDetails/CatalogItem/PartNum
ber/GlobalPartNumber/ProdDescription</InputFrom>
<OutputTo></OutputTo>
<Value/>
</Field>
```

Use Reserved XMLC IDs

Apart from the customizations listed in the "Add Repeating Elements in the Form" section, you can also use the predefined XMLC IDs to display information or links.

The following XMLC IDs are available and can be used during customization of the page. The values get populated automatically when you use these IDs. Restrict changes to following ids only.

For Primary Products

Prefix all IDs with the text "MAIN". Attributes can be added using the format "MAIN<ATTRIBUTENAME>".

- "MAINMODDATE": Mod Date
- "MAINMODVERSION": Mod Version
- "MAINMODMEMBER": Mod Member Name
- "MAINCREATIONDATE": Creation Date
- "MAINREJECTIONEDITLINK": Create a Rejection Edit Link
- "MAINREJECTIONVIEWLINK": Create a Rejection View Link
- "MAINPRODUCTEDITLINK": Create a Product Edit Link
- "MAINPRODUCTVIEWLINK": Create a Product View Link
- "MAINREJECTIONEDITLINK": Create a Rejection Edit Link
- "MAINREJECTIONEDITLINK": Create a Rejection Edit Link
- "MAINCOMPARELINK": Create a Product Compare Link

For Associated Products

Prefix all IDs with the text "ASSOC". Attributes can be added using the format "ASSOC<ATTRIBUTENAME>".

- "ASSOCMODDATE": Mod Date.
- "ASSOCMODVERSION": Mod Version.
- "ASSOCMODMEMBER": Mod Member Name.
- "ASSOCCREATIONDATE": Creation Date.
- "ASSOCRELATIONSHIPTYPE": Relationship type.
- "ASSOCREJECTIONEDITLINK": Create a Rejection Edit Link.
- "ASSOCREJECTIONVIEWLINK": Create a Rejection View Link.
- "ASSOCPRODUCTEDITLINK": Create a Record Edit Link.
- "ASSOCPRODUCTVIEWLINK": Create a Record View Link.
- "ASSOCPRODUCTCOMPARELINK": Create a Compare Product Link.

For Products Related to Associated Products

Prefix all IDs with the text "REL". Attributes can be added using the format "REL<ATTRIBUTENAME>".

Use Reserved XMLC IDs

Record data from the previous Confirmed version can be displayed on a work item by specifying the element with ID as "showPreviousProduct". The value for this element must be set to ‘Y’.

For example:

```
<input id="showPreviousProduct" type="hidden" name="showPreviousProduct" value="Y">
```

XMLC IDS

The following XMLC IDs are available and can be used. Prefix all IDs with the text “PMAIN”.

Reserved XMLC IDs

XMLC ID	Description
PMAIN<ATTRIBUTENAME>	Attributes
PMAINMODDATE	Modification date
PMAINMODVERSION	Modification version
PMAINMODMEMBER	Mod member name
PMAINCREATIONDATE	Creation date

Note:

- Links are not supported.
- Same support is available for associated and related records by using prefix of PASSOC and PREL.

Other Data

- SubmitButtonHiddenOnErrors: Set to "Y" to hide the work item submit button when errors are present.

  For example:

  ```
  <input id=" SubmitButtonHiddenOnErrors " type="hidden" name="
  SubmitButtonHiddenOnErrors " value="Y">
  ```

- ExpansionOption: Set to EXPANDALL to specify the default behavior when record view/edit is clicked from work item; COLLAPSEALL otherwise.

  ```
  <input id="ExpansionOption" type="hidden" name="ExpansionOption"
  value="COLLAPSEALL">
  ```

- Buyer Catalog Link: Add an element to include Buyer Catalog link on a CatalogApproval work item.

  For example:

  ```
  <input type="hidden" name="BCurl" value="" id="BCLink">
  <input type="hidden" name="browseurl" value="" id="HeaderLink">
  ```

- Previous Alert: Alerts from the previous step cab are displayed on the work item. An element with id "showPreviousAlerts" gives an option to indicate if alerts from the previous step must be shown or not. If a previous approver has rejected a work item, alerts are displayed in the current approver's work item if "showPreviousAlert" is set to Y.

  For example:

  ```
  <input id="ApproverForm" type="hidden" name="ApproverForm" value="Y">
  <input id="showPreviousAlert" type="hidden" name="showPreviousAlert" value="Y">
  ```

Email Template Customization

The only difference in case of Email templates is the configurable subject line.

Configurations are described in Write a Custom Work Item Form and Add Repeating Elements in the Form.

Email Subject Line Configuration

The subject line is coded into the HTML template being used. The line that is used for setting the subject has a subject ID.

For example:

```
<div id="subject"><span id="defaultSubject">Approval
Requested</span> for Product : <span
id="SubHeaderProductID">#filler#</span> - <span
id="SubHeaderProductExtension">#filler#</span></div>
```

The system fills in data from the source XML using the Xpath corresponding to the ID specified in the HTML template. The system builds the subject line left to right until it hits the first ID resolving to a NULL, or the end of the 'div' tag. The 'div' tag is removed from the body of the Email template.

The following is the list of reserved IDs used by the system:

- 'subject': This ID is used by the system to identify the section to be used for building the subject line.
- 'defaultSubject': This ID represents the static portion of the subject line that always appears in the subject.
- 'ManagedWorkItemMsg': This ID is used to place the work item delegation or reassignment messages.
- 'commentData': This ID is used to place the work item comments within a given event.

Note:

- The predefined XMLc IDs do not work on the input XML document.
- Use of AssociatedProduct IDs, add data, and links works for all related records.
- Avoid 'Coding repeating elements' when using AssociatedProduct IDs.

HTML Labels Using Resource Bundles Customization

Like the rest of the application user interface, the HTML labels in work item forms and Email templates can also be customized for each individual installation.

See Localize Text Strings.

Changing the Edit Box to Multiline Edit Box

For the Record Modify and Add screens, you can define the threshold length above which an edit box must change to a multiline edit box. Set the following properties in the Configurator. This only affects the Record user interface.

Procedure

1. Log in to the Configurator.
2. Navigate to **Advanced > UI Customization > Multiline Display Cutoff.**
3. Configure the number of characters to 400.

   When the number of characters in a line are > 400, the edit box is changed to a multiline edit box.
4. Configure the number of minimum and maximum limits on the number of lines in a multiline box. These properties control the height of the multiline box:

   Advanced > **UI Customization** > **Multiline Display Min Lines** = 3

   Advanced > **UI Customization** > **Multiline Display Max Lines** = 5

Suppressing the Mouse Over Help

Suppress the help shown on the repository and record screens (Add, Modify, and View) by setting the property in the Configurator to ‘false’.

Procedure

1. Log in to the Configurator.
2. Navigate to **Advanced > UI Customization > Record Attribute Mouse Over Help.**

Changing the Refresh Rate for Event Log

The default browser refresh for the Event Log page is 0.

Procedure

1. Log in to the Configurator.
2. Change the value of **User Interface Settings >** Event Details Refresh Time.
3. To enable autorefresh of the Event Detail Log page, set this property to a value greater than 0. The value must be specified in seconds and for best results, it must be any number greater than 30.

Changing the Number of Days for Event List

The default number of days the list of events is displayed on the Event Log page is 30 days.

Procedure

1. Log in to the Configurator.
2. Change the value of the Days Event Log Stored property.

Repository Management

- Default Attribute Length: While defining String attributes for repository, data

sources or synchronization formats, if length is not specified, this default length is used. Specified in number of characters.

- Import Error Threshold: Default error count threshold for number of rows found to be in error in an import.
- Record Compare Mode: If 'All' is specified, all attributes are shown on the record compare page. If more than 2 records are selected, the View Option = Changes is not supported, All is used.
- Import Parent Record Version State: Parameter to indicate which previous record version must be used to merge incoming record data during Import. If value is LATEST, the incoming record data is merged with the latest Confirmed/UnConfirmed version of the record. If value is set to CONFIRMED, the incoming record data is merged with the latest Confirmed version. Default value is CONFIRMED.
- Text Indexing Enabled: Setting this flag allows you to use text indexing and searching of the records. Valid choices for this value are:
    - NONE: Text Indexing and Text Searching are disabled.
    - OFFLINE: Text Indexing is done during off-peak hours using a console run utility scheduled by an administrator. Text Searching is enabled.
    - ONLINE: Text Indexing is automatically initiated while the data is changed. Text Searching is enabled.

Changing the Default Number of Rows

You can specify the default number of rows to be displayed in various pages by setting the properties in the Configurator.

Procedure

1. Log in to the Configurator.
2. Set the following properties:
    - Synchronization Profile List Default Rows
    - Event Detail Log Default Rows
    - Event Log Default Rows

- Inbox Default Rows
- Record History Default Rows
- Record List Default Rows
- Text Search Result Default Rows

3. Save the values.

Changing the Display Date and Time Formats

You can specify the default display date and time formats to be displayed in various pages by setting the properties in the Configurator.

Procedure

1. Log in to the Configurator.
2. Set the following properties:
   - Default Display Date Format
   - Default Display Time Format
3. Save the values.

Customization of Repository

The repository customization includes:

- Customizing Attributes and Attribute Groups
- Record Security Customization
- Manage Relationships
- Security of Relationship Attributes Configuration
- Suppress Informational and Warning Validation Messages
- Validation Using a Java Class Customization

Become familiar with the concepts before you customize the repositories.

Customizing Attributes and Attribute Groups

You can create an attribute or an attribute group through the user interface.

For more details about creating attributes or attribute groups, see the *TIBCO MDM User's Guide*.

You cannot modify a pre-defined attribute in a repository through the user interface. To modify the size or length of a predefined attribute, modify the metadata of the attribute in the database.

Procedure

1. Navigate to the repositories page.
2. View the repository, and modify the attribute length.
3. Save the repository.

Result

The new length of the attribute is shown on the user interface. Do not change anything on the user interface while modifying the repository.

Note: Ensure that both the above SQL entries have the same size for length.

For repositories that have user-defined table names and custom database column names for attributes, the statements are:

- ALTER TABLE <table name> MODIFY (<DB_COLUMN_NAME> VARCHAR2 (10));
- UPDATE CATALOGATTRIBUTE set ATTRIBUTELENGTH=10 where CATALOGID=<repository ID> and NAME='<ATTRIBUTE-NAME>';

Attribute and Attribute Group Customization Examples

This example changes the attribute length to 10.

ALTER TABLE MCT_<repository ID> MODIFY (C<ATTRIBUTE-NAME> VARCHAR2 (10));

UPDATE CATALOGATTRIBUTE set ATTRIBUTELENGTH=10 where CATALOGID=<repository ID> and NAME='<ATTRIBUTE-NAME>';

This example shows how to modify the length of PRODUCTID and SHORTDESC to 10 and 50 respectively.

```
ALTER TABLE MCT_3323 MODIFY (CPRODUCTID  VARCHAR2(10));
UPDATE CATALOGATTRIBUTE set ATTRIBUTELENGTH=10 where  CATALOGID=3323 and
NAME='PRODUCTID';
```

Navigate to the repositories page and view the repository.

Modify the attribute length and save the repository.

The new length of the attribute is shown on the user interface. You are not required to change anything on the user interface while modifying the repository.

Customization of the Record UI

The current record UI renders each attribute group as a separate tab but there are instances when repositories containing many attribute groups must be combined into a single group.

The valid values are displayed as multivalue drop-down list and requires you to select each item and only selected items are displayed. Also, the attribute with valid values list are displayed as a drop-down list using which you can select a single item. If the valid values are displayed as a multivalue check box and grouped as radio buttons, it is easy for selection.

Similarly, if a specific relationship is rendered within a relationship tree, you must expand the relationship tree to add or remove related records but if it is displayed as a single relationship tab it can be easily edited. You can also combine a relationship tab within another attribute group or a combined group.

> ⚠ **Caution:** The previous customization feature through the recordui-render.xml file is deprecated. In the 9.1.0 release, the customization of the record UI is supported only through the UI Builder feature of TIBCO MDM Studio. For information about using UI Builder, see *TIBCO® MDM Studio UI Builder User's Guide*.

Displaying Attributes in a Relationship Tree for a Parent Record

Similar to displaying attributes for children records in the relationship tree, you can configure attributes other than *Record ID* and *Extension* displayed in the relationship tree for a parent record. To perform this, add the com.tibco.mdm.repository.displayattributelist.enterprisename.repositoryname property using the Configurator.

Procedure

1. Log in to the Configurator.
2. Select the cluster level. For example, InitialConfig.
3. Select the **Advanced** configuration outline.

4. Select the **UI Settings** category. The Configuration and Setup For InitialConfig - UI Settings is displayed on the right side.
5. Click **Add New Property**. The Add New Property window is displayed.
6. Enter details for the new property in the following fields:

Configuration Value Definition

Field Name	Value
Configuration Value Name	Type the configuration value name. For example, Display Attributes.
Internal Name	Type the internal property name in the following format: com.tibco.mdm.repository.displayattributelist.*enterprisename.repositoryname*. **Note:** • The property name is case-sensitive and must be in lower case. For example, com.tibco.mdm.repository.displayattributelist.*name.person*
Version	Select the version from the drop-down list.
Visibility	Select the **Advanced** option.
Read Only	Select the **Read Only** check box.
Set as Hidden Property	Select the **Set as Hidden Property** check box to hide this property.
Description	Type the description. For example, Attributes to be displayed on the relationship tree for a parent record.
Value Type	Select the **List** option from the Value Type drop-down list. The other valid values are String, Numeric, Boolean, Enumeration, and Password.

Field Name	Value
Current Value	Type the record attributes in the Current Value field. For example, for the Person repository tree, you want to display John and Smith attributes.
Default Value	Type the record attributes in the Default Value field. You can set the default value to current value.

7. Click **Finish**. The newly added configuration value is displayed in the Configurator.

Displaying Attributes in a Relationship Tree for Children Records

In the relationship tree, by default the *Record ID* and *Extension* are displayed. Instead, you can configure the record attributes and relationship attributes to be displayed in the tree. To perform this, add the com.tibco.mdm.repository.displayattributelist.*enterprisename.repositoryname. relationshipname* property using the Configurator. You can add this property for each relationship.

Procedure

1. Log in to the Configurator.
2. Select the cluster level. For example, InitialConfig.
3. Select the **Advanced** configuration outline.
4. Select the **UI Settings** category. The Configuration and Setup For InitialConfig - UI Settings is displayed on the right side.
5. Click **Add New Property**. The Add New Property dialog box is displayed.
6. Enter details for the new property in the following fields:

Configuration Value Definition

Field Name	Value
Configuration	Type the configuration value name. For example, Display

Field Name	Value
Value Name	Attributes.
Internal Name	Type the internal property name in the following format: com.tibco.mdm.repository.displayattributelist.*enterprisename*.*repositoryname*.*relationshipname* **Note:** • The property name is case-sensitive and must be in lower case. For example, com.tibco.mdm.repository.displayattributelist.*techpubs.person.persontoacc* • Special characters are not allowed in the property name. For example, if you specify '-'(hyphen) in the *relationshipname*, the customization does not work.
Version	Select the version from the drop-down list. For example, 8.3.2.
Visibility	Select the **Advanced** option.
Read Only	Select the **Read Only** check box.
Set as Hidden Property	Select the **Set as Hidden Property** check box to hide this property.
Description	Type the description. For example, Attributes to be displayed on the relationship tree for children records.
Value Type	Select the **List** option from the Value Type drop-down list. The other valid values are String, Numeric, Boolean, Enumeration, and Password.

Field Name	Value
Current Value	Type the record attributes in the Current Value field. For example, for the PersonToAcc relationship tree, you want to display Nominees, Loan, and ATMCardNo attributes.
Default Value	Type the record attributes in the Default Value field. You can set the default value to current value.

7. Click **Finish**. The newly added configuration value is displayed in the Configurator.

Display Attributes in Record Header

In the record header, the record ID/Extension, state and versions are shown by default. In addition to these attributes, you can configure additional attributes on the record header. Add the following property using the Configurator. A property can be added for each repository.

```
com.tibco.mdm.repository.displayinheader.<enterprisename>.<repositoryName>
```

Note: The property name is case-sensitive and must be in lower case.

An example of the ConfigValues.xml entry after the property is added is shown below:

```
<ConfValue description="Attributes to be displayed on the record
header" name="displayAttributes" propname="com.tibco.mdm.repository.displayinheader.corpA.person" readonly="true"
sinceVersion="8.0.1" visibility="Advanced">

<ConfList>                          <ConfListString value="dateofbirth"/>

    <ConfListString value="married"/>

 </ConfList>
```

```
</ConfValue>
```

These attributes can be displayed in the following screens.

- Add Record
- View Record
- Modify Record
- Copy Record
- Delete Record
- Restore Record

Note:
- All those attributes which are not a part of a repository are ignored.
- Multivalue attributes are not supported.

Customizing Mandatory Indicator

The red asterisk (*) sign is shown for the mandatory attributes on the record UI and work item page of the TIBCO MDM UI. You can change the style and color of the asterisk (*) sign using the CSS file.
For information about mandatory fields, see the Mandatory Fields section in the *TIBCO MDM User's Guide*.

Procedure

1. Navigate to ECM.ear > EML.war > css > default.
2. Open the styles.css file.
3. Change the following values:

```
[isrequired=true]:before {
   content: "*";
```

```
    color: #ff0000;
}
```

4. Save the styles.css file.

Record Security Customization

The record security customization includes:

- Value Based Security for Record List Configuration
- Restrictions on Record Access Conditions
- Manage Relationships
- Security of Relationship Attributes Configuration
- Suppress Informational and Warning Validation Messages

Value Based Security for Record List Configuration

You can secure the record list screens (output of repository browse, subsets, and so on.) based on the value of attributes in the records itself. This security applies to the list of records displayed in the relationship section of the View/Modify record.

With this customization, it is possible to completely hide a record from a user using “hide_record” access, or restrict a user’s edit access to a record by using “view_record”. You must enhance the repository validation rulebase to do so.

see the *TIBCO MDM Studio Rulebase Designer User’s Guide* for more information.

Restrictions on Record Access Conditions

For performance reasons, the condition of rules containing record level access actions are converted to SQL in some cases. Therefore, only functions that can be translated into SQL can be used.

These include: **and, or, not, eq, neq, lt, leq, gt, geq, defined, undefined**.

Conditions containing expressions other than those listed above, fail. See the *TIBCO MDM Studio Rulebase Designer User’s Guide* for more information.

Manage Relationships

Relationships are used to see other record data items.

For example, you can specify that a can, say a can of soda, is contained by a case, and that cases are contained in a pallet. Relationships can also be used to define record substitutes. TIBCO MDM allows you to name these relationships, and determine whether they are unidirectional or bidirectional. Bidirectional relationships are visible from both the parent and child record. Unidirectional relationships are valid only from parent to child and not vice-versa, for example, cross-sale.

Controlling the Relationships Shown for an Item

While creating a record and associating it with other records, you can control which relationships apply at a specific step or level.

For example, for the main record, all relationships might be valid but for the associated child record, only some of the relationships might apply. You can also control the order in which relationships are listed.

Procedure

1. For each relationship type you include for an item, define a constraint in the catalogvalidation.xml file. This file shows relationships for items in all repositories.
2. To include a relationship type that is different from relationships in the rest of the repositories, add a constraint (specific to the repository) to the CatalogValidation.xml file. The file is located in the MQ_COMMON_DIR/ <enterprise lternalName>/catalog/master/<specific repository ID> directory.
3. To define this constraint:
    a. Declare a variable named RELATIONSHIP_LIST, with the usage attribute declared as "output".

```
<declare usage="output">
 <var>RELATIONSHIP_LIST</var>
</declare>
```

    b. Declare a variable named RECORD_ACTION.

```
<declare>
  <var>RECORD_ACTION</var>
</declare>
```

c. In the <name> element of the constraint, name the type of relationship you want to include. In the description element of the constraint, enter a description.

d. Define a condition element, similar to other conditions defined for other constraints in the Rulebase.

e. To show a different relationship list for RECORD_ACTION, define a separate constraint. The possible values for this RECORD_ACTION constraint are ADD, EDIT, and COPY. Optionally, you can use record attributes to control the list of valid relationships.

f. Define an action element for this condition, to assign the "RELATIONSHIPLIST" output variable. The following code sample defines a constraint for the 'Contains' relationship. You can add constraints for all of the relationships that you want to include for an item.

```
<constraint>
 <name>contains</name>
    <description>
    Default relationship if Record Action
    is"ADD" or "EDIT" or "COPY"
    </description>
 <condition>
    <or>
    <eq>
        <var>RECORD_ACTION</var>
        <const type="string">ADD</const>
    </eq>
    <eq>
            <var>RECORD_ACTION</var>
            <const type="string">EDIT</const>
    </eq>
    <eq>
            <var>RECORD_ACTION</var>
            <const type="string">COPY</const>
    </eq>
```

```
        <eq>
<!-Does Record_type attribute has value of VENDOR?>
            <var>RECORD_TYPE</var>
        <const type="string">VENDOR</const>
    </eq>
    </or>
</condition>
<action>
        <assign>
    <var>RELATIONSHIP_LIST</var>
    <const type="string">CONTAINS</const>
    <const type="string">CONTAINEDBY</const>
        </assign>
</action>
</constraint>
```

4. To show a relationship that is specific to RECORD_ACTION ='EDIT', define the constraint as "a general constraint that applies to RECORD_ACTION = 'ADD' or 'COPY.

Result

```
<constraint>
<name>contains</name>
    <description>
    Default relationship if Record Action is "ADD" or
    or "COPY"
    </description>
<condition>
<or>
<eq>
    <var>RECORD_ACTION</var>
    <const type="string">ADD</const>
</eq>
<eq>
        <var>RECORD_ACTION</var>
        <const type="string">COPY</const>
</eq>
    </or>
</condition>
    <action>
<assign>
    <var>RELATIONSHIP_LIST</var>
    <const type="string">CONTAINS</const>
    <const type="string">CONTAINEDBY
```

```
    </const>
    </assign>
</action>
</constraint>
```

The following constraint is specific to the EDIT action:

```
<constraint>
<name>containsforEdit</name>
<description>
Default relationship if Record Action is "EDIT"
</description>
<condition>
<eq>
    <var>RECORD_ACTION</var>
    <const type="string">EDIT</const>
</eq>
</condition>
<action>
<assign>
    <var>RELATIONSHIP_LIST</var>
    <consttype="string">CONTAINS</const>
</assign>
</action>
</constraint>
```

Security of Relationship Attributes Configuration

Rulebase security access control includes relationship-based objects. Within the rulebase, you can set **hide**, read-only **view**, and **modify** access to relationship attributes. The only attribute currently defined for a relationship is “Quantity.”

The **access** tag controls the **read/write/show** access to a particular relationship quantity, based on the **mode** attribute. This feature allows certain users to have restricted access to the quantity field for relationship attributes. For example, for a CASE that contains 12 of EACH, some users might have permissions to modify the quantity of EACH, while others might have read-only access.

Note that each attribute can only be assigned one **access mode**, and the first access mode assignment in the rulebase takes precedence.

Levels of Access and their Effects

The following table lists the levels of access and their effects.

Levels of Access and their Effects

Attributes	Effects
<access mode="hide">	The attribute does not appear on the page.
<access mode="view">	The attribute appears on the page as read-only.
<access mode="modify">	The attribute appears on the page and can be modified by the user.
<access mode="view_record">	The record appears in the record list, but cannot be modified.
<access mode="hide_record">	The record does not appear in the record list or relationships.

The relationship attribute (quantity) can be declared in one of the following ways:

```
<!-- Declare the following constraint into Relationship(RCT_*) Catalogvalidation.xml file to specify the access to the relationship attribute -->
<constraint>
<name>EachAccess</name>
<description>If UOM is EACH then change the access for contains Quantity to hide.</description>
<condition>
<eq>
  <var>PARENT/UOM</var>
```

```
    <const type="string">EACH</const>
</eq>
</condition>
<action>
<access mode="hide">QUANTITY</access>
</action>
</constraint>
```

```
<!--Declare the relationship attribute and then specify the access to the relationship attribute -->
<declare>
    <var>CONTAINSRELQTY</var>
    <link type="relationship">
     <literal>Contains</literal>
     <literal>Quantity</literal>
    </link>
</declare>
<constraint>
```

```
<name>EachAccess</name>
<description>If UOM is EACH then change the access for
contains Quantity to hide.</description>
<condition>
 <eq>
  <var>UOM</var>
  <const type="string">EACH</const>
 </eq>
</condition>
<action>
 <access mode="hide">CONTAINSRELQTY</access>
</action>
</constraint>
```

Suppress Informational and Warning Validation Messages

A level is assigned to each informational message. The threshold level, above which the messages are to be displayed in the user interface, might be configured in the rulebase or through the workflow.

Similarly, during validations, warnings of certain severities can be suppressed by setting a warning message threshold in the rulebase or through the workflow.

The three ways to set thresholds:

- By defining default thresholds in the rulebase header.
- By setting thresholds through work items (workflow).
- By computing the threshold in the rulebase.

Define Default Thresholds in the Rulebase Header

A sample section of the rulebase is shown as follows:

```
<information_threshold>1</information_threshold>
<warning_threshold>5</warning_threshold>
<severity>3</severity>
```

Set Thresholds Through Work Item (Workflow)

A sample section of workflow that sets the rulebase information threshold is shown as follows:

```
<Activity Name="CheckPrices">
                <action>Workitem</action>
                <Parameter direction="in" name="InformationThreshold" eval="constant"
type="long">5</Parameter><warning_threshold>5</warning_threshold>
                    <severity>3</severity>
```

You can set the following thresholds:

- InformationThreshold
- StepInformationThreshold
- WarningThreshold
- StepWarningThreshold

Step-thresholds are used to set thresholds that are relevant to the current step.

Validation Using a Java Class Customization

While saving a record, the application performs the following standard validations:

- PRODUCTID must be specified.
- PRODUCTID and PRODUCTIDEXT must be unique within a repository.
- For Boolean type attribute, value must be either TRUE or FALSE.

To extend the validations and cleansing rules defined using a rulebase, you could consider using a validator class. The use of this class is considered advanced customization and must be approached only if customization cannot be done using rulebase and rulebase custom functions.

The standard validation applied on the records can be customized across all the repositories or for a specific repository by placing the class in the appropriate directory.

The application looks for a CatalogProductValidator class in the following order:

- $MQ_COMMON_DIR/<enterpriseInternalName>/catalog/master/<id> directory specific to a repository. If a class is found, it is used for validation of the specified repository.
- If it is not found, the application checks whether any class has been defined for all repositories.

  $MQ_COMMON_DIR/<enterpriseInternalName>/catalog/master
- If no custom class is found, the standard record validations provided out-of-box are applied.

Customizing Standard Record Validations

Procedure

1. Write the Java class which extends the StandardCatalogProductValidator, which is the base implementation for ICatalogProductValidator interface. The sample class can be copied from $MQ_COMMON_DIR/standard/catalog/master/CatalogProductValidator.java.

   The ICatalogProductValidator interface must be implemented by the class that provides the custom validation. This interface has following methods:

   - public HashMap preValidate(ICatalogProduct cp, ICatalogProduct oldCP,

HashMap actions)

This callback method is called before rulebase validations are done.

- public HashMap validate(ICatalogProduct cp, ICatalogProduct oldCP, HashMap actions)

  This method implements record validations. These validations are applied after rulebase validations are done.

- public ArrayList isDuplicate(ICatalogProduct cp, ArrayList attrList, ArrayList valueList, int eventId, int ownerType)

  This method checks for duplicates. It overrides the standard duplicate check method.

- public ArrayList isDuplicate(ICatalogProduct cp, ArrayList attrList, ArrayList valueList)

  This method checks for duplicates. It overrides the standard duplicate check method. This callback is used to verify duplicate when event context is not available.

  Both isDuplicate methods must be implemented.

- public boolean getDuplicateCheckSQL(ArrayList attrList, ArrayList valueList, StringBuilder sql)

  This method builds the where clause for a duplicate check. For customization, override this method.

2. Compile the class and copy it to $MQ_COMMON_DIR/<enterpriseInternalName>/catalog/master/<catalogid>.

Customization of Data Types

The data types are defined as a string name and ID for the predefined data types. Whereas TIBCO MDM needs a large number of data types, both primitives and derived. You can customize the existing data types and add your own validation and formatting pertaining to the data type and page (optional).

You can include the predefined data types in TIBCO MDM. The data types are not enterprise-specific. The predefined data types are File, String, Integer, Decimal, Custom Decimal, Timestamp, Date, Amount, and Boolean, Long, and URL. For more information about the predefined data types, see *TIBCO MDM User's Guide*.

Modifying Data Type Operations

Procedure

1. Navigate to *$MQ_HOME* and extract EML.war from ECM.ear.
2. Navigate to the *$MQ_HOME*\ECM.ear\EML.war\javascripts\datatype folder.
3. Open the DataTypeClient.js file and modify the following custom methods:
    a. Custom operation with respect to data type and a particular page ID:
    DataTypeClient.prototype.*OPERATION_DATATYPE_SCREENID* = function(){ // Own implementation
    b. Custom operation with respect to data type for all screens:
    DataTypeClient.prototype.*OPERATION_DATATYPE* = function(){ // Own implementation }
    Where,
        - *OPERATION* = {formatForDisplay / validate}
        - *DATATYPE* = {4 / 11 / 100}
        - *SCREENID* = {850 / 900 / 950}
4. Archive EML.war to add the new methods.
5. Archive ECM.ear to add the updated EML.war.
6. Redeploy the ECM.ear file and restart the application server.

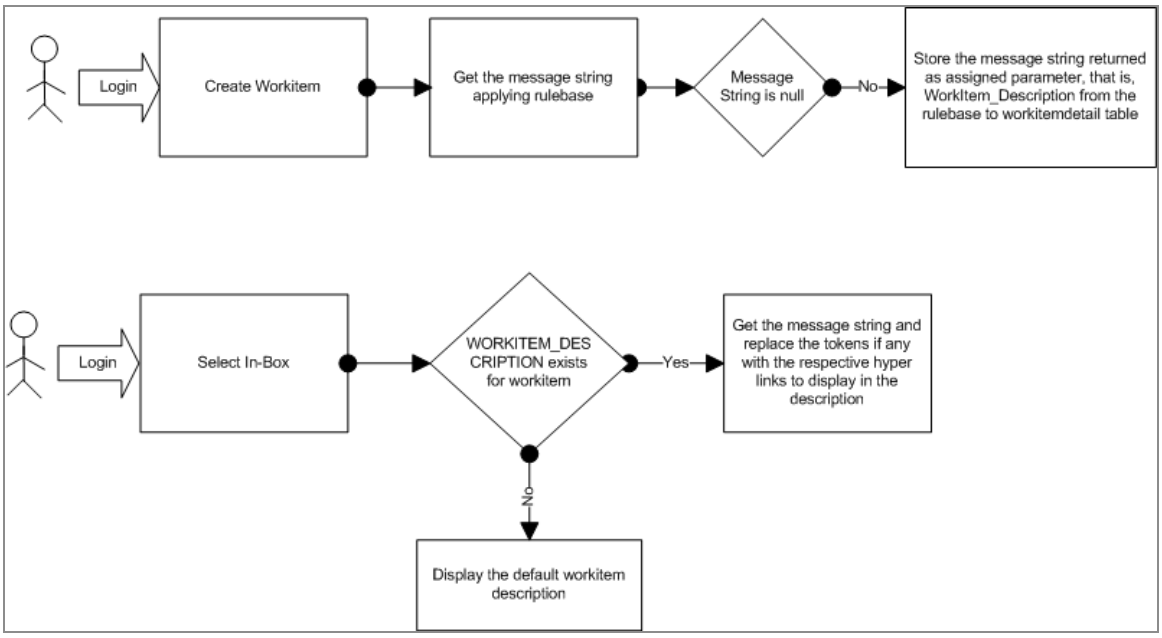
Customization of Inbox

Inbox consists of a list of work items. Therefore, to customize the inbox, you must customize various modules of work items.

Work Item Workflow

Out-of-the-box work item descriptions are derived using the rulebase rbworkitemdescription.xml stored under $MQ_COMMON_DIR/standard/rulebase.

The overall workflow is:



Customizing Work Item Descriptions

Procedure

1. Copy the rbworkitemdescription.xml file to $MQ_COMMON_DIR/<enterprise internal Name>/rulebase. Do not change the file name.

2. Modify the descriptions specified in rbworkitemdescription.xml. See following sub-sections for details.
3. Save the file.

Variables Available for Rulebase

The following table lists all the variables which can be used in rulebase conditions or in message descriptions.

- All values are available in session using SESSION/.
- All record attribute values are for the primary record. These attributes are accessed by simply specifying the name of the attribute.

The rulebase returns a message description as the output. This message is stored in the Work-item Detail table. The output parameter name is WORK-ITEM_DESCRIPTION.

Rulebase Attributes

Rulebase Attribute Name	Description
WORKITEM /MASTER_ CATALOG_NAME	Name of the repository.
WORKITEM /MASTER_ CATALOG_VERSION	Version of the repository.
WORKITEM /CATALOG_NAME	Name of the catalog.
WORKITEM /CATALOG_ VERSION	Version of the catalog.
WORKITEM /DOCTYPE	DocType.
WORKITEM /DOCSUBTYPE	DocSubType.
WORKITEM /INTENT	Intent of the work item.
WORKITEM /ERRORS	Number of errors in the reference step. If none specified, 0.

Rulebase Attribute Name	Description
WORKITEM /REJECTIONS	Number of rejections in the reference step. If none specified, 0.
WORKITEM /WARNINGS	Number of warnings in the reference step. If none specified, 0.
WORKITEM /TRADING_ PARTNER	Name of trading partners.
WORKITEM /TRADING_ PARTNER_TYPE	Type of trading partner organization: RETAILER, SUPPLIER or DATAPOOL.
WORKITEM /MARKETPLACE_ NAME	Name of the data pool.
WORKITEM /RECORD_COUNT	Number of records associated.
WORKITEM /SEVERITY	Severity specified for the work item.
WORKITEM /STEPSEVERITY	Step severity specified for the work item.
WORKITEM /ACTIVITY_NAME	Name of the work item activity.
WORKITEM/CustomMerge	Matching Record Merge Approval
RECORD_ID	Record ID.
RECORD_VERSION	Record mod version.
RECORD_KEYID	Record key ID.
RECORD_IDEXT	Record ID extension.

Additionally, all work item details saved can also be inquired by specifying the name as: WORK-ITEM /<detailName>.

If you are required to save any additional work item details to include in a work item description, you can specify these as input to the work item activity:

```
<Parameter direction="in" name="CUSTOM_text" type="String" eval="constant">my custom text</Parameter>
```

Note the prefix “CUSTOM_”. It must be specified exactly as in the example. These custom details can be accessed as any other work item detail.

```
WORK-ITEM /CUSTOM_text
```

Hyperlinks

You can include hyperlinks to any of the following objects in your work item description. To do this, you must include one of the following tokens in the description. You can have only one hyperlink per object in a message.

Hyperlinks

Token	Description	Exception Conditions
$MasterCatalog$	Use this token to create a hyperlink with the repository. The name of the repository appears in place of the token, and is hyperlinked to the repository browsing page. The repository is associated with most of the work items, however, if it is not available, the token is replaced by blanks.	Not available when repository is not defined. Unknown system alert.
$Catalog$	Use this token to create a hyperlink with the catalog. The name of the catalog appears in place of the token, and is hyper linked to the latest version of the catalog. If no catalog is available, the token is replaced by blanks.	Not available when no catalog is applicable. Add, modify, delete a record. (GDSN only) CIN message notifications received by

Token	Description	Exception Conditions
		retailers. (GDSN only) CIC received by suppliers which were not sent to retailers using the catalog. Unknown partner alert. Repository import approval.
$TradingPartner$	Use this token to create a hyperlink with the Trading partner profile view. The name of the trading partner appears in place of the token and is hyper linked to the latest version of the profile. If no trading partner is available, the token is replaced by blanks.	Not available when there is no trading partner associated with an event: Add, modify, delete a record. Repository import approval. Synchronization with data pool, with no trading partner. Unknown partner alert.
$Marketplace$	This token is to be used to create a hyperlink with the data pool organization profile. The name of the data pool appears in place of the token and is hyper linked to the latest version	Not available when there is no data pool associated with an

Token	Description	Exception Conditions
	of the profile. If no data pool is available, the token is replaced by blanks.	event: Add, modify, delete a record. Repository import approval. Synchronization with trading partners without going through the data pool.
$PrimaryRecord$	The attributes takes the value of the RecordId/ProductID if nothing is specified, otherwise it picks up the mentioned record attributes value.	Not available when there is no record involved in the event. Add, Modify, Delete and synchronization of the records.
$Hierarchy$	Use this token to create a hyperlink with the hierarchy. The token is replaced with the display name of a hierarchy and is hyperlinked to the hierarchy viewer. For the Link Approval work item, this token is hyperlinked to the browse linkages page. If no hierarchy is available, the token is replaced by blank. To customize the hierarchy work item descriptions, the rules are defined in the rbworkitemdescription_hierarchy.xml rulebase file.	Not available when the hierarchy associated with the work item does not exist in the database.

Token	Description	Exception Conditions
	This rulebase file is linked in the rbworkitemdescription.xml file. Both the files are located at $MQ_COMMON_DIR/standard/rulebase.	

Adding a New Search Attribute

You can customize the attributes on which you would like to search work items.

Procedure

1. Identify the name of the search attribute. The name must be an attribute name in the repository.
2. Identify the workflow and the work item activity to which this parameter must be added.
3. You can find the workflow files in $MQ_COMMON_DIR/standard/workflow/ folder. You must look for the workflow that creates the work item.
4. Define the parameter as follows. Use uppercase for the name of the attribute. If there is already a record attribute parameter defined, add a running number as in *RecordAttributeName1*.

```
<Parameter direction="in" name="RecordAttributeName" type="String”
eval="constant">ATTRIBUTENAME</Parameter>
```

Enabling Work Item Locking

Procedure

1. Log in to Configurator.
2. Set the **InitialConfig > Workflow Settings > Enable Work Item Locking** property to **true**.

   The default is false.

Enabling Automatic Locking

You can enable automatic locking of work items when they are opened from the UI. When this property is set to true, it instructs the application to lock the work item when the user opens the work item using the UI. It has no impact on web services.

Procedure

1. Log in to the Configurator.
2. Set the **InitialConfig > Workflow Settings > Enable Automatic Work Item Locking** property to **true**.

   The default is false.

Work Item Locking Configuration

Using the Configurator, you can configure the following properties related to work item locking:

- **Default Work Item Lock Expiry Method** – Valid values are **None** and **Relative**. **None** indicates that the lock never expires. **Relative** indicates that the locks expire after a specified interval. **None** is the default.
- **Default Work Item Lock Interval** – Indicates the default work item lock interval in seconds. This is relevant only if **Default Work Item Lock Expiry Method** is **Relative**.
- **Enable Release of Work Item Lock on Timeout** – On timeout of work items, the lock of a work item is released by default. To disable it, set the **InitialConfig > Workflow Settings > Enable Release of work Item Lock on Timeout** property in the Configurator to **false**. The default is **true**.

Customizing Work Item Approval Message

You can customize the message that is displayed when a work item is quickly approved. The message is shown in the **Process Comment History** box of the Record Introduction page.

Before you begin

The work item must have been approved using the **Approve** icon on the Inbox page.

Procedure

1. Navigate to $MQ_HOME/ECM.ear/lib/ECMClasses.jar/com/tibco/mdm/properties/mdm.
2. Open the SharedStringResources.properties file.
3. Search for the following string: UI_WORKLIST_HANDLER_WORKITEM_ APPROVE_ DEFAULT_COMMENT=This Work Item was quick approved.
4. Customize the message as desired and save the SharedStringResources.properties file.

   The customized message is displayed in the **Process Comment History** box of the Record Introduction page.

Customization of Business Processes

Using Business Processes Rules you can customize the business processes in TIBCO MDM. The application internally uses TIBCO BusinessEvents for evaluating business rules.

The following are some useful concepts for customizing the business process:

- Rule Definitions: Rules are defined as a series of conditions and actions. If the conditions are satisfied (evaluate to 'true'), the actions are executed. For example, here is a rule with 3 conditions and 1 action:

```
IF
    MasterCatalog is 'MastCat1' and
    Product Department is 'ProdDept A' and
    Document Type is 'Add Product'
THEN
Approver is 'Manager_123'
```

- Rule Domain (Also called Rule Meta Model): A rule domain is a collection of all conditions and actions required for processing a unit of the business process. It is defined in XML.
- Rule Template (Also called Rule Model): A rule template is a subset of the conditions and actions from a rule domain. It has a selection of the conditions and actions that would be used to define the rules. The templates are defined using the user interface.
- Rule Instance (Or just Rule): A rule is an instance of a rule template. The user interface is used to define the rule.
- Default Rule: A default rule is a special rule in which all conditions are set to 'Any'. This rule is fired only if no other rules are fired in that domain. There can be only one default rule per domain.

Adding New Business Process Domains

Business process domains are the contexts in which rules are applied. Each business process domain contains one or more templates.

When a new enterprise is created, all the domains listed in $MQ_HOME/config/security/defaultdata.xml (under the rulemetamodel section) are created.

Procedure

1. Define a new rule domain.
2. Identify a list of conditions and actions needed. If needed, define new conditions and actions.
3. Add the newly created domain to the required enterprises.

Defining a New Rule Domain

Procedure

1. Identify all the conditions and actions that might be needed.
2. Configure the identified conditions and actions into the rule domain definition under $MQ_HOME/config/rules.
3. To validate the correctness of the rule domain, you can use the $MQ_HOME/config/rules/RuleModel.dtd schema.

   For example, the following shows a rule domain with 10 conditions and 7 actions:

```
<RuleModel OrganizationID="234" RulesetName="Product Edit
Approval" RuleModeName="None" Priority="0" RuleID="-1" RuleType="Standard">

  <Description><![CDATA[Conditions and actions defined by the "Product Edit Approval" builder can be used to
create various rule templates. Rules created from the templates define list of participants involved during the
approval process.</Description>

  <Imports>

    <Import Import="com.tibco.mdm.integration.ruleengine.IAttributable;"/>

  </Imports>
```

```
<Conditions>
  <ImportConditions>
    <ImportCondition ImportCondition="CONDITION_DefaultRuleCheck.xml" default="Y"/>
    <ImportCondition ImportCondition="CONDITION_Organization.xml" default="Y"/>
    <ImportCondition ImportCondition="CONDITION_MasterCatalog-CatalogAction.xml"
default="Y"/>
    <ImportCondition ImportCondition="CONDITION_Action.xml" default="Y"/>
    <ImportCondition ImportCondition="CONDITION_CatalogProductUOM.xml"/>
    <ImportCondition ImportCondition="CONDITION_CatalogProductSupplierID.xml"/>
    <ImportCondition ImportCondition="CONDITION_CatalogChangeCode.xml"/>
    <ImportCondition ImportCondition="CONDITION_DocumentTypeProd.xml"/>
    <ImportCondition ImportCondition="CONDITION_ConvState.xml" default="Y"/>
    <ImportCondition ImportCondition="CONDITION_DeploymentMode.xml" default="Y" />
  </ImportConditions>
</Conditions>
<Actions>
  <ImportActions>
```

```
        <ImportAction ImportAction="ACTION_DefaultRuleCheck.xml" default="Y"/>
        <ImportAction ImportAction="ACTION_RoleParticipant.xml"/>
        <ImportAction ImportAction="ACTION_RoleType.xml"/>
        <ImportAction ImportAction="ACTION_MemberParticipant.xml" default="Y"/>
        <ImportAction ImportAction="ACTION_EnterpriseMemberParticipant.xml"/>
        <ImportAction ImportAction="ACTION_MemberType.xml" default="Y"/>
        <ImportAction ImportAction="ACTION_ConvState.xml" default="Y"/>
      </ImportActions>
    </Actions>
</RuleModel>
```

Note: The following is a mandatory condition: <ImportCondition ImportCondition="CONDITION_DefaultRuleCheck.xml" default="Y"/>

Do not remove this condition while defining a new business process domain, else an error occurs.

4. Edit the defaultdata.xml ($MQ_HOME/config/security/defaultdata.xml) file. Add the following entries. These add the rule domain and the 'default' rule templates for new enterprises created henceforth.

   Each section corresponds to the type of enterprise you are creating.

   - For a supplier, make an entry to the following sections:

```
<mq_default table="rulemetamodel" orgType="SUPPLIER">
<mq_default table="rulemodel" orgType="SUPPLIER">
```

For a retailer, make an entry to the following sections:

```
<mq_default table="rulemetamodel" orgType="RETAILER">
<mq_default table="rulemodel" orgType="RETAILER">
```

- For a data pool, make an entry to the following sections:

```
<mq_default table="rulemetamodel" orgType="MARKETPLACE">
<mq_default table="rulemodel" orgType="MARKETPLACE">
```

Defining New Conditions and Actions

A standard XML definition is used to define a condition or action.

Procedure

1. Validate the correctness of the XML definition. You can use the $MQ_HOME/config/rules/RuleModel.dtd schema.

```
<?xml version="1.0" encoding="UTF-8"?>
<Condition ID="RejectedOrChangedAttrGrps" Visible="true">
  <Type>String</Type>
  <PresentationFormat PresentationFormat="the rejected/changed attribute group is ( an attribute group)"/>
  <ConditionOperand>
    <GetOperand>
```

```
        <ClassName ClassName="com.tibco.mdm.integration.ruleengine.IAttributable"/>
          <AttributeName AttributeName="RejectedOrChangedAttrGrps"/>
          <DataSourceType Type="OBJECT"/>
          <DataSourceValue Value="PRODUCT"/>
        </GetOperand>
    </ConditionOperand>
    <BooleanOperator ProgrammingOperator="hasValue"/>
    <ConditionOperand>
        <ParameterOperand>
          <SQLQuery Dependents="MasterCatalog" SQLQuery="select ag.name, ag.description from ATtributegroup
ag where ag.ownerid = -1 #MasterCatalog#  or ag.ownerid = $MasterCatalog$ and ag.active = 'Y'  order by name
#MasterCatalog# "/>
        </ParameterOperand>
    </ConditionOperand>
</Condition>
```

2. Change the following details for a new condition:

Result

- **ID** – Must be unique across all conditions/actions defined under $MQ_HOME/config/rules.
- **Type** – Data type
- **PresentationFormat** – The text that is shown while showing/defining a rule.

> **Note:** The open and close braces in the text, that is, (and) are mandatory, and if you do not have them, the rules don't get saved.

- **AttributeName** – This is the same as the name of the attribute in the workflow 'parameter' definition.
- **DataSourceType** - Different ways to access the data:

  XPATH – Xpath into the mlXML document.

  OBJECT – Data access uses TIBCO MDM defined business objects. Following is the list of objects and their supported DataSource values

  - PRODUCT:

    RejectedOrChangedAttrGrps - Gets the list of rejected or changed attribute groups within the current event.

    ChangedAttrGrps - List of attribute groups that have changed within the current event.

    ConflictAttrGrps - List of attribute groups that have a conflict (during merge).
  - PROFILE

    TimedOutParticipantRoles - Gets the list of roles of users whose work items have timed out.
- DataSourceValue

  For DataSourceType XPATH – the actual XPATH into the mlXML document to fetch the required data.

  For DataSourceType OBJECT – the command word to fetch the required data from the business object.
- **ProgrammingOperator** – The list of supported operators is as follows:
  - equals – valid for all data types.
  - contains – valid for 'String' data types. Checks if a substring exists.
  - greaterThan – valid for all data types.
  - greaterThanEqual – valid for all data types.
  - lessThan – valid for all data types.
  - doesntHaveValue – Checks to see that a particular value is not among the

array of values picked up by the DataSourceValue.

- hasValue – Checks if a value exists in an array of values picked up by the DataSourceValue.
- isEmpty – Checks to see if no value was found.

- **SQLQuery** - This is used by the application to show a drop down list of possible values for a condition/action. In the given example, a list of repositories is shown.

Adding New Domain to an Existing Enterprise

Procedure

1. Navigate to the $MQ_HOME/bin/migration directory.
2. Update addNewRules.xml file with the list of rule domains that must be inserted. (See $MQ_HOME/config/security/defaultdata.xml for sample entries.)
   - 'rulemetamodel' is the same as the name of the domain definition file under $MQ_HOME/config/rules.

     For example, $MQ_HOME/config/rules/NewProductIntroductionEdit.xml.
   - 'active' is set to 'Y.'
   - 'description' is a description of the rule domain.
   - 'name' is the same as the 'RulesetName' attribute in the domain definition file.
   - 'type' is 'BUSINESSPROCESS'.
3. Update orglist.txt. Enter the list of organization IDs to which you want to add the domains (the rule domains listed in the rulelist.xml file).
4. Create a new rule file in $MQ_HOME/config/rules by running the following command:

   ./MigrateRules.sh -addNewRules

   In case of errors or warnings, you can correct them and try again as the script can be re-run without generating duplicates.

```
<?xml version="1.0" encoding="utf-8" ?>
- <mlXML>
- <mq_default table="rulemetamodel">
- <default_row>
  <rulemetamodel>CustomProtocol123</rulemetamodel>
```

```
<active>Y</active>
<memberid id="memberid" />
<description>Rulebase to be used for Custom Protocol123.</description>
<name>Custom Protocol 123</name>
<type>BUSINESSPROCESS</type>
<restrictto>3</restrictto>
<orgid id="orgid" />
</default_row>
</mq_default>
- <mq_default table="rulemodel">
- <default_row>
<rulemetamodel>Custom Protocol123</rulemetamodel>
<description>Default System Template123.</description>
<memberid id="memberid" />
<name>Default</name>
<orgid id="orgid" />
</default_row>
</mq_default>
</mlXML>
```

Adding New Conditions and Actions to Existing Rule Domains

Procedure

1. Create new Conditions and Actions. For more information, see Defining New Conditions and Actions.
2. Go to $MQ_HOME/bin.
3. Run ./MigrateRules.sh or migrateRulesDomain.

Creating New Business Process Templates

Procedure

1. Click **Business Process > Business Processes**.
2. From the **Business Processes** page, click a process name link.
3. From the **Rule Template List** page, select **Create**. Please see the note at the end

of this section if you do not see a create link.

4. In the Rule Template Builder page, enter a template name and description in the appropriate fields.
5. Select the required condition from the **Condition Selection** list and click to add it to the **Selected Conditions** pane. Do the same for actions.
6. Repeat the previous two steps if necessary, and click **Save**.

Allowing Multiple Templates in a Domain

If you create a "default" template for a business process, and cannot create more templates after the first one, use this procedure to create more templates.

Procedure

1. Navigate to $MQ_HOME/config/security.
2. Open the defaultdata.xml file.
3. Search for the process for which you want to create more than one template. For example, "New Product Introduction Edit" can be an appropriate search text. You might find multiple instances of this. For example:

```
<default_row>
 <rulemetamodel>NewProductIntroductionEdit</rulemetamodel>
 <active>Y</active>
 <memberid id="memberid"/>
 <description>Retailer person or role that will edit newly introduced products.</description>
 <name>New Product Introduction Edit</name>
 <type>BUSINESSPROCESS</type>
 <restrictto>1</restrictto>
 <orgid id="orgid"/>
</default_row>
```

4. Look for a parameter called <restrictto>1</restrictto>. Change this number to change the limit for the number of rule templates. If you don’t want to limit the number of rule templates, remove this entry.
5. Do not log out or restart the server to see this change the graphical user interface.

Defining a Custom Workflow Selection

Procedure

1. Make an entry of the workflow in the CONFIGURATIONDEFINITION table. Set the member ID of the organization for which customization is to be done.

   ```
   INSERT INTO CONFIGURATIONDEFINITION(ID, "TYPE", OWNERID, GLOBAL, "NAME",
   SELECTOR, DESCRIPTION, DEFINITIONTYPE, DEFINITION, ACTIVE, MODMEMBERID,
   MODDATE, MODVERSION) VALUES ('<maxID+1>', 'PROCESSNAME', '1', 'Y',
   'wfin26dqcatsourcev1.xml', 'WORKFLOW', 'Process for data source upload and import with
   DQ', 'File', 'standard/workflow/wfin26dqcatsourcev1.xml', 'Y', '1', TO_DATE('15-03-2006
   04:01:00 pm','DD-MM-YYYY HH:MI:SS AM'), '1');
   ```

2. If the DOCTYPE and DOCSUBTYPE are different (not provided out-of-box), make an entry in the DOMAINENTRY table with domaintype as DOCTYPE and DOCSUBTYPE.

   ```
   INSERT INTO DOMAINENTRY(DOMAINTYPE, "VALUE", DESCRIPTION) VALUES
   ('DOCSUBTYPE', '<CustomValue>', '<CustomDescription>');
   INSERT INTO DOMAINENTRY(DOMAINTYPE, "VALUE", DESCRIPTION) VALUES
   ('DOCTYPE', '<CustomValue>', '<CustomDecscription>');
   ```

3. Log in to the application.
4. Click the **Business Processes** link in the menu and select the **Process Definition Selection** rule. If the out-of-box condition is not as per the requirement, add a new condition provided in the templates.
5. Create a rule through the rule creation menu using the DOCTYPE, DOCSUBTYPE, and custom workflow name parameters.

Result

Note: Rules are executed in order of definition. So, a default rule could override 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.

Customization of FileWatcher

TIBCO MDM uses the FileWatcher utility that monitors incoming files containing data for initiation of processes within the application. FileWatcher runs as a background thread in the same JVM as the server.

FileWatcher provides a file-based integration method for some of the workflow operations. After configuration, workflows can be triggered when new files are detected and input files are processed.

FileWatcher polls the directories specified in the configuration file for any new files created there. The polling is done at a configured interval specified in the configuration file. When a new file is detected, it is processed according to the specifications defined in a configuration file.

After the file is successfully processed, it is moved to a /done directory. This prevents the file from being picked up on subsequent polls. FileWatcher has an option to detect duplicate files by the filename. If a previously processed file (name) is copied to /incoming, it is not processed again, and is moved to /Rejected.

Prerequisites

To work with FileWatcher, you must possess the following skills and access privileges:

- Working knowledge of XML
- Working knowledge of TIBCO MDM
- Access to $MQ_HOME/config directory
- Privileges to modify FileWatcher.xml

FileWatcher Configuration File

The FileWatcher configuration file specifies how an input file must be processed, and the directory location for the file after processing is complete.

The location of the FileWatcher configuration file is defined in the Configurator (**Configuration Files** > **File Watcher Configuration File**). The default is $MQ_HOME/config/FileWatcher.xml.

When the application is installed, the default configuration is placed in $MQ_HOME/config/FileWatcher.xml. Although it is not a good practice, you can relocate this file to any other location relative to $MQ_HOME.

When the application is running, and if any change is made to FileWatcher.xml, it is automatically deployed the next time the file is accessed. Although each instance deployed in a cluster can have different FileWatcher configuration, it is a good practice that all servers share the same configuration.

FileWatcher Configuration Parameters

FileWatcher configuration is defined by the Global and DataSet parameters. These parameters control various actions.

- Global Parameters
- DataSet Parameters

Global Parameters

Global parameters are required for all actions. These parameters define:

- How input files are managed.
- How to identify the target enterprise and organization.

You can set the following FileWatcher global parameters:

- Polling Interval
- Directory Names
- In Progress Suffix
- LockFile

Polling Interval

The PollingInterval parameter defines how often FileWatcher checks input directories for new files. The interval is specified in seconds. For faster response time, set the PollingInterval to 60 seconds.

```
<!-- Polling Interval in seconds 3600 = 1 hour -->
<PollingInterval>3600</PollingInterval>
```

Directory Names

FileWatcher works on a set of directories for file processing.

- The directory for incoming files (incoming).
- The directory for successfully uploaded files (done).
- The directory for files that fail (rejected).

These directories are specified in the FileWatcher configuration file.

Parameter	Default Directory Name	Description
InComingDir	incoming	Location where new files appear. These directories are polled at specified intervals.
DoneDir	done	Location where successfully uploaded files appear.
RejectedDir	rejected	Location for failed files.

Each directory can be specified with an absolute path or one relative to the MQ_COMMON_DIR or MQ_HOME or MQ_LOG environment variables. To specify an absolute path, omit the <Relative> tag.

In the following example, the data set is defined, and the new files are placed in the $MQ_COMMON_DIR/EAI/PrimData/incoming directory.

```
<DataSet type="single" >
      <URIInfo scheme="local">
         <Relative>MQ_COMMON_DIR</Relative>
         <URI>EAI/PrimData</URI>
      </URIInfo>
</DataSet>
```

FileWatcher maintains a list of files processed during a run. If the same file is resubmitted for processing, it is rejected. Duplicates are detected by file name.

Setting InProgressSuffix

When large files are transferred to the incoming directory, FileWatcher might detect the file before it is completely written to disk. In this situation, FileWatcher might read the partial file, and the processing might fail. To prevent this, you can use InProgessSuffix to stop processing of the file.

Procedure

1. Set the InProgressSuffix in the configuration file to ignore files that end in this value.
2. When a file is being created, add the suffix to the file name.
3. After the file write is complete, rename the file by removing the suffix.

Result

This procedure is not necessary for small files:

```
<CatalogPoller version="1.0">
<IncomingDir>incoming</IncomingDir>
<DoneDir>done</DoneDir>
<RejectedDir>rejected</RejectedDir>
<InProgressSuffix>.inprogress</InProgressSuffix>
```

For example:

```
datafile123.dat.inprogress - while it is being written
datafile123.dat - rename to this when done.
<InProgressSuffix>.inprogress</InProgressSuffix>
```

In this case, the file is initially named as datafile123.dat.inprogress.

FileWatcher detects the file, but ignores it, because the file ends with the “.inprogress” suffix. After the file is written, it is renamed to datafile123.dat and FileWatcher picks it up during its next polling interval.

LockFile

The LockFile can be specified to synchronize processing of incoming files to ensure that file is processed by only one FileWatcher at a time thereby avoiding race conditions.

```
<LockFile>.lock</LockFile>
```

For more information about how the lock mechanism works for concurrent file loading, see the section Concurrent File Loading.

DataSet Parameters

Each configuration is called a DataSet. One DataSet is set up for each type of processing.

Types of DataSets

You can specify two types of DataSets.

- Single – One file that acts as both the data and trigger files. For example,

```
<DataSet type="single">
 <Name>Purge</Name>
 <Credential domain="ZZ">
   <Identity>GLOBAL</Identity>
 </Credential>
 <Action>Purge</Action>
 <RetentionUOM>MONTH</RetentionUOM>
 <RetentionUnits>6</RetentionUnits>
 <URIInfo scheme="local">
   <Relative>MQ_COMMON_DIR</Relative>
   <URI>/Work/purge</URI>
 </URIInfo>
</DataSet>
```

- Multiple – A multiple data set is a trigger file written to the incoming directory, consisting of a list of file names, and the absolute paths to their locations. Each file name is matched to a datasource in the DataSet and is uploaded from the specified (absolute path) location. The order of the filenames in the trigger file must match with the order of the data sources in the DataSet. Multiple DataSets are not supported for all actions. For the Multiple DataSet type, specify the list value. For

example,

```
<DataSet type="list">
<Name>MBVLoad</Name>
                    <EnterpriseName>k</EnterpriseName>
 <Credential domain="MartQuestNet">
     <Identity>k</Identity>
 </Credential>
  <Action>Load</Action>
  <URIInfo scheme="local">
     <Relative>MQ_COMMON_DIR</Relative>
     <URI>standard/MBVData</URI>
  /URIInfo>
  <DataSource>
      <RevisionID>
      <BaseName>M1</BaseName>
       </RevisionID>
  </DataSource>
  <DataSource>
     <RevisionID>
     <BaseName>M2</BaseName>
     </RevisionID>
     </DataSource>
  </DataSet>
```

Name of DataSet

Each DataSet has a name for informational purposes. This name must be unique.

```
<DataSet type="single">
        <Name>PRIM</Name>
```

Parameters

The DataSet parameters include the following information and define a complete configuration:

- Action Parameters
- Catalog
- CheckDuplicateFile
- ClassificationScheme

- Credential
- DataSource
- Incremental
- InputMap
- MasterCatalog
- ProcessID
- RetentionUOM
- RetentionUnits
- URIInfo

If the default directories are used, do not specify the following parameters in the DataSet configuration:

- Incoming
- Done
- Rejected

Files are copied from the incoming directory to the done directory. Duplicates and failures are placed in the rejected directory. These default values can be overwritten by each DataSet.

Note: Enterprise name must be specified when a DataSet is defined with the martQuestNet DomainType.

Action Parameters

The Action parameter specifies the ways in which a data source can be used. Each action initiates a workflow.

Note: All actions initiated are integrated with the built-in workflow engine.

Parameter	Description	Workflow	Multiple Data sets supported?
ExportRecords	Export data from a repository or subset.	Initiates the Export data workflow.	N
Load	Upload the data source.	Initiates a data source upload workflow.	Y
LoadImport	Upload data sources and import into the repository.	Initiates a data source upload and repository import workflow.	Y
Import	Import records into the repository.	Initiates a repository import workflow.	N
ImportClassificationScheme	Import classification codes, taxonomy, and reclassify products.	Initiates a classification code upload workflow. For a custom classification scheme, this workflow can optionally reclassify the records in a repository.	N
Publish	Initiate a synchronization.	Initiates a catalog synchronization workflow.	N
Validate	Initiate a validation for synchronization.	Initiates a catalog validation workflow.	N
DataServiceUpdate	Initiate metadata import.	Initiates import of metadata workflows.	N
DataServiceQuery	Initiate metadata export.	Initiates export of metadata workflows.	N
Purge	Initiate a data purge workflow. On purge, an email is sent to the address specified in the Company Profile page. However, note that it is necessary to set an email message completion rule for receiving this purge email.	Initiates the Purge workflow. **Note:** Work/Purge directory must be created manually in $MQ_COMMON_DIR.	N

Each action might need different parameters to be specified. For example, if the action is Import, you are required to specify MasterCatalog.

Catalog

The Catalog parameter identifies the name of a catalog for which synchronization is to be initiated. To uniquely identify the Catalog, you must also specify the repository. The Catalog parameters are *only relevant if* the specified action is Publish or Validate.

```
<Catalog>
        <RevisionID>
                <BaseName>MYCATALOG</BaseName>
        </RevisionID>
</Catalog>
```

BaseName specifies the name of an existing catalog. This catalog must already be created.

Note: When the specified action is Publish, the Catalog element must refer to the synchronization profile name.

CheckDuplicateFile

By default, whenever FileWatcher picks up a file, it checks whether the file has already been processed by comparing it against the last 100 files processed. You can disable this check using the CheckDuplicateFile parameter.

```
<CheckDuplicateFile>No</CheckDuplicateFile>
```

If this parameter is used, File Watcher processes the file irrespective of whether the file existed previously or not.

The default value of CheckDuplicateFile is Yes and it checks for duplicate entries in the history file and rejects it if it exists.

Note: This parameter can be specified for each DataSet.

FileWatcher.xml with CheckDuplicateFile

The following sample section is from FileWatcher.xml. In this example, the CheckDuplicateFile parameter has been added to the FileWatcher configuration file.

```
<DataSet type="single">
 <Name>PRIM</Name>
 <Credential domain="GLN">
   <Identity>0065064183212</Identity>
 </Credential>
 <Action>Load</Action>
 <CheckDuplicateFile>No</CheckDuplicateFile>
 <URIInfo scheme="local">
   <Relative>MQ_COMMON_DIR</Relative>
   <URI>EAI/PrimData</URI>
 </URIInfo>
 <DataSource>
   <RevisionID>
     <BaseName>UCCV3</BaseName>
   </RevisionID>
 </DataSource>
</DataSet>
```

ClassificationScheme

The ClassificationScheme parameter identifies the name of the classification scheme.

If no MasterCatalog parameter is specified, the classification scheme name must refer to a standard classification scheme. If a MasterCatalog parameter is specified, the classification scheme name must refer to one of the custom classifications defined for the repository.

```
<ClassificationScheme>
        <RevisionID>
                <BaseName>UDEX</BaseName>
        </RevisionID>
</ClassificationScheme>
```

Credential

The Credential parameter identifies the enterprise that is applicable to a specific configuration.The credential specified in the Data Set must match a credential defined in the Company Profile page.

This parameter is required for all actions. The credentials specified in FileWatcher and JMS messages are case insensitive. For more information about case sensitivity, see the Login section in the *TIBCO MDM User's Guide*.

```
<DataSet type="single" >
        <Credential domain="GLN">
            <Identity>0065064183212</Identity>
        </Credential>
```

DataSource

The DataSource parameter specifies the name of the data source to use. To specify a unique data source, use the data source name in conjunction with the Organization name specified in the Credential parameter.

The data source name is used in conjunction with Credentials to identify a data source defined in the application.

```
<DataSource>
<    RevisionID>
<BaseName>SupplierV3</BaseName>
</RevisionID>
</DataSource>
```

BaseName specifies the name of the existing data source.

Note: This data source must be defined using the application user interface.

Incremental

The Incremental parameter is applicable only for the ImportClassificationScheme action.

For ImportClassificationScheme, this is the only mode currently supported, and it supports incremental loads of the classification code as follows:

- new classification codes are imported.
- existing classification codes are updated.

Note: Existing classification codes cannot be deleted.

```
<Incremental>Yes</Incremental>
```

InputMap

The InputMap parameter identifies the Input map to be used for data import. It is applicable for Import and LoadImport actions. If the Input map name is not specified, then Input map name ‘DEFAULT ‘is assumed. You must specify which Input Map must be used.

```
<InputMap>
                <RevisionID>
                    <BaseName>Inputmap</BaseName>
                </RevisionID>
            </InputMap>
```

MasterCatalog

The MasterCatalog parameter specifies the name of the repository. The repository name must be used in conjunction with the Organization specified by the Credential to specify a unique repository to use. The MasterCatalog parameter is *only relevant* if the specified **Action** is one of the following:

- LoadImport
- Load
- Publish
- Purge
- Validate
- ImportClassification
- ExportRecords

```
<MasterCatalog>
```

```
        <RevisionID>
                <BaseName>SupplierV3</BaseName>
        </RevisionID>
</MasterCatalog>
```

BaseName specifies the name of an existing repository.

ProcessID

Using the ProcessID parameter for a data set, you can specify a workflow name in the FileWatcher.xml file. This is an optional element.

For example, you can set up two different data sets, each one importing to the Customer repository. However, the import requirement for each source can be different and a different workflow is needed. In such a case, you can specify the workflow in the data set definition itself. When the FileWatcher fires a workflow, it takes the specified workflow and skips the workflow selection using business process rules.

Note: This feature supports all import modes except Database Loader because it does not use any workflow in the import operation. The supported import modes are Direct Load, Split/Approval Required, and Split/No Approval.

Example

```
<DataSet type="single">
  <Name>ENT2 Import</Name>
  <Credential domain="GLN">
  <Identity>0065064183212</Identity>
  </Credential>
  <Action>DataServiceUpdate</Action>
  <URIInfo scheme="local">
    <Relative>MQ_COMMON_DIR</Relative>
    <URI>fwtest/DataForImportdata</URI>
  </URIInfo>
  <ProcessID>fwtest/workflow/wfin26dataservicev2</ProcessID>
</DataSet>
```

Note: Even though you specify the workflow name, you must deploy the workflow through TIBCO MDM Studio.

RetentionUOM

The RetentionUOM parameter specifies the units of measure for RetentionUnits. This parameter is applicable *only if* the action specified is Purge. The valid values are MONTH (default) and DAY.

RetentionUnits

The RetentionUnits parameter specifies the number of days or months to purge. It is applicable *only if* the action specified is Purge. Any integer more than 0 is valid. The Default is 6.

URIInfo

The URIInfo parameter specifies the location of the root directory for a DataSet. It also specifies how the information is encoded. Only local schemes are currently implemented.

This parameter is required for all actions.

Parameter	Description
Relative	An environment variable name. Valid options are MQ_HOME, MQ_COMMON_DIR and MQ_LOG.
URI	Directory path appended to the Relative environment variable.

The Relative parameter is an environment variable associated with URIInfo. The URI information can be added following the Relative Parameter.

In the following example, the Relative parameter specifies the *TIBCO MDM*environment variable MQ_COMMON_DIR, and appends to the EAI/PrimData directory. Assuming MQ_

COMMON_DIR translates to /usr/local/velosel, the final root directory would be: /usr/local/velosel/EAI/PrimData.

```
<DataSet type="single" >
        <URIInfo scheme="local">
            <Relative>MQ_COMMON_DIR</Relative>
            <URI>EAI/PrimData</URI>
        </URIInfo>
```

The History file is stored in the root directory. If the incoming directory is not specified, the default directory is used. If the incoming directory is specified as the default, input files are copied to: /usr/local/velosel/EAI/PrimData/incoming.

The URIInfo parameter describes the location of the file. It specifies the relative path, the enterprise directory, and the directory where the FileWatcher directory structure is created.

For example:

```
<URIInfo>
  <Relative>MQ_COMMON_DIR</Relative>
  <URI>internal-directory/SUPP/FILEWATCHER/Import</URI>
</URIInfo>
```

To implement the URIInfo in the example above, ensure that the following directory structure exists under the enterprise internal name directory:

```
SUPP
FILEWATCHER
```

Export

done

incoming

rejected

Import

done

incoming

rejected

Subset must be included. It applies only for export records. Subset name is unique for repository.

Actions

The actions include:

- ExportRecords Action
- Load Action
- LoadImport Action
- Import Action
- DataServiceQuery and DataServiceUpdate Actions
- ImportClassificationScheme Action
- Purge Action
- Publish and Validate Actions

ExportRecords Action

The ExportRecords action is used to export data from a repository. Either all the records in the repository can be exported or records can be selected using a subset. The default workflow extracts all records related to selected records, including records in other repositories.

If only specific relationships are to be extracted, the workflow must be modified to specify the relationship name. The relationships to be extracted must be specified in the workflow as follows:

```
<Parameter direction="in" name="RelationshipName" type="string"
eval="constant">Rel6</Parameter>
```

If no related records are to be extracted, specify a non-existing relationship as follows:

```
<Parameter direction="in" name="RelationshipName" type="string" eval="constant">NON_
EXISTANT_RELATIONSHIP_X</Parameter>
```

Any number of relationships can be specified using multiple parameters (*RelationshipName1*, *RelationshipName2*, and so on).

You can also control whether the latest (last confirmed or unconfirmed) version is to be exported or only the last confirmed version is to be exported. This is done by specifying VersionOption as follows:

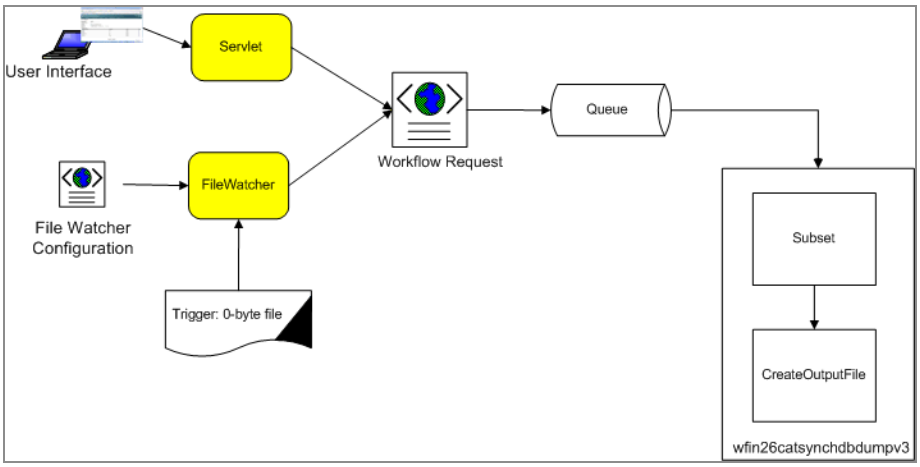
```
<Parameter direction="in" name="VersionOption" type="string"
eval="constant">LATEST</Parameter>
```

The default is LATEST, that is, the last confirmed or unconfirmed version is exported.

To initiate the export, add a 0-byte file to the directory pointed to by incoming folder of URI in the FileWatcher data set definition.

The output is a zip file containing the following:

- One data export file per repository.
- One relationship file containing record relationship information.
- If related records are extracted, related relationships are also codified in the CONTAINS attribute which can be imported back into the application.
- One index file containing information of the relationship file and the data export files.



ExportRecords Parameters

Parameter	Description	Valid Values	Mandatory
MasterCatalog	Name of the repository.	Any existing repository name	Y
Subset	If a subset name is specified using this parameter, only records qualifying the subset are exported. If a subset is not specified, all records from the repository are exported. **Note:** To specify a subset, only one repository must be specified.	Any existing subset name	N

```
<DataSet type="single">
 <Name>ExportRepository</Name>
 <URIInfo scheme="local">
    <Relative>MQ_COMMON_DIR</Relative>
 <URI>/ent1/export</URI>
 <Credential domain="ZZ">
    <Identity>GLOBAL</Identity>
 </Credential>
<Action>ExportRecords</Action>
        <MasterCatalog>
            <RevisionID>
                <BaseName>PRIMDATA</BaseName>
            </RevisionID>
        </MasterCatalog>
            <Subset>
            <RevisionID>                                    <BaseName>MYSET</BaseName>
            </RevisionID>
            </Subset>
</DataSet>
```

Load Action

The Load action is used to initiate a datasource upload.

Parameter	Description	Mandatory	Default
DataSource	Name of the datasource to upload.	Y	-

```
<DataSet type="single" >
 <Name>PRIM</Name>
 <Credential domain="GLN">
   <Identity>0065064183212</Identity>
 </Credential>
 <Action>Load</Action>
 <URIInfo scheme="local">
    <Relative>MQ_COMMON_DIR</Relative>
    <URI>EAI/PrimData</URI>
 </URIInfo>
 <DataSource>
    <RevisionID>
       <BaseName>V3</BaseName>
    </RevisionID>
 </DataSource>
</DataSet>
```

LoadImport Action

The LoadImport action is used to initiate a datasource upload and import into the repository. This allows for automated data feeds into the application, when data change is initiated in other systems.

To initiate the LoadImport workflow event and to spawn the **catalog record add** event through FileWatcher, put at least one file in the $Relative/URI/incoming directory. This file must have the records that must be imported.

Another requirement for LoadImport is that the <Datasource> parameter must be specified.

FileWatcher polls for a file in the incoming directory and then processes the DataSets. For LoadImport, a file containing the records to be imported is required in the incoming directory. The file is moved to the done or rejected folder after the action is completed.

Prerequisites

Create the repository.

Create the datasource. see the Load Action section for more details on the format of the data source.

Create the input map between the datasource and the repository.

To initiate the LoadImport event:

Set the FileWatcher configuration file. Include the DataSource parameter in the DataSet.

Place any file in the $Relative/URI/incoming folder and the Import event is initiated.

Data Transfer Action

A new sub action DataTransfer has been added under LoadImport Action to initiate Data Transfer through FileWatcher. For this, the Export Archive must be put in the configured incoming directory.

The only prerequisite for this action is that repositories metadata must be present in the target Enterprise.

This action can be specified as follows, after <URIInfo>

```
<ImportAction>DataTransfer</ImportAction>
```

No other parameter such as MasterCatalog, DataSource, InputMap etc. are needed in this special case of LoadImport Action.

LoadImport Action Parameters

Parameter	Description	Valid Values	Mandatory	Default
DataSource	Name of the datasource to upload.	Any existing datasource name	Y	-
MasterCatalog	Name of the	Any existing	Y	-

Parameter	Description	Valid Values	Mandatory	Default
	repository.	repository name		
Incremental	Indicates if the import must be incremental.	Yes, No	N	Yes
InputMap	Specify the name of input map for data import	Any existing input map within MasterCatalog specified above	N	DEFAULT
ValidateOnly	Validates the records that are being importing without saving those records, and generates an error report and a log file.	true or false	N	false
ConfirmImmediately	If specified true, the ImportCatalogRecords activity confirms the imported records immediately.	true or false	N	false
ImportAction	Initiates Data Transfer	Data Transfer	Y	-

```
<DataSet type="single">
        <Name>PRIM</Name>
        <Credential domain="GLN">
            <Identity>0065064183212</Identity>
        </Credential>
        <Action>LoadImport</Action>
        <URIInfo scheme="local">
            <Relative>MQ_COMMON_DIR</Relative>
            <URI>EAI/PrimData</URI>
```

```
        </URIInfo>
        <DataSource>
            <RevisionID>
                <BaseName>UCCV3</BaseName>
            </RevisionID>
        </DataSource>
        <MasterCatalog>
            <RevisionID>
                <BaseName>PRIMDATA</BaseName>
            </RevisionID>
        </MasterCatalog>
        <InputMap>
            <RevisionID>
                <BaseName>MyInputMap</BaseName>
            </RevisionID>
        </InputMap>
        <ValidateOnly>true</ValidateOnly>
    </DataSet>
```

DataSet sample for Data Transfer

```
<DataSet type="single">
<Name>data-transfer</Name>
     <EnterpriseName>target</EnterpriseName>
 <Credential domain="MartQuestNet">
    <Identity>admin</Identity>
 </Credential>
    <Action>LoadImport</Action>
    <URIInfo scheme="local">
    <Relative>MQ_COMMON_DIR</Relative>
    <URI>target/datatransfer</URI>
    </URIInfo>
    <ImportAction>DataTransfer</ImportAction>
</DataSet>
```

Import Action

Import action is useful when there is an already uploaded Data Source to import.

To initiate the Import workflow event and to spawn the **catalog record add** event through FileWatcher, put at least one file in the $Relative/URI/incoming directory. This file can be empty.

FileWatcher polls for a file in the incoming directory and then processes the DataSets. If the <Action> tag requires the usage of the file (as in Load/LoadImport), the file is used. Otherwise, the file is not used. The file is moved to the folder specified in the <Done> or <Rejected> tag after the action is completed.

Note: The ImportCatalog activity is not supported for in-memory workflows.

Prerequisites

Create the repository.

Create the datasource.

Upload the data into the datasource.

Create the input map between the datasource and the repository.

Initiating the Import Event

Procedure

1. Set the FileWatcher configuration file with the DataSet as described above.
2. Place any file in the $Relative/URI/incoming folder and the Import event is initiated.

Note: No security is applied when import is initiated from FileWatcher. However, if import permissions are denied, you cannot import records from the user interface.

Import Action Parameters

Parameter	Description	Valid values	Mandatory	Default
MasterCatalog	The repository name.	Any existing	Y	-

Parameter	Description	Valid values	Mandatory	Default
		repository name		
Incremental	Indicates if the import must be incremental.	Yes, No	N	Yes
InputMap	Specify the name of input map for data import.	Any existing input map name.	N	DEFAULT
ValidateOnly	Validates the records that are being importing without saving those records, and generates an error report and a log file.	true or false	N	false
ConfirmImmediately	If specified true, the ImportCatalogRecords activity confirms the imported records immediately.	frue or false	N	false

Note: No security is applied when import is initiated from FileWatcher. However, if import permissions are denied, you cannot import records from the user interface.

```
<DataSet type="single" >
   <Name>PRIM</Name>
   <Credential domain="GLN">
      <Identity>0065064183212</Identity>
   </Credential>
   <Action>Import</Action>
   <URIInfo scheme="local">
     <Relative>MQ_COMMON_DIR</Relative>
```

```
      <URI>EAI/PrimData</URI>
    </URIInfo>
    <Incremental>Yes</Incremental>
    <MasterCatalog>
      <RevisionID>
        <BaseName>PRIMDATA</BaseName>
      </RevisionID>
    </MasterCatalog>
    <InputMap>
      <RevisionID>
        <BaseName>DEFAULT</BaseName>
      </RevisionID>
     </InputMap>
     <ValidateOnly>true</ValidateOnly>
</DataSet>
```

DataServiceQuery and DataServiceUpdate Actions

These actions initiate workflows for export and import of metadata. All the input parameters are specified in the input file itself and no other parameters must be specified.

No specific parameters are required.

```
<DataSet type="single">
    <Name>ENT1 Import</Name>
    <Credential domain="GLN">
      <Identity>0040885020007</Identity>
    </Credential>
    <Action>DataServiceUpdate</Action>
    <URIInfo scheme="local">
       <Relative>MQ_COMMON_DIR</Relative>
       <URI>/ent1/metadata/import</URI>
    </URIInfo>
</DataSet>
<DataSet type="single">
    <Name>ENT1 Export</Name>
    <Credential domain="GLN">
       <Identity>0040885020007</Identity>
    </Credential>
    <Action>DataServiceQuery</Action>
    <URIInfo scheme="local">
      <Relative>MQ_COMMON_DIR</Relative>
      <URI>/ent1/metadata/export</URI>
```

```
    </URIInfo>
</DataSet>
```

ImportClassificationScheme Action

This action is used to import classification codes for an existing classification scheme. Currently, only an incremental mode is supported:

- Existing codes descriptions are updated
- New codes are added

If the classification scheme is a custom classification scheme, existing records are reclassified.

Note: The ImportClassificationScheme and ReclassifyRecord activities are not supported for in-memory workflows.

The classification upload is based on the data source load. The input file must have the following format and the number of levels must match with the number of classification code levels defined for the classification scheme. The file format must be one of the supported formats for data source upload. When you want to set up the classification loads, ensure to test the file upload using a data source.

ImportClassificationScheme Action Parameters

Level1	Level1 Description	Level2	Level2 Description	Level3	Level3 Description
Food	Food products	Produce	Farm products	Fresh	Fresh Produce
Food	Food products	Produce	Farm products	Frozen	Frozen products
Food	Food	Processed	Processed	Canned	Canned

Level1	Level1 Description	Level2	Level2 Description	Level3	Level3 Description
	products		food		products
Food	Food products	Processed	Processed food	Packaged	Packaged products

The attributes (column headers in the input file) can be any name as long as the file format matches with the format described above. Note that higher level codes repeat for different values of low level codes.

Parameter	Description	Valid Values	Mandatory	Default
DataSource	Name of the data source to upload.	Any existing data source name.	Y	-
MasterCatalog	The repository name. Required if the classification scheme is not a pre-defined classification scheme.	Any existing repository name.	N	-
ClassificationScheme	Name of the classification scheme.	Any existing pre-defined classification scheme or a custom scheme for the repository.	Y	-
Incremental	Indicates if the import must be incremental.	Yes, No	N	Yes

Parameter	Description	Valid Values	Mandatory	Default
	Currently only 'Yes' is supported.			

```
<DataSet type="single" >
            <Name>PRIM</Name>
            <Credential domain="GLN">
                <Identity>0065064183212</Identity>
            </Credential>
            <Action>ImportClassificationScheme</Action>
            <URIInfo scheme="local">
                <Relative>MQ_COMMON_DIR</Relative>
                <URI>EAI/PrimData</URI>
            </URIInfo>
            <DataSource>
                <RevisionID>
                    <BaseName>UCCV3</BaseName>
                </RevisionID>
            </DataSource>
            <Incremental>Yes</Incremental>
            <MasterCatalog>
                 <RevisionID>
                    <BaseName>PRIMDATA</BaseName>
                </RevisionID>
            </MasterCatalog>
             <ClassificationScheme>
              <RevisionID>
                <BaseName>UDEX</BaseName>
              </RevisionID>
            </ClassificationScheme>
        </DataSet>
```

Purge Action

This action is used to initiate a data purge workflow. On purge, an email is sent to the address specified in the Company Profile page. However, note that it is necessary to set an email message completion rule for receiving this purge email.

The following directory structure must be created under $MQ_COMMON_DIR/Work/purge:

```
done
incoming
rejected
```

For Purge to start, you must add an empty file in the incoming directory.

Note: The Purge activity is not supported for in-memory workflows.

Purge Action Parameters

Parameter	Description	Valid Values	Mandatory	Default
DeleteRecordVersions	Specifies whether records must be deleted. If this parameter is set to Yes, old record versions are deleted. **Note:** To delete record versions, you must specify DeleteRecordVersions as Y in File Watcher and pass the DeleteRecordVersions parameter in the workflow.	Y Yes N No	No	No
EnterpriseName	Specifies the enterprise to be purged. The credential specified must belong to the enterprise specified. If this parameter is not specified, all enterprises are purged.	ALL (Default) Any enterprise name in the application.	No	ALL
MasterCatalog	Specifies the repositories from which record versions and historical data is to be purged. **Note:** If one or more repositories are specified, purge is limited to history and data of these catalogs only and also restricted to the current enterprise. Enterprise name is required if the repository name is specified. All repositories must belong to ONE enterprise only.	Any valid repository name.	No	Default is ALL catalogs.
RetentionUnits	Specifies the number of days or months from the current date beyond which data is purged. The cutoff date must be greater than the current date, that is, RetentionUnits must be greater than 0.	Number greater than 0.	No	6
RetentionUOM	Specifies the unit of measure for RetentionUnits.	MONTH DAY	No	MONTH
RecordsProcessed	The number of records purged during the run. Does not include records which are automatically removed due to database referential integrity constraints.			

The parameter, EnterpriseName, does not work with the default DataSet. To specify the enterprise name or a repository of an enterprise in the DataSet, you must specify a DataSet similar to the following example:

```
<DataSet type="single">
   <Name>Purge</Name>
   <Credential domain="GLN">
      <Identity>0065064444443</Identity>
   </Credential>
   <Action>Purge</Action>
```

```
    <RetentionUOM>DAY</RetentionUOM>
    <RetentionUnits>1</RetentionUnits>
    <EnterpriseName>CorpA</EnterpriseName>
    <MasterCatalog>
        <RevisionID>
            <BaseName>Precedence</BaseName>
        </RevisionID>
    </MasterCatalog>
    <URIInfo scheme="local">
        <Relative>MQ_COMMON_DIR</Relative>
        <URI>/Work/purge</URI>
    </URIInfo>
</DataSet>
```

Change the values of the following elements as per your requirement:

- Identity: specify the GLN as specified in Company Profile of your setup.
- EnterpriseName: specify the name of the enterprise whose data you want to purge.
- MasterCatalog: specify the name of the repository whose data you want to purge.

Advanced Purge

The AdvancedPurge activity purges all historical data from the system (except data involved in BCT tables) and a list of files to be removed is generated.

For more details about configuring Advanced purge and how it works, see the *TIBCO MDM System Administrator's Guide*.

For more details about the parameters and usage rules of the AdvancedPurge activity, see the Activities chapter in the *TIBCO MDM Workflow Reference.*

Usage Guidelines

Some usage guidelines for the AdvancedPurge activity are:

- Use the AdvancedPurge activity when the focus is on cleaning the data quickly rather than precisely.
- Use AdvancedPurge in situations where you want more control on the purge process. Since the AdvancedPurge activity uses Oracle's stored procedure, you can customize the procedure to change the purge process easily. The standard Purge activity is not easily customizable like the AdvancedPurge activity.
- AdvancedPurge might purge some of the GeneralDocument entries which a ProcessLog might be referring to. So, if accuracy of purging is of high importance, do not use AdvancedPurge.

Workflow Customization

You can customize the Oracle stored procedure used by the AdvancedPurge activity. The script, Create_Purge_Package.sql, used to create the procedure is supplied as a sample in the $MQ_HOME\common\standard\samples\scripts directory.

You can also customize the Java class, com.tibco.mdm.workflow.engine.activities.FilesAndDirectoriesCleaner, which purges the Work directory content and Temp directory content. This class is used in the PurgeFilesThroughShellScript and PurgeTempFiles workflow activities. A sample Java class, FilesAndDirectoriesCleaner.java, is located in the $MQ_HOME\common\standard\samples\workflow directory.

Invoking the Activity From the Standard Purge Workflow

The isAdvanced parameter of the UpdateEvent activity is commented by default. To invoke the AdvancedPurge activity from the workflow, uncomment the isAdvanced parameter.

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

Publish and Validate Actions

These actions initiate a synchronization workflow for the specified catalog. Specify an existing catalog name. This works exactly the same as synchronizations initiated from the user interface. These actions do not require any data to be present in the input file. The input file simply acts as a trigger and could be 0 bytes. The most common implementation is to generate a 0-byte file by using the UNIX command ‘touch’. Ensure the file name is unique. This trigger file must be created in the directory identified by the URI.

When the action is specified as Publish, the Catalog element must refer to the synchronization profile name.

Publish and Validate Actions Parameters

Parameter	Description	Valid Values	Mandatory	Default
MasterCatalog	The repository name.	Any existing repository name.	Y	-
Catalog	Name.	Any existing catalog name.	Y	-
Action	The action to be taken.	PUBLISH or VALIDATE	Y	-

```
<DataSet type="single" >
   <Name>PRIM</Name>
   <Credential domain="GLN">
      <Identity>0065064183212</Identity>
   </Credential>
   <Action>PUBLISH</Action>
   <URIInfo scheme="local">
      <Relative>MQ_COMMON_DIR</Relative>
      <URI>EAI/PrimData</URI>
   </URIInfo>
   <MasterCatalog>
      <RevisionID>
         <BaseName>PRIMDATA</BaseName>
      </RevisionID>
   </MasterCatalog>
   <Catalog>
      <RevisionID>
         <BaseName>MYCATALOG</BaseName>
      </RevisionID>
   </Catalog>
</DataSet>
```

About Using FileWatcher

FileWatcher is a timer-based process that starts up automatically when TIBCO MDM is started. During the initialization process, it checks the **Credential** of every DataSet.

Refer the following to know the operations of FileWatcher and the procedure for controlling the operations:

- Start FileWatcher
- Disable FileWatcher
- Re-Initialize the Configuration File
- Change the FileWatcher.xml file
- Duplicate Files
- Concurrent File Loading
- Import Data using FileWatcher
- DataServiceUpdate Actions

- Error Reports
- Processing
- Archiving Files
- Move Files
- Configure FileWatcher to Import or Export Metadata

Start FileWatcher

If a Credential does not exist, FileWatcher does not start. This is the most common configuration error for the FileWatcher. The application startup fails if FileWatcher fails.

Note:

- If the FileWatcher configuration file has errors and is invalid, the application might not start. it is advisable to ensure that the configuration file is a valid XML and can be validated using the schema supplied in $MQ_HOME/schema/config/filewatcher/1.1/FileWatcher.xsd.
- If the configuration file points to a non-existent directory or URI, FileWatcher initiation does not fail. If the directory pointed to by the URI does not exist, FileWatcher attempts to create it.

Disable FileWatcher

FileWatcher can be disabled by either changing the polling interval to be very high or by removing the FileWatcher from the initialization class list.

Re-Initialize the Configuration File

The FileWatcher process activates after specific intervals specified in the PollingInterval.

Before processing configurations, it first checks the FileWatcher.xml configuration file for any changes (The PollingInterval is specified in FileWatcher.xml). If there are changes, the file is re-read, and all parameters are re-initialized. The changed polling interval is effective for the next polling.

Change the FileWatcher.xml file

Make the necessary changes and wait for the next PollingInterval to refresh the configuration. If the PollingInterval is excessively long, you can restart TIBCO MDM to force FileWatcher to re-initialize.

Duplicate Files

FileWatcher does not load any file it has previously processed. It maintains a history of the last 100 files encountered, and checks each new file against the history list. If a duplicate file is detected, an email with the error message is sent to the administrator, and the duplicate file is moved to the rejected directory.

Concurrent File Loading

In a clustered environment, if more than one FileWatcher attempts to process an incoming file, it would lead to concurrency issues and possible data corruption. This is addressed by the lock mechanism which establishes a semaphore by creating a lock file for the DataSet and releasing the lock once the processing is done.

Before processing an incoming file, FileWatcher attempts to acquire a lock on the lock file specified as a part of DataSet parameters. To acquire a lock, it attempts to create a file with the DataSet name suffixed with name specified in the LockFile parameter (<DataSet name>_<LockFile>). If the lock is acquired for the specified DataSet, FileWatcher processes the incoming files for that DataSet and deletes the lock file after the processing is done.

If the LockFile for a DataSet already exists and lock cannot be established, the FileWatcher skips the DataSet processing and works on any other DataSets that have been configured.

The lock file is created in the <Relative>/<URI> path provided in the URIInfo section of the DataSet. It contains information about the instance which acquired the lock so that the abandoned lock can be cleaned up by the correct instance. The instance ID used in the lock file is given by the JVM argument NODE_ID.

A LockFile can be specified for each DataSet. If no LockFile parameter is specified in the DataSet information, the global LockFile parameter is used to determine lock file extension.

Import Data using FileWatcher

Data is imported into the catalog by defining data sources and mapping the data sources into a repository. New data is imported through the user interface (UI); you upload a data source, and then import it into the repository.

FileWatcher expedites importing data by polling for new files, and uploading them automatically. These files typically contain master data, which can be uploaded into a data source and then imported into the repository.

Users can initiate imports either manually through the TIBCO MDM user interface (UI), or via FileWatcher in an automated lights-out manner. When Import Workflow is initiated, a draft is created for each imported record.

While a newly uploaded file is being processed, the file ending is changed to the ending defined in the InProgressSuffix, to indicate that the file must not be processed by any other program. When a file is successfully processed it is moved into the 'done' directory. If the processing fails, the file is moved into the 'rejected' directory to facilitate reprocessing after the error condition has been resolved.

One example for a processing failure is when you try to upload a file with the same file name as a previously loaded file. The application assumes that the same file name indicates the same data and prohibits this scenario. For regular uploads it is advisable to devise a file naming convention which ensures uniqueness of a file name. For example, date and time could be a part of the file name.

If the same file is being attempted to be processed, FileWatcher accesses the previously processed file names (which are in a file named .history) and determines whether the file has been loaded for this DataSet before.

When one or more files are added for upload, one of the application cluster instances acquires a lock in the root directory of the DataSet. It creates a file with the file name specified in the LockFile property of the respective DataSet configuration. This file lock contains information about the instance which acquired the lock, so that an abandoned lock can be cleaned up by the correct instance. The instance ID used in the lock file is given by the JVM argument NODE_ID, which must be different for each member of the cluster.

The instance which successfully acquired the lock accesses all the files in the directory and moves them into the done or rejected directory depending on the processing outcome.

Only after the instance has finished processing the set of files it found while initiating processing, it releases the file lock. The next instance can then acquire the lock and process newly arrived files.

The files are processed in order of the arrival date (last modification timestamp) on the file system.

A data import through FileWatcher can lead to many simultaneous uploads if many data files have been added. To avoid parallel imports the engine allows only one import workflow on the same repository, data source or input map to proceed at the same time. The other simultaneous processes are queued for later import. When the first process exits workflow processing the queue is checked and if not empty the next import starts the import process.

DataServiceUpdate Actions

Metadata from the system is exported in the form of JAR, ZIP, or XML files. Such export files are used as input to the FileWatcher 'DataServiceUpdate' action.

While processing DataSerivceUpdate, FileWatcher extracts the contents of the file into a directory such as:

MQ_COMMON_DIR/Work/year/month/day/hour/unique_dir_name (where unique_dir_name is a directory created by the system with unique ID).

Sample JAR File Entries

The following are the contents of a sample JAR file to be used as an input to the DataServiceUpdate action. At the root of the JAR is the command request-response.xml file. It also contains two attachments referred by the data.csv and DataCustodian.xml command files.

```
request-response.xml$MQ_COMMON_DIR/Work/2004/Nov/19/16/55224-0888234/data.csv$MQ_
HOME/config/rules/DataCustodian.xml
```

These entries might be extracted into the following directories:

```
$MQ_COMMON_DIR/Work/2004/Nov/19/16/1MDNU41Q3G8TJAJT/request-response.xml$MQ_COMMON_
DIR/Work/2004/Nov/19/16/1MDNU41Q3G8TJAJT /commondir/Work/2004/Nov/19/16/55224-0888234/data.csv$MQ_
COMMON_DIR/Work/2004/Nov/19/16/1MDNU41Q3G8TJAJT/mqhome/config/rules/DataCustodian.xml
```

Error Reports

When an error occurs (including credential/validation errors), FileWatcher sends a message to the address, specified in the Configurator (**Email** > **Email Receiver Address**).

This email address must be routed to someone who accesses his or her mail on a regular basis.

Note that the errors reported to this email address are technical errors, that is, configuration errors. This does not include emails generated by initiated workflows.

If the Application does not start, an error exists in FileWatcher initialization. All DataSets and users referred to in the configuration must exist.

Processing

When FileWatcher loads the configuration, it parses the configuration, and ensures that it is a valid XML file.

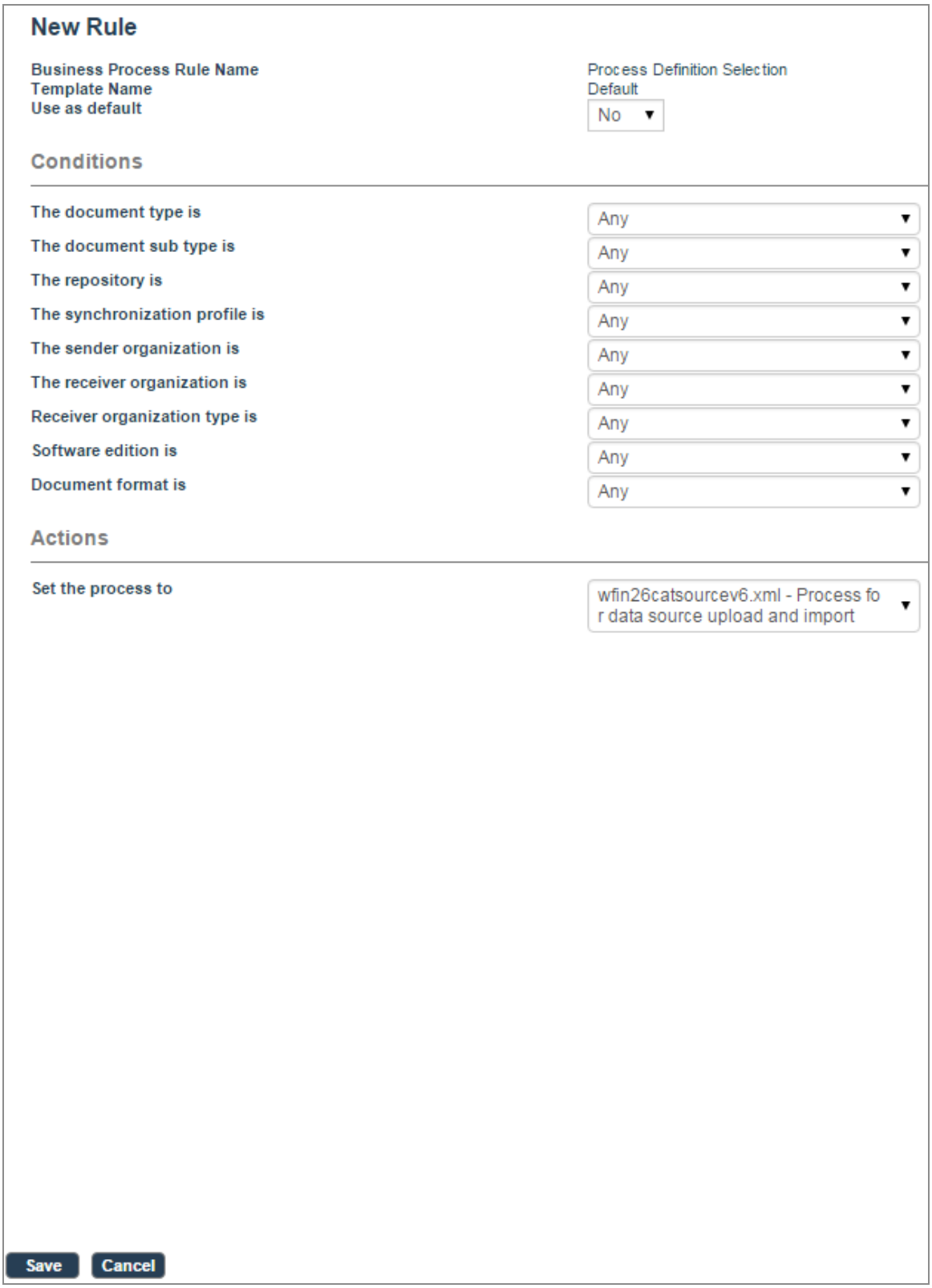
It then scans each DataSet configuration and verifies that:

- Directories specified for present and valid
- Credentials are valid

When a file is detected, FileWatcher constructs a workflow request event, and submits for processing.

The workflows requested by FileWatcher are treated in the same way as the ones initiated by other input channels, that is, the UI.

Normal workflow selection and routing methods are applicable. For example, using the process definition selection in Business Process Rule (BPR), you can create a new rule that contains the following logic:


New Rule
Business Process Rule Name
Template Name
Use as default
Process Definition Selection
Default
No
Conditions
The document type is
Any
The document sub type is
Any
The repository is
Any
The synchronization profile is
Any
The sender organization is
Any
The receiver organization is
Any
Receiver organization type is
Any
Software edition is
Any
Document format is
Any
Actions
Set the process to
wfin26catsourcev6.xml - Process for data source upload and import
Save
Cancel

All data source upload events are routed to the wfin26catsourcev5.xml workflow file. After the Event is sent, FileWatcher moves the file to the done directory, and considers the job completed.

Archiving Files

The done and rejected directories accumulate files over time, and must be cleaned out on a regular basis. Files are copied to these directories after processing. They can be deleted, because the original files exist in their directories of origin.

Purge these directories:

Procedure

1. Back up all directories.
2. Delete all files (older than 30 days) in the done and rejected directories.

Result

When using a multiple DataSet, archive your data files. The location of your archive is at your discretion.

Move Files

For a DataSet of type multiple, the following happens if a file is moved to a directory, assuming that a data directory holds all of the files. The files can be stored anywhere on the network that is accessible to FileWatcher.

Normal File Processing – Using Multiple DataSet

A comma separated file is copied into the data directory.

```
$MQ_COMMOND_DIR/catalogupload/prim
data
mcdata_20021107135621.csv
/incoming
done
rejected
```

When copying is complete, write the trigger file:

```
/data
mcdata_20021107135621.csv
/incoming
list_20021107135621.txt
/done
/rejected
```

- If your file is large, set up the inprogress flag, and name the file as list_20021107135621.txt.inprogress. After the file is completely written, rename it to list_20021107135621.txt.
- The contents of the list_20021107135621.txt file must be: /velosel/commondir/catalogupload/prim/data/mcdata_20021107135621.csv

Watchdog wakes up and polls the incoming directory. It finds the list_20021107135621.txt file in the incoming directory. It processes the file, and then moves it to the done directory.

In the following example, the data file is not moved, because it does not trigger the Watchdog.

```
/data
    mcdata_20021107135621.csv
                /incoming
                /done
                list_20021107135621.txt
                /rejected
```

The file has been successfully processed.

Duplicate File Processing

When copying is complete, write the trigger file:

```
        /data
            mcdata_20021107135621.csv
            /incoming
            list_20021107135621.txt
            /done
        /rejected
```

TIBCO MDM remembers the last 100 files that were processed, and does not allow the same file to be processed twice.

Watchdog moves the file into the rejected directory:

```
/data
    mcdata_20021107135621.csv
    /incoming
    /done
    /rejected
    list_20021107135621.txt
```

If the system administrator decides to process this file, it is sufficient to copy the file back into the incoming directory under a different name. In this example, a -2 suffix is added (symbolizing version 2). The file processes normally.

```
 /data
mcdata_20021107135621.csv
/incoming
list_20021107135621-2.txt
/done
/rejected
list_20021107135621.txt
```

Configure FileWatcher to Import or Export Metadata

Using copy metadata you can export or import metadata of business objects such as Catalog, CatalogType, and ClassificationScheme.

Objects that can be exported or imported are:

- Components/Objects
- SubComponents/SubObject (implicit of Objects)
- Repository
    - Attributes
    - Attributes Group
    - Inputmap
    - Output map
    - Classification Schemes
    - RelationShip Definition

- BusinessProcess Rules
    - RuleMetaModel
    - RuleMode
    - Rules/RuleInstances
- CatalogFormat
    - Attributes
    - Attributes Group
- Data sources
    - Attributes
    - Attributes Group
    - Data Source File
- OutputMap
    - Attributes
    - Attributes Group
- CatalogInputMap
    - Attributes
    - Attributes Group

Exporting Metadata Sample

You can export the metadata from the source enterprise, for any given object within the request XML.

Here is a sample XML:

```
<DataService>
        <Transaction>
```

```
<Command type="Query">
<MasterCatalog etype="Entity">
<ExternalKeys>
<Key name="name" type="string">CUSTOMER</Key>
</ExternalKeys>
</MasterCatalog>
</Command>
</Transaction>
<Transaction>
<Command type="Query">
<DataSource etype="Entity">
<ExternalKeys>
<Key name="name" type="string">CUSTOMER_DS_2</Key>
</ExternalKeys>
</DataSource>
</Command>
</Transaction>
<Transaction>
```

```
<Command type="Query">
<DataSource etype="Entity">
<ExternalKeys>
<Key name="name" type="string">ACCOUNT_DS_1</Key>
</ExternalKeys>
</DataSource>
</Command>
</Transaction>
<Transaction>
<Command type="Query">
CatalogFormat etype="Entity">
<ExternalKeys>
<Key name="name" type="string">CUSTOMER_FORMAT</Key>
</ExternalKeys>
</CatalogFormat>
</Command>
</Transaction>
<Transaction>
```

```
                    <Command type="Query">
                        <CatalogFormat etype="Entity">
                            <ExternalKeys>
                            <Key name="name" type="string">ACCOUNT_FORMAT</Key>
                            </ExternalKeys>
                        </CatalogFormat>
                    </Command>
                </Transaction>
                <Transaction>
                    <Command type="Query">
                        <CatalogInputMap etype="Entity">
                            <ExternalKeys>
                            <Key name="name" type="string">ABC</Key>
                            <Key name="catalog" type="string">MC8</Key>
                            </ExternalKeys>
                        </CatalogInputMap>
                    </Command>
                </Transaction>
```

```
<Transaction>
<Command type="Query">
<OutputMap etype="Entity">
<ExternalKeys>
<Key name="name" type="string">CUSTOM1</Key>
<Key name="catalog" type="string">MCA</Key>
</ExternalKeys>
</OutputMap>
</Command>
</Transaction>
<Transaction>
<Command type="Query">
<OutputMap etype="Entity">
<ExternalKeys>
<Key name="name" type="string">CUSTOM2</Key>
<Key name="catalog" type="string">MCA</Key>
</ExternalKeys>
</OutputMap>
```

```
        </Command>
    </Transaction>
    <Transaction>
      <Command type="Query">
       <OutputMap etype="Entity">
         <ExternalKeys>
          <Key name="name" type="string">HSB1</Key>
           <Key name="catalog" type="string">CATALOGX</Key>
         </ExternalKeys>
        </OutputMap>
      </Command>
    </Transaction>
    <Transaction>
      <Command type="Query">
       <BusinessProcessRule etype="Entity">
         <ExternalKeys>
          <Key name="name" type="string">Product Notification</Key>
        </ExternalKeys>
```

```
                </BusinessProcessRule>
            </Command>
        </Transaction>
        <Transaction>
            <Command type="Query">
                <CatalogFormat etype="Entity">
                    <ExternalKeys>
                        <Key name="name" type="string">COUNTER_PARTY_PUBLISH_OUTPUT_
FORMAT</Key>
                    </ExternalKeys>
                </CatalogFormat>
            </Command>
        </Transaction>
        <Transaction>
            <Command type="Query">
                <CatalogFormat etype="Entity">
                    <ExternalKeys>
                        <Key name="name" type="string">EAN.UCC</Key>
```

```
                    <Key name="objectType" type="string">predefined</Key>
                    </ExternalKeys>
                    </CatalogFormat>
                </Command>
                </Transaction>
</DataService>
```

Exporting Metadata

Procedure

1. Place the request XML (similar to CatalogInput-OutputMap-Sample-CopyMetaData-Request.xml) in the directory accessible by FileWatcher.
2. Ensure that the object name specified in the request XML exists in the source enterprise.
3. FileWatcher initiates the copy metadata process using the request XML.
4. The system parses the request XML.
5. The system exports metadata objects into the response XML file.
6. This response XML is generated and packaged in a jar file.
7. The Jar file is placed in the work directory.

Result

Note: The export process ends abruptly if:

- You try to export an object on the source enterprise.
- If the object does not exist on the source enterprise. It starts exporting the next object specified in the request XML.

Importing Metadata

If the object exported must be imported with changes, you can extract the Request XML out of the JAR file, edit it and then bundle it back to the JAR for import.

Note: If you add new attributes to a synchronization format, the attributes are displayed in the synchronization format. However, when you modify a predefined synchronization format through import metadata, the attributes are not displayed in the repository attribute list. To get the new attributes, select and deselect output format.

Procedure

1. Place the JAR file created by copy metadata export, data service query request for the object in the directory accessible to FileWatcher.
2. Ensure that the associated object exists on the source enterprise or is a part of the exported JAR file.
3. FileWatcher initiates copy metadata processing using the JAR file.
4. System parses the response XML extracted from the JAR file.
5. The object is created or modified.
6. Response XML is generated.
7. The object is created or modified without any error.

Result

Note: The system ends import process abruptly if:

- If there is a dependency on any other object.
- If the object is not a part of the import JAR or does not exist in the source enterprise.

Customization of Roles Menus and Access Rights

TIBCO MDM uses role-based security to restrict and grant access to specific menus. This feature controls the actions the user can or cannot perform, such as editing or creating certain records in a record list.

Role-based Security

Users' access privileges are defined by the roles assigned to the user by the TIBCO MDM Administrator. Privileges are based on the functions performed by a role.

Users might be assigned one or more roles, with each role granting the user a different access privilege and level. TIBCO MDM includes standard pre-defined, out-of-the-box user roles. You can define your own custom roles in addition to those to meet your specific business needs.

Role-based security in TIBCO MDM is determined by *negative logic*. TIBCO MDM checks which functions are *not* allowed. Role-based security works on functions associated with HTML elements. Functions identify a logical group of work, primarily menu items. For example, the repository function groups all repository related functions together.

There are two aspects of role-based security:

- **Dynamic menu generation**: If you change the user roles, the menus are updated the next time a page is refreshed.
- **Filtering HTML elements**: Certain HTML elements can be added or removed

based on assigned user roles.

HTML elements are the hyper links such as the *add new record* link. You cannot control access to action links like ‘modify’, ’copy’, and so on that appear against a list entry on a page. Security is applied after the page is built.

Customization of Menus

The TIBCO MDM menus for any given user are generated dynamically, based on the role or roles assigned to that user. Menus are built each time a page is displayed. If you change the user roles, the menus are updated the next time a page is refreshed.

Adding a New Menu

Consider the scenario where you want to add a link so that you can access the user guide from the menu.

Procedure

1. Include the following entry in the allmenu.xml file to add **User Guide** menu item to the menu:

   ```
   <menuitem class="security" id="custom_menu_user_guide">
   <displayname>User Guide</displayname>
   <url>temp/index.htm</url>
   </menuitem>
   ```

   When you click the **User Guide** menu item, an HTML copy of the User Guide opens that is placed in the temp folder.

   The allmenu.xml file lists all menus and URLs, and refers to functions by name. Creating a new menu requires one of the following:

   - Use of existing screens. No merging of applications is required in this case.
   - Use of new applications. Merging of a new application or servlet class is required in this case. The following steps show how to merge servlet classes.

     A typical menu class entry is as follows:

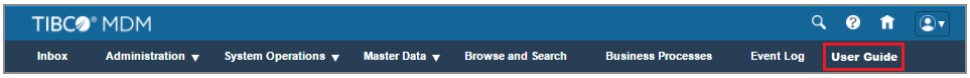
```
<menuitem class="security" id="catalogmenu_mass_update">
<displayname>Mass Update</displayname>
    <url>EmbeddedGIView?embed=4</url>
</menuitem>
```

2. Go to $MQ_HOME and extract EML.war from ECM.ear.
3. If any servlet classes are required, place the compiled servlet classes in the $MQ_HOME\ECM.ear\EML.war\WEB-INF\classes\com.tibco.mdm.ui.util folder.
4. Update the $MQ_HOME\ECM.ear\EML.war\WEB-INF\web.xml file to include the new servlet classes (<servlet> and <servlet-mapping> tags for the servlet class that you have added). The web.xml file is an index of all servlet classes used by the application.

```
- <servlet id=".....">
<servlet-name>.....</servlet-name>
<servlet-class>.....</servlet-class>
</servlet>
- <servlet-mapping id=".....">
<servlet-name>.....</servlet-name>
<url-pattern>.....</url-pattern>
</servlet-mapping>
```

5. Copy the temp directory containing all the HTML files in the ECM.ear\EML.war directory.
6. Archive EML.war to add the new classes.
7. Archive ECM.ear to add the updated EML.war.
8. Redeploy the ECM.ear file and restart the application server.

   The **User Guide** link is displayed on the menu bar. Using this link, you can browse through an HTML version of the User Guide.



9. Update the allmenu.xml file to add the new servlet classes as added in the web.xml file.
10. Redeploy the ECM.ear file and restart the application server.

Changing Permissions for the New Menu Based on Role

You can ensure that the new menu is available or hidden depending on the role.

For more information about customizing roles, see Customization of Roles.

By default, the new menu works for all new enterprises.

Procedure

1. As the roles and the ROLE2FUNCMAP table are created at the time of creating an enterprise, pre-existing enterprises *will not receive* the new menu correctly. You can work around this constraint by writing update scripts, based on default data, to insert records in the ROLE2FUNCMAP table. Contact TIBCO Support to get a sample script.
2. To hide the new menu item for some roles, add an entry in the FUNC2IDMAP table for the custom menu item that you have added.
3. To ensure that the new menu is available only to authorized roles update the defaultdata.xml file located at $MQ_HOME\config\security\.

   For the GDSN edition, the file is named as defaultdata.gdsn.xml.

Customization of Roles

Roles define groups of users with common privileges and responsibilities. You can use roles to further define Workflow, GUI, and Catalog access behavior.

> **Note:** To add a new role or modify the predefined roles, use the Import Metadata feature of TIBCO MDM.
>
> For more information, see Importing Users, Roles, and Permissions section of the *TIBCO MDM User's Guide*.

The following table describes the role configuration table name and its description:

Role Configuration Tables

Table Name	Description
ROLE	Holds all the roles. When a new enterprise is created, all roles for the enterprise are created, based on the $MQ_HOME/config/security/defaultdata.xml file. For the GDSN edition, the file is named defaultdata.gdsn.xml.
ROLE2FUNCMAP	Associates a role with functions that are not allowed for the role. The entries in this table are populated based on $MQ_HOME/config/security/defaultdata.xml when an enterprise is created. For the GDSN edition, the file is named defaultdata.gdsn.xml.
FUNCTION	Identifies menus and pseudo functions assigned to HTML elements. It is either copied from seed data or populated.
FUNC2IDMAP	Relates functions to HTML elements. It is either copied from seed data or populated.
HTMLELEMENTID	Stores all HTML elements, other than menus, which are to be managed.

Filtering HTML Elements

Use HTML filtering to hide a link shown on the page based on roles assigned to the user. You cannot control access to action links such as, ‘modify’, ’copy’, and so on that appear against a list entry on a page.

The following example shows how to remove the **add new record** link from the **TaxonomyViewServlet** page for a user with an **admin** role:

Procedure

1. Find the HTML name on which the link appears.
2. Open the HTML file and find the element ID which must be controlled. If the element that you want to control does not have an ID, you must contact TIBCO to assign an ID to it. In this case, it is found to be acreateNP.
3. If not already present, insert a corresponding new function in the FUNCTION table

as follows:

```
INSERT INTO FUNCTION ( ID, NAME, DESCRIPTION, FUNCTION ) VALUES ( 66, 'add_
new_record', 'Add New Record link', 'add_new_record');
```

4. If not already present, add a corresponding new element in the HTMLELEMENTID table.

   ```
   INSERT INTO HTMLELEMENTID ( ID, NAME, PAGENAME ) VALUES ( 32480, 'acreateNP',
   'com.tibco.mdm.ui.repository.taxonomy.TaxonomyView');
   ```

   where NAME is the HTML element ID. Contact TIBCO Customer Support to get the PAGENAME.

5. Add a corresponding record in the FUNC2IDMAP table.

   ```
   INSERT INTO FUNC2IDMAP ( ID, FUNCID, NAME, ELEMENTID, ELEMENTTYPE,
   SECURITY ) VALUES ( 100, 66, 'add_new_record', 32480, 1, 3);
   ```

6. Identify the ID of the role for which you want hide the HTML element:
   a. Find the enterprise you want to modify and note the enterprise ID. In this example, 1234 is the enterprise ID. Name is case sensitive.

      ```
      Select * from enterprise where name = '<<YourEnterpriseName>>'.
      ```

   b. Find all of the roles created for the enterprise. A list of roles for the enterprise is displayed. The name is the name of the role that is assigned to a user.

      ```
      Select * from role where enterpriseID = ‘<<YourEnterpriseID>>’
      ```

   c. Note the role name and the ID of the role you want to modify. This example uses 32870 as the role ID.

7. Add an entry for this role (identified in step 6), and the function (created in step 3) to the ROLE2FUNCMAP table.

   ```
   INSERT INTO ROLE2FUNCMAP (ID, ROLEID, FUNCID, NAME, PRIORITY) VALUES (100,
   32870, 'add_new_record', 32480, 1);
   ```

8. Restart the application server.

Note: If you do not restart the application server, the links are not hidden from the page.

Modifying or Deleting a Users Role Assignment

Before you begin

You must have the appropriate access permissions.

Procedure

1. Navigate to **Administration > Roles**, select the user and then click the **Show Users** link to the right of the appropriate user role.
2. Click **Modify** to modify a user who is assigned to a role. The Modify User page is displayed.
3. Click **Delete** to delete a user assigned to a role. The Delete User page is displayed.
4. From the Modify User (or Delete User) page, change the user account information as needed.
5. Click Save to apply the modification or deletion to the user account and role assignments.

Track History of User Role Assignments and Permissions

You can manage the history of permissions, user, and role assignments through history tables. These history tables keep track of all changes to the appropriate tables. When changes are made to the table, the changes are captured by triggers and copied to the history tables.

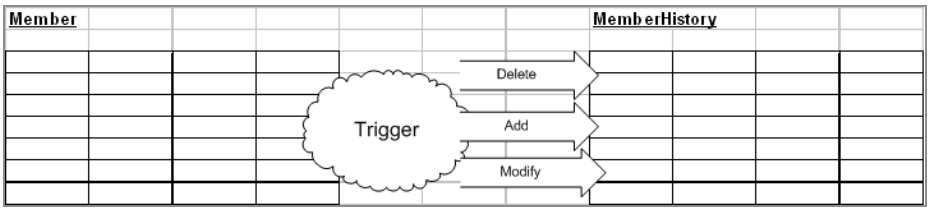
The following table describes the table name and history table name for users, role assignments, and permissions:

History Tables

Name	Table	History Table
Users	MEMBER	MEMBERHISTORY
Role Assignments	ROLEASSIGNMENT	ROLEASSIGHISTORY
Permissions	RESOURCEACL	RESOURCEACLHISTORY

Users (MEMBERHISTORY Table)

The MEMBER table is used to store user information. When changes are made to the MEMBER table, the old information is stored in the MEMBERHISTORY table.

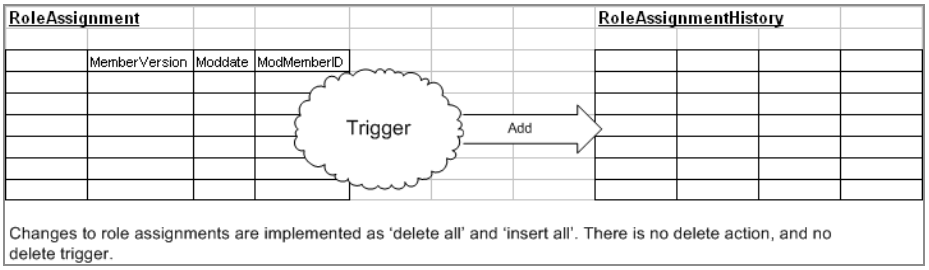


The triggers capture information when:

- Data is inserted into the MEMBER table.
- Data from the MEMBER table is modified (an entry is added to the table).
- Data is deleted from the MEMBER table. In this case, an inactive entry is added to the history table. However, the timestamp is the same as the system date and modmemberID is the same as the old member record.
- When an user is deleted, it is done by updating the Active column to N. This invokes the update trigger. However, no entries are made into the ROLEASSIGHISTORY and ResourceACLHistory tables.

Role Assignments (ROLEASSIGHISTORY Table)

The ROLEASSIGMENT table maintains assignment of roles to users. When changes are made to the ROLEASSIGNMENT table, the old information is stored in the ROLEASSIGHISTORY table.



A change to the ROLEASSIGMENT table role is always accompanied with a change to the MEMBER table (the member version is incremented). The ROLEASSIGHISTORY table inherits the member version from the MEMBER table.

This history table has MEMBERID, MEMBERVERSION as the set keys. All roles assigned for the specific version can be found for this combination.

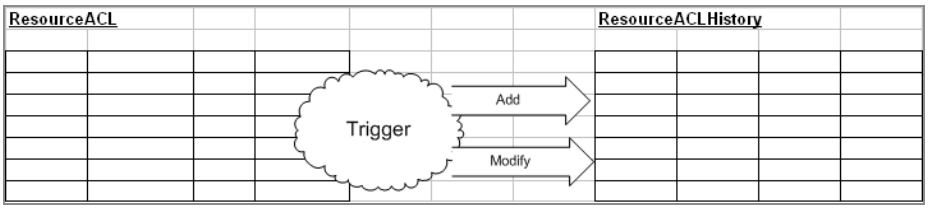
Permissions (RESOURCEACLHISTORY Table)

When resource ACLs are modified, all the ACLs are assigned the same time stamp so that they are identified as one complete set. The timestamp is the only way to identify a set.

The history table maintains the full set of permissions associated with the member or role. Set keys are RESOURCETYPE, GRANTEEID, and MODDATE.

To find out permissions granted to a resource, identify one complete set of permissions using RESOURCETYPE, GRANTEEID, and MODDATE. Select the set which has the same time stamp for a GRANTEEID and RESOURCETYPE. This set represents the total permissions for the grantee of that resource type. As "Create permissions" are stored with resource ID = 0, you can have two resource IDs in this set, that is, 0 and the specific resource ID.

When permissions are deleted for a grantee through the UI, the existing set is added to the history table with active = N. In this case, the history table has the corresponding deleted entries as the trigger is also fired for an update. However, when permissions are changed, permissions are deleted and re-added. In this case, the history table only has the new set.



Configuring Password Policy Rulebase

You can apply certain policies for valid passwords. A sample rulebase file (standard/rulebase/rbpasswdval.xml) is available. To use this file, you must copy it to the enterprise-specific directory.

Procedure

1. Define a rulebase file that contains the validation rules for the user passwords.
2. Upload this file to the $MQ_COMMON_DIR/*enterprisename*/rulebase/ directory.
3. Specify the name of the file in the Configurator (**Rule Base > Password Validation Rules File**).

   > **Note:** Ensure that you specify the default file name (rbpasswdval.xml) whenever possible.

   The application uses this rulebase file to ensure that the specified password is according to the rules.

   - If the file name is not specified in the Configurator, no validation is done.
   - If a file name is specified, and the file is not found, a warning message is issued and the file is ignored.
   - The rulebase is used to generate a Java script which is added to the password fields. It does not have access to any other context variables, such as, the SESSION context variables. For more details about the context variables that can be used in the rulebase, see *TIBCO® MDM StudioRulebase Designer User's Guide*.

Differences between Software Editions

Review the differences between the GDSN and MDM software editions. The default edition is MDM.

Default Terminology Differences

Both the editions include the following terminology differences in both the editions:

Default Term Changes

GDSN Name	MDM Name
Synchronize	Publish
Trading Partner	Backend System
Datapool	

Roles (GDSN only)

Roles specific to the GDSN edition are: **Channel Manager** and **Sales Rep**.

Enterprise Creation (GDSN only)

For the MDM edition, no options are shown while creating an enterprise.

The following Enterprise Type names are specific to the GDSN edition.

- Supplier
- Retailer
- Datapool

Business Process Rules

Some Business Process rules are specific to the software editions as listed in the following table:

Business Process Rule and Rule Set Changes

GDSN Name	MDM Name
Master Catalog Selection	Repository Selection
Unknown Partner Alert	Unknown System Alert

Customization of Data Synchronization

Information related to Marketplace, Retailer, Rulebase operations, and so on is specific to GDSN only.

Terminology (such as Supplier, Marketplace, and so on) used in this chapter might vary depending on customization done at your site.

Customization of Format-Specific Attributes on GUI

The customization of format-specific attributes on GUI includes:

- Configuration of Format-Specific Attributes
- Configuration of ATTRIBUTE_LIST
- Configuration of Operation Groups
- Configuration of Operation Identifiers
- Configuration of ResponseHandlers
- Configuration of MessagingHandlers
- Role Derivation Rulebase
- Synchronization Validation Rulebase
- Protocol Derivation Rulebase
- Configuration of the Protocol
- Configuring Queues

Configuration of Format-Specific Attributes

You can configure the trading partner-specific Synchronization profile graphical user interface (GUI) so that the users can specify different input variables.

You can determine which portions (or, format-specific attributes) of the trading partner GUI apply to which input values for the trading partner Catalog. You can create, modify, publish, and view the Catalog GUI.

Use the Rulebase to modify these format-specific attributes, based on the following input parameters:

- OUTPUT_FORMAT
- MARKETPLACE_NAME
- ORGANIZATION_TYPE
- DELIVER_TO

An example of a variable declaration for an input parameters is:

```
<declare usage="input">
        <var>OUTPUT_FORMAT</var>
</declare>
```

The values of these input parameters are *always* uppercase, and must be compared to uppercase strings. Based on the input parameters above, the following variables can be assigned for customization:

- ATTRIBUTE_LIST
- DELIVERY_GROUP
- OPERATIONS

An example of the variable declaration is:

```
<declare usage="output">
  <var>ATTRIBUTE_LIST</var>
 </declare>
```

Configuration of ATTRIBUTE_LIST

The values assigned to the ATTRIBUTE_LIST output variable are used to populate format-specific attributes of the Catalog. You use a constraint tag, based on a condition, to modify the ATTRIBUTE_LIST output variable.

The following attributes can be configured for both SUPPLIER and RETAILER organizations

- CLASSIFICATIONSCHEME
- DELIVERCATALOG
- MINIMIZENEWVERSIONS
- OPERATION
- VALIDFROM
- VALIDUNTIL

The following format-specific attributes are configurable for the SUPPLIER only:

- SUPPLIERCREDENTIAL
- BUYERCREDENTIAL

One format-specific attribute is configurable for the RETAILER only:

- RETAILERCREDENTIAL

Example: Valid Attribute List Output

In the following example, CatalogFormat = 1SYNC, Organization_Type = SUPPLIER, and Deliver_To = Marketplace. A list of format-specific attributes is defined for the ATTRIBUTE_LIST output variable.

These attributes appear under format-specific attributes when the user logs into the 'SUPPLIER' organization, and chooses the Deliver_To option of the 'Marketplace', and the CatalogFormat option '1SYNC,' while creating, viewing, or modifying the Catalog.

```
For CatalogFormat = 1SYNC
   Organization_Type = SUPPLIER and
   Deliver_To = Marketplace,
   Show the format specific attributes
   CLASSIFICATIONSCHEME,DELIVERCATALOG,
   OPERATION, SUPPLIERCREDENTIAL, VALIDFROM,
   VALIDUNTIL
<constraint>
<name>
MarketPlaceAndCatalogFormatAttributesForTransora
</name>
 <description>
 Standard format-specific attributes for the
 supported Trading partners and catalog
 formats applicable for the supplier.
 </description>
```

```
 <condition>
  <and>
  <eq>
   <var>OUTPUT_FORMAT</var>
   <const type="string">TRANSORA</const>
  </eq>
  <eq>
   <var>ORGANIZATION_TYPE</var>
   <const type="string">SUPPLIER</const>
  </eq>
  <eq>
   <var>DELIVER_TO</var>
   <const type="string">MARKETPLACE</const>
  </eq>
  </and>
 </condition>
<action>
  <assign>
   <var>ATTRIBUTE_LIST</var>
   <const  type="string">
   CLASSIFICATIONSCHEME</const>
   <const type="string">
   DELIVERCATALOG</const>
   <const type="string">OPERATION</const>
   <const type="string">
   SUPPLIERCREDENTIAL</const>
   <const type="string">VALIDFROM</const>
   <const type="string">VALIDUNTIL</const>
  </assign>
 </action>
</constraint>
```

Configuration of Operation Groups

There are three operation groups that can be configured, two for SUPPLIER organizations and one for RETAILER organizations.

You can modify these operation groups if OPERATION is specified in the format-specific attributes.

The operation groups specific to SUPPLIER organizations are:

- ADDGROUP
- PUBLISHGROUP

The operation group specific to the RETAILER organization is:

- CONFIRMGROUP: This is an additional operations group.

Configuration of Connector Plug-ins

All connector plug-ins use two variables connectorProtocol and connectorRole. Based on these two variables, Configurator (ConfigValues.xml) is configured.

Plug-Ins Used for Synchronization

The following are the plug-ins used for synchronization:

- **ResponseHandlers** – They are the gateway to the application. They help in routing of new messages, and also handle response to messages sent out.
- **Operation Identifiers** – This plug-in uses a rulebase to identify synchronization operations to perform, based on the synchronization history of a record in a given data pool.
- **Messaging Handlers** – This plug-in decides on the order in which messages must be sent. It also determines the records that must go together in a particular message.

Configurable Variables Used in the Plug-In Selection

The following are the configurable variables used in the plug-in selection:

- **ConnectorRole** –This variable identifies the role an organization plays on a given data pool, namely, Supplier, Retailer. The ConnectorRole used can be configured using a rulebase pointed to by the Configurator > **Connector** > **Connector Role Rulebase** option. It is also possible to specify the role directly as a workflow parameter to the IdentifyProtocolOperations activity. The rulebase itself uses 4 variables:SENDER_ORGANIZATION_NAME, SENDER_ORGANIZATION_TYPE, RECEIVER_ORGANIZATION_NAME and RECEIVER_ORGANIZATION_TYPE. Where a sender organization is the organization that is trying to send the message (usually a SUPPLIER or RETAILER). The receiver organization is usually the marketplace or the trading partner who is to receive the message.
- **ConnectorProtocol** – This variable identifies the protocol to be used while

communicating with a data pool. It differs based on the data pool, that is, 1Sync or Agentrics. The ConnectorRole used can be configured using the rulebase pointed to by the Configurator > **Connector** > **Connector Role Rulebase** option.

Configuration of Operation Identifiers

The kind of OperationsIdentifier picked depends upon the two variables: ConnetorRole and ConnectorProtocol.

Operations and Sub Operations

TIBCO MDM allows you to control the operations and sub-operations generated for data synchronization. Operations and sub-operations are returned in the format: OPERATION:SUB-OPERATION.

Ensure that the operation returned is valid. see the following table.

Valid Operations

Operation	Sub-Operation
ADD	Change, New, Correct, Cancel, Delete, Discontinue, Reinstate, RELOAD
PUBLISH	NEW, CANCEL, IMPLICIT_CANCEL, DISCONTINUE, IMPLICIT_ DISCONTINUE, CHANGE, CORRECT, IMPLICIT
CONFIRM	ACCEPTED, REJECTED, REVIEW, SYNCHRONISED
NOTIFY-CONFIRMATION	ADD, CHANGE, CANCEL, DISCONTINUE
NOTIFY-PRODUCT CONTENT	ADD, CHANGE, CANCEL, DISCONTINUE
LINK	NONE, UNLINK, CORRECT

Defining the Rulebase

You define the rulebase by mapping the marketplace that you want to customize.

Procedure

1. Open the ConfigValues.xml file in a text editor and go to the **Integration Setup - External** section.
2. Copy the following properties:

```
<ConfValue name="Default Protocol Identifier"
propname="com.tibco.cim.connector.plugin.mapping.identifyoperations.*.*"
sinceVersion="7.0" visibility="All">
<ConfString value="com.tibco.cim.init.connector.plugin.identifyoperations.default"
default="com.tibco.cim.init.connector.plugin.identifyoperations.default"/>
</ConfValue>
<ConfValue name="Default Protocol Class"
propname="com.tibco.cim.init.connector.plugin.identifyoperations.default.class"
sinceVersion="7.0" visibility="All">
<ConfString
value="com.tibco.mdm.integration.messaging.connector.ProtocolOperationsIdentifier"
default="com.tibco.mdm.integration.messaging.connector.ProtocolOperationsIdentifier"/>
</ConfValue>
<ConfValue name="Default Rulebase File"
propname="com.tibco.cim.init.connector.plugin.identifyoperations.default.rulebaseFile"
sinceVersion="7.0" visibility="All">
<ConfString value="standard/rulebase/rbdefault.xml"
default="standard/rulebase/rbdefault.xml"/>
</ConfValue>
```

3. Change the properties as shown, where MYDATAPOOL is the name of the data pool to which you want to synchronize:

```
<ConfValue name="Default Protocol Identifier"
propname="com.tibco.cim.connector.plugin.mapping.identifyoperations.*.MYDATAPOOL"
sinceVersion="7.0" visibility="All">
<ConfString value="com.tibco.cim.init.connector.plugin.identifyoperations.mydatapool"
default="com.tibco.cim.init.connector.plugin.identifyoperations.default"/>
</ConfValue>
<ConfValue name="Default Protocol Class"
propname="com.tibco.cim.init.connector.plugin.identifyoperations.MYDATAPOOL.class"
sinceVersion="7.0" visibility="All">
<ConfString
```

```
value="com.tibco.mdm.integration.messaging.connector.ProtocolOperationsIdentifier"
default="com.tibco.mdm.integration.messaging.connector.ProtocolOperationsIdentifier"/>
</ConfValue>
<ConfValue name="Default Rulebase File"
propname="com.tibco.cim.init.connector.plugin.identifyoperations.MYDATAPOOL.rulebaseFile" sinceVersion="7.0" visibility="All">
<ConfString value="<enterprise>/rulebase/rbMYDATAPOOL.xml"
default="standard/rulebase/rbdefault.xml"/>
</ConfValue>
```

4. Edit the rbMYDATAPOOL.xml file and place it in the $MQ_COMMON_DIR/<enterprise>/rulebase directory.

   Use a lower priority for default assignments. The default rulebase priority is 1, so set the priority to 0 for default assignment.

   Optionally, you can set a different priority for each assignment. Any assignment of the same or higher priority replaces the previous value. If there is more than one constraint for an operation, and all assignments have the same priority, the last assignment is retained.

Determine Synchronization History

Use the following keywords to determine synchronization history and identify the state of the product in the system:

- **HAS_PREVIOUSLY_PUBLISHED_PARENT:** returns TRUE if any of the records reached by traversing the reverse relationship of the IS_PUBLISHED record are TRUE.
- **IS_ACCEPTED:** checks the product status and determines whether a confirmation was sent or received with an ACCEPTED status.
- **IS_ADDED:** identifies whether a product is already added to the Registry. It returns FALSE if the latest version has been successfully deleted from the registry (that is, if a product is no longer in the registry).
- **IS_DELETED:** returns TRUE if the last registry operation deleted the product from the registry.
- **IS_DISCONTINUED:** returns TRUE if the GTIN has been discontinued. Theoretically, this product cannot be synchronized in the future.
- **IS_LINKED:** identifies whether the product has already been linked to other

related products in the registry.

- **IS_PUBLISHED:** checks whether the product has ever been published explicitly to a particular trading partner. If the publication was CANCELED after the publication, this check returns FALSE. If the product has been successfully published, this check returns TRUE.
- **IS_PUBLISHED_TO_ANY_PARTNER:** determines whether the product has been published successfully to any trading partner. You can use this check to find out if the product is listed.
- **IS_REVIEWED:** checks the product status and determines whether a confirmation was sent or received with a REVIEW status.
- **IS_REJECTED:** checks the product status and determines whether a confirmation was sent or received with a REJECTED status.
- **IS_SYNCHRONISED:** checks the product status and determines whether a confirmation was sent or received with a SYNCHRONISED status.
- **IS_ADD_DISCONTINUED**: returns TRUE if the last registry operation is ADD + DISCONTINUED.
- **IS_ADD_CANCELLED**: returns TRUE if the last registry operation is ADD + CANCELLED.
- **IS_PUBLISH_CANCELLED**: returns TRUE if the GTIN has been CANCELLED. Theoretically, this product cannot be synchronized in the future.
- **IS_PUBLISH_DISCONTINUED**: returns TRUE if the GTIN has been DISCONTINUED. Theoretically, this product cannot be synchronized in the future.

Evaluate the State of the Record

Use these keywords to evaluate the current state of the record:

- **HAS_ATTRIBUTE_SET_CHANGED**: determines whether any of the attributes specified in the format have been modified.
- **HAS_LINKED_RELATION_CHANGED**: determines whether any of the relationships defined for a record have been changed. If the product had no relationship earlier, but a new relationship has been defined, or vice versa, this returns TRUE. If any relationship has changed, this returns TRUE. If the related products remain the same, but the quantity of the products has changed, this returns FALSE.

- HAS_LINKED_QUANTITY_CHANGED: determines whether or not any of the previously linked records quantity defined for the record have been changed. Returns TRUE if quantity is changed for the linked record. If no change in quantity, it returns FALSE.
- HAS_PUBLISHED_RELATION_CHANGED: determines whether or not any of the previously published records relation defined for the record have been changed. Returns TRUE if relation is changed for published record. If no change in relation, it returns FALSE.

Identify Product Quantity Changes

Use these keywords to identify the product quantity changes:

- HAS_ANY_CHILD_CHANGED: returns TRUE if any of child products (based on traversal of the given forward relationship) have changed since they were last registered with the data pool.
- HAS_CHILDREN: returns TRUE if the product has relationships defined, and has child records.
- HAS_LINKED_QUANTITY_CHANGED: determines whether the quantity of a particular relationship has changed from an earlier defined quantity. It also returns TRUE if HAS_LINKED_RELATION_CHANGED is TRUE.
- HAS_PUBLISHED_QUANTITY_CHANGED: determines whether or not any of the previously published records quantity defined for the record have been changed. Returns TRUE if quantity is changed for publishing a record. If no change in quantity, it returns FALSE.
- IS_ROOT_RECORD: returns TRUE if the product has been explicitly selected for synchronization.
- IS_PARENT: returns TRUE for all records reached using a reverse relationship starting from the root record.
- IS_CHILD: returns TRUE for all records reached using specified relationships starting from the root record.

Example:

In this example, the user selected A for synchronization, and selected the *Contains and ContainedBy* relationships for processing. ContainedBy is defined as a reverse relationship of Contains.

While evaluating A, all related records are collected in the record bundle, with A as the root record. All records selected by traversal of the given relationships are included in the bundle. see the *Conditional Parent/Child Relationships* table that follows.

- A contains B
- B contains C
- Z contains A
- Y contains Z
- Z contains D
- B, C, Z, Y and D are implicitly included.

Conditional Parent and Child Relationships

Conditional Parent and Child Relationships

Record	IS_ROOT_ RECORD	IS_ PARENT	IS_ CHILD	HAS_PREVIOUSLY_ PUBLISHED_PARENT
A	True	False	False	True if IS_PUBLISHED is true for either Z or Y.
B	False	False	True	True if IS_PUBLISHED is true for any of Z, A, or Y.
C	False	False	True	True if IS_PUBLISHED is true for any of Z, A, B, or Y.
Z	False	True	False	True if IS_PUBLISHED is true for Y.
Y	False	True	False	False.
D	False	False	False	True if IS_PUBLISHED is true for either Z or Y.

Incorporate User Input

Use these keywords to incorporate user input from the Catalog:

- **IS_ACCEPT_REQUESTED:** returns TRUE if the user chose to send the ACCEPTED message on the Catalog.
- **IS_ADD_REQUESTED:** returns TRUE if the user selected the ADD option on the Catalog.
- **IS_CANCEL_REQUESTED:** returns TRUE if the user selected the WITHDRAW option on the Catalog.
- **IS_CORRECTION_REQUESTED:** returns TRUE if the user selected the Minimize Version option on the Catalog.
- **IS_DELETE_REQUESTED:** returns TRUE if the user selected the Delete Version option on the Catalog.
- **IS_DISCONTINUE_REQUESTED:** returns TRUE if the user selected the DELIST option on the Catalog.
- **IS_INCREMENTAL_REQUESTED:** returns TRUE if the user selected the Incremental Flag option on the Catalog.
- **IS_PUBLISH_REQUESTED:** returns TRUE if the user selected the PUBLISH option on the Catalog.
- **IS_REJECT_REQUESTED:** returns TRUE if the user chose to send the REJECTED message on the Catalog.
- **IS_RELOAD_REQUESTED:** returns TRUE if the user selected the INITIAL LOAD option on the Catalog.
- **IS_REVIEW_REQUESTED:** returns TRUE if the user chose to send the REVIEW message on the Catalog.
- **IS_SYNCHRONISE_REQUESTED:** returns TRUE if the user chose to send the SYNCHRONISED message on the Catalog.
- **IS_ADD_DELETE_REQUESTED**: return true if user selects ADD and DELETE operations on Catalog.
- **IS_ADD_REINSTATE_REQUESTED**: return true if user selects ADD and REINSTATE operations on Catalog.
- **IS_ADD_DISCONTINUE_REQUESTED**: return true if user selects ADD and DISCONTINUE operations on Catalog.

- **IS_ADD_CANCEL_REQUESTED**: return true if user selects ADD and CANCEL operations on Catalog.
- **IS_RFCIN_REQUESTED**: return true if user selects RFCIN option on Catalog.
- **IS_PUBLISH_CANCEL_REQUESTED**: return true if user selects PUBLISH and CANCEL operations on Catalog.
- **IS_PUBLISH_DISCONTINUE_REQUESTED**: return true if user selects PUBLISH and DISCONTINUE operations on Catalog.

Configuration of ResponseHandlers

The response handler plug-in used depends upon the ConnectorProtocol. There are predefined response handlers for protocols - Agentrics and 1Sync.

All custom protocols go through a default response handler that simply forwards a received message (or response) to the workflow. To define a custom protocol you must configure the rulebase.

Configuration of MessagingHandlers

There are predefined messaging handlers for protocols - Agentrics and 1Sync. All custom protocols go through a default messaging handler that goes sequentially through all the identified operations without regard to the order of records selected or the order of operations.

Role Derivation Rulebase

Role is one of the two variables used for selecting connector plug-ins used in synchronization workflows.

Configuration of the Rulebase

The Role used can be configured using the rulebase pointed to by the property Configurator > **Connector** > **Connector Role Rulebase** (velosel.connector.rolederivationrulebase).

Synchronization Validation Rulebase

TIBCO MDM supplies sample validations for various data pools. For each data pool, two validation files are provided.

Validations for a Specific Datapool

To apply validations for a specific data pool when records are created and modified, install the corresponding validation file.

All validation files are located in /standard/rulebase/.

Ensure that if you modify these files, you do not change the constraint names. This allows you to reapply any changes in sample files.

You must select an appropriate rulebase while creating a catalog.

If you do want to apply additional validations during sync, you have 2 choices:

- Modify the sample rulebase. It is a good practice to copy these files under your enterprise directory before modifying. Also, leave the TIBCO MDM defined constraint names as is, for easy upgrade.
- Define additional validations in a separate rulebase.

Applying Two Validations

To apply two validations, modify the synchronization workflow.

Procedure

1. Add a new ApplyRulebase step before the existing ApplyRulebase step.
2. Specify the name of the sample file in the new step and pick the additional validation file in the catalog.

Protocol Derivation Rulebase

Protocol is one of the two variables used for picking up connector plug-ins to be used in a synchronization workflow.

Configuration of the Protocol

The Protocol used can be configured using a rulebase pointed to by the Configurator > **Connector** > **Connector Rulebase**.

Configuring Queues

For communication between two different enterprises, a couple of gateways are needed to transport messages across two firewalls, and to transfer messages from the Q_ECM_INTGR_STD_OUTBOUND_INTGR_MSG queue of the sender instance to the Q_ECM_INTGR_STD_INBOUND_INTGR_MSG queue of the receiver instance.

The same physical queue (Q_ECM_INTGR_STD_INBOUND_INTGR_MSG) hosted on one of the JMS servers (the one connected to TIBCO MDM M/P) can be used, and you can map it to the following:

- logical queue (StandardOutboundIntgrMsg) used for sending outbound integration messages from the Brand Owner instance, and
- logical queue (StandardInboundIntgrMsg) used for receiving inbound integration messages in the TIBCO MDM M/P instance.

This is possible because the messaging framework allows:

- Mapping between logical queue names and physical queue names.
- Connectivity to multiple JMS servers from the same instance.

Since the Brand Owner must connect to two different queue managers, it requires two different clusters.

Note: The support for MQSeriesCluster has been removed from the TIBCO MDM 9.1.0 release. Therefore, this section is deprecated.

Procedure

1. If the Brand Owner instance is using MQSeriesCluster, add the following properties using the Configurator . This is just a copy of the MQSeriesCluster section with a different queue manager (VeloseIMPJMSQMgr) and a different IP address (VeloseIMPJMSServerIPAddress):

a. Properties needed by the messaging framework:

- com.tibco.cim.queue.cluster.MQSeriesCluster2=\
  inherit:com.tibco.cim.queue.cluster.DefCluster
- com.tibco.cim.queue.cluster.MQSeriesCluster2.clusterLiaison.class=com.tibco.mdm.integration.messaging.queue.ibm.MQSeriesClusterLiaison
- com.tibco.cim.queue.cluster.MQSeriesCluster2.replaceAllConnOnFailure=true

a. Vendor specific properties:

- com.tibco.cim.bus.cluster.MQSeriesCluster.clusterLiaison.clusteredQMgrs=LocalhostQMgr
- com.tibco.cim.queue.cluster.MQSeriesCluster2.clusterLiaison.clusteredQMgr.LocalhostQMgr.connFactory.mqQMgr=VeloseIMPJMSQMgr
- com.tibco.cim.queue.cluster.MQSeriesCluster2.clusterLiaison.clusteredQMgr.LocalhostQMgr.connFactory.mqHost=VeloseIMPJMSServerIPAddress
- com.tibco.cim.queue.cluster.MQSeriesCluster2.clusterLiaison.clusteredQMgr.LocalhostQMgr.connFactory.mqPort=
- com.tibco.cim.queue.cluster.MQSeriesCluster2.clusterLiaison.clusteredQMgr.LocalhostQMgr.connFactory.mqChannel=SCC_ECM

2. Change the mapping of the logical queue to the physical queue on the Brand Owner instance to use Q_ECM_INTGR_STD_INBOUND_INTGR_MSG (instead of Q_ECM_INTGR_STD_OUTBOUND_INTGR_MSG) hosted on the JMS server connected to the TIBCO MDM M/P instance (the queue is hosted by MQSeriesCluster2).
3. Set the following properties in the Configurator:

a. Properties needed by the messaging framework:

- com.tibco.cim.queue.queue.StandardOutboundIntgrMsg=\inherit:com.tibco.cim.queue.queue.DefOutboundIntgrQueue
- com.tibco.cim..queue.queue.StandardOutboundIntgrMsg.cluster=MQSeriesCluster2
- com.tibco.cim.queue.queue.StandardOutboundIntgrMsg.msgIO=\inherit:com.tibco.cim.queue.msgIO.process.OutboundIntgrMsgIOProcess
- com.tibco.cim.queue.queue.StandardOutboundIntgrMsg.addToJNDI=true

a. Cluster specific properties - JNDI:

- com.tibco.cim.queue.queue.StandardOutboundIntgrMsg.cluster.JNDICluster2.jndiQueueName=VeloseI_Queue_StandardInboundIntgrMsg

a. Cluster specific properties - MQSeries:

- com.tibco.cim.queue.queue.StandardOutboundIntgrMsg.cluster.MQSeriesCluster2.mqBaseQueue=Q_ECM_INTGR_STD_INBOUND_INTGR_MSG

4. Change the mapping of the logical queue to the physical queue on the Brand Owner instance to use Q_ECM_INTGR_STD_OUTBOUND_INTGR_MSG (instead of Q_ECM_INTGR_STD_INBOUND_INTGR_MSG) hosted on the JMS server connected to the TIBCO MDM M/P instance (the queue is hosted by MQSeriesCluster2).

Customization of Generic Page

The generic page is available from the Record View page. An action link is available for SendMessage based on the user privileges (Role).

After clicking the action link, the page is customizable based on the underlined rulebase rbsendMessage.xml. Currently, it is used to send an RFCIN message and generate a fact sheet.

> **Warning:** Information related to Marketplace, Retailer, and RFCIN is specific to GDSN only.

The following customizations are possible:

- Customize the product attributes that can be displayed.
- Customize action that can be performed.
- Customize the MarketPlace and TradingPartner Credentials.
- Customize the user defined attributes.

> **Note:** These changes are applied to all repositories in an enterprise.

Generic Page Configuration

You can configure the generic page graphical user interface (GUI) so that the users can specify different input variables.

Use the Rulebase to modify generic page based on the following input parameters:

- OUTPUT_FORMAT
- MASTERCATALOGNAME
- ORGANIZATION_TYPE
- ACTION

An example of a variable declaration for input parameters is the values of these input parameters are always uppercase and must be compared to uppercase strings. Based on the input parameters above, the following variables can be assigned for customization:

- ATTRIBUTE_LIST
- ACTION_LIST
- USERDEFINED_LIST
- CREDENTIAL_LIST

An example of the variable declaration is configuring the ATTRIBUTE_LIST. The values assigned to the ATTRIBUTE_LIST output variable are used to populate format-specific attributes of the catalog. You use a constraint tag, based on a condition, to modify the ATTRIBUTE_LIST output variable.

```
<declare usage="input">
<var>OUTPUT_FORMAT</var>
</declare>
<declare usage="output">
<var>ATTRIBUTE_LIST</var>
</declare>
```

Product Attributes Customizations

All the attributes that are available in the repository are valid values for ATTRIBUTE_ LIST.

They can be assigned as shown in the following example:

You can assign as many attributes as required. All the display names of these attributes are shown on the GUI.

```
<constraint>
    <name>Display attribute List</name>
    <description>Standard format specific attributes.</description>
    <condition>
        <eq>
        <var>MASTERCATALOGNAME</var>
        <const type="string">EANUCCSUBSCRIPTIONCATALOG</const>
        </eq>
```

```
    </condition>
     <action>
      <assign>
         <var>ATTRIBUTE_LIST</var>
         <const type="string">SHORTDESC</const>
         <const type="string">ADDITIONAL_CLASS_CAT_CODE</const>
         <const type="string">GTIN</const>
         <const type="string">TARGET_MARKET_ID</const>
         <const type="string">SUPPLIERID</const>
       </assign>
      </action>
 </constraint>
```

Customization of Actions

You can assign the following valid values to the ACTION_LIST variable:

- RFCIN: On submitting the form with the action as RFCIN, it sends an RFCIN message to the Supplier based on the credentials. RFCIN is only valid for the retailer.
- FACTSHEET: On submitting the form with the action as FACTSHEET, it generates the PDF format of the product.

It is a good practice to use the constraints according to the following example:

```
<constraint>
      <name>Action List</name>
      <description>list of all actions possible with the organization.</description>
      <condition>
      <eq>
         <var>ORGANIZATION_TYPE</var>
         <const type="string">SUPPLIER</const>
       </eq>
     </condition>
      <action>
       <assign>
         <var>ACTION_LIST</var>
         <const type="string">FACTSHEET- Generate factsheet</const>
       </assign>
      </action>
 </constraint>
<constraint>
```

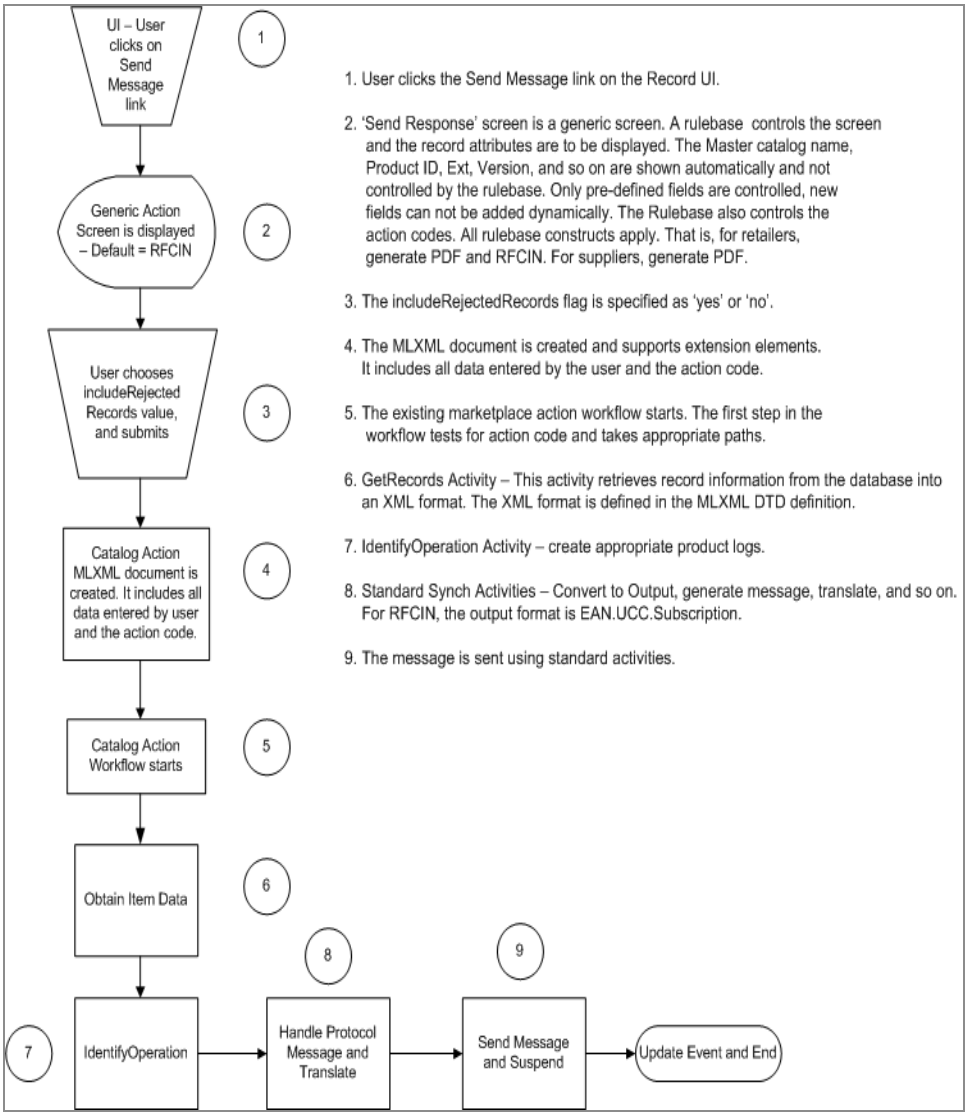
```
      <name>Action List 1</name>
      <description>list of all actions possible with the organization.</description>
      <condition>
      <and>
      <eq>
          <var>ORGANIZATION_TYPE</var>
          <const type="string">RETAILER</const>
       </eq>
          <eq>
          <var>ACTION</var>
          <const type="string">RFCIN</const>
       </eq>
 </and>
    </condition>
      <action>
       <assign>
          <var>ACTION_LIST</var>
          <const type="string">RFCIN- Generate RFCIN Message</const>
          <const type="string">FACTSHEET- Generate factsheet</const>
    </assign>
      </action>
</constraint>
<constraint>
      <name>Action List 2</name>
      <description>list of all actions possible with the organization.</description>
      <condition>
    <and>
      <eq>
          <var>ORGANIZATION_TYPE</var>
          <const type="string">RETAILER</const>
       </eq>
          <eq>
          <var>ACTION</var>
          <const type="string">FACTSHEET</const>
       </eq>
     </and>
    </condition>
      <action>
       <assign>
          <var>ACTION_LIST</var>
          <const type="string">FACTSHEET- Generate factsheet</const>
          <const type="string">RFCIN- Generate RFCIN Message</const>
          </assign>
      </action>
</constraint>
```

In the ACTION_LIST, FACTSHEET and RFCIN are must and description after "-"is Configurable. We can change the description as required. Default value populated for ACTION on the page is RFCIN on retailer enterprise and FACTSHEET on supply side.

RFCIN Handling - Retailer Side

The following is a pictorial representation of how RFCIN messages are handled on the Retailer side:

RFCIN Handling - Retailer Side



Customization MarketPlace and TradingPartner Credentials

You can view or hide the marketplace credentials by populating CREDENTIAL_LIST.

The following are valid values:

- Marketplace
- TradingPartner

You can either populate just market place or both marketplace and trading partner, but just trading partner is invalid. For example:

```
<assign>
    <var>CREDENTIAL_LIST</var>
        <const type="string">Marketplace</const>
        <const type="string">TradingPartner</const>
    </assign>
```

Customization of User Defined Attributes

There are eight user defined attributes as follows:

- two Strings
- two dates
- two numbers
- two booleans

All these variables can be named as required by the user. The following illustrates the assigning of user defined attributes:

```
<assign>
    <var>USERDEFINED_LIST</var>
    <const type="string">Date1_Name:Start Date</const>
    <const type="string">Date2_Name:End date</const>
    <const type="string">boolean1:boolean1</const>
    <const type="string">boolean2:boolean2</const>
    <const type="string">String1_Name:Special Instructions</const>
    <const type="string">String2_Name:String2_Name</const>
```

```
    <const type="string">Integer1_Name:Integer1_Name</const>
    <const type="string">Integer2_Name:Integer2_Name</const>
<assign>
```

Names on the left of ':' are constants and cannot be changed. Those to the right of ':' are configurable and are displayed on the GUI.

Setting up GPC Classification Scheme Data Load (GDSN Only)

The information is applicable only for the GDSN version of the software.

Uploading GPS Codes

The GPC data source is provided as an out-of-the-box setup in the TIBCOCIM enterprise.

Procedure

1. Add the following dataset to FileWatcher.xml.

```
<DataSet type="single">
<!--
```

There must be no MasterCatalog Element when importing a Standard Classification Scheme such as GPC, UDEX.

```
-->
<Name>DataSetS2</Name>
<Credential domain="ZZ">
<!-- This is a pre-defined credential-->
<Identity>GLOBAL</Identity>
</Credential>
<Action>ImportClassificationScheme</Action>
<URIInfo scheme="local">
<Relative>MQ_COMMON_DIR</Relative>
<!-- This directory is created outOfBox -->
<URI>velosel/GPC</URI>
</URIInfo>
```

```
<Incremental>Yes</Incremental>
<DataSource>
<RevisionID>
<!-- This a pre-defined data source in Velosel enterprise -->
<BaseName>GPC</BaseName>
</RevisionID>
</DataSource>
<ClassificationScheme>
<RevisionID>
<!-- This a pre-defined classification scheme -->
<BaseName>GPC</BaseName>
</RevisionID>
</ClassificationScheme>
</DataSet>
```

2. Ensure that the following sub-directories exist under the $MQ_COMMON_DIR/velosel/GPC/ directory.

```
incoming
rejected
done
```

3. Place the data file (.csv) in the $MQ_COMMON_DIR/velosel/GPC/incoming directory. The .csv file is used for upload when the FileWatcher polling interval time is over. All the rows from the .csv file are uploaded into the GPC data source. If the upload is successful, the .csv file is copied into the done directory.

Adding GPC Drop-Downs for Editing Records in a Repository

Procedure

1. For dependent drop-downs, add the following section to your catalog validation rulebase:

```
<declare>
<var>GPC_classification</var>
    <link type="datasource">
```

```
            <literal>GPC</literal>
        </link>
    </declare>
    <constraint>
        <name>GPC CODE</name>
        <description>GPC CODE dropdown</description>
        <usefor>
            <var>GPC_SEGMENT_CODE</var>
        </usefor>
        <action>
            <select novalue="option">
            <table source="classification">
                <literal>GPC_classification/SEGMENT_CODE</literal>
                <literal>GPC_classification/SEGMENT_DESC</literal>
            </table>
            </select>
        </action>
    </constraint>
    <constraint>
        <name>FAMILY CODE</name>
        <description>GPC Hierarchy Level 2 dropdown</description>
        <usefor>
            <var>GPC_FAMILY_CODE</var>
        </usefor>
        <condition>
            <defined>
                <var>GPC_SEGMENT_CODE</var>
            </defined>
        </condition>
        <action>
            <select novalue="default">
                <table source="datasource">
                    <literal>GPC</literal>
                    <literal>FAMILY_CODE</literal>
                    <literal>FAMILY_DESC</literal>
                    <where>
                        <sql>
                        <eq>
                <literal>SEGMENT_CODE</literal>
                <const type="string">?</const>
                        </eq>
                        </sql>
                        <var>GPC_SEGMENT_CODE</var>
```

```
                    </where>
                </table>
            </select>
        </action>
</constraint>
<constraint>
        <name>CLASS CODE</name>
        <description>GPC Hierarchy Level 3 dropdown</description>
        <usefor>
            <var>GPC_CLASS_CODE</var>
        </usefor>
        <condition>
            <defined>
                <var>GPC_FAMILY_CODE</var>
            </defined>
        </condition>
        <action>
            <select novalue="default">
                <table source="datasource">
                    <literal>GPC</literal>
                    <literal>CLASS_CODE</literal>
                    <literal>CLASS_DESC</literal>
                    <where>
                        <sql>
                            <and>
                            <eq>
                            <literal>SEGMENT_CODE</literal>
                            <const type="string">?</const>
                            </eq>
                            <eq>
                    <literal>FAMILY_CODE</literal>
                    <const type="string">?</const>
                            </eq>
                            </and>
                        </sql>
                        <var>GPC_SEGMENT_CODE</var>
                        <var>GPC_FAMILY_CODE</var>
                    </where>
                </table>
            </select>
        </action>
</constraint>
<constraint>
```

```
        <name>BRICK CODE</name>
        <description>GPC Hierarchy Level 4</description>
        <usefor>
            <var>ADDITIONAL_CLASS_CAT_CODE</var>
        </usefor>
        <condition>
            <defined>
                <var>GPC_CLASS_CODE</var>
            </defined>
        </condition>
        <action>
            <select novalue="default">
                <table source="datasource">
                    <literal>GPC</literal>
                    <literal>BRICK_CODE</literal>
                    <literal>BRICK_DESC</literal>
                    <where>
                        <sql>
                            <and>
                            <eq>
                    <literal>SEGMENT_CODE</literal>
                    <const type="string">?</const>
                            </eq>
                            <eq>
                    <literal>FAMILY_CODE</literal>
                    <const type="string">?</const>
                            </eq>
                            <eq>
                    <literal>CLASS_CODE</literal>
                    <const type="string">?</const>
                            </eq>
                            </and>
                        </sql>
                        <var>GPC_SEGMENT_CODE</var>
                        <var>GPC_FAMILY_CODE</var>
                        <var>GPC_CLASS_CODE</var>
                    </where>
                </table>
            </select>
        </action>
    </constraint>
```

2. Save the catalog validation rulebase.

Data Quality Process

The Data Quality process is used to derive unique, standardized, and complete master data. Data Quality routines ensure that the data entered in a repository is "golden" so that data can be managed appropriately. If data quality is low, the repository contains two or more records for the same logical item, resulting in duplication of data.

To avoid duplicate records, you must standardize all similar records in a consistent form. After data standardization, you can successfully check for duplicates. This process of standardizing the data is also known as data cleansing.

Even after data cleansing, it can sometimes be difficult to determine whether a record is new or a variation (that is, a version) of an existing record. You might have to implement a mix of automated decisions (for most of the records) and some human intervention to decide whether a record is new or existing. This is often not a simple decision. For example, deciding whether two persons are the same is difficult when a reliable (or unique) ID is missing. It, typically, depends on the nature of the data and, in particular, which attributes are used for identification. In case a reliable or unique ID is supplied, deduplication is not required. However, this represents an ideal scenario, which rarely occurs in the real world.

Match-and-find options to locate duplicate records

In TIBCO MDM, you can use the following options to match and find duplicate records:

- **Out-of-the-box matching:** Performs matching using the and operator for all the specified attributes names. You can conduct "similarity or fuzzy" searches based on certain identifying attributes and detect a duplicate in the data and merge the data during a single or bulk record. For example, "FIRST_NAME" = "tom" And "LAST_NAME" ="Campbell". See Duplicate Record Detection Process.
- **Custom matching:** Integrates the custom matcher. You can implement the customized Netrics query by writing the custom code in Java. Compile and merge the custom code in ECM.ear. For example, "FIRST_NAME" = "tom" And "LAST_NAME" ="Campbell" "OR" "FIRST_NAME" = "tom" And "SSN_ID" ="987 XX XXXX". See Custom Netrics Query.
- **Custom matching using the query builder platform of TIBCO® Patterns - Search:** You can use the query builder platform tool in TIBCO Patterns - Search to search for records. The query is built and passed in an XML file. This XML file is

passed to the MatchRecord activity. TIBCO MDM uses the TIBCO Patterns - Search query and locates matching records. After matching records, the records are merged automatically based on the configuration defined in the auto-selection record configuration file. See Custom Matching Using Query Builder Platform of TIBCO Patterns - Search.

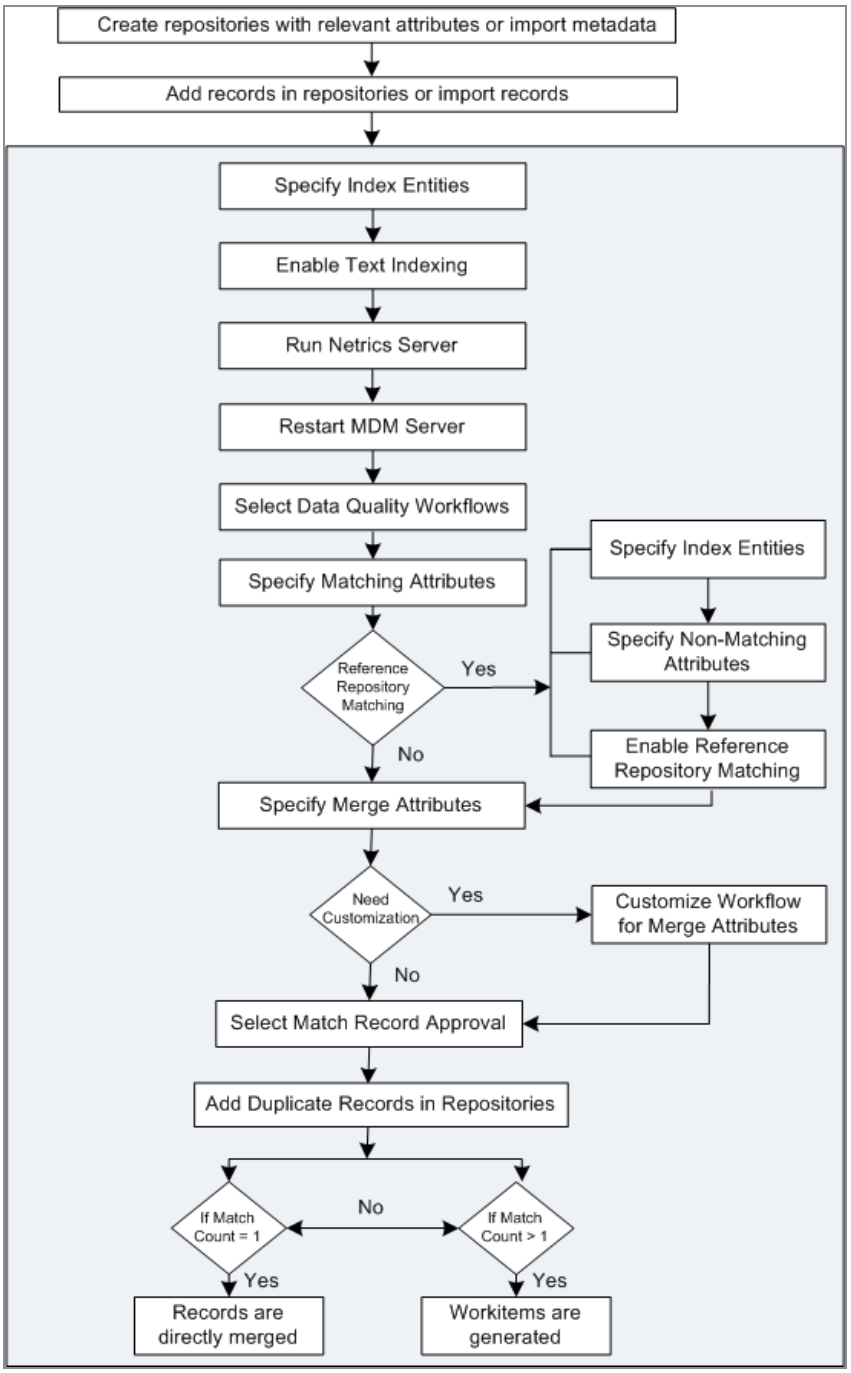
For information about the custom matching process by using the query builder platform tool, see Custom Matching Process.

Duplicate Record Detection Process

To detect duplicates, you can conduct "similarity or fuzzy" searches based on certain identifying attributes.

You can detect a duplicate with small variations in the data and merge the data during a single or bulk record add or modify workflow. With the duplicate detection routine, you can run a fuzzy query based on a configurable record attribute and then return one or more potential matches. TIBCO Patterns - Search or Advanced Matching Engine is the search engine used for text search and record matching. You can match and deduplicate records in bulk. Using the business process rule process definition selection, you can override the default upload or import process to choose the data quality import process. This direct load import process calls the match and merge activities to detect duplicates and deduplicate.

Duplicate Record Detection Process



The following components are involved in this process:

- Index Entities
- Enabling Text Indexing
- Running TIBCO Patterns - Search Server and Restarting TIBCO MDM Server

- Selecting Data Quality Workflows
- Specifying Matching Attributes
- Enabling Reference Repository Matching
- Specify Merge Attributes
- Customizing Workflow for Merge Attributes
- Selecting Match Record Approval
- MergeRecord Activity

Index Entities

To match and locate duplicate records in a repository, index the repositories in an IndexerConfig.xml file. In case of Reference Repository Matching, index both the source and target repositories. The specified attributes in this file are indexed.

For more information about index configuration, see the *TIBCO MDM System Administration* guide.

Cross-Repository Matching

Scenario 1

Single parent contains a single child

```
<IndexEntity joinTable="true">
  <Name>CustomerAddressIndexEntity</Name>
  <EnterpriseName>ABC</EnterpriseName>
  <Repository>
   <RepositoryName>CUSTOMER</RepositoryName>
    <AttributeList>
     <Attribute>
       <AttributeName>FIRSTNAME</AttributeName>
     </Attribute>
     <Attribute>
       <AttributeName>LASTNAME</AttributeName>
     </Attribute>
    </AttributeList>
    <Relationship>
```

```
      <RelationshipName>RESIDENCEADDRESS</RelationshipName>
      <RelatedRepository>ADDRESS</RelatedRepository>
    </Relationship>
  </Repository>
  <Repository>
   <RepositoryName>ADDRESS</RepositoryName>
    <AttributeList>
     <Attribute>
      <AttributeName>CITY</AttributeName>
     </Attribute>
     <Attribute>
      <AttributeName>ZIP</AttributeName>
     </Attribute>
    </AttributeList>
  </Repository>
</IndexEntity>
```

Scenario 2

Single parent contains multiple children (Cross-repository overlapping)

Specify indexing as follows:

Customer > ResidenceAddress> Address

Customer > CustomerToAcc > Account

Ensure that the Index entity name is unique. Example is as follows:

```
<IndexEntity joinTable="true">
  <Name>CustomerAddressIndexEntity</Name>
  <EnterpriseName>ABC</EnterpriseName>
  <Repository>
   <RepositoryName>CUSTOMER</RepositoryName>
    <AttributeList>
     <Attribute>
      <AttributeName>FIRSTNAME</AttributeName>
     </Attribute>
     <Attribute>
      <AttributeName>LASTNAME</AttributeName>
     </Attribute>
    </AttributeList>
    <Relationship>
      <RelationshipName>RESIDENCEADDRESS</RelationshipName>
      <RelatedRepository>ADDRESS</RelatedRepository>
    </Relationship>
```

```
  </Repository>
  <Repository>
   <RepositoryName>ADDRESS</RepositoryName>
    <AttributeList>
     <Attribute>
      <AttributeName>CITY</AttributeName>
     </Attribute>
     <Attribute>
      <AttributeName>ZIP</AttributeName>
     </Attribute>
     </AttributeList>
  </Repository>
</IndexEntity>
<IndexEntity joinTable="true">
  <Name>CustomerBankIndexEntity</Name>
  <EnterpriseName>ABC</EnterpriseName>
  <Repository>
   <RepositoryName>CUSTOMER</RepositoryName>
    <AttributeList>
     <Attribute>
      <AttributeName>FIRSTNAME</AttributeName>
     </Attribute>
     <Attribute>
      <AttributeName>LASTNAME</AttributeName>
     </Attribute>
    </AttributeList>
    <Relationship>
      <RelationshipName>CUSTOMERTOACC</RelationshipName>
      <RelatedRepository>ACCOUNT</RelatedRepository>
    </Relationship>
  </Repository>
  <Repository>
   <RepositoryName>ACCOUNT</RepositoryName>
    <AttributeList>
     <Attribute>
      <AttributeName>BANKNAME</AttributeName>
     </Attribute>
     </AttributeList>
  </Repository>
</IndexEntity>
```

Reference Repository Matching

```
<IndexEntity joinTable="false">
   <Name>CustomerStagingIndexEntity</Name>
```

```
    <EnterpriseName>ABC</EnterpriseName>
     <Repository>
      <RepositoryName>CustomerStaging</RepositoryName>
     <AttributeList>
       <Attribute>
         <AttributeName>FirstName</AttributeName>
       </Attribute>
       <Attribute>
         <AttributeName>LastName</AttributeName>
       </Attribute>
     </AttributeList>
     </Repository>
</IndexEntity>
<IndexEntity joinTable="false">
   <Name>CustomerReferenceEntity</Name>
   <EnterpriseName>ABC</EnterpriseName>
     <Repository>
       <RepositoryName>CustomerReference</RepositoryName>
    <AttributeList>
       <Attribute>
         <AttributeName>FName</AttributeName>
       </Attribute>
       <Attribute>
         <AttributeName>LName</AttributeName>
       </Attribute>
     </AttributeList>
     </Repository>
</IndexEntity>
```

Enabling Text Indexing

Procedure

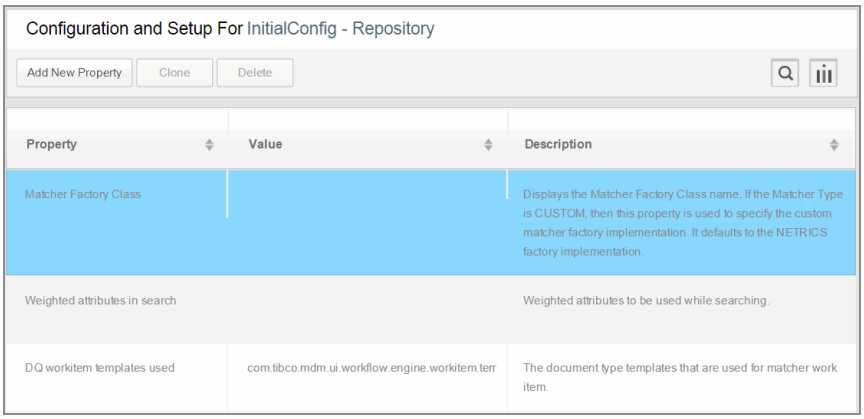
1. Start Configurator.
2. Navigate to **InitialConfig > Repository**. By default, **Advanced Matching Engine** is selected for the Matcher Type property.
3. Select **ONLINE** or **OFFLINE** for the Text Indexing Enabled property.
4. Click the Node ID from Cluster Outline.
5. Go to **Async Task Management** in the Advanced configuration outline.
6. Locate and set the value of the Text Indexing Receiver Pool Size to 1.

The text indexing is enabled.

CUSTOM Configuration

You can select the CUSTOM value for the MatcherType property. The matcher implementation uses the **MatchingScore input** parameter of the MatchRecord activity to perform the match operation.

For the CUSTOM type, use the Matcher Factory Class configuration property to specify the custom match factory implementation.



The custom factory implementation extends the MatcherFactory class and overrides the following method:

```
public IMatcher getMatcher() throws MqException;
```

For more information about indexing, see the Search and Matching chapter of the *TIBCO MDM System Administration*.

Running TIBCO Patterns - Search Server and Restarting TIBCO MDM Server

Procedure

1. After enabling text indexing, run the netricsServer.bat or netricsServer.sh utility.

   The utility is available in the $MQ_HOME/bin folder.

For more information about the options available in the utility, see the Search and Matching chapter of the TIBCO MDM *System Administration*.

2. After updating the IndexerConfig.xml file, restart the TIBCO MDM server or run the textIndexMigration.sh or textIndexMigration.bat utility using the -cf option. For more information about Text Index Migration, see the TIBCO MDM *System Administration*.

Selecting Data Quality Workflows

A deduplication process is available out-of-the-box and can be adapted for a specific implementation. The Process Definition Selection business process rule allows you to select the out-of-the-box Data Quality workflows.

Procedure

1. Click **Business Processes** in the menu.

   The Business Processes page displays a list of all available business processes.

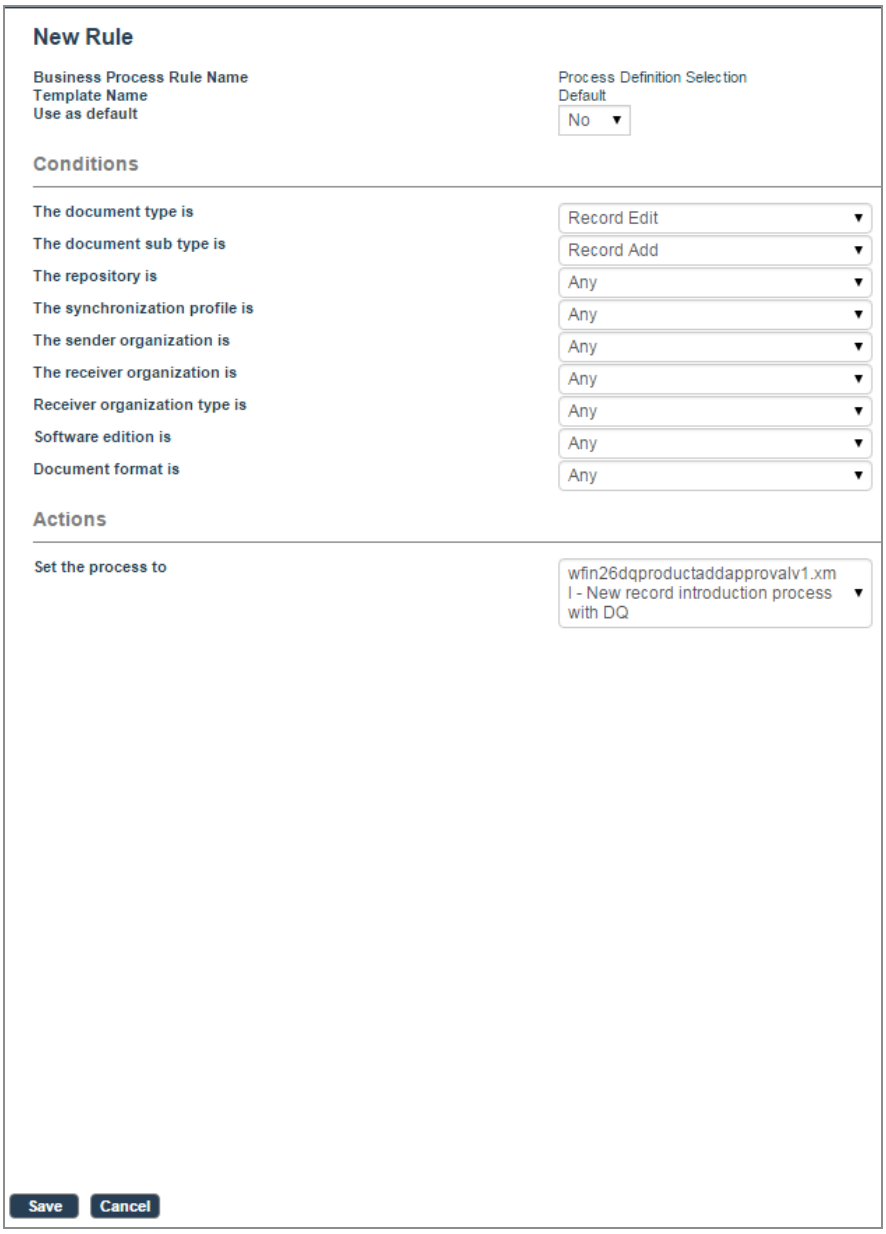
2. Click the **Process Definition Selection** rule.

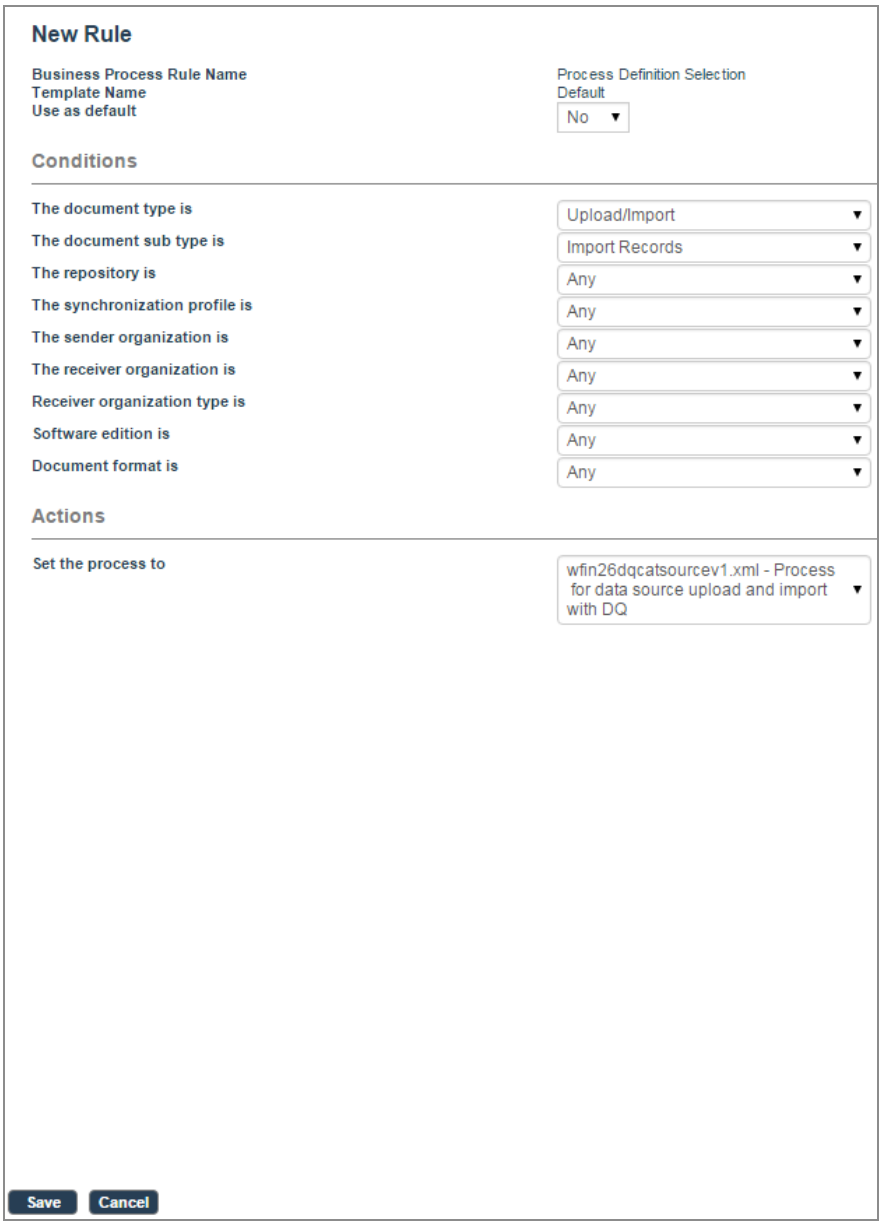
   The Rule Template page is displayed.

3. Click the **Default** template.

   The Rule List page is displayed.

4. Click **Create**. The New Rule page is displayed.


New Rule
Business Process Rule Name
Template Name
Use as default
Process Definition Selection
Default
No
Conditions
The document type is
Record Edit
The document sub type is
Record Add
The repository is
Any
The synchronization profile is
Any
The sender organization is
Any
The receiver organization is
Any
Receiver organization type is
Any
Software edition is
Any
Document format is
Any
Actions
Set the process to
wfin26dqproductaddapprovalv1.xml - New record introduction process with DQ
Save
Cancel



Select the following conditions:

For Process	Description	Conditions and Actions
Record Add Process	when a new record arrives, the process	• Document type > Record Edit

For Process	Description	Conditions and Actions
	determines whether similar records exist in the repository.	• Document subtype > Record Add • Set the process to > wfin26dqproductaddapprovalv1.xml Or • Document type > Record Edit • Document subtype > Import Sub-workflow • Set the process to > wfin26dqcatAddRecordApprovalv1.xml
Direct Load Import Process	In the case of bulk import, it is assumed that the incoming records are sufficiently standardized for duplicate detection.	• Document Type > Upload/Import • Document subtype > Import Records • Set the process to > wfin26dqcatsourcev1.xml or wfin26dqcatsourceimportv1.xml

Note: Select the wfin26dqcatAddRecordApprovalv1.xml and wfin26dqcatsourceimportv1.xml workflows to process import records sequentially and to detect duplicate records in the imported file.

5. Click **Save**.

   The created rules are saved in the Rule list.

Specifying Matching Attributes

Procedure

1. To match records, specify matching attributes in a rulebase file.

   For example, see the MatchAttrList.xml file located in the $MQ_COMMON_DIR/standard/rulebase folder. Using matching attributes, you can search for close, not necessarily exact, matching records.

Note: If you copy the MatchAttrList.xml file in $MQ_COMMON_DIR*enterprise internal Name*\rulebase folder, you must change its location in relevant workflow files.

Note: While specifying index entity attributes in the rulebase:

- Ensure that the number of attributes specified in MatchAttributeArray and IndexEntityAttributes must match, otherwise the IndexEntityAttributes are ignored.
- Specify the RepositoryName followed by the attribute name. Entity name specified for a target repository is the target repository name specified in the IndexerConfig.xml file.
- In the case of RRM (reference repository matching), specify the MatchingAttributes and IndexEntityAttributes in the hierarchy level because the MatchRecord activity uses the MatchAttributesArray and IndexEntityAttributes in the exact specified order. For example, if CustomerStaging:FIRSTNAME is the first attribute specified under MatchAttributesArray, and CustomerReferenceEntity:FNAME is the first attribute specified under IndexEntityAttributes, the MatchRecord activity accesses the FIRSTNAME attribute's value (For example, JOHN) from the CustomerStaging record and uses this value to query the CustomerReference index entity using the FName attribute.

Simple Matching

In the simple matching process, specify only record attributes as matching criteria in the rulebase file.

For example, for a Customer repository, the matching attributes might be FirstName, MiddleName, and LastName. If any of these attributes' values match an existing record up to the specified minimum matching threshold, the record can be a matching record candidate. The specified record attributes are searched in all repositories and the matching result displays the values of matching attributes for all repositories.

MatchAttributesArray

Match Attributes Array lists the record attributes to be used as matching criteria. For example,

```
<constraint>
   <name>Attribute</name>
   <description>match attribute List</description>
    <action>
      <assign>
         <var>MatchAttributeArray</var>
         <const type="string">FIRSTNAME</const>
         <const type="string">MIDDLENAME</const>
         <const type="string">LASTNAME</const>
      </assign>
   </action>
</constraint>
```

Cross-Repository Matching and Overlapping

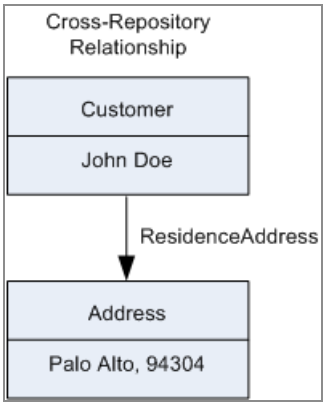
In the cross-repository matching process, specify attributes across related repositories as matching criteria.

Scenario 1

Single parent contains single child.

Consider that a Customer repository is related to the address repository by ResidenceAddress relationship. You want to detect near-duplicate or duplicate records for a customer John Doe, who has a residence address Palo Alto, 94304.

Cross-Repository Matching



- **MatchAttributesArray**: List the record attributes of both the repositories to be used as matching criteria. For example, FirstName and LastName from the customer

repository and City and ZIP from the address repository.

You can also specify the weightage of an attribute. You can decide which attribute you want to provide more weightage for matching records. For example, the last name of a person is unique and not repeated frequently. Hence, specify weightage for the LASTNAME attribute. To specify weightage, use the caret sign (^). For example, LASTNAME^0.7 as in the following code block:

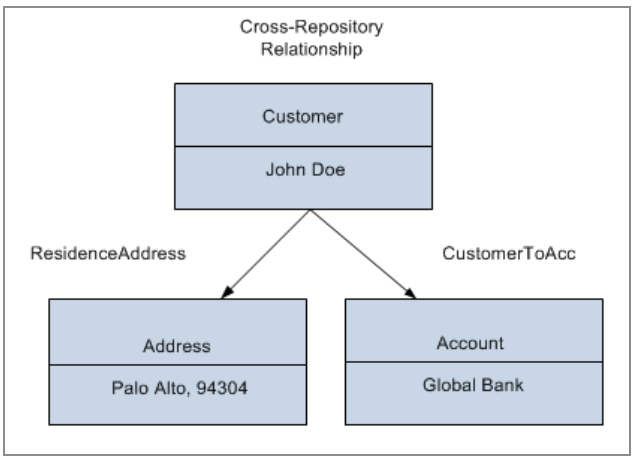
```
<constraint>
   <name>Attribute</name>
   <description>match attribute List</description>
    <action>
      <assign>
        <var>MatchAttributeArray</var>
        <const type="string">Customer:FIRSTNAME</const>
        <const type="string">Customer:LASTNAME^0.7</const>
        <const type="string">Address:CITY</const>
        <const type="string">Address:ZIP</const>
      </assign>
   </action>
</constraint>
```

Scenario 2

Single parent contains multiple children (Cross-repository Overlapping).

Consider that the Customer repository is related to the address and account repositories by the **ResidenceAddress** and **CustomerToAcc** relationships.

Cross-Repository Overlapping



- **MatchAttributesArray**: List the record attributes to be used as matching criteria

across multiple related repositories. For example,

```
<constraint>
   <name>Attribute</name>
   <description>match attribute List</description>
    <action>
      <assign>
         <var>MatchAttributeArray</var>
         <const type="string">Customer:FIRSTNAME</const>
         <const type="string">Customer:LASTNAME</const>
         <const type="string">Address:CITY</const>
         <const type="string">Address:ZIP</const>
         <const type="string">Account:BANKNAME</const>
      </assign>
   </action>
</constraint>
```

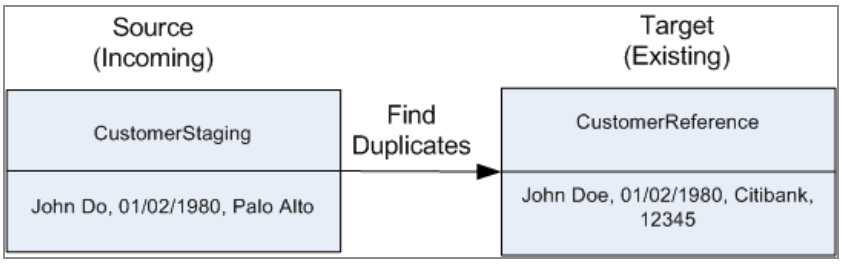
Reference Repository Matching

In the reference repository matching process, you can locate duplicate records in different repositories.

- If a single match is found in the target repository, go to direct merge. Based on the specified merge attributes, records are automatically merged.
- If multiple duplicate records are found in the target repository, records are displayed in the matcher work item. Based on the merge attributes, you can merge the records.

> **Note:** Relationships are not supported in the Reference Repository matching process.

Reference Repository Matching



For different repositories, you must specify matching attributes criteria of one repository and index entity attributes of another repository against which you want to match records. These two sets of repository attributes correspond to one another while matching the records. MatchAttributesArray and IndexEntityAttributes are in one file, MatchAttrList.xml.

- **MatchAttributesArray**: List the record attributes of a source repository to be used as matching criteria.

```
<constraint>
 <name>Attribute</name>
  <description>match attribute List</description>
   <action>
    <assign>
     <var>MatchAttributeArray</var>
      <const type="string">CustomerStaging:FIRSTNAME</const>
      <const type="string">CustomerStaging:LASTNAME</const>
      <const type="string">CustomerStaging:DateofBirth</const>
    </assign>
   </action>
</constraint>
```

  - **IndexEntityAttributes**: List the record attributes of a target repository to be mapped to the IndexEntityName table attributes of a target repository.

    Note: IndexEntityAttributes are declared in another constraint.

```
<assign>
        <var>IndexEntityAttributes</var>
   <const type="string">CustomerReferenceEntity:FNAME</const>
   <const type="string">CustomerReferenceEntity:LNAME</const>
   <const type="string">CustomerReferenceEntity:DOB</const>
     </assign>
```

Specifying NonMatching Attributes

To display non-matching attributes of two different repositories in the work item, specify those in the catalogvalidation.xml file of the source repository.

Procedure

1. Navigate to the $MQ_COMMON_DIR/<enterprise IternalName>/catalog/master/<specific repositoryID> directory.
2. Open the catalogvalidation.xml file.
3. Specify the non-matching attributes.

   For example,

```
<name>AttributeRRMMappingAttributes</name>
    <description>match attribute List RRMMappingAttributes. Each attribute must be qualified by EntityName.</description>
  <action>
    <assign>
      <var>RRMMappingAttributes</var>
      <const type="string">CustomerStaging:CITY=CustomerReference:BANKNAME</const>
    </assign>
  </action>
```

> **Note:** The Array name must be RRMMappingAttributes.

Based on this list, the attributes are displayed in the work item. You can merge matching and non-matching attributes.

> **Note:** You cannot merge multiple attributes of a source repository into a single attribute of a target repository. Hence, in this example you cannot map the AccNo attribute of the CustomerReference repository to the CustomerStaging repository. The non-mapping attributes are not displayed in the work item.

Enabling Reference Repository Matching

By default, the wfin26productaddinternaleditv1.xml workflow is used for both cross-repository overlapping and reference repository matching. However, you must explicitly enable the Reference Repository Matching settings in this workflow.

Procedure

1. Open the wfin26productaddinternaleditv1.xml file and comment the following lines:

```
<!--Transition FromActivity="Matcher" ToActivity="EmptyActivity"><Rule> <Parameter
direction="in" eval="variable" name="CrossRepositoryMatch" type="boolean"
>crossRepoMatch</Parameter> <Parameter direction="out" name="result" type="boolean"
></Parameter> <Condition format="bsh" ><![CDATA[ result = (CrossRepositoryMatch
!=null); System.out.println("CrossRepositoryMatch detected result : "+result); </Condition>
</Rule></Transition-->
```

2. Uncomment the following line in the EvaluateRulebase and MatchRecord actions:

```
<Parameter name="IndexEntityAttributes" direction="out" eval="variable"
type="arraylist">IndexEntityAttributes</Parameter>
```

Specify Merge Attributes

In the case of a large number of attributes, you might want to skip a few attributes and allow the rest of the attributes. Accordingly, you can specify either allow or skip attributes in a rulebase file.

To merge records, specify merging attributes based on any one of the two constraints: allow merge or skip merge.

In the case of Direct merge, if you do not specify allow or skip merge attributes in the rulebase, all attributes from source record except PRODUCTID and PRODUCTIDEXT are merged into the target record. However, in the case of Reference Repository Matching, only matching attributes are merged.

Specifying Allow Merge Attribute

Procedure

1. Navigate to the $MQ_COMMON_DIR/standard/rulebase folder.
2. Open the sample allowAttrList.xml rulebase file.
3. Specify the attribute name.

Note: For the reference repository matching, you must specify allow merge attributes in the following format:

```
<const type="string">CUSTOMERSTAGING:FirstName=CUSTOMERREFERENCE:FName</const>
<const type="string">CUSTOMERSTAGING:LastName=CUSTOMERREFERENCE:LName</const>
```

Specifying Skip Merge Attribute

Procedure

1. Navigate to the $MQ_COMMON_DIR/standard/rulebase folder.
2. Open the sample skipAttrList.xml rulebase file.
3. Specify the attribute name.

Note: For the reference repository matching, skip Merge is not supported.

Customizing Workflow for Merge Attributes

The SkipMergeAttributeList and AllowMergeAttributeList parameters in the MergeRecord activity evaluates merge or skip attributes. By default, the

SkipMergeAttributeList parameter is specified in the MergeRecord activity. For the allow merge functionality, you must customize the workflow.

Procedure

1. Navigate to the $MQ_HOME\common\standard\workflow folder.
2. Open the wfin26productaddinternaleditv1.xml file.

Result

- For Allow merge:
    - Add that the following parameter in MergeRecord action:

      ```
      <Parameter name="AllowMergeAttributeList" direction="in" eval="variable"
      type="arraylist">allowMergeAttributeList</Parameter>
      ```

    - Add the following parameter in EvaluateRuleBase action:

      ```
      <Parameter name="AllowMergeAttributeArray" direction="out" eval="variable"
      type="arraylist">allowMergeAttributeList</Parameter>
      ```

    - Specify the path of allow merge attribute rulebase file in EvaluateRuleBase action. For example,

      ```
      <Parameter name="Rulebase" direction="in" eval="constant" type="string">$MQ_
      COMMON_DIR/standard/rulebase/allowAttrList.xml</Parameter>
      ```

- For Skip merge:
    - SkipMergeAttributeArray is not supported for Reference Repository Matching and therefore, perform the following:

      Add comment to the following parameter in the MergeRecord action:

      ```
      <Parameter name="Rulebase" direction="in" eval="constant"
      type="string">standard/rulebase/skipAttrList.xml</Parameter>
      ```

      Add comment to the following parameter in the EvaluateRuleBase action:

      ```
      <Parameter name="SkipMergeAttributeList" direction="in" eval="variable"
      ```

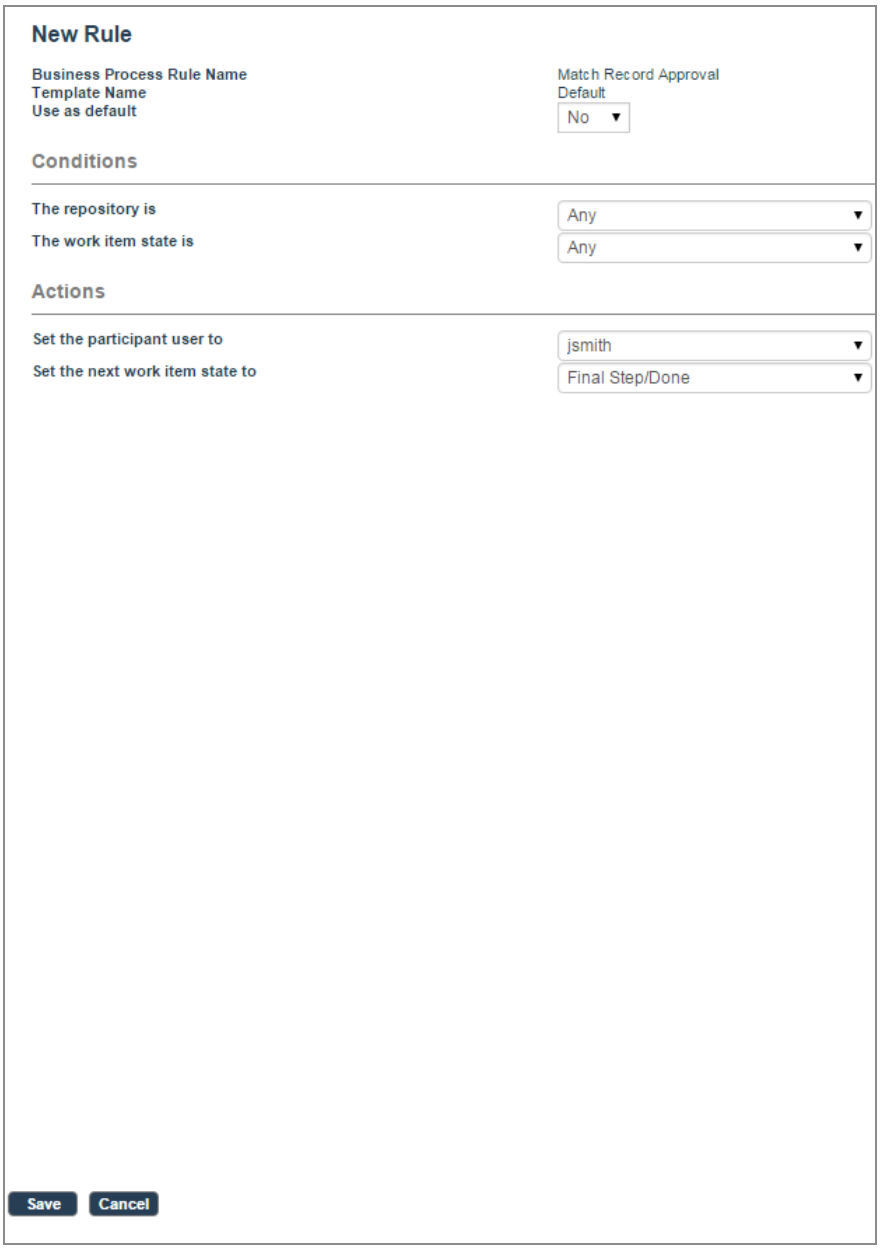
```
type="arraylist">skipMergeAttributeList</Parameter>
```

Based on the specified allow or skip attributes in the rulebase, the MergeRecord activity merges or skips attribute data. For more information about these parameters, see the MergeRecord Activity section in the *TIBCO MDM Workflow Reference* guide.

Selecting Match Record Approval

Procedure

1. Click **Business Processes** in the menu. The Business Processes page displays a list of all available business processes.
2. Click the **Match Record Approval** rule. The Rule Template page is displayed.
3. Click **Default** template. The Rule List page is displayed.
4. Click **Create**. The New Rule page is displayed. Select the following conditions:
   - Set the participant user to <*your user name*>
   - Set the next work item state to Final Step/Done



5. Click **Save**. The created rule is saved in the Rule list.

MatchRecord Activity

The MatchRecord activity performs the following:

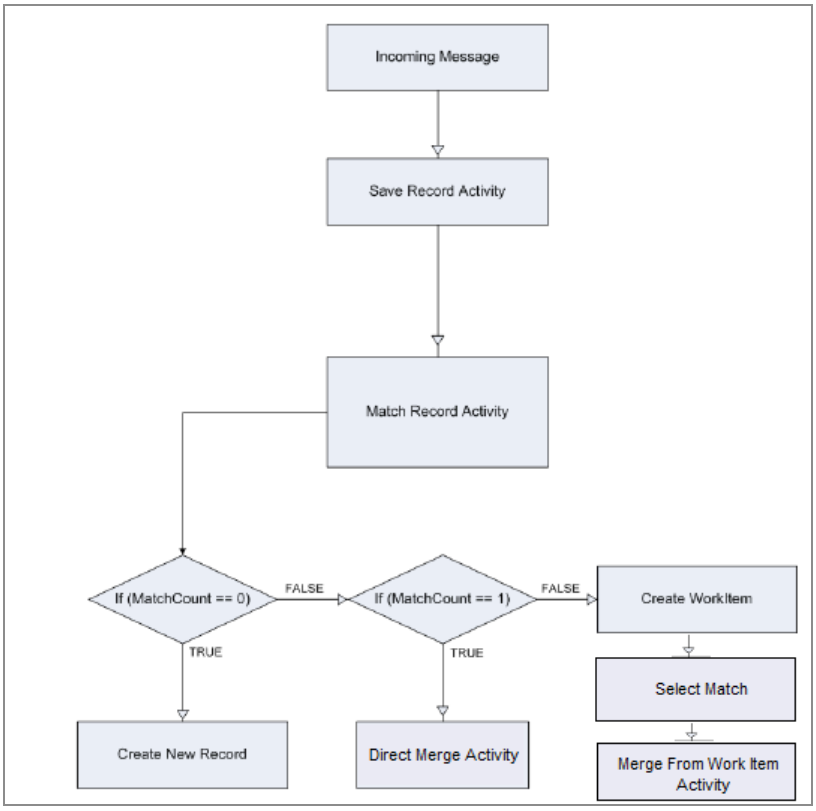
- Retrieves a source record list as an input and matches each source record against the existing repository data.

- Runs a fuzzy search query and returns potential matches.
- Identifies a matching record provided in a set of matching attribute criteria.

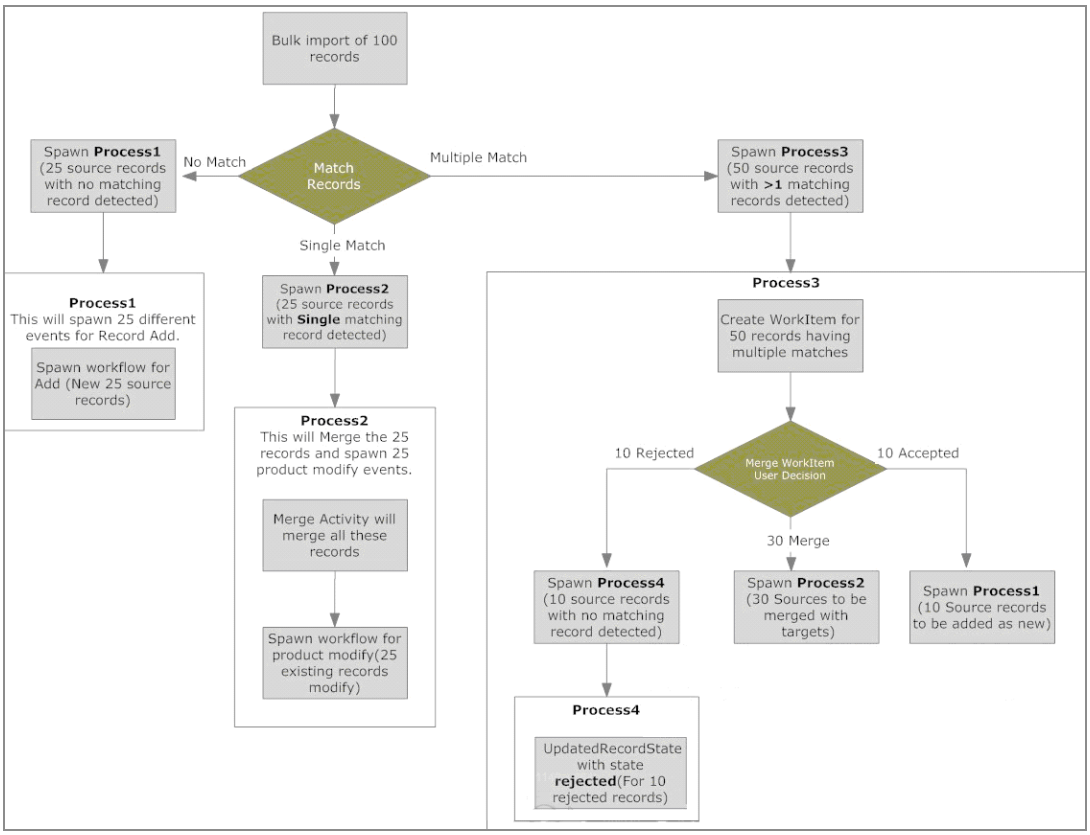
Based on the MatchingThreshold set in the MatchRecord activity, the output of the workflow is as follows:

- If one match is found, it is assumed that the match describes the same semantic record and the new record is merged into the existing record as a new version.
- If no match is found, the record is assumed to be a new one and is introduced as a new record in the repository.
- If multiple matches are found, a work item page is displayed where you can select one of the choices displayed. For more information, see Work Item.

The workflow for adding a single record for the Data Quality process is shown in the following diagram:

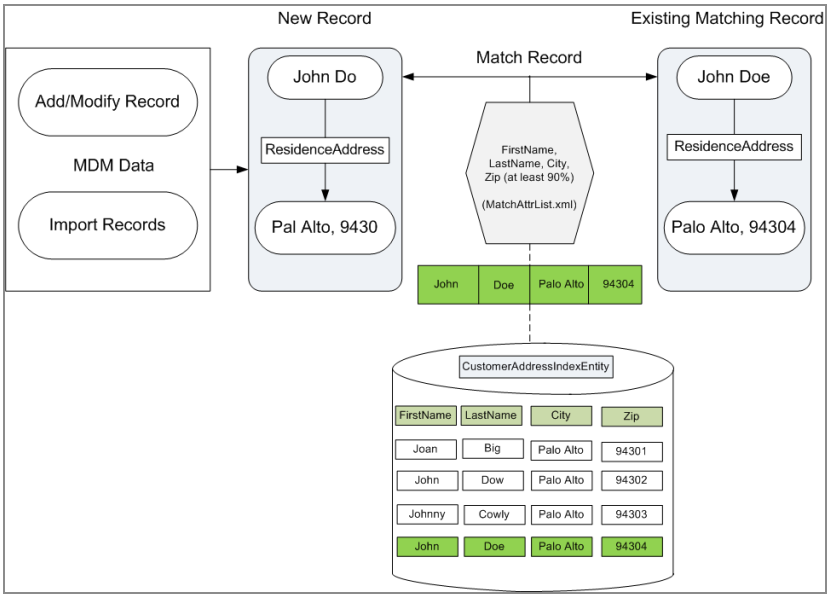


The workflow for bulk import is shown in the following diagram:



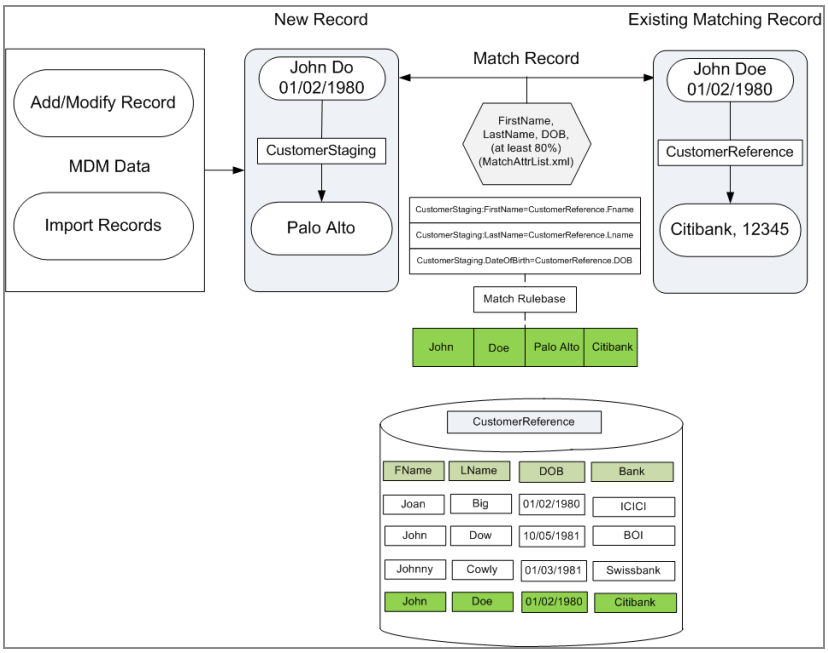
Cross-Repository Matching

The following workflow illustrates the Cross-Repository Matching process using the MatchRecord activity:



Reference Repository Matching

The following workflow illustrates the Reference Repository Matching process using MatchRecord activity:



For more details about the MatchRecord activity, see the *TIBCO MDM Workflow Reference* guide.

MergeRecord Activity

The MergeRecord activity accepts the output of the MatchRecord activity, and then merges the source record with the target record. The source record refers to a new record, and the target record refers to an existing record.

Mutation of Records

If PRODUCTID or EXTN is specified in the release, the record is mutated based on the Allow Merge constraint. For more information about mutation, see the Modifying Records section of the *TIBCO MDM User's Guide*.

Condition for Mutation

Mutation is possible if a record with the same ID and EXTN does not exist in a repository.

MergeRecord Activity Modes

The MergeRecord activity works in the following three modes:

- Bulk
- InDocument
- Legacy

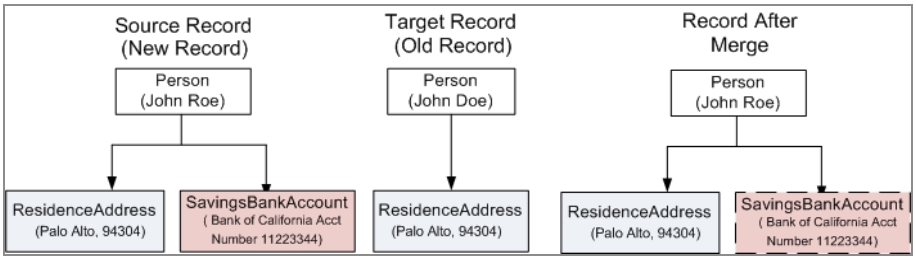
Bulk Mode

You can merge multiple records using the Bulk Mode. For example, you have imported 100 records and found a match for 50 source records. The work items are generated for 50 matched source records provided the MatchRecordApproval business process rule is set. For more details about merging records using work items, see Work Item.

During the merge process, the ReferenceStepID of the MergeRecord activity is mapped to the out parameter MatchRecordProcessLogID of the MatchRecord activity.

InDocument Mode

InDocument refers to the mIXML document that contains the source record to be merged with the target record. The InDocument mode allows a single record merge, relationship merge, and hierarchy merge. For example, you have added a new record through UI or the JMS message. You can verify whether this new record already exists in the application using the MatchRecord activity. If you found the exact match for this new record, you can directly merge the source record with the target record. For example, consider the following record hierarchy:



In this case, a new record, John Roe, has two related records: Address record Palo Alto, 94304 and BankAccount record Bank of California Acct Number 11223344. John Doe has an existing record, one related ResidenceAddress record Palo Alto, 94304 , which is found matching. However, the SavingsBankAccount relationship does not exist. Hence, while merging the records, the following records are merged:

- Primary Person record is merged.
- Attributes of the matching ResidenceAddress repository are merged.
- By default, the true value is specified for ImplicitRelationshipMerge parameter, therefore non-matching SavingsBankAccount relationship is also merged.

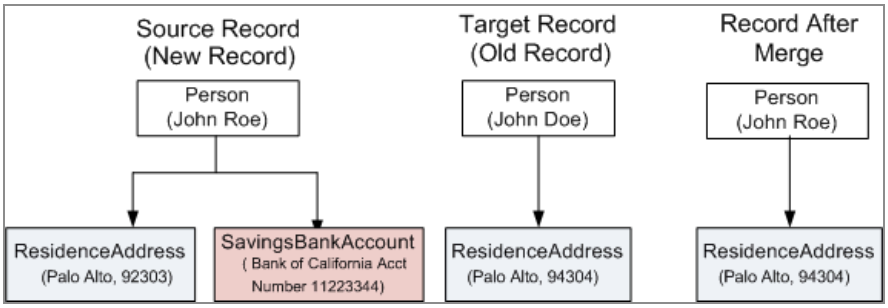
ImplicitRelationshipMerge works only for Direct merge. You can merge non-matching relationship records using the Implicit Relationship Merge. You can select not to merge such non-matching relationships implicitly by specifying the false value for the ImplicitRelationshipMerge parameter of the MergeRecord activity.

Legacy Mode

The Legacy mode is supported only with the release version 8.2 or later. Legacy mode uses the MergeRecordUsingLegacyProcess parameter of the MergeRecord activity. If you found a source record matching with the existing record, you can do two things. You can directly process it for merge or you can pass source and target records directly to the merge without evaluating the MatchRecord activity.

Note: The MatchRecord activity is optional for the legacy mode.

In the legacy mode, you can merge only one level of data. For example, consider the following record hierarchy:



In this case, a new record, John Roe has two related records: Address record Palo Alto, 92303 and BankAccount record Bank of California Acct Number 11223344 and the existing record John Doe has one related ResidenceAddress record Palo Alto, 94304.

If you specify the true value for the MergeRecordUsingLegacyProcess parameter, only root data, that is only Person record data is merged. Merging of hierarchy data and relationship merge are not supported, that is, attributes of the matching ResidenceAddress repository are not merged. The non-matching SavingsBankAccount relationship is also not merged.

For more details about the parameters of the MergeRecord activity, see the *TIBCO MDM Workflow Reference* guide.

Transitions

When the matcher work item activity is complete, that is, the work item is submitted, the process performs the following activities:

- MatcherWork Item >> SpawnWorkflowForAddNewRecords

  If there are some source records marked "Accept as new" on the work item, they are processed for an add new record workflow.

- MatcherWorkItem >> UpdateRecordStateAsRejected

  If there are some source records marked "Rejected" on the work item, they are processed for updating their state as **Rejected**.

- MatcherWork Item >> MergeRecordBulk

  If there are some source records marked "Merged" with existing records on the work item, they are processed for a merge decision.

This processing is based on the output parameters set by the work item activity on submission. The following output parameters are generated by the CreateWorkItem activity (see the MatchRecordWorkItemBulk activity in the bulk workflow wfin26productmergeapprovalv1.xml) to the process state when the work item gets submitted.

- RecordListForNew: Arraylist of recordkey IDs.
- RecordListForReject: Arraylist of recordkey IDs.
- MergeCount: Count of source records to be merged.
- MatchingRecordKey: Output only when a single input source record is merged with a target record, that is, the MergeCount is 1 and the RecordListForNew and RecordListForReject are empty/null.

```
<Transition FromActivity="MatchRecordWorkItemBulk"
ToActivity="UpdateRecordStateAsRejectedFromWI">
 <Description>reject records.</Description>
 <Rule><Parameter direction="in" eval="variable" name="RecordListForReject"
type="arraylist">RecordListForReject</Parameter>
 <Parameter direction="out" name="result" type="boolean" ></Parameter>
 <Condition format="java">com.tibco.mdm.workflow.engine.transition.
     WfSharedConditionTransition.isNotEmptyOrNull</Condition>
 </Rule>
</Transition>
<Transition FromActivity="MatchRecordWorkItemBulk"
ToActivity="SpawnWorkflowForAddNewRecordsFromWI">
 <Description>add records.</Description>
<Rule><Parameter direction="in" eval="variable" name="RecordListForNew"
          type="arraylist" >RecordListForNew</Parameter>
<Parameter direction="out" name="result" type="boolean" ></Parameter>
 <Condition format="java">com.tibco.mdm.workflow.engine.transition.
     WfSharedConditionTransition.isNotEmptyOrNull</Condition>
 </Rule>
</Transition>
<Transition FromActivity="MatchRecordWorkItemBulk"
ToActivity="MergeRecordBulkFromWI">
 <Description>merge records.</Description>
 <Rule><Parameter direction="in" eval="variable" name="MergeCount"
     type="long" >MergeCount </Parameter>
<Parameter direction="out" name="result" type="boolean"></Parameter>
<Condition format="java">com.tibco.mdm.workflow.engine.transition.
 WfSharedConditionTransition.isGreaterThanZero</Condition>
</Rule>
</Transition>
```

MultiMerge

With MultiMerge, you can merge one or multiple matched record hierarchies into one. Multiple records can be selected and merged from the work item or selected manually and merged from **Browse and Search**.

Work Item

When multiple matches are found, a work item page is displayed, where you can complete the following tasks:

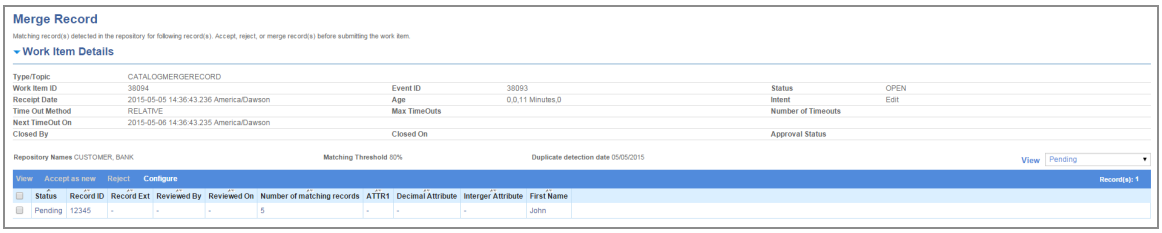
- Accept the record as a new record
- Merge a few attributes from the source record into the target record (attribute level merge)
- Merge the entire record as a new target record version (blanket merge)
- Reject the source record

Note: If you have customized the workflow for matcher work item, change the value of the DQ workitem templates used parameter to **com.tibco.mdm.ui.workflow.engine.workitem.templates.DQMatcherWorkItem** in the Configurator.

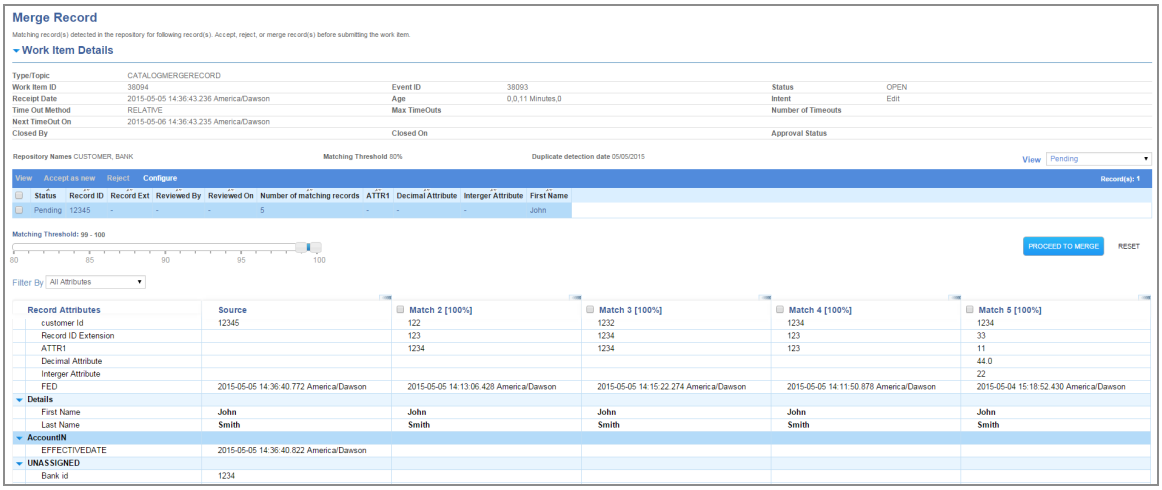
Types of Work Items

Based on your matching and configuration criteria, the following types of work item screens are displayed:

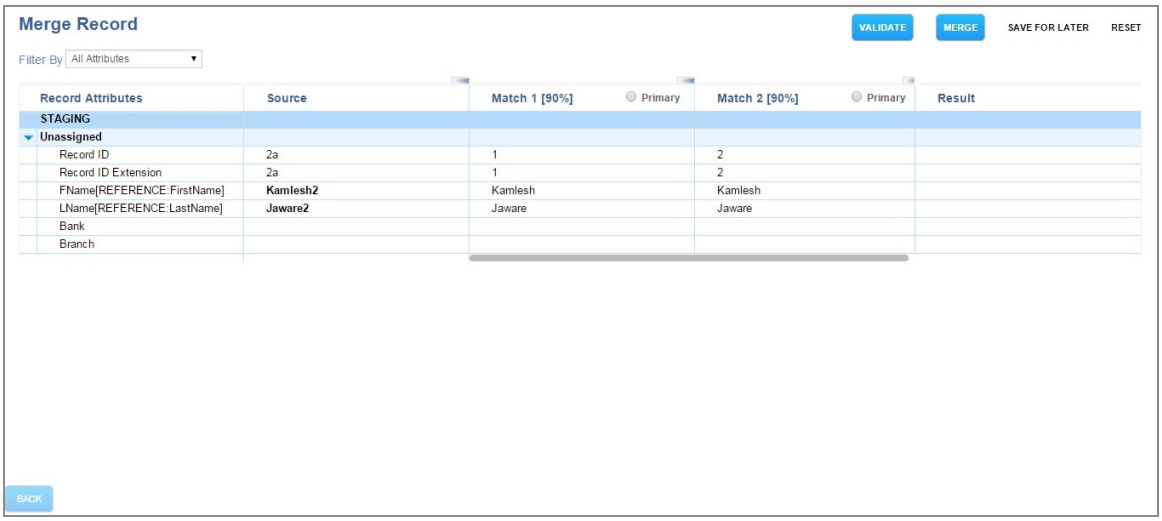
Cross-Repository Simple Matching Work Item



Cross-Repository Overlapping Work Item



Reference Repository Matching Work Item



Merge Record Page

When you open the work item, the Merge Record page is displayed.

The Merge Record page displays:

- Work item and repository details. For example, work item ID, event ID, time out details, repository name, and date of duplicate records detected.
- Record information in a tabular format. For example, the Record ID, Status,

Record Ext, Reviewed By, Reviewed On, and Number of matching records details are displayed.

- Matching Record details. For example, matching threshold and duplicate detection date.

From this page, you can do the following:

- Adjust the matching threshold to see only some of the records.
- Configure which attributes to be shown on pending record details.
- View the status of records displayed in the table based on whether the records are accepted as new, rejected, merged, or pending from the **View** drop-down list.
- Drag and drop values from the source record and matched records to the result column.
- Use the **VALIDATE** button to perform the validation process.
- Use the **RESET** button to reset the records selected for merging.
- Use the **SAVE FOR LATER** button to save the changes made. This option is not for manual merge. It is only available for merging a source record with an existing matching record.
- Accept a source record as a new record. See Accepting a Source Record as a New Record.
- Reject a record. See Rejecting Source Records.

Accepting a Source Record as a New Record

Procedure

1. Click **Accept as new** , then click **Process** to accept the selected source record as a new record.

   To accept all source records as new records in the repository, you can do one of the following:

   - Click **Done** directly without taking any other action. A confirmation dialog appears.
   - Click **Ok** to accept all source records as new in the repository. Accept as new

is the default action for all pending source records.

- Select all the check boxes by clicking the header check box.
- Click **Process**.

Rejecting Source Records

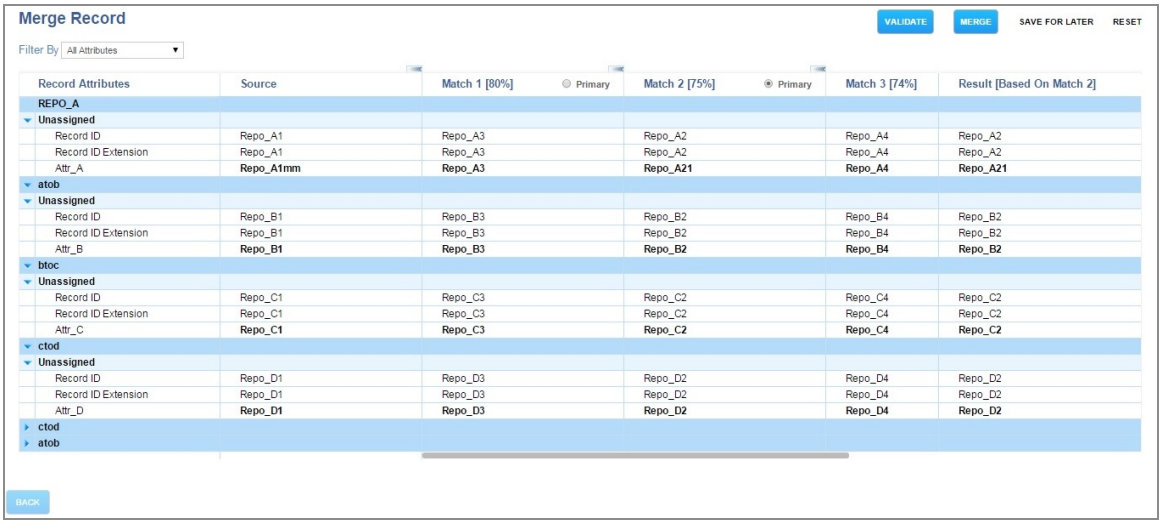
Procedure

1. Select the source record. To reject multiple source records, select the check box of the respective records.
2. Click **Reject**.
3. Click **Process** to reject the selected source record.

Merging Source with an Existing Matching Record

Procedure

1. Click the record to be merged. All available matching records are displayed as separate columns in this table.



Record Attributes	Source	Match 1 [80%]	Primary	Match 2 [75%]	Primary	Match 3 [74%]	Result [Based On Match 2]
REPO_A							
Unassigned							
Record ID	Repo_A1	Repo_A3		Repo_A2		Repo_A4	Repo_A2
Record ID Extension	Repo_A1	Repo_A3		Repo_A2		Repo_A4	Repo_A2
Attr_A	**Repo_A1mm**	**Repo_A3**		**Repo_A21**		**Repo_A4**	**Repo_A21**
atob							
Unassigned							
Record ID	Repo_B1	Repo_B3		Repo_B2		Repo_B4	Repo_B2
Record ID Extension	Repo_B1	Repo_B3		Repo_B2		Repo_B4	Repo_B2
Attr_B	**Repo_B1**	**Repo_B3**		**Repo_B2**		**Repo_B4**	**Repo_B2**
btoc							
Unassigned							
Record ID	Repo_C1	Repo_C3		Repo_C2		Repo_C4	Repo_C2
Record ID Extension	Repo_C1	Repo_C3		Repo_C2		Repo_C4	Repo_C2
Attr_C	**Repo_C1**	**Repo_C3**		**Repo_C2**		**Repo_C4**	**Repo_C2**
ctod							
Unassigned							
Record ID	Repo_D1	Repo_D3		Repo_D2		Repo_D4	Repo_D2
Record ID Extension	Repo_D1	Repo_D3		Repo_D2		Repo_D4	Repo_D2
Attr_D	**Repo_D1**	**Repo_D3**		**Repo_D2**		**Repo_D4**	**Repo_D2**
ctod							
atob							

2. Select the Target Matching Record by clicking the radio button of the appropriate column that you want to merge with the source record.
3. Click **Merge Record**.

The **Merge Record Details** page is displayed with the target record values.

By default, the **Show All** option is displayed in the **View** drop-down list. The other options are **Show Matcher Attributes** and **Show Different Values**. The highlighted rows in the Source Record values column are matcher attributes.

4. Select the source attribute value by selecting the check box in the **Source Record values** column.

   As you select individual source attributes, the values displayed in the **Target Record values** column change dynamically. The highlighted row in the **Target Record values** column represents the final value of the merged record.

5. Optional steps:
   - Click Add (+) to add a new relationship instead of merging attributes. This adds a new bulk of new related records.
   - Skip sub child record or, using the (x) icon, you can remove the sub child record.
   - To copy the cell values to the clipboard, click the **Copy cell value to the clipboard** icon. The icon is displayed when you move the mouse pointer over on a cell.
   - Drag and drop values from the source or matched record to the result column.
   - Edit any of the attributes in the result column by double clicking on the cell. To go back to the original value, click the arrow

     icon.
   - Click **VALIDATE** to perform the normal validation process.
   - Click **RESET** to reset the records selected for merging.
   - Click **SAVE FOR LATER** to save the changes made.
6. Click **Merge**.

> **Warning:** If you do not select any attribute and click **Merge**, all source record values apart from PRODUCTID and PRODUCTIDEXT are merged in the target record values. However, if you have specified any attributes as VIEW attributes in the rule base, those are not merged.

7. Optional: After clicking Merge, the Summary page displays the source records, modified record(s) text message, matched records. If needed, you can reject the source record.
8. Click **Submit**.

Result

The record is merged and the work item is closed.

TIBCO Patterns - Search Joins

Using the joined tables feature of TIBCO Patterns - Search to do a joined search eliminates the issues with de-normalization and merging of separate query results. With TIBCO Patterns - Search Joins, you can search data spread across multiple tables in a way that the fields of each table appear to be merged into a single table.

For implementation of TIBCO Patterns - Search Joins, the normalized attribute has been added to Index entity:

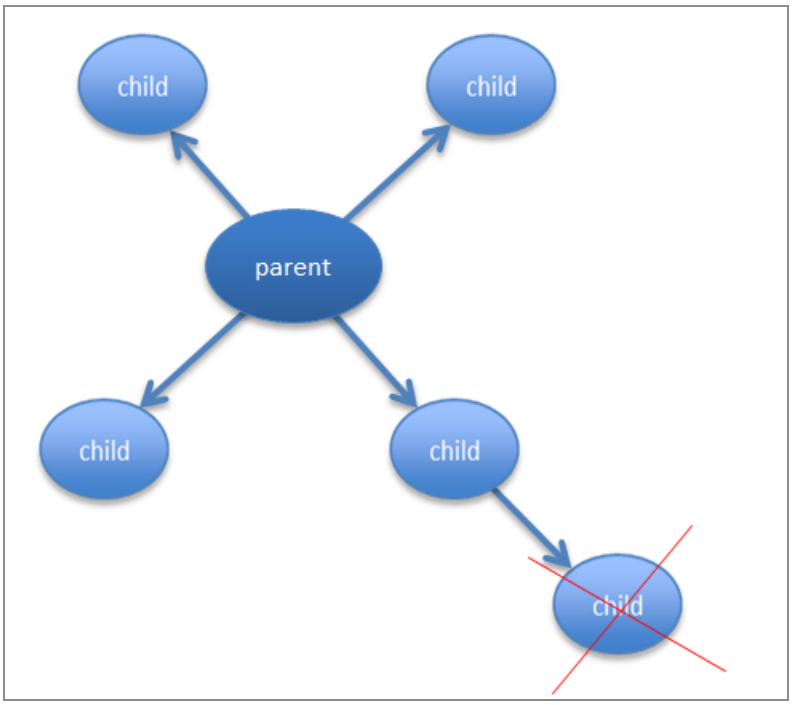
```
<IndexEntity joinTable="true" normalized=“true">
```

TIBCO Patterns - Search Joins allows one query for an accurate result. TIBCO Patterns - Search Joins provides separate tables for each repository. Each repository is named by the entity name and the partition number, for example CustomerAddress_ p0.

The parent repository can have multiple relationships. Relationships between tables and records are established when the tables and records are first created.

Only star schemas are supported:

- A parent table might have any number of child tables.
- A child table has only one parent.
- A child table cannot be a parent.



Left join is supported out-of-the-box.

Tables

The three types of tables are standard tables, parent tables, and child tables. The type of table is set at the time the table is created and does not change throughout the life of the table.

Standard Table

A standard table is one which does not share a parent or child relationship with any other table. Joins operations cannot be performed on standard tables.

Parent Table

A table designated as a parent table when it is created, can act as the root, or parent in a join relationship. A parent table can have multiple child tables. The child table is linked to its parent table when it is created. So, it is necessary to create the parent table before creating the child table. A parent table cannot be deleted if one or more child tables are linked to it. A parent table cannot be a child table.

Child Table

A table must be designated as a child table when it is created. The parent table of the child table is specified at the time it is created and is fixed for the life of the table and cannot be modified. You cannot have a child table without a parent table.

Table Operations

You can create, delete, rename or checkpoint and restore tables.

Table Creation

Relationships between tables and records are established when the tables and records are first created. The parent table must be designated as such when it is created. The child table must be designated as such when it is created, and the parent table must be specified for the child table.

Table Deletion

A parent table cannot be deleted if any child tables for that parent exist. A child table can be deleted at any time. Deleting a parent record causes all child records of that parent record to become orphans. The parent key must be known and used to access the orphaned child record.

Table Rename

Parent and child tables can be renamed at any time.

Table Checkpoint and Restoration

Checkpoint and restore a parent and all of its child tables as a unit.

Records

Child tables might have both standard and child records. Only a child table might have child records. A child record is identified by the combination of the parent record key and the child record key.

Standard Record

A standard record can be added to any of the table types. When added to a parent table, a standard record becomes a parent record. However, if added to a child table, a standard record remains a standard record. It is never linked to a parent record.

Parent Record

A parent record is a standard record with the additional feature that it allows child records to be linked to it. Adding a record to a parent table makes it a parent record. Therefore, all records in a parent table are parent records. Standard and child tables cannot hold parent records. Add, update, and delete operations are performed in exactly the same way as for standard records. Parent records can have multiple child records.

Child Record

A child record is a standard record with the addition of a parent record key. A standard record has a single key value that forms the unique key for the record. A child record has two key values: the record key and the parent record key. The combination of the two forms the unique key value for a child record. A child record has only one parent key value.

Note: Adding a parent record with the same key as the old parent does not re-establish a child-parent relationship.

Limitation of TIBCO Patterns - Search New Joins

All linkages in TIBCO Patterns - Search Joins are one-to-many linkages. Many-to-many relationships between records are not supported.

That is, TIBCO Patterns - Search Joins only supports the STAR schema, which means that a child repository cannot be a parent and a grandchild repository cannot be created.

Custom Netrics Query

The Custom Netrics is implemented on the Customer Repository model to understand the MatchCase, QueryLet, Grouping and Penalty scores features. Using the Customer

Repository sample, you can integrate other repository models and design your Custom Netrics Query.

- **QueryLet:** Executes the TIBCO Patterns - Search query at one time and refers to the results in the other locations instead of running the same TIBCO Patterns - Search query in multiple occurrences.
- **Penalty Score:** Defines the penalty score if wrong score is computed in case of child record or data is not found in the matching attributes.
- **MatchCase:** Produces a match strength score if a set of information is available that can be split into categories. Each Category is called as a MatchCase.

For more details about QueryLet, see the "Named Querylets" section in TIBCO Patterns - Search *Programmer's Guide*.

Configuring Custom Netrics Query

By default, the Installer configures the custom Netrics query. To manually configure, follow the steps.

Procedure

1. Import customer.zip metadata from $MQ_HOME\templates\customer\.
2. Copy and modify the CustomQueryConfig.xml file from MQ_HOME\templates\customer\queryConfig to MQ_COMMON_DIR\<enterprisename>\rulebase\.
3. The thesaurus file must be correctly defined in the CustomQueryConfig.xml available in the MQ_HOME\templates\customer\queryConfig and the thesaurus files nicknames.csv and street.csv are available in MQ_HOME\templates\customer\thesaurus .
4. Copy the Rulebase file from MQ_HOME\templates\customer\queryConfig\MatchAttrList.xml to MQ_COMMON_DIR\<enterprisename>\rulebase\.
5. Using the TIBCO MDM Studio, update the wfin26productaddinternaleditv1.xml. In Matcher activity, enable the custom configuration <Parameter name="UseCustomQueryForNetrics" direction="in" eval="constant" type="String">true</Parameter> <!-- com.tibco.mdm.repository.search.INetricsQueryBuilder --> <Parameter name="CustomNetricsQueryBuilderImpl" direction="in" eval="constant" type="String">com.tibco.dq.customQuery.PatternsCustomQueryImpl</Parameter>

Custom Query Config

The CustomQueryConfig.xml is used for the weights and the match cases to the Customer model. This file is available in MQ_COMMON_DIR\<EnterpriseName>\rulebase or MQ_COMMON_DIR\standard\rulebase.

The various tags in the configuration file are explained in the following table:

Tag Names	Description
QletList	Specify the list of query lets in this field.
MatchCaseList	Specify the weights for the matchcase in this field.
QletConfig	Specify the weights, queries, and thesaurus in the query let configuration.
QletClassName	Specify the query builder class name which must be instantiated. This class must extend AINetricsQueryBuilder.
Description	Specify the description of the query let.
QletInputRepName	Specify the base name of the repository as it appears in the search entity. It is used by the query builder class. If the parent entity is used then this field must be empty or null.
QletOutputTableName	Specify the table name as it appears in Patterns.
QletIndexEntityName	Specify the index entity name if the tablename is not defined.
QletFields	Specify only If you want remapping of base field names. You can use entirely different field names by using this tag. The name must be given as known in the query builder class. That is, its default field name, and the new name.
QletInputFields	Specify the input remapping fields.
QletFieldDef	Specify the default name and the new name of the remapping. The names must be as known to the query builder class.

Tag Names	Description
QletOutputFields	Specify the output remapping fields.
QletParams	Specify the extra parameters that must be used like thesaurus.
MatchCase	Specify the matchcase defined as per the use case.
MatchStrength	Specify the strength score for the match case defined.
MatchThresholds	Specify the thresholds score for the match case defined.
MatchWeights	Specify the weights score for the match case defined.
MatchPenalties	Specify the penalties score for the match case defined.

Customer Data Model

The Customer data model is a sample B2C (business-to-consumer) metadata that is used for the Custom Netrics Query feature. Custom Netrics Query is run at the time of matching records.

The Customer data model comprises the following components:

Repositories	Data Sources	Data Files	Input Maps	Classification Schemes
Address			ContactAddress_InputMap1	ADDRESSTYPE
	Account Source	AccountSource.csv		
		Account.csv		
Delivery				
	Contact	Contact.csv	ContactPhone_InputMap1	
Email			ContactEmail_InputMap1	EMAILTYPES
	Country			
		CountryCode.csv		
Household				
	Language Codes	LanguageCodes.csv		
Person			InitialPersonImport	
	Occupation	Occupation.csv		
PersonProfile **Note:** This is the main repository.	Standard Customer	StandardCustomer.csv		
	Role Values	Roles.csv		
Phone			ContactPhone_InputMap1	PHONETYPE
	State	State.csv		

Repositories	Data Sources	Data Files	Input Maps	Classification Schemes
Preference			Preference_InputMap1	
		Ratings.csv		
Source Reference			Reference_InputMap1	

Rulebases

- cust_ADDRESS_catalogvalidation.rul
- cust_ADDRESS_searchcontrolrules.rul
- cust_EMAIL_catalogvalidation.rul
- cust_EMAIL_searchcontrolrules.rul
- cust_PERSON_catalogvalidation.rul
- cust_PERSONPROFILE_catalogvalidation.rul
- cust_PHONE_catalogvalidation.rul
- cust_PHONE_searchcontrolrules.rul
- cust_SFIdentityRulebase.rul
- cust_SFImportRulebase.rul
- default.rul
- test_catalogvalidation.rul

Note: Apart from the repositories, the thesaurus files nickname.csv and street.csvare available in TIBCO Cloud MDM. You can download these files from **Configuration Viewer > config > thesaurus** folder.

- The nickname.csv file contains the short names of persons for the Person repository.
- The street.csv contains the street names for the Address repository

Defining Match Cases

The Custom Netrics Query is a sample for the Custom Query which is implemented on Customer Repository model. To understand the problems with data matching, TIBCO Patterns - Search came up with MatchCase, QueryLet, Grouping and Penalty scores functionality. Using this feature, the Customer model has been implemented. The weights can be configured and all the use cases can be managed.

A common matching problem involves matching entities where there is a set of information describing the entity that can be split into categories. For example, the standard customer model is used for the record matching person's use cases. The customer model matching query uses the following categories:

1. Name
2. Date of Birth
3. Gender
4. ID (SSN)
5. Phone Number
6. Email Address
7. Address

For each category a QueryLet is defined that produces a match strength score for that category. The problem arises while trying to combine the category match strength scores into an overall match score. The standard AND combiner is not sufficient to express the complexities of determining a match based on the match strengths of each category. There are two primary reasons for this:

1. **Scores are not comparable:** A match strength score of 0.85 for an Address might represent a fairly good match for a street address, but a match strength score of 0.85 for an SSN, phone number or email address probably represents a fairly poor match. Basically, we want to ascertain that the two records represent the same value. Different categories are likely to have different match strength criteria before we can say they are probably the same value.
2. **Categories are not interchangeable:** Some categories are required to determine a match, others are almost irrelevant, or play only supporting roles. In the example, it does not matter how well the other categories match, if there is no Name or ID

match, you cannot say the two records represent the same person.

The current AND structure can capture some of this complexity through the use of the ignore scores, reject scores, and querylet weighting features. However, it cannot deal with scores comparable scenarios. The current constructs cannot capture all of the aspects of merging category scores to determine the likelihood of a match.

The following match cases are defined for each category:

- **Case 1:** Name and DOB as core QueryLet.
- **Case 2:** Name and ID as core QueryLet.
- **Case 3:** Name and Email as core QueryLet.
- **Case 4:** Name and Address as core QueryLet.
- **Case 5:** Name, DOB and ID as core QueryLet.

For each match case, six QueryLets are defined.

- **QueryLet 1 - Name:** Values defined for the Firtsname, Last Name, Middle Name and NameSuffix is considered as one QueryLet.
- **QueryLet 2 - DOB:** Values defined for the Date of Birth is considered as one QueryLet.
- **QueryLet 3 - IDs:** Values defined for the NationalReferenceNumber is considered as one QueryLet.
- **QueryLet 4 - Email:** Child record values of the Email is considered as one QueryLet.
- **QueryLet 5 - Address:** Child record values of the Address such as street pincode, and so on is considered as one QueryLet.
- **QueryLet 6 - Phone:** Child record values of the PhoneNumber is considered as one QueryLet.

Based on the record data the weights are applied and results returned. These results are shown in the matcher work item. The Customer query is as follows:

```
And(
  Person.first = “<Person.first>”,
  Person.last = “<Person.last>”,
  Or(
    Email1.address = “<Email1.address>”,
```

```
    Email2.address = “<Email2.address>”,
    Email3.address = “<Email3.address>”),
  Or(
    Phone1.number = “<Phone1.number>”,
    Phone2.number = “<Phone2.number>”),
  Or(
    And(
      Address1.street = “<Address1.street>”,
      Address1.city = “<Address1.city>”,
      Address1.state = “<Address1.state>”),
    And(
      Address2.street = “<Address2.street>”,
      Address2.city = “<Address2.city>”,
      Address2.state = “<Address2.state>”),
    And(
      Address3.street = “<Address3.street>”,
      Address3.city = “<Address3.city>”,
      Address3.state = “<Address3.state>”),
    And(
      Address4.street = “<Address4.street>”,
      Address4.city = “<Address4.city>”,
      Address4.state = “<Address4.state>”)
  )
)
```

Customizing Customer Model

The Customer model can be customized based on your requirement.

The sample Java codes are available in common\standard\samples\PatternsCustomQuery\src\com\tibco\dq\customQuery. Modify the Java codes as per your requirement and merge it with **ECM.ear**.

Custom Matching Using Query Builder Platform of TIBCO Patterns - Search

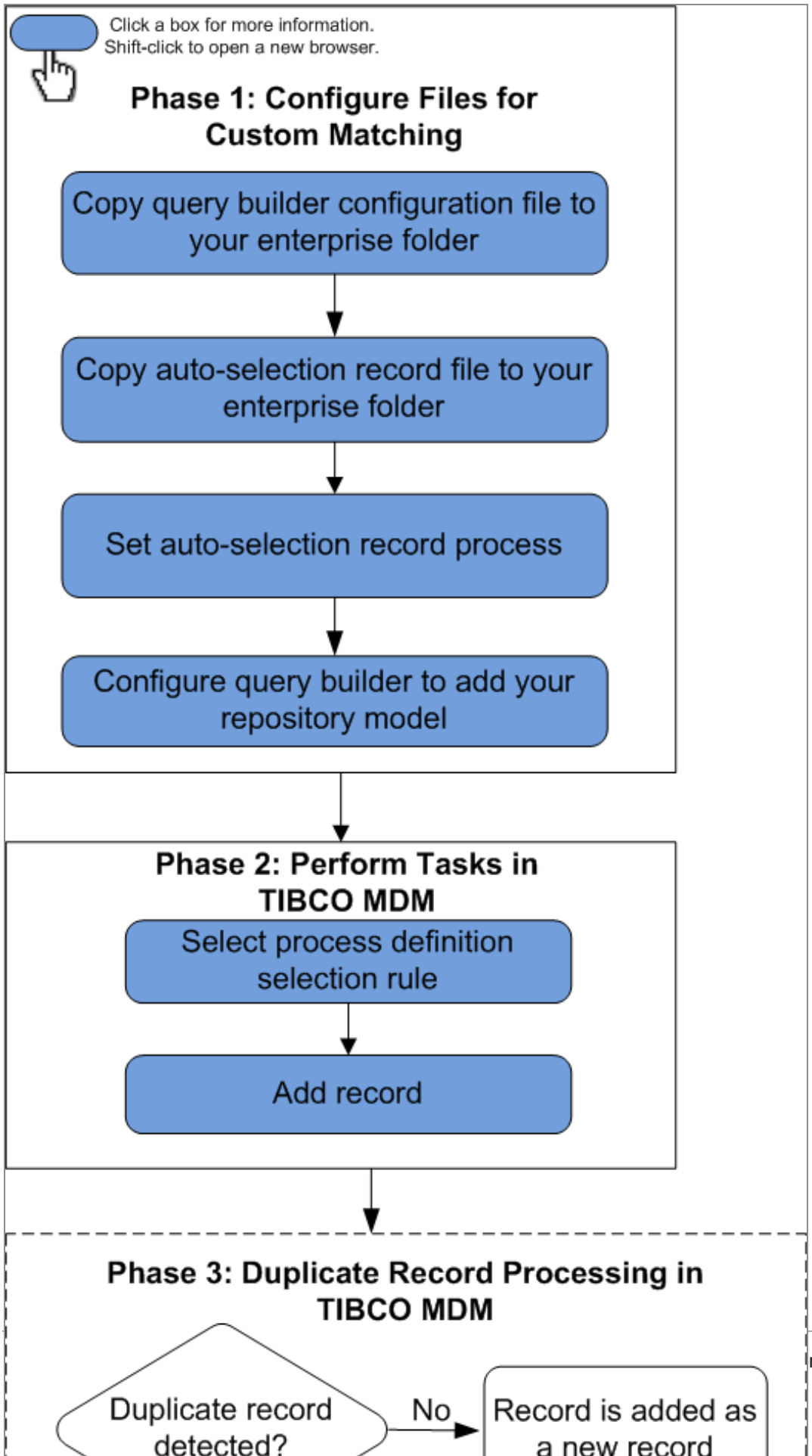
TIBCO Patterns - Search includes the Query Builder Platform (a Java based tool) that eases the process of designing, testing, and administering TIBCO Patterns - Search record matching queries. This tool eliminates or greatly reduces the need to write Java code to define and run record matching queries.

In TIBCO MDM, the Query Builder Platform tool works in the interpretative mode, wherein, you specify the QueryBuilderConfig.xml configuration file and pass it to TIBCO MDM. TIBCO MDM calls the query builder TIBCO Patterns - Search API and returns the TIBCO Patterns - Search query. You can pass the query to the MatchRecord Activity. TIBCO MDM then uses the query and finds matching records.

For information about the TIBCO Patterns - Search query, see Custom Netrics Query. For information about the Query Builder Platform tool, see *TIBCO Patterns - Search Concepts*.

Custom Matching Process

The custom matching process using Query Builder Platform is divided into three phases.


Click a box for more information.
Shift-click to open a new browser.
Phase 1: Configure Files for Custom Matching
Copy query builder configuration file to your enterprise folder
Copy auto-selection record file to your enterprise folder
Set auto-selection record process
Configure query builder to add your repository model
Phase 2: Perform Tasks in TIBCO MDM
Select process definition selection rule
Add record
Phase 3: Duplicate Record Processing in TIBCO MDM
Duplicate record detected?
No
Record is added as a new record

- In the phase one, you must first configure the query builder file and auto-selection record files. Based on the configuration defined in the QueryBuilderConfig.xml file, TIBCO Patterns - Search query is formed and a search is performed. Based on the configuration defined in the AutoRecordSelectionConfig.xml file, the surviving or target records are auto-selected to merge with the source record.
- In phase two, you must select the wfin26dqautorecordselectionv1.xml workflow through the Process Definition Selection business process rule in TIBCO MDM. The Process Definition Selection business process rule includes the com.tibco.dq.customQuery.queryBuilder.PatternsQueryBuilderImpl Java class for the CustomNetricsQueryBuilderImpl parameter in the Match Record activity.
- In phase three, after detecting duplicate records, the records are passed through the SurvivingRecordSelection activity and the MergeRecord activity, and the record is merged based on the auto-selection criteria.

You can view the record merge details on the Event Log page.

For information about parameters of the Match Record, Merge Record, and Surviving Record Selection activities, see *TIBCO MDMWorkflow Reference*.

Query Builder Configuration File

The query builder configuration file includes the query information. The configuration file is used as a part of the query builder implementation. This file is taken as an input parameter in the MatchRecord activity.

The QueryBuilderConfig.xml configuration file is available in $MQ_COMMON_DIR\standard \rulebase folder. The QueryDef.xsd schema file (provided with TIBCO Patterns - Search) includes details about the format of the configuration file that defines a query. The QueryBuilderConfig.xml configuration file consists the following items:

- Table information: Includes the parent and child information. Name of the table is retrieved based on the entity-name mentioned in the IndexerConfig.xml file. The table is available in the TIBCO Patterns - Search index. TIBCO MDM does not provide index table information. The parent and child table information is added directly to the GeneralcomponentQuerybuilder object.
- Querylet class name: Implementation class is auto-generated based on the querylet description.
- Query description and MatchCase description: This is similar to the TIBCO

Patterns - Search query builder configuration XML file.

- Query string: It is used to prepare the TIBCO Patterns - Search query. Be careful when adding attribute names in the query string. For example, <queryString>${FIRSTNAME}</queryString> or <queryString>${AREACODE} ${PHONE}</queryString>.

For information about configuring the query builder file, see TIBCO Patterns - Search documentation.

Customizing Query Builder

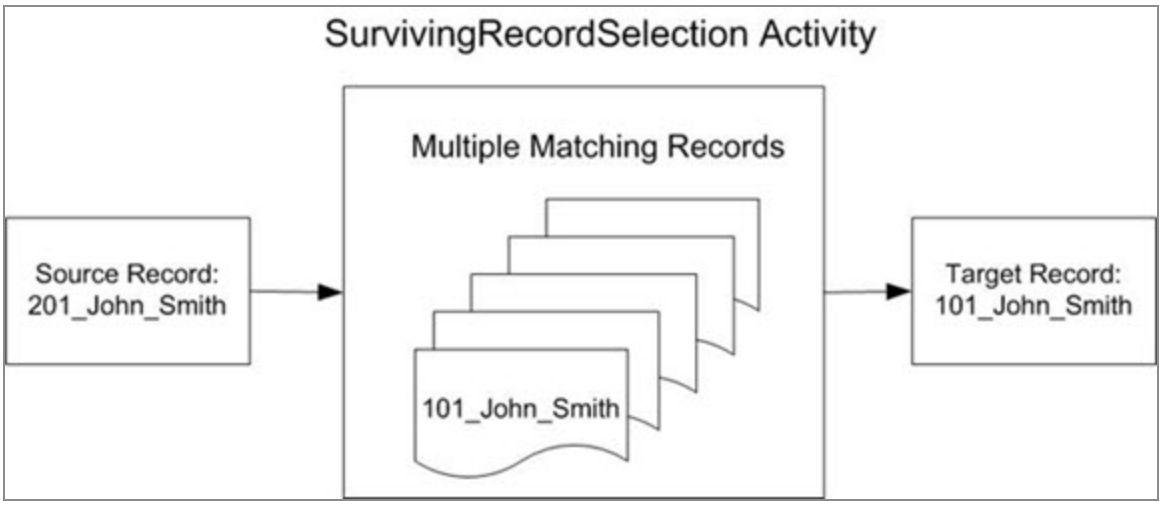
You can customize the query builder as required.

The sample Java codes are available at $MQ_COMMON_DIR\standard\samples\PatternsCustomQuery\src\com\tibco\dq\customQuery. Modify the Java codes (as required), and then compile and merge it with **ECM.ear**.

Note: You must restart TIBCO MDM after deploying the updated ECM.ear file.

Auto-merge Selection

When you want to merge source records with the existing matching records, use the auto-merge selection process to automate the selection of a target or surviving matching record.



The SurvivingRecordSelection activity selects one target record from the detected duplicate records and saves all data in the Merge tables. The selection of a target record is based on the Auto-selection Process Rules defined in the auto records selection configuration file.

1. The source and target records are passed to the MergeRecord Activity for the merging process.
2. The MergeRecord Activity identifies the records from the Merge tables and merges the records.
3. Auto-merge selects only one record as the survivor.

   The survivor record is merged with the incoming record as blanket merge.

For information about the parameters of the SurvivingRecordSelection activity and MergeRecord activity, see *TIBCO MDMWorkflow Reference*.

Auto-selection Process Rules

You can define the auto-selection process rules and their parameters in the AutoRecordSelectionConfig.xml file located in the $MQ_COMMON_DIR/standard/rulebase folder.

Note: The AutoRecordSelectionConfig.xml file is defined at an enterprise level. Thus, the auto-selection process rules defined are applied to the records in one enterprise (and not to records that exist outside an enterprise).

You can define the following selection processes:

- **AutoSelectionProcess**: The auto selection process rules define which surviving or target record must be selected from the multiple matching records to merge with the source record.

  Out of all the auto-selection rules that are available, you can select only two at a time. Based on the selected rules, the target record is selected.

Rule Name	Description
High Score or Best Score	Records are sorted based on the highest matching score. The highest score is returned from TIBCO Patterns - Search. The

Rule Name	Description
	maximum score received for the record is considered as surviving or target record. If two or more records have the same matching score, the value of ConflictResolutionProcess is considered for evaluation.
Mostly New	By default, records are sorted based on the ModDate attribute. The newest (latest) record is considered as a surviving or target record. If two or more records have the same ModDate attribute, the value of ConflictResolutionProcess is considered for evaluation. You can add your own attributes apart from ModDate. The data types of attributes can be date, long, or string. For information about the data type, see "Data Type for Attributes" in *TIBCO MDM User's Guide*. Additionally, you can define new or old sorting order to select the newest or oldest records. Sorting is done based on ascending and descending order.
Mostly Merge	Records are sorted based on the record linkage count available in the RecordLinkage table. The record linkage count indicates how many times the record is merged with different records. The record that is most often merged is considered for a surviving or target record. If two or more records have the same record linkage count, the value of ConflictResolutionProcess is considered for evaluation.
Precedence	Records are sorted based on the incoming source. For example, if five duplicate records are detected, a record received from a social security source takes precedence over the record received from a social media source. If precedence is defined, the source ID is considered. If the precedence is not defined, attribute name and its order of values is considered to select the surviving record.

Rule Name	Description
	Remember: The precedence calculations are not applied during auto-selection of records. For more information about how precedence works, see Auto-selection Process Rules.
Complete Data	Record that contains the maximum number of attribute values is considered as a surviving or a target record. You must specify the attribute list in the AllowedAttributes parameter list. If two or more records have the same values specified in the attribute list, the value of ConflictResolutionProcess is considered for evaluation.

- **ConflictResolutionProcess**: If conflict arises when selecting the auto-selection process rule, the value of the ConflictResolutionProcess is considered. The values are similar to the auto-selection process rule.

Auto-selection Process Parameters

The auto-selection process rules define which surviving or target record must be selected from the multiple matching records to merge with the source record. Out of all the auto-selection rules that are available, you can select only two at a time. Based on the selected rules, the target record is selected.

Parameters

For Selection Process	Parameter and Description
Mostly New	• **AttributeName:** Specify the attribute name to find the new record. By default, the ModDate attribute is specified. • **SortOrder:** Define NEW or OLD sorting order to select the newest or oldest records.
Mostly Merge	**CheckMode:** The valid values are RecordLinkage and the ModVersion. By default, the RecordLinkage value is specified.

For Selection Process	Parameter and Description
Precedence	• **AttributeName:** Specify the attribute name to find the new record. The AttributeName parameter is ignored if the PrecedenceBased value is specified for the PrecedenceMode parameter. • **PrecedenceMode:** Specify AttributeBased or PrecedenceBased value. You must enable the precedence to work with the PrecedenceBased mode. If precedence is specified, the source ID is considered. **Parameter List** OrderOfValues. Considered only for AttributeBased value. For example, see AutoRecordSelectionConfig.xml file.
Complete Data	See Auto-selection Process Rules. **Parameter List** AllowedAttributes For example, see AutoRecordSelectionConfig.xml file.
ActionOnMatchedRecords	**DELETE:** By default, the DELETE action is commented in the AutoRecordSelectionConfig.xml file. To delete records from the repository, uncomment the following line: <!-- ActionOnMatchedRecords>DELETE</ActionOnMatchedRecords -->. Additionally, in the wfin26dqautorecordselectionv1.xml workflow, append the Delete Record activity after the Merge Record activity.

Customization of Precedence Management

Precedence algorithm or parts of it can be customized. Precedence management supports loading of custom algorithms and weight calculators.

You can modify the existing attributes and also add new attributes in the predefined data model. The new attributes are passed as is to the custom components. Also, you can extend the data model defined for AttributeQuality and PrecedenceDefinition by adding new attributes. These attributes become available to you for custom weight and precedence calculations.

Precedence Management Interfaces

To customize the precedence management, TIBCO MDM provides default implementation classes (PrecedenceEvaluator.class and AttributeWeightCalculator.class) in the ECM.ear\ECMClasses\com\tibco\dq\precedence directory. You can customize these classes in the enterprise directories, for example, CustomPrecedenceEvalutor and CustomAttributeWeightCalculator. Ensure that the Quick Configuration Update property is set to **true** in the Configurator. The property auto reloads the classes.

To support extension of logic and other weight calculators, implement the following interfaces by replacing the default implementation:

- IPrecedenceEvalutor.compute
- IAttributeWeightCalculator.computeWeight

For information about using precedence management, see the "Precedence Management" section in *TIBCO MDM User's Guide*.

How the Algorithm Works

The algorithm within Precedence Management computes the changes made, calculates the new weight based on quality, compares the weight, and decides which value to retain.

The algorithm works as follows:

- Computes the changes for the attributes that have been changed.
    - For each attribute, looks up the history to find the most recent change in the source.
    - For each change, calculates the old weight based on old source quality definition, apply precedence trust factor if applicable.
- For each change, calculates the new weight based on the quality definition. The

weight calculation takes decay into account. If there is no definition for an attribute, no quality computation can be done for that attribute (change has 100% weight and is always accepted). For more information, see Weight Computation.

- For each change, calculates old weight based on the old source quality definition. Applying the precedence trust factor is applicable.
- Compares this weight and decide which value to retain (old or new).
- Generates assignment actions so that values can be reverted by the rulebase processing logic.

Weight Computation

The weight calculation takes decay type into account, whether this decay is linear or half life.

Linear

This decay type is taken into account when quality declines linearly over a set period of time. For example, if decay is defined as ten months, in five months the weight becomes half of the initial quality. In ten months, the weight becomes zero.

Example for the LINEAR decay type:

```
if ageOfDataInMillis > decayPeriodInMilliscomputedWeight = 0elsecomputedWeight = Math.round
(given Weight * (1 - ((float) ageOfDataInMillis / (float) decayPeriodInMillis)))]
```

Half Life

With this decay type, the quality never becomes zero. It gets lower as more intervals expire. For example, if the decay has a half life of six months and the initial weight is 70, in six months the weight becomes 35. In another six months, the weight becomes 17, and so on.

Example for the HALFLIFE decay type:

```
[float n = ((float) ageOfDataInMillis / (float) decayPeriodInMillis);computedWeight = Math.round(
(float)(given Weight / (Math.pow(2, n))))]
```

The following example shows how Precedence is applied:

- If the old weight > the new weight, it reverts to the old data.
- If the old weight <= the new weight, it continues with the new data.

Ensure the Absolute Quality of an Attribute

If the absolute quality is defined, it is used on new weight before comparing it with the existing data. The following example shows how to ensure the absolute quality of an attribute:

1. John is an existing name in the system.
2. A new value is received: J. The source from where J comes from is more reliable than the previous source.

In the current implementation, J overwrites John. You can have a rule which can reject this record but as this record contains other useful information, you do not want to reject it. You would rather just change J to John. This is where the concept of absolute quality data comes in. In this example, the quality of J is less than the quality of John.

A capability to assign semantics and support it with a way to add rules for quality computation is needed. This can be done in one of the following ways:

- When defining the precedence data for each attribute, attach a rule for absolute quality computation. This rule is fired when precedence is calculated.
- Have an action in the rulebase, assignQuality, which assigns a numeric value attribute quality. This value is retained and subsequently used for precedence calculations.
- Similar to the first option, but instead of defining a rule per attribute, define a rulebase which computes absolutequality for all of the attributes.

When absolute quality is computed, it is multiplied by the source trust factor to arrive at the applicable quality of the data before comparing it with the existing data.

How Rulebase is Defined to Measure Absolute Quality

- Precedence action in rulebase accepts a rulebase as an additional input similar to the Include action. This rulebase executes exactly like included rulebase, however

the following action occurs:

- The rulebase must produce absolute quality for each attribute by assigning values to output variables. The output variables must follow the naming pattern: <attributeName>_WEIGHT.
- The rulebase has all the context and variables which are in calling rulebase.
- All of the weight produced by this rulebase are removed from the context after they are used in the precedence calculations.

- Similar to the Include action, you can specify a rulebase or decision table and a logical name for the constraints.
- If the rulebase does not have to modify the weight, it does not have to produce any weight or produce 100.

  For example, if the rulebase produces 70 as the weight for FIRSTNAME and the configured weight for FIRSTNAME is 80, the weight becomes 70.
- After the weight is computed, it is used to decide which value to keep.

Note: After the value is accepted, the quality is not stored for the next time.

Configuration Properties for Attribute History

Purpose	Configuration Property and Values
To generate attribute history only on a subset of attributes within a repository, which are part of the precedence management.	To enable attribute history, add the following property in the Configurator: • Configuration value name: Limit Attribute History Entries • Internal name: com.tibco.mdm.precedence.limit.attributehistory.for.repositories • Version: 9.1.2 • Description: Enable Attribute History only for the attributes that are part of Precedence Management • Value Type: List • Current Value: TEST.EMPLOYEE • Visibility: Advanced • Category: Miscellaneous **Note:** The format of the value is ENTERPRISENAME.REPOSITORYNAME.
To compute precedence when the same value for an attribute comes from different sources. In such scenario, the values that come from the source with the higher weightage are always considered and the decay factor is ignored.	To log attribute history, add the following property in the Configurator: • Configuration value name: Log attribute history entries for an attribute even if the value of the attribute does not change. • Internal name: com.tibco.mdm.create.attributehistory.for.unchanged.data • Version: 9.1.2 • Description: Log attribute history entries for an attribute even if the value of the attribute does not change. Works in conjunction with the com.tibco.mdm.precedence.limit.attributehistory.for.repositories configuration property. • Value Type: String • Current Value: true • Default Value: false • Visibility: Advanced • Category: Miscellaneous

Version Selection

In most cases, the version in which precedence is considered is the version that is served as the base. However, this might not be the case in some use cases.

When import is done, the draft record is created. This draft record might be modified to create additional versions in the workflow. To ensure that each subsequent modification does not violate the precedence criterion, the use of version, which is previously confirmed or unconfirmed (to this version), is more appropriate.

To apply flexibility in precedence logic, it is possible to configure the workflow to have a separate step, EvaluateRulebase, applied to precedence. In this case, irrespective of what changes have happened during workflow for the draft records, it is possible to look up the previously confirmed or latest version.

To support these use cases, the precedence action can accept an argument to indicate which version must be used. The version keywords are the same as those used with CHANGED condition: PREVIOUS_VERSION and PREVIOUS_CONFIRMED_ VERSION. If nothing is specified, the parent version is used.

Limitations

Review the limitations before using this feature.

- Precedence is not supported for relationship attributes.
- Precedence is not supported in the PrepareForImport activity; the ApplyPrecedence rulebase does not work with the PrepareForImport activity.
- FILE type attributes cannot be used with Precedence Management.

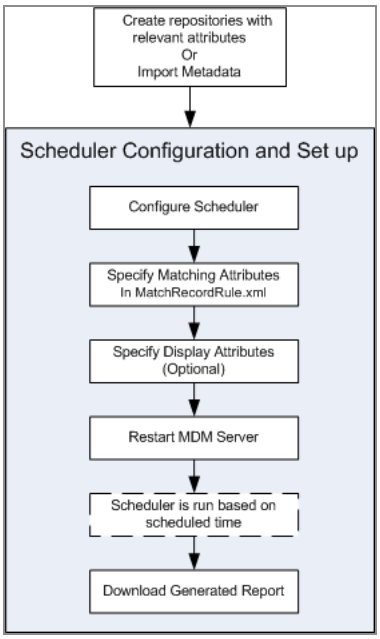
Scheduler Duplicate Detection Process

Tip: If you prefer a video of the *Scheduler Duplicate Detection* process, visit https://youtu.be/2NsDwp7WfjA.

The scheduler duplicate detection process allows you to locate duplicate records within an existing repository. In TIBCO MDM, data is retrieved and loaded from many sources such as Web Services, bulk import, Database Loader, and so on.

In this process, the database might contain duplicate records. Therefore, you must perform the duplicate detection on these records on a timely basis.

Scheduler Duplicate Detection Process



The following components are involved in this process:

- Configuring Scheduler
- Specifying Matching Attributes
- Specifying Display Attributes
- Restarting MDM Server
- Downloading Generated Report

Configuring Scheduler

To schedule the duplicate detection process in a timely manner, use Configurator as well as the CronSchedules.xml file.

For using the Configurator , see the section "Scheduling Jobs Using Configurator" in *TIBCO MDM System Administration*. For using the CronSchedules.xml file, perform the subsequent steps.

Procedure

1. Navigate to the $MQ_HOME/config folder.
2. Open the CronSchedules.xml file.
3. Specify the following properties in the CronSchedules.xml file under the Schedule tag:

Property Name	Description
Name	Specify the name of the scheduler. For example, Scheduler Duplicate Detection for XYZ Enterprise.
Jobs	Specify the jobs that must be initiated and invoked. You can include more than one job tag.
Job	Specify the job that informs the time and job (task) to be initiated. • **Name**: Specify the job name. For example, FED Job. • **TriggerExpression**: Specify the cron trigger expression based on which the job is invoked. For example, 0 0/5 * * * ?, which refers to Seconds Minutes Hours Day-of-Week Year. The trigger expression can contain more than one expression. For more information, see the Scheduler Configuration chapter in the *TIBCO MDM System Administrator's* guide. • **JobDetailsClass**: Specify the job class that implements the task. For example, For Future Effective Date scheduler job: com.tibco.mdm.infrastructure.scheduler.FEDScheduler For Scheduler Duplicate Detection job: com.tibco.dq.scheduler.DuplicateDetectionJob or implement com.tibco.mdm.infrastructure.scheduler.CimSchedeulerjob. The JobDetailsClass must implement

Property Name	Description
	com.tibco.mdm.infrastructure.scheduler.CimSchedeulerjob interface. When a trigger is executed in the Java class, the execute() method is called. You can extend a job by specifying the com.tibco.mdm.infrastructure.scheduler.CimSchedulerJob class and implement the executeJob methods by passing the context object. The Scheduler framework provides the context object to the implemented job. The context object contains the job name, trigger name, its description, and so on. The signature of the executJob method is public void executeJob (JobExecutionContextjobExecutionContext) throws MqException{…}
ExecuteonStartup	If you specify true value, the scheduler runs the job at the start of the application. By default, the value is false.
Job Inputs	• **EnterpriseName**: Specify the enterprise name on which the duplicate detection process must invoke. • **UserName**: Specify the user name. The User name is case sensitive. • **JobPolicy**: Specify the XML file name that is parsed for repository attributes. For example, the MatchRecordRule.xml file. • **ParserClass**: Specify the class that parses through the XML file and sends the values in a form of match. For example, com.tibco.dq.scheduler.MatchRecordRuleParser. If you have a different MatchRecordRule format, the parser class can be customized according to the XML file. The job inputs are retrieved from the JobDataMap object in the job class with the key as IRuleParser.GET_ PARAMETER. A custom policy parser must be defined to implement the IRuleParser interface.

Property Name	Description
	Note: MadCap:autonum="Note: ">The job input tag is optional for other schedulers.

You can customize other scheduled jobs. These can be defined for any periodic executable task. For example, the Future Effective Date job or an email job to send out a summary of pending work items every Friday evening. For more information, see the samples located in the $MQ_COMMON_DIR/standard/samples/Scheduler sample folder.

Specifying Matching Attributes

To match and locate duplicate records, specify the repository attributes to be used as matching criteria in an XML file.

Procedure

1. Navigate to the $MQ_COMMON_DIR/samples/DQ process folder.
2. Copy the sample MatchRecordRule.xml to the $MQ_COMMON_ DIR/<enterprise_ internal_name>/rule base folder. You can change the file name.

   The defined repositories and their attributes are processed for the duplicate detection process.

MatchRecordRule.xml Tags and Values

The sample MatchRecordRule.xml file includes the following tags and values:

Tag Names and Description

Tag Name	Element Name	Attribute Name	Description	Value
	<MatchRecordRule>		Refers to the first node of the MatchRecordRule file. It contains the SearchEntity node.	
	SearchEntity		Includes the SearchMode, DataProvider, Repository details and their attributes that are considered in the duplicate detection process.	
		Type	Specifies the search entity type. You can search records based on the search entity types based on the repository type.	The valid value is table or view. The default value is table. • **Table**: Used for a single repository duplicate detection • **View**: Used for a cross-repository duplicate detection
SearchMode			Specifies the search mode for scheduler duplicate detection. You can specify the following two modes: • **Complete**: Specifies that for the first time search is performed on all data that exists in a repository. Next time onwards, the server verifies whether a scheduler duplicate detection (SDD) job was executed earlier, if yes, only new or changed records are searched. For example, the SDD job was executed on 12th April and the current SDD job execution date is 16th April. In this case, while detecting duplicate records, records are searched between 12th April and 16th April. • **Incremental**: Specifies that only new or changed data is searched except in one case, that is, if an SDD job was not executed earlier, the server starts with the complete mode and the search is performed on the entire data.	The valid value is Incremental or Complete. The default value is Incremental.
DataProvider			The class name of the data provider that returns a set of	The valid value is any data provider class name.

Tag Name	Element Name	Attribute Name	Description	Value

			records in a batch.
			Any custom data provider implementation must implement the IRecordExtractor interface and provide implementation of the following two methods:
			`public IBatchIterator getRecords(long catalogId,String searchMode,long eventId,String viewName) throws MqException;`
			`public void closeIterator() throws MqException;`
				For example, com.tibco.dq.dao.DupDetectJobDataExtractor

MatchAttributes			Name of the matching attributes and weightage. For example, • **Name**: The name of the attribute. For example, BANKNAME • **Weight**: Weight of the character that is specified in the Name attribute. Enter a decimal number between 0.00 to 1. This is the weightage of an attribute for matching records. For example, 0.80.	The valid value for the Name attribute is any valid string. The valid value for the Weight attribute is a fraction between 0 and 1.
DisplayAttributes			Name of the attributes that you want to display in the report.	The valid value is any valid attribute name.
Relationship			Refers to the relationship name and related repository name.	The valid value is any valid string.
MatchingThreshold			Refers to the matching score of the record. The value indicates a minimum matching expectation. For example, if the MatchingThreshold value is set to 0.75, only records matching 75% or more are returned by the matching process.	The valid value is the minimum matching score value between 0 and 1.
UseCustomQueryForNetrics (Optional)	<UseCustomQueryForNetrics>		If specified as True, the custom Netrics query specified in the CustomNetricsQueryBuilderImpl parameter is used to get results from the TIBCO Patterns - Search indexes.	True False (Default)
CustomNetricsQueryBuilderImpl (Optional)	<CustomNetricsQueryBuilderImpl>		Use the custom implementation class to query the TIBCO Patterns - Search table. It must implement com.tibco.mdm.repository.search.INetricsQueryBuilder interface.	The Java class name that you want to load.
IndexEntityNameToSearch (Optional) - Do not work with View Type.	<IndexEntityNameToSearch>		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.	Index entity name that is specified in the IndexerConfig.xml file. For example, Customer_IndexEntity.

Specifying Display Attributes

Besides the matching attributes, if you want other attributes to be displayed in the report, you can specify these attributes.

Procedure

1. Navigate to the $MQ_COMMON_DIR/samples/DQ process folder.
2. Open the MatchRecordRule.xml file.
3. Specify the attributes in the DisplayAttributes tag.

Restarting MDM Server

After you update the CronSchedules.xml file, restart the TIBCO MDM server.

Downloading Generated Report

After you specify the matching attributes in the MatchRecordRule.xml file and the scheduler job configuration details in the CronSchedules.xml file, the GenerateReportForSDD activity generates a report for the scheduler duplicate detection process.

This activity uses the text format. The job runs in the scheduled time and the report is generated.

You can use the JasperReports tool to generate dynamic content. The JasperReports generator sample file is provided at $MQ_HOME/common/standard/samples/DQprocess/JasperReportGenerator.java. To compile and merge the Jasper code, see the Word document, "Merging Jasper Report.doc" located at $MQ_HOME/common/standard/samples/DQprocess/

Procedure

1. Click the **Auto Duplicate Detection Post Process** link in the Event Log page.
2. Click **File Download**. The File Download dialog is displayed to save or open the text file.
3. Click **Save** to download the file.
4. Open the text file to view the duplicate records. If duplicate records do not exist,

the text file contains only the specified enterprise name and matching attributes.

For more details about the GenerateReportForSDD activity, see the *TIBCO MDM Workflow Reference* guide.

Limitations

Manual Merge

Rulebase does not work with child repositories and relationship attributes. When a child record is added from the source or other matched records, then the validation process is executed, the newly added child records get validated, but the existing record does not get validated.

Index Entities

The Matching across repositories is based on the indexing of multiple repositories in a single TIBCO Patterns - Search table as a composite entity.

This approach of using a denormalized table for indexing has several fundamental disadvantages, such as replicating data, requirement of more memory for too many record bundles, and so on. These disadvantages counteract with the fast searches on the denormalized table, which do not require any joins.

Therefore, you must carefully specify the composite entities and limit their complexity. The following are the various limitations that you must be aware of while specifying the index entities:

- Composite index entities cannot be defined across a self relationship.
- The repositories in a composite index entity must be specified in a linear way. The index entity does not allow two relationships in one repository, which is implemented in the Index configuration file. Each repository tag inside the index entity can only have one associated relationship.
- Relationship attributes cannot be used in matching expressions. However, the Text Search web service allows specifying relationship attributes for a search criterion.
- Defining an index across reverse relationships is not possible. Similarly, matching and text searching across reverse relationships are not supported.

Deduplication Tool

TIBCO MDM provides an external tool named deduplication to locate duplicate records before the data reaches TIBCO MDM. This tool is provided as a sample for the Customer data model, and you can customize the tool for other data models.

The tool creates the TIBCO Patterns - Search table and builds the query. The provided sample data is inserted into the TIBCO Patterns - Search table and finds duplicate records accordingly. Later, the results are grouped into groups based on the duplicate data.

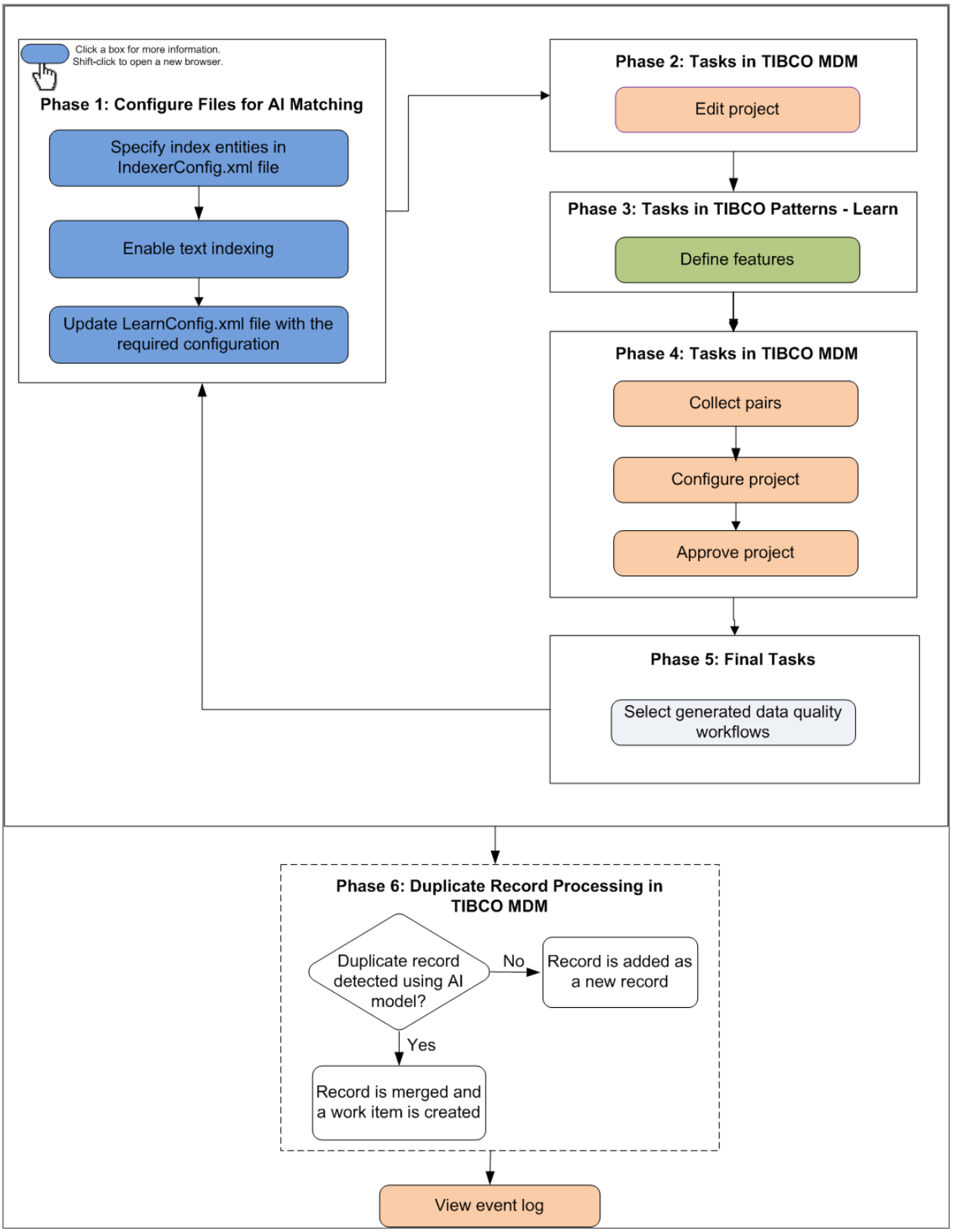
The final file is generated with the groups of duplicate data. You can use the outcome file to analyze the data and remove the duplicate records manually or automate the scripts.

The deduplication tool sample and the *Deduplication_tool* Word file are located at $MQ_HOME/common/standard/samples/DQprocess/customerDedup. To know more about the tool and how to use it, see the Word file.

AI Matching Process

The AI matching process is divided into multiple phases.

Note: This interactive feature only works in the HTML format and remains a static flow chart in PDF.


Click a box for more information.
Shift-click to open a new browser.
Phase 1: Configure Files for AI Matching
Specify index entities in IndexerConfig.xml file
Enable text indexing
Update LearnConfig.xml file with the required configuration
Phase 2: Tasks in TIBCO MDM
Edit project
Phase 3: Tasks in TIBCO Patterns - Learn
Define features
Phase 4: Tasks in TIBCO MDM
Collect pairs
Configure project
Approve project
Phase 5: Final Tasks
Select generated data quality workflows
Phase 6: Duplicate Record Processing in TIBCO MDM
Duplicate record detected using AI model?
No
Record is added as a new record
Yes
Record is merged and a work item is created
View event log

AI Matching Configuration File

The AI matching configuration (LearnConfig.xml) file includes the sample record matching configuration for the standard Customer data model. You can customize your repository model based on this sample data model. The configuration includes fields from the PersonProfile (parent table) repository, and Email, Phone, and Address (child tables) repositories.

You must use the sample LearnConfig.xml file in combination with the IndexerConfig.xml where index entities are defined for the standard Customer data model.

To customize the AI Matching learn configuration (LearnConfig.xml) file for your data model, change values of the following parameters to match your requirements:

- RootDirectory: Specify the path of your project to store and use it accordingly
- EditDirectory: Specify the path of the exported learn project

The LearnConfig.xml file is at $MQ_HOME\templates\Customer\learn. The MDMLearnConfig.xsd schema file (located at $MQ_HOME\config and schema\learn\1.0) includes details about the format of the configuration file that defines a query. After defining properties in the LearnConfig.xml file, save the file in the $MQ_HOME\config directory.

The LearnConfig.xml file consists of the following items:

Property	Description
RepositoryName	The parent or root repository name for which you want to configure the AI model.
OrigWorkflowConfigFile	The path of the original data quality workflow files.
LearnWorkflowConfigFile	The path of a workflow file generated by the AI model. It contains new Matcher activity configuration files.
ProjectName	The unique project name. The specified project names are listed on the AI Matching UI. You can specify multiple projects for a workflow file.
RootDirectory	All active and deployed models are stored in this root directory of the project. TIBCO Patterns - Learn does not

Property	Description
	create AI models in the TIBCO MDM database.
AutoDeploy	Collects pairs and auto deploys the project if the score is good. **Default value:** true **Valid values:** true and false • **true:** AI Matching verifies that the new model performs better than the currently deployed model and automatically approves the newly retrained AI model. • **false:** AI Matching recommends the next action by setting the State of the Active project instance. For information about the state of projects, see *TIBCO MDMUser's Guide*.
MaxSavedProjectHistory	The number of projects to be saved. **Default value:** 5
SampleDataFile	Create your own sample data file or the system automatically creates a sample data file. The file contains all repository names. For information about the data file, see *TIBCO Patterns - Search Learn UI Guide*.
KeyFieldName	The column or field name has unique values. Refers to the ID specified in the sample data file. The unique identifier for the rows to identify.
EditDirectory	The project path where features and data pairs are stored. If the location is not mentioned, the AI model generates the new path. If the project is specified, a new project is created to enable editing from the TIBCO Patterns - Learn UI.
MatchConfigInfo	Specify the Matcher activity details that retrieve duplicate

Property	Description
	records such as attribute name, workflow files, and rulebase files.
OrigMatchConfigFile	The path of the original workflow file, for example, common/*enterprise_internal_name*/workflow/wfin26productaddinternaleditv1.xml
LearnMatchConfigFile	The path of the workflow file modified for the repository, for example, common/*enterprise_internal_name*/workflow/wfin26productaddinternaledit-*repository_name*.xml.
OrigAttrConfigFile	The path of the original match attributes rulebase file, for example, common/*enterprise_internal_name*/rulebase/MatchAttrList.xml.
LearnAttrConfigFile	The path of the match attributes rulebase file modified for the repository, for example, common/*enterprise_internal_name*/rulebase/MatchAttrList-*repository_name*.xml.
MatchTableInfo	Includes the index entity names, parent and child repository names, and their attributes. The name of the index entity must match with the entity name mentioned in the IndexerConfig.xml file.
ModelName	The AI model name that is to be deployed on the TIBCO Patterns - Search server.
UIQueryClassName	This class is to be generated by AI Matching. It contains a query based on the defined features in TIBCO Patterns - Learn.
MDMQueryClassName	This class is to be generated by TIBCO MDM AI Matching UI. The custom query is called by the Matcher activity. The query contains all repository names and their attribute names.

Localize Text Strings

The application supports multiple locales of the application. Currently, the application uses HTML templates to generate the user interface which is viewed in a browser on the client side.

These templates contain static text written in English. The text also exists in other locations such as JavaScript functions and images displayed on the user interface, database tables, rulebase, and workflow templates.

This release provides a framework which helps you reduce the workload in maintaining multiple locales. It provides a mechanism to localize the static text in various files to the required locale.

You can localize the strings in the following areas in various languages:

- HTML pages
- JavaScript files
- Database tables
- Strings displayed on the UI
- Error messages displayed on the UI
- GI screens
- Predefined attribute groups

> **Note:** The following areas cannot be customized:
>
> - Email templates
> - Business Process Rule templates

Resource Bundles

You can localize the strings from the following components:

- HTML pages
- JavaScript files
- Database tables
- Strings displayed on the UI
- Error messages displayed on the UI

To localize these strings, use the following resource bundles:

Resource Bundles

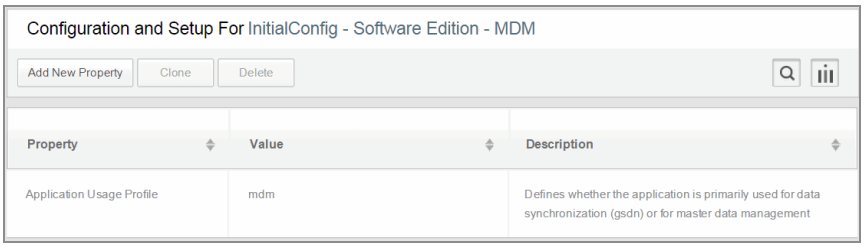
Resource Bundle	Description
htmlresources.properties	Resource bundle that contains static strings from all HTML pages in the UTF-8 format.
jsresources.properties	Resource bundle that contains strings from JavaScript files and the JavaScript section of the HTML files.
SharedDBStringResources.properties	Resource bundle that contains user interface strings from selected tables of the database.
SharedStringResources.properties	Resource bundle that contains: • Display text of all menu items • All retailer or supplier terminologies • Descriptions and variable names defined in business process rules, rulebase, and workflows which are displayed on UI. • Any generic text displayed on UI. • User-specified text which needs translation and is displayed on UI.
UserText.properties	Resource bundle that contains dynamically created informational, error, or warning messages displayed on the user interface.

Resource Bundles Specific to the Software Edition

Resource bundles specific to the software edition are bundled within the ECMClasses.jar of the deployable application ECM.ear:

- For GDSN – ECM.ear/lib/ECMClasses.jar/com/tibco/mdm/properties/gdsn
- For MDM – ECM.ear/lib/ECMClasses.jar/com/tibco/mdm/properties/mdm

The appropriate resource bundles are retrieved based on the software edition specified for the Application Usage Profile property in the Configurator (Go to **InitialConfig** > **Basic** > **Software Edition**).



htmlresources.properties

This resource bundle contains static strings from all HTML template pages in the UTF-8 format.

The format of an entry in this resource bundle is:

Default packagename from HtmlTranslator.properties.HTML filename.Unique ID for string=String

For example,

```
com.tibco.mdm.ui.infrastructure.ObjectCopy.ObjectCopy_4=Enter Copy Details
```

Where:

- com.tibco.mdm.ui.infrastructure.ObjectCopy

  Refers to the default package name from the HtmlTranslator.properties file.
- ObjectCopy

  Refers to the name of the HTML file.
- ObjectCopy_4

Refers to the unique ID for the string.

- Enter Copy Details

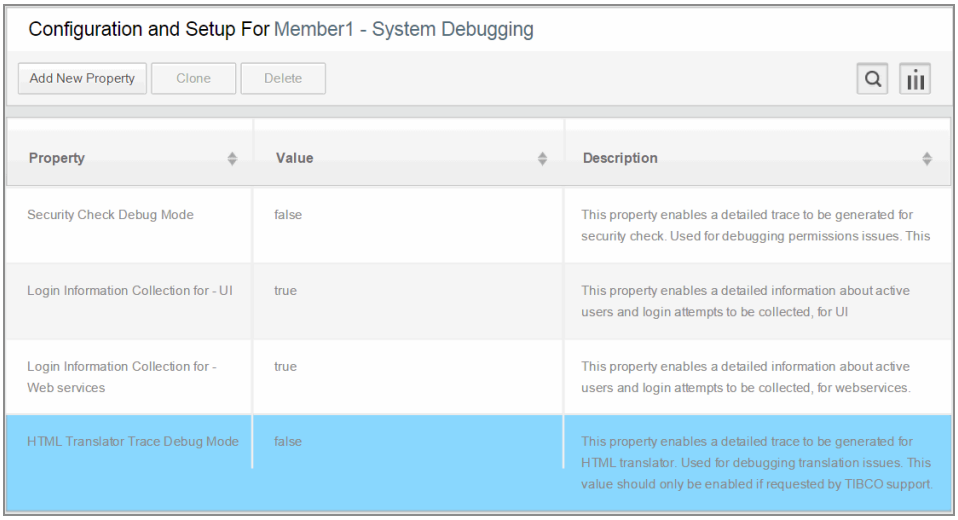
  Refers to the string.

Enabling HTML Translator Trace

To enable a detailed trace generated by HTML translator and reduce the logging of missing translations, the **HTML Translator Trace Debug Mode** property is defined in the Configurator.

- *NodeID* > **System Debugging**

Use this property for debugging translation issues. By default, the false value is defined. If TIBCO support personnel requests you to enable the property, change the value to true.



jsresources.properties

This resource bundle contains strings from JavaScript files.

The format of an entry in this resource bundle is:

Name of the JS or HTML file.Unique ID for string=String

For example, reldefui.msg2 = Relationship name is blank.\\n

Where:

- reldefui

  Refers to the name of the Javascript file.
- msg2

  Refers to the unique ID for the string, and
- Relationship name is blank.

  Refers to the string.

SharedDBStringResources.properties

This resource bundle contains strings from the database tables. For this release, only the following specific column values of database tables which are rendered on the UI have been externalized.

The columns mentioned in the following table are accessible through the resource bundle only. If there are SQL queries and Java files that access these columns, they have been modified. These columns no longer come from the result set:

Externalized Columns

Table	Column of Table
RECORDAPPROVAL	Status
RULEMETAMODEL	Name, Description
DOMAINENTRY	Description
DOMAINLINK	Description
RESOURCEACCESS	Description

Table	Column of Table
SUPPLIERSTATE	Name

The format of an entry in this resource bundle is:

TableName in upper case_PrimaryKey1 in upper case_PrimaryKey2 in upper case_ . . . _PrimaryKeyN in upper case = String

Note: Special characters (: - ? .) in the primary key column name of the database table are replaced with the underscore character (_) and then converted to upper case. Special characters (<blank> and /) are replaced by _ _.

For example, Show Record Attribute Help? is converted to SHOW__ RECORD__ATTRIBUTE__HELP_.

For example,

```
DOMAINENTRY_CURRENCY_DEM=Deutsche Mark
```

Where:

- DOMAINENTRY
  Refers to the database table.
- CURRENCY and DEM
  Refers to the primary keys.
- Deutsche Mark
  Refers to the string.

SharedStringResources.properties

This file contains static informational, error, or warning messages displayed on the user interface. It also contains text, such as the title of the page, soft link name, common drop-down items, product status, and any other text displayed on the UI.

The format of an entry in this file is:

Unique ID=String

For example,

```
UI_USER_SELECT_TIME_FORMAT=Select Time Format
```

Where:

- UI_USER_SELECT_TIME_FORMAT
  Refers to the unique ID.
- Select Time Format
  Refers to the string.

UserText.properties

This file contains dynamically created informational, error, or warning messages displayed on the UI.

The format of an entry in this file is:

Unique ID=String with Parameter name= substitute_parameter_name

The unique ID must be attached with the following prefix depending on the type of an error:

Prefix for Various Errors

Prefix	Type of Error
CAT-	Catalog error
SEC-	Security error
RUL-	Rulebase Error
GEN-	General error
SQL-	Database error

Prefix	Type of Error
WFL-	Workflow error
ADM-	Administration error
JAV-	Java Error
SVC-	Service Framework error
COM-	Communication error
CFG-	Configuration error
DQ-	Data Quality error
RB-	Rulebase error
COM-	Communication error

For example,

```
CAT-1001=Synchronization failed. Additional information: <Parameter
name='EXCEPTIONMESSAGE'>.
```

Where:

- CAT-1001

  Refers to the unique ID for a catalog error.

- Synchronization failed. Additional information: <Parameter name='EXCEPTIONMESSAGE'>

  Refers to the string.

- <Parameter name='EXCEPTIONMESSAGE'>

  Refers to the actual parameter name.

Information messages that involve runtime append or prepend of values exist in the UserText.properties file, and parameters are placed in the messages to retrieve the contextual information at runtime.

TIBCO General Interface (GI) Pages

GI based pages (input map and output map) have also been enabled for localization. Information on various styles is externalized to the styleforwidgets6_dynamic.xml file and the text strings are externalized to the sharedStringResources.xml file using the GI builder utility.

The input map and output map being GI pages identify a different type of resource bundle than those identified by HTMLs and Java servlets. For example, the resource bundles present under com/tibco/mdm/properties/mdm cannot be used by GI pages because these resource bundles are in the properties resource bundle format. GI identifies a specific XML format for resource bundles and it is bundled within EML.war of the deployable ECM.ear application. For example, JSXAPPS\sharedStringResources.xml.

To have a Japanese resource bundle, place a resource bundle consisting of Japanese strings (for example, sharedStringResources.ja.xml) under the same location. The resource bundle is picked up based on the locale set. If you have set "Japanese" as the preferred language in the profile, the Japanese resource bundle is initialized if the sharedStringResource.ja.xml file is present.

Locale Stored in Database

The locale selected by the user when creating the user account is stored in the database table Member (Locale, Language).

If the user does not choose a locale while creating or modifying the account, the fields in the database are kept blank and default JVM locale or the language selected at the login page is assigned for such users.

Customization of Resource Bundles

Customization of resource bundles signifies overriding the resource values of the default resource bundles.

You can customize the resource bundles using the following two ways:

- Customize resource values for the entire application.
- Customize resource values which apply only for an enterprise within the application. The order of retrieving resource values from the resource bundles

always starts from the enterprise-specific resource bundles, custom specific resource bundles, and then default resource bundles.

Default resource bundles

By default, TIBCO MDM provides the following resource bundles:

- htmlresources.properties
- jsresources.properties
- SharedStringResources.properties
- SharedDBStringResources.properties
- UserText.properties

Customizing Resource Bundles on JBoss WildFly Application Server

You must perform the specific steps to customize the resource bundles on the JBoss WildFly Application Server.

Before you begin

Create a custom resource bundle using the Resource Bundle editor in TIBCO MDM Studio and then deploy the customized resource bundle at the enterprise or an application level in TIBCO MDM. For information, see *TIBCO MDM Studio Repository Designer User's Guide*.

After you deploy the custom resource bundle in TIBCO MDM, the custom resources are created in the following location:

- For Windows: %MQ_HOME%/custom/resources/com/tibco/mdm/custom/main/com/tibco/mdm/properties/mdm
- For Linux: $MQ_HOME\custom\resources\com\tibco\mdm\custom\main\com\tibco\mdm\properties\mdm

The module.xml file is created in

- For Windows: %MQ_HOME%/custom/resources/com/tibco/mdm/custom/main
- For Linux: $MQ_HOME\custom\resources\com\tibco\mdm\custom\main

Procedure

1. Navigate to $JBOSS_HOME/bin or %JBOSS_HOME%\bin directory and open the standalone.sh or standalone.bat file.
2. Add the following entry:
   - For standalone.bat:

     ```
     if "x%JBOSS_MODULEPATH%" == "x" (
     set "JBOSS_MODULEPATH=%JBOSS_HOME%\modules;%MQ_
     HOME%\custom\resources"
     )
     ```

   - For standalone.sh:

     ```
     if [ "x$JBOSS_MODULEPATH" = "x" ]; then
        JBOSS_MODULEPATH="$JBOSS_HOME/modules:$MQ_
     HOME/custom/resources"
     fi
     ```

3. Save the standalone.bat or standalone.sh file.
4. Navigate to $JBOSS_HOME/standalone/configuration or %JBOSS_HOME%\standalone\configuration directory and open the standalone.xml file.
5. Add the following global module entry:

   ```
   <subsystem xmlns="urn:jboss:domain:ee:4.0">
   <global-modules>
   <module name="com.tibco.mdm.custom" slot="main"/>
   </global-modules>
   ```

6. Save the standalone.xml file.

What to do next

1. Restart TIBCO MDM server.
2. Log in to TIBCO MDM and verify the customized text.

Customizing Resource Bundles on WebSphere and WebLogic Application Servers

Before you begin

Create a custom resource bundle using the Resource Bundle editor in TIBCO MDM Studio and then deploy the customized resource bundle at the enterprise or an application level in TIBCO MDM. For information, see *TIBCO MDM Studio Repository Designer User's Guide*.

After you deploy the custom resource bundle in TIBCO MDM, the custom resources are created in the following location:

- For Windows: %MQ_HOME%/custom/resources/com/tibco/mdm/properties/mdm
- For Linux: $MQ_HOME\custom\resources\com\tibco\mdm\properties\mdm

Procedure

1. Add $MQ_HOME/custom/resources to the CLASSPATH of application server startup script.
    - For WebLogic application server: add $MQ_HOME/custom/resources to the CLASSPATH in environment file or the application server startup script.
    - For WebSphere application server:
        a. Log in to the Administrative console of WebSphere Application Server.
        b. In the left panel, expand **Servers > Server Types** and click **WebSphere application servers**. The Application servers panel is displayed on the right.
        c. Under **Preferences**, click *servername*. For example, **server1**. The **Configuration** tab is displayed.
        d. Under **Server Infrastructure**, expand **Java and Process Management** and click the **Process definition** link. The **Configuration** tab is displayed.
        e. Under **Additional Properties**, click the **Java Virtual Machine** link. The **Configuration** tab is displayed.
        f. In the **Classpath** field, enter ${MQ_HOME}/custom/resources.
2. Click **OK**.

A message is displayed with the Save and Review options.

3. Click the **Save** link to save changes to the master configuration.

What to do next

1. Restart TIBCO MDM server.
2. Log in to TIBCO MDM and verify the customized text.

Example 1 Customizing Company Logo on Login Screen

Procedure

1. Create an image of size **1920** x **427** pixels, that is, Width is equal to **1920** and Height is equal to **427**.
2. Save the image file name as **back_login** and save as type **JPEG image (*.jpg)** to the local directory.
3. Go to $MQ_HOME and double-click the ECM.ear file. The various JAR files and sub-folders are displayed.
4. Double-click the EML.war file, and then open the images folder.
5. Copy the new back_login.jpg from the local directory and replace it with the existing back_login.jpg image.
6. Restart the application server.

   The customized company logo is displayed on the Login page.

Example 2 Customizing Company Logo for Entire Application

To customize company logo for entire application:

Procedure

1. Create an image of size **1022** x **38** pixels, that is, Width is equal to **1022** and

Height is equal to **38**.

2. Save the image file name as **custom_header_logo** and save as type **JPEG image (*.jpg)** to the local directory.
3. Go to $MQ_COMMON_DIR and create the following directory structure:

   htmlprops > css > images
4. Copy the custom_header_logo.jpg image file from the local directory and place it in the htmlpropscssimages folder.
5. Create the custom.css file and copy the following css rule:

```
#logoFrame {
   background: url("/eml/Download?type=img&downloaddoc= $MQ_COMMON_
DIR/htmlprops/images/custom_header_logo.jpg") no-repeat scroll 0 0 #5682BF;
   color: #CCEEFF;}
```

   Note: The $MQ_COMMON_DIR/htmlprops/images/custom_header_logo.jpg refers to the absolute path of an image file. In a clustered environment, ensure that the shared file system path is mapped to the same volume.
6. Save the custom.css file.
7. Restart the application server.

   The customized company logo is displayed on the application.

Example 3 Customizing Company Logo for an Enterprise

To customize company logo for an enterprise, perform the steps listed in Example 2 Customizing Company Logo for Entire Application except for placing the image file path.

For an enterprise, place the customized logo in $MQ_COMMON_DIR/*enterprisename*/htmlprops folder and restart the application server. The customized company logo is displayed for an enterprise.

Creating Locale-Specific Resource Bundles

Procedure

1. Copy of the resource bundle with the English strings. Suffix the copied resource bundle with the language code and country code (optional). For example, if you plan to create Japanese locale resource bundle, copy the htmlresources.properties file to htmlresources_ja.properties.
2. Translate the strings in the resource bundles manually into the required user-specific locale.
3. Save the property file in the UTF-8 format. Convert it into the ASCII format using the Ant build script (native2ascii).
4. Rebuild the ECM.ear file.

Result

On run-time, the locale-specific property file is read and the text in the HTML elements of the original template is replaced with the translated strings.

Externalizing Buttons

You can display the text on a button in a different language, if required. The text displayed on a button is picked up from the jsresources.properties resource bundle and can be displayed in multiple languages.

Procedure

1. Add an input tag and Javascript as described in Input Tag and Javascript Tag.
2. Add a DIV tag as described in Adding a DIV Tag.
3. Create an entry in the jsresources.properties resource bundle as described in Creating an Entry in jsresources.properties.

   To display a button on the user interface, use the *INPUT* tag with the TYPE as button or submit in the HTML template. For more information, see Input Tag and Javascript Tag.

Input Tag and Javascript Tag

The input tag can be used under enclosing Anchor Tag <a> (optional). The method call buttonClicked(this) is used as the first method on the event onClick() for providing the look and feel of clicking a button. The overlibWrapper('Clear') function shows the externalized tool tip. Clear is passed as the key for the tool tip whose externalized value is present in jsresources.properties file.

- TYPE=button: Shows a button without any submit action. To submit the page using button type input tag, set the submit action on the onClick event of <INPUT> or <ANCHOR> tag.

  ```
  <INPUT ID="btnClearImage" NAME="clear" TYPE="button" VALUE="Clear"
  onClick="buttonClicked(this); someMethod ();” CLASS="buttonCls" onMouseOver="return
  overlibWrapper('Clear');" onMouseOut="return nd();">
  ```

- TYPE=submit: Shows a button with submit action. In this case, do not set any explicit submit action. For the submit input type, ensure that you do not call submit action again in any of the methods set on the event onClick().

  ```
  <INPUT ID="btnClearImage" NAME="clear" TYPE="submit" VALUE="Clear"
  CLASS="buttonCls" onClick=" buttonClicked(this); someMethod ();" onMouseOver="return
  overlibWrapper('Clear');" onMouseOut="return nd();">
  ```

Each <INPUT> tag has one associated <SCRIPT> tag, which is responsible for showing the externalized text on the button.

The following JavaScript section checks whether the ID is present or deleted in the HTML template. If it is present, the externalized text from the jsresources.properties file is set. The function getLocalizedString("Clear") is used to show the externalized button value. ‘Clear’ is the key for the button text which is present in jsresources.properties file.

```
<SCRIPT> if(document.getElementById("btnClearImage") != null)
document.getElementById("btnClearImage").value = getLocalizedString("Clear")
</SCRIPT>
```

Sample (Type=button)

```
<INPUT ID="btnClearImage" NAME="clear" TYPE="button" VALUE="Clear"
onClick="buttonClicked(this); someMethod ();” CLASS="buttonCls" onMouseOver="return
```

```
overlibWrapper('Clear');" onMouseOut="return nd();">
<SCRIPT>   if(document.getElementById("btnClearImage") != null)
document.getElementById("btnClearImage").value = getLocalizedString("Clear")
</SCRIPT>
```

Sample (Type=button: With anchor tag)

```
<A ID="btnClearHRef" onClick="buttonClicked(this); someMethod (); clearSearch();">
<INPUT ID="btnClearImage" NAME="clear" TYPE="button" VALUE="Clear" CLASS="buttonCls"
onMouseOver="return overlibWrapper('Clear');" onMouseOut="return nd();">
<SCRIPT>   if(document.getElementById("btnClearImage") != null)
document.getElementById("btnClearImage").value = getLocalizedString("Clear")
</SCRIPT>
</A>
```

Sample (Type=submit)

```
<INPUT ID="btnClearImage" NAME="clear" TYPE="submit" VALUE="Clear" CLASS="buttonCls"
onClick=" buttonClicked(this);" onMouseOver="return overlibWrapper('Clear');"
onMouseOut="return nd();">
<SCRIPT>   if(document.getElementById("btnClearImage") != null)
document.getElementById("btnClearImage").value = getLocalizedString("Clear")
</SCRIPT>
```

Sample (Type=button: With anchor tag)

Do not use an anchor tag if we are using an input tag with type ‘submit’.

Adding a DIV Tag

Check whether DIV ID="overDiv" tag is present in the HTML Template file. If it is not present, add the tag in all the HTML files. This DIV tag is used by the overlib.js file.

```
<DIV ID="overDiv" STYLE="position:absolute; visibility:hidden; z-index:1000;"> </DIV>
```

Creating an Entry in jsresources.properties

Procedure

1. Create an entry for the value corresponding to the button key in the resource bundle. For example, Back=Back.
2. Create another entry of the key suffixed with '_help_info' to show the tooltip on mouse over. For example, Back_help_info=Back. This value might be different from the button text.

```
# Externalized Buttons Text
Back=Back
Back_help_info=Back
Cancel=Cancel
Cancel_help_info=Cancel
```

Translate Dynamically Generated Text Created by Rulebase

Text constructed within a rulebase can be localized and extracted into the UserText.properties resource bundle.

The text within <op func="concat"> is dynamically constructed by the rulebase engine and saved within the WORKITEMDETAIL database table during the work item creation process. This text does not get translated when you switch to a different locale.

To translate dynamically-generated text created by a rulebase engine, use the following rulebase function which takes message text, parameter list, and the resource ID.

```
<op func="message">
    <text resourceid="">Text Message</text>
    <parameter>
          <name>parameter name</name>
        <var>parameter value to be fetch from a variable</var>
    </parameter>
</op>
```

From this rulebase function, the resource ID is assigned to the message and it is then extracted into the UserText.properties resource bundle. The text value specified is the default text to be used when no resource ID entry is found in the resource bundle.

The aforementioned snippet is then replaced with the new rulebase function as follows:

```
<action>
    <assign>
        <var>WORKITEM_DESCRIPTION</var>
        <op func = "message">
    <text resourceid="RB-1001"><![CDATA[Record $PrimaryRecord$ in Master Catalog
$MasterCatalog$ is being added. Your approval is requested. ( Total records in this workitem:
<Parameter name='RECORD_COUNT'/>)</text>
            <parameter>
                <name>RECORD_COUNT</name>
                <var>WORKITEM/RECORD_COUNT</var>
            </parameter>
        </op>
    </assign>
</action>
```

After changing the rulebase files, the resource strings with resource IDs RB-1001 must be added into the UserText.properties resource bundle.

Translating Role Names and Descriptions

Procedure

1. Form the following two keys:
    - ROLENAME_<rolename>
    - ROLEDESC_<rolename>

      For example:

      ```
      ROLENAME_ADMIN=Administrator
      ROLEDESC_ADMIN=Manages users, roles, permissions
      ```

      If any key contains blank or space (Work Supervisor), replace it with "__" (two underscores).

      If the key contains any of : (colon), / (slash), ? (question mark), . (full stop), -

(dash), replace such characters with a "_" (single underscore).

The keys must be in upper case.

2. Add the keys and translation in the sharedStringResources.prop resource bundle.

Result

The translated role name and description is displayed on the appropriate screens.

Translating Relationship Names

You can specify a display name for a relationship so that it is meaningful in a relationship tree. You can translate both the forward and reverse relationship names. This works for self relationship as well as cross-repository relationship.

Note: Specify relationship name in uppercase.

Procedure

1. Define the relationship name in sharedStringResources.properties as follows:

   REL_NAME__*repository ID*__*relationship name*=*relationship name to be displayed*

   For example:

   REL_NAME__44359__CONTAINEDBY= Address and Customer Relationship
   REL_NAME__44359__CONTAINS= Customer and Address Relationship

2. Add all customizations to a custom resource bundle.
3. Merge the bundles.

Externalize Text Displayed by Workflows

The workflow descriptions and activity descriptions can be externalized.

Workflow descriptions are defined in workflow definition files (XML). The workflow description is shown in the Event Log > Event Log Details > Description > against the steps which show the process ID.

The translation scheme is as follows:

- Translation keys: Translation keys are defined as follows.
  - For workflow description: <workflowName>_DESC
  - For workflow activity descriptions: <workflowName>_<activityName>.
- If any key contains blank ("Work Supervisor"), replace it with "__" (two underscores).
- If the key contains any of : (colon), / (slash), ? (question mark), . (full stop), - (dash), replace such characters with "_" (single underscore).
- This key must have all upper case.

Add the translation to the SharedStringResources.properties file. If no translation is found, the text provided in the XML file is displayed.

Example

Workflow is:

```
<Workflow Version="1.3" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="Workflow.xsd">
<Owner>TIBCO</Owner>
<Name>wfin26dataservicev2</Name>
<Description lang="en">Process for extraction/Import of metadata</Description>
```

Activity is:

```
<Activity Name="SetStatusToSuccess">
<Action>UpdateEvent</Action>
<Description lang="en">Set the event status 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>
```

The key is WFIN26DATASERVICEV2_DESC and WFIN26DATASERVICEV2_SETSTATUSTOSUCCESS.

Add the translation to the SharedStringResources.properties file as follows:

```
WFIN26DATASERVICEV2_SETSTATUSTOSUCCESS=Set the event status to Success/Done
WFIN26DATASERVICEV2_DESC=Process for extraction/Import of metadata
If no translation is found, the text provided in the XML file is displayed.
```

Translate Metadata

Translating the Display Name of an Attribute

While defining repository metadata, you can specify an attribute display name. This display name is shown in the UI whenever an attribute is displayed.

Procedure

1. Create a key based on attribute name (not based on display name) as follows:

   RAD__<catalogID> __<ATTRIBUTENAME>.

   RAD stands for "repository attribute display name".

   If any key contains blank (Work Supervisor), replace it with "__" (two underscores).

   If the key contains any of : (colon), / (slash), ? (question mark), . (full stop), - (dash), replace such characters with "_" (single underscore).

   The keys must be in upper case.

2. Add the translation for the key in the SharedStringResources.properties file as follows:

   ```
   RAD__34961__EFFDATE=Effective date
   ```

   The display name is then picked up from the resource bundle, whenever applicable.

   Note: This only applies to UI. Web services, rulebase, and workflow continue to use attribute names as currently practiced. If no translation is provided, the display name entered while defining the attribute is used.

Translating Repository Names

When repository metadata is defined, the repository name is provided. This name can also be translated and displayed in the language preferred by the user.

Procedure

1. Create a key based on the repository name.

   ```
   REPO_NAME__"<repository name>"
   ```

2. Add the translation for the key in the SharedStringResources.properties file as follows:

   ```
   REPO_NAME__CUSTOMER=Customers
   ```

Result

The translated name is then picked up from the resource bundle, whenever applicable.

Note:

- This only applies to UI. Web services, rulebase, and workflow continue to use repository names as currently practiced. If no translation is provided, repository name as provided in metadata is shown.
- The metadata UI to view repository metadata uses translated names whereas to modify repository metadata uses original names.
- This feature works on all screens except for Business process rules and Work Item forms.

Translating Repository Descriptions

When repository metadata is defined, a repository description is provided. This content can also be translated and displayed in the language preferred by the user.

Procedure

1. Create a key based on the repository name.

```
REPO_DESC__"<repository description>"
```

2. Add the translation for the key in the SharedStringResources.properties file as follows:

```
REPO_DESC__CUSTOMER=Customers
```

Result

The translated description is then picked up from the resource bundle, whenever applicable.

Note:
- This only applies to UI. Web services, rulebase, and workflow continue to use repository names as currently practiced. If no translation is provided, repository description as provided in metadata is shown.
- The metadata UI to view repository metadata uses translated descriptions whereas to modify repository descriptions metadata uses original descriptions.

Translating Attribute Group Names

When repository metadata is defined, an attribute group name is provided. This content can also be translated and displayed in the language preferred by the user.

Procedure

1. Create a key based on an attribute group name.

```
RAG_NAME__"attributegroupname"
```

2. Add the translation for the key in the SharedStringResources.properties file as follows:

   RAG_NAME__FINANCE=Finance

Note: The metadata UI to view or edit repository metadata does not use translated descriptions.

Translating Attribute Group Description

For each attribute group, a description is provided. This content can also be translated and displayed in the language preferred by the user.

Procedure

1. Create a key based on an attribute group name.

   ```
   RAG_NAME__"attributegroupname"_DESC
   ```

2. For predefined attribute groups, create the following key:

   ```
   "attributegroupname"_DESC
   ```

   The predefined groups include UNASSIGNED, SYSTEM, RELATIONSHIPS, LINKAGES, RELATIONSHIP_GROUP, CLASSIFICATIONS, and RELATIONSHIP_ATTRIBUTES.

3. Add the translation for the key in the SharedStringResources.properties file as follows:

   ```
   RAG_NAME__FINANCE_FINANCEDOMAIN=FinanceDomain
   ```

Result

The translated description is then picked up from the resource bundle, whenever applicable.

Translating Perspective Names

Perspective defines a subset of the relationships of a repository. The perspective is created for a root repository using the TIBCO MDM Studio. Later, it is deployed to TIBCO MDM Server.

You can browse perspectives on the View Record page. For more information, see *TIBCO MDMUser's Guide*.

Procedure

1. Create a key based on perspective name.

   ```
   PERSP_NAME__repositoryID__<perspectiveName>
   ```

2. Add the translation for the key in the SharedStringResources_ *supportedlocale*.properties file as follows:

   PERSP_NAME__12345__<perspectiveName>=OrderPerspective

   The translated name is then retrieved from the resource bundle, whenever applicable.

   Note: This feature works on record UI and resource security UI.

Load Resource Bundles for Plug-Ins

TIBCO MDM plug-ins such as GDSN Plug-in provides incremental resource bundles, such as SharedStringResources.properties, SharedDBStringResources.properties, htmlresources.properties, UserText.properties, and jsresource.properties.

These resource bundles are packaged within the plug-ins library under the standard package name, that is, com.tibco.mdm.properties.*plugin_name*.

The com.tibco.mdm.plugins property in the ConfigValues.xml file determines the order in which the resource bundle must be loaded so that a correct resource value is fetched from the right resource bundle of a plug-in.

When any plug-in is installed, a post-configuration step prepends the com.tibco.mdm.plugins property value with the plug-in name.

For example, after installing the GDSN Plug-in, the post-configuration step pre-pends "GDSN" to the com.tibco.mdm.plugins property value. A sample ConfigValues.xml looks as follows:

```
<Category description="Plugins properties" name="Plugins" visibility="Advanced">
  <ConfValue description="List of plugins installed."       isHotDeployable="false" name="Plugins
List"       propname="com.tibco.mdm.plugins" sinceVersion="8.0"       visibility="Advanced">
                <ConfString default="" value="GDSN"/>
     </ConfValue>
</Category>
```

After installing the plug-ins, the entry for com.tibco.mdm.plugins property in the ConfigValues.xml looks as follows:

```
<Category description="Plugins properties" name="Plugins" visibility="Advanced">
  <ConfValue description="List of plugins installed."     isHotDeployable="false" name="Plugins List"
   propname="com.tibco.mdm.plugins" sinceVersion="8.0"     visibility="Advanced">
          <ConfString default="" value="PluginA,GDSN"/>
      </ConfValue>
</Category>
```

In this case, the order of loading resource bundle is:

PluginA resource bundle:com.tibco.mdm.properties.pluginA

GDSN Plug-in resource bundle: com.tibco.mdm.properties.gdsn

Base product resource bundle: com.tibco.mdm.properties.mdm

Setting up Data Extraction

The Data Extractor is a framework for extracting and processing data from TIBCO MDM by providing custom implementation. You can extract data from any TIBCO MDM data source such as database, matching engines, for example TIBCO Patterns - Search or file system.

This Data Extractor framework provides interfaces where you can plug-in custom implementation for extracting and processing data. This framework needs custom implementation of these interfaces for execution.

The Extract Data web service is provided for triggering Data Extractor. For details, see *TIBCO MDM Web Services*. You can execute the extractor either by synchronously or asynchronously. Event is spawned and can be used for tracking the progress.

Data Extractor Interfaces and Concrete Implementations

The interface and implementation details are:

- Data provider interface (IDataProvider) must be implemented for extracting data. For details, see IDataProvider.
- Data processor interface (IDataProcessor) must be implemented for processing data. For details, see IDataProcessor.
- Batch Iterator (IBatchIterator) and Row Mapper (IRowMapper) are optional interfaces, which can be used for further customizing the framework. Concrete implementation of these interfaces are already provided so you can use these implementation out-of-the-box in the IDataProvider.
- Concrete implementation for IDataProvider ; AbstractDBDataProvider is provided for extracting data from database. Custom implementation can extend this for extracting data from TIBCO MDM database.

Setting up Data Extractor

Procedure

1. Create implementation for IDataProvider.
2. Create implementation for IDataProcessor.
3. Package these implementations in a JAR file and merge it with TIBCO MDM EAR.
4. Add ECMClasses.jar to the classpath. IDataProvider and IDataProcess can be found in com.tibco.mdm.repository.engine.dataextractor package.
5. Create implementation for IDataProvider. For data extraction from database extend AbstractDataDBProcessor.
6. Create implementation for IDataProcessor.
7. Package these implementations in a JAR file and merge it with TIBCO MDM EAR.
8. Send or Use Webservice Data Extractor web request.

Context Variables

IDataProvider (init,cleanup) and IDataProcessor (start,process,end) are provided with following runtime context variables:

- CONTEXT_EXEC_MODE: Execution Mode Sync or async.
- CONTEXT_BATCH_SIZE: Batch size for async messages.
- CONTEXT_EVENTID: Event ID.
- CONTEXT_PROCESSID: Process ID.
- CONTEXT_BATCH_NUMBER: Batch number of the async message.
- CONTEXT_UNIQUEID: Unique ID identifies the async message.
- CONTEXT_MEMBERID: Member ID of the member who has initiated the process.
- CONTEXT_ERRORMSG: Any execution errors in current context.
- CONTEXT_ISWORKFLOWPROCESS: Yes, if the process is triggered from workflow.
- CONTEXT_ACTIVITYNAME: Activity name, if triggered from workflow.

IDataProvider

IDataProvider interface must be implemented for data extraction. Concrete implementation can provide implementation for extracting data from any data source like RDBMS, data matching engines such as TIBCO Patterns - Search or files.

Data Extractor framework executes IDataProvider concrete implementation for data extraction. IDataProvider provides the following methods:

- **init** – Invoked only once and must be used for one time initialization such as opening files and so on. Init receives the parameters from web service request and context.
- **execute** – Must contain the logic to fetch data from the underlying data sources. It must return a Batch Iterator.
- **getTotalCount** – Must return total count of the records to be extracted in the process.
- **getParams** – Returns the parameters specified in init.
- **cleanUp** – Invoked only once and must be used clean in the end such as closing connection and so on.

For details about the IDataProvider interface, click the Help icon in the TIBCO MDM application. In the Contents section, click **API Reference** > **Java API Reference Pages**. Click the Java API Reference link in the left pane. The detailed description of each class and method in the TIBCO MDM Java API references are provided.

IDataProcessor

IDataProcessor must be implemented for processing extracted data such as writing to files, cleaning, sending emails, and so on. Data Extractor uses IDataProcessor concrete implementation for processing data.

In async mode, framework sends async messages and each message contains IDataProcessor. IDataProcessor provides the following APIs:

- **start** – Invoked only once and must be used for one time processing such as opening connections, file and so on. Start receives parameters from web service request and context.
- **process** – Must contain logic for processing data. In async processing, process is

invoked each time async Message is de serialized. Process receives parameters from web service request and context.

- **end** – Invoked only once at the end of processing. Must be used for one time processing in the end such as closing a file.

For details about the IDataProcessor interface, click the Help icon in the TIBCO MDM application. In the Contents section, click **API Reference** > **Java API Reference Pages**. Click the Java API Reference link in the left pane. The detailed description of each class and method in the TIBCO MDM Java API references are provided.

AbstractDbDataProvider

AbstractDbDataProvider is an abstract concrete implementation of IDataProvider for extracting data from TIBCO MDM database. Concrete implementation can extend this class for database related data extraction.

AbstractDataDBProvider provides implementation for IDataProvider method and also introduces two abstract methods which must be extended by concrete implementations:

- **getQuery** – Must return a SQL query.
- **setParameters** – Sets SQL Query parameter and returns a MqDebbugable statement.

The concrete implementation can override getRowMapper if custom IRowMapper implementation is used.

For details about the AbstractDbDataProvider interface, click the Help icon in the TIBCO MDM application. In the Contents section, click **API Reference** > **Java API Reference Pages**. Click the Java API Reference link in the left pane. The detailed description of each class and method in the TIBCO MDM Java API references are provided.

IBatchIterator

This is an optional interface. Similar to a java iterator interface but iterates over a collection in batches. Batch size can be passed externally from web service requests.

CollectionBatchIterator and ResultSetBatchIterator are two concrete implementations of IBatchIterator for iterating over collection and result set.

For details about the IBatchIterator interface, click the Help icon in the TIBCO MDM application. In the Contents section, click **API Reference** > **Java API Reference Pages**. Click the Java API Reference link in the left pane. The detailed description of each class and method in the TIBCO MDM Java API references are provided.

IRowMapper

IRowMapper is used by AbstractDBDataProvider for mapping single row level data from SQL result set to any Java object such as Map or List or custom value objects.

Default row mapper implementation is provided, which maps row level data to a Map.

For details about the IRowMapper interface, click the Help icon in the TIBCO MDM application. In the Contents section, click **API Reference** > **Java API Reference Pages**. Click the Java API Reference link in the left pane. The detailed description of each class and method in the TIBCO MDM Java API references are provided.

Custom Attributes in TIBCO MDM

You can create user-defined attributes in a repository cache using IFastCacheCustomOperation.

Consider a scenario, where a retailer wants to store information about the customer, which is not present in TIBCO MDM. For example, customer type, loyalty points, payback points, and so on. This information is referred to as custom attributes. To add custom attributes in TIBCO MDM, you must implement the IFastCacheCustomOperation interface.

IFastCacheCustomOperation Interface

By using the IFastCacheCustomOperation interface, you can get the external attributes and their values. You must configure the implementation class by using the com.tibco.mdm.fastcache.customattribute.implementation.class property. You can merge the property with ECM.ear or add it to the application server classpath.

The IFastCacheCustomOperation interface contains the following two methods:

- getAttributes: returns the custom attributes for the specified repository name and an enterprise, which must be created at the server startup.
- getValue: returns the value of the specified product ID and extension. This method is invoked whenever a golden copy record is added or updated in golden record cache.

For details about the IFastCacheCustomOperation interface, click the **Help** icon in the TIBCO MDM application. In the Contents section, click **API Reference > Java API Reference Pages**, and click the **Java API Reference** link in the left pane. The detailed description of each class and method is displayed in the right pane.

Customization of TIBCO GeoAnalytics

TIBCO MDM provides a standard GeoAnalytics interface through which you can create and configure your own GeoAnalytics service, and then plug it into TIBCO MDM.

The GeoAnalytics service uses the Maporama solutions, which leverage location intelligence and geospatial analytics. By using TIBCO GeoAnalytics in TIBCO MDM UI, you can map address data, and complete the partially entered information. If a new address is added, retrieval of the full address is needed. For information about using GeoAnalytics, see the chapter, "GeoAnalytics" in *TIBCO MDM User's Guide*.

MDMGeoAnalyticsService Interface

To use the GeoAnalytics service, TIBCO MDM provides a default implementation class, Geocoding Service Implementation Class(com.tibco.mdm.geo.service.impl) property in the Configurator. The property uses TIBCO GeoAnalytics (Maporama) REST APIs.

The MDMGeoAnalyticsService interface contains the following two methods:

- geocode: used for a single record. Includes the following parameters:
    - serviceUrl: the full-service URL for a single record geocoding or reverse geocoding with key or credentials. You can configure the URL using the Geo Analytics URL property in the Configurator.
    - addParams: a map of "standard address attribute" to "its value in record" to be cleansed or geocoded. The standard address attributes are COUNTRY, ZIP, CITY, STATE, STREET, and so on.
- batchGeocode: used for bulk record. Includes the following parameters:
    - serviceUrl: the full-service URL for bulk records geocoding or reverse geocoding with a key or key-value credentials. You can configure the URL by using the Geo Analytics Batch URL property in the Configurator.
    - allRecordsAddresses: a collection of a map of "standard address attribute" to "its value in record" to be cleansed or geocoded. However, a map for this method contains one additional property key-value, that is, "ID"-"*UNIQUE_RECORD_ID*". TIBCO MDMPRODUCTKEYID is used for the value of the ID.

Note:

- A GeoAnalytics service needs an account with an appropriate key or credentials to access the service or its APIs. Configure the key using the aforementioned serviceUrl parameters.
- It is a good practice to use a maximum of 200 records in one batch request in the TIBCO GeoAnalytics Maporama service. In other words, a maximum batch size for records in a workflow can be 200, and must be tuned as the recommendation of the respective service.

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.