Populating an RMC Library using XML Import

This article describes how to migrate information from XML files into Rational® Method Composer.

1 Overview

Some users of RMC like to initially define their library structure and method elements in other tools before documenting the method in RMC. Examples are Rational System Architect, Websphere Business Modeler, and even Microsoft Excel.

Other users of RMC may be migrating to using the tool for the first time, and they have a significant amount of information in another format.

Being able to import existing method information can save a lot of time. This article describes how to do that.

2 A simple example of RMC XML format

The following is a very simple example of an RMC XML file that can be used to create a library with a single plug-in, package, task, and role.

```
<?xml version="1.0" encoding="UTF-8"?>
<uma:MethodLibrary xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:uma="http://www.eclipse.org/epf/uma/1.0.6" tool="rmc=7.5.0;epf=1.5.0">
  <MethodElementProperty name="library_synFree" value="true"/>
  <MethodPlugin name="example_plugin" userChangeable="true" id="_plugin1">
    <MethodPackage xsi:type="uma:ContentPackage" name="example_package" id="_package1" >
      <ContentElement xsi:type="uma:Artifact" name="example_artifact" id="_artifact1" />
      <ContentElement xsi:type="uma:Role" name="example_role" id="_role1" />
      <ContentElement xsi:type="uma:Task" name="example_task1" id="_task1" >
             <MandatoryInput>_artifact1</MandatoryInput>
       <PerformedBy>_role1</PerformedBy>
       </ContentElement>
    </MethodPackage>
  </MethodPlugin>
</uma:MethodLibrary>
```

Here is an explanation of the XML:

The `<uma:MethodLibrary>` tag creates a Library.

```
<MethodElementProperty name="library_synFree" value="true"/>
```

- This library attribute sets the library to “autosynchronize” processes. This is the recommended default for most customers, as it ensures that when a task is used in a delivery process that it the task descriptor is kept consistent with the task from which it was created. If you don't know what this means, then just set this property to “true” as that will generally produce the results you expect.

The `<MethodPlugin>` tag creates the Plugin.

```
userChangeable="true"
```

- This ensures that the plug-in can be modified after being imported (otherwise the plug-in is marked as “locked”).

The `<MethodPackage>` tag creates a Package.

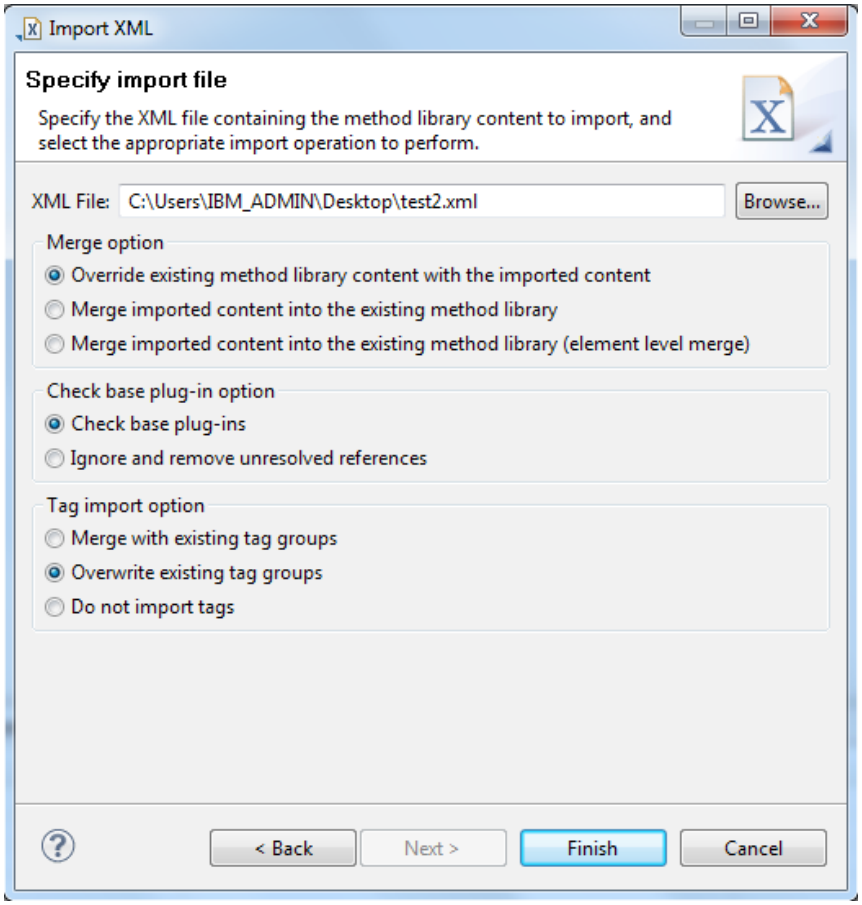
The `<ContentElement>` tags create an artifact, role, and task.

The `<MandatoryInput>` tag creates a relationship between a task and an artifact. It uses the artifact's “id” to create the relationship.

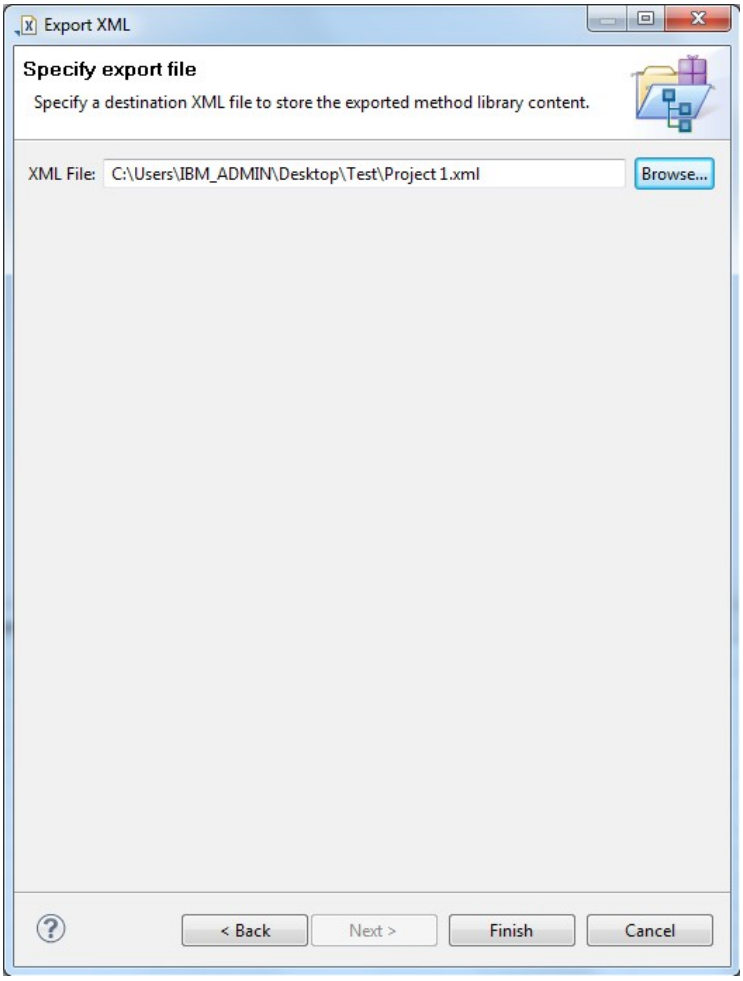
The `<PerformedBy>` tag creates a relationship between a task and a role. It uses the role's “id” to create the relationship.

3 Importing and exporting

To import an XML file into RMC, click **File > Import > Method Authoring > XML**. Browse to the XML file and click **Finish**. This process will create RMC elements from XML.



To export an XML file from RMC, click **File > Export > Method Authoring > XML**. Select Export the entire method library, then click **Next**. Then, specify a destination XML file, and click **Finish**. Give any file name for the destination XML file and it will be created. This process will now translate your entire method library into XML.



4 Importing and exporting html

The rich text (HTML) attributes of elements are stored in an “escaped” format. If you wish to work with proper HTML, you can export a plug-in or library as a set of HTML files, one file per element:

File > Export > HTML

You can then update the contents of those files and reimport them into the library. This is typically easier than working with the escaped HTML stored in the XML format.

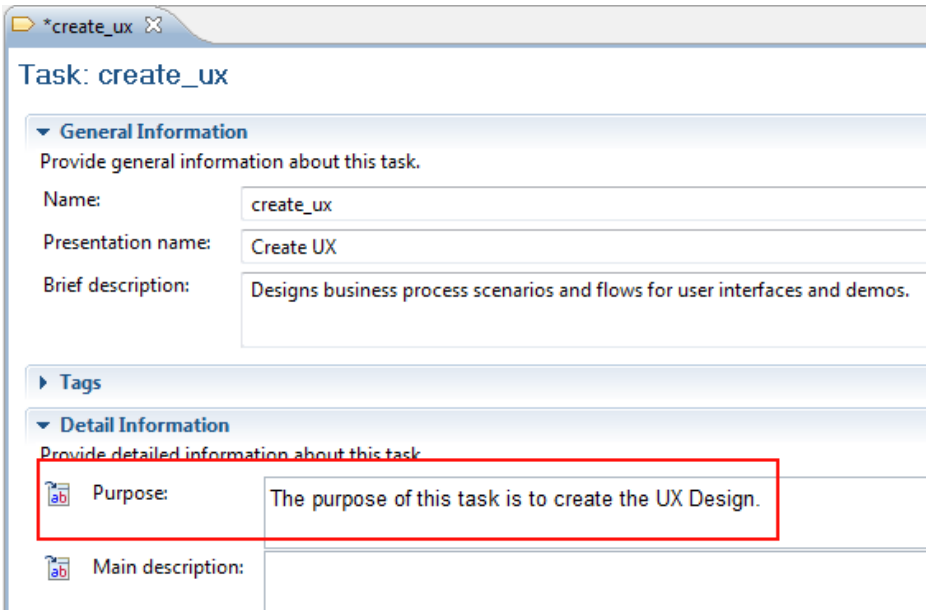
5 Understanding the RMC XML format

There are two ways to create more complex examples. The first is to study the XML schema, which is published here.

However, it may be easiest to simply create an example in RMC of the kinds of elements with the kinds of attributes and relationships that you want to import, and then export to XML. This will then generate an XML file that you can use as a template.

For example, suppose you want to create a bunch of tasks and populate each one with a "purpose".

First, create a task in RMC, and put some text in the "purpose" attribute.



Then export the library into XML. You will see the following:

```
 <ContentElement xsi:type="uma:Task" name="create_ux" briefDescription="Designs business
process scenarios and flows for user interfaces and demos." id="_qOx_0BnYEeW1T4kHCDNidA"
orderingGuide="" presentationName="Create UX" suppressed="false" isAbstract="false"
variabilityType="na">
        <MethodElementProperty name="me_edited" value="true"/>
        <Presentation xsi:type="uma:TaskDescription"
name="create_ux,_qOx_0BnYEeW1T4kHCDNidA" briefDescription="" id="-0uOKNs7ylO_wnPNxk_IYIQ"
orderingGuide="" presentationName="" suppressed="false" authors="" changeDate="2015-08-
18T01:54:14" changeDescription="" version="" externalId="">
          <KeyConsiderations></KeyConsiderations>
          <Alternatives></Alternatives>
          <Purpose><![CDATA[The purpose of this task is to create the UX Design.]]>
          </Purpose>
        </Presentation>
        <PerformedBy>_bSsl0Bk6EeWTWvUldBUvlw</PerformedBy>
        <MandatoryInput>_ZA4UMBk6EeWTWvUldBUvlw</MandatoryInput>
      </ContentElement>
```

5.1 Content Element

Content elements must be nested in content packages.

<ContentElement xsi:type="uma:*Type*" name="example_name" id="*typeID*" />
The *Type* can be an Artifact, Role, Task, Guideline, etc. The full list of types can be viewed here.

Each type has attributes, such as the following:

- briefDescription="This is a description"
- id=""
- orderingGuide=""

- presentationName="This is a presentation name"
- suppressed="false"
- isAbstract="false"
- variabilityType="na"

5.1.1 Task Description

Nested within the content element tag, you can define a description.

<ContentElement xsi:type="uma:Task">
<Presentation xsi:type="uma:TaskDescription" >
<MainDescription><![CDATA[This is the description text.]]>
</MainDescription>
</Presentation>
</ContentElement>

5.1.2 Role

Nested within the content element tag, you can define roles.
<ContentElement xsi:type="uma:Task">
<PerformedBy>*role*</PerformedBy>
</ContentElement>
The *role* is actually an "id" attribute.
Thus, if you have a
`<ContentElement xsi:type="uma:Role" name="product_owner" id="_role1" />`
Then, the performed by tag for <ContentElement xsi:type="uma:Task" name="example_task" />
would be <PerformedBy>_role1</PerformedBy>

5.1.3 Work product

Nested within the content element tag, you can define inputs and outputs.
<ContentElement xsi:type="uma:Task">
`<MandatoryInput>`*input*`</MandatoryInput>`
<Output>*output*</Output>
<OptionalInput>*input*</OptionalInput>
</ContentElement>
The *input* and *output* are actually "id" attributes.
Thus, if you have a
`<ContentElement xsi:type="uma:Artifact" name="example_artifact" id="_artifact1" />`
Then, the input tag for `<ContentElement xsi:type="uma:Task" name="example_task" />` would be
`<MandatoryInput>_artifact1</MandatoryInput>`

5.1.4 Guidance

Nested within the content element tag, you can define guidance elements.
<ContentElement xsi:type="uma:Task">
<SupportingMaterial>*guidance*</SupportingMaterial>
</ContentElement>
The *guidance* is actually an "id" attribute.

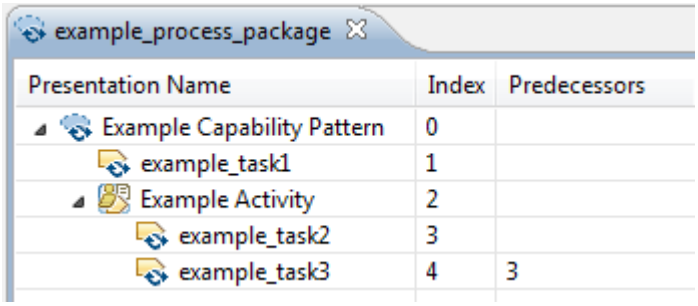
Thus, if you have a
<ContentElement xsi:type="uma:Guideline" name="example_guideline" id="_guideline1" />
Then, the supporting material tag for <ContentElement xsi:type="uma:Task" name="example_task" /> would be <SupportingMaterial>_guideline1</SupportingMaterial>

5.1.5 User-defined type

User-defined types are Practice elements and are defined in XML as follows:
<ContentElement xsi:type="uma:Practice" name="work_item_type" id="_TtO" presentationName="Work Item Type">

5.2 Process

The Process tag creates a capability pattern or delivery process. In this example, the following capability pattern is defined:



<include a picture showing an example capability pattern with one activity that in turn has two tasks nested inside, with a finish to start dependency from task 1 to task 2. >

The XML is as follows:

```
<MethodPackage xsi:type="uma:ProcessComponent" name="example_process_package"
id="_processpkg1" >
    <Process xsi:type="uma:CapabilityPattern" name="example_capability_pattern" id="_capability1"
presentationName="Example Capability Pattern" >
    <BreakdownElement xsi:type="uma:TaskDescriptor">
       <SuperActivity>_capability1</SuperActivity>
       <Task>_task1</Task>
    </BreakdownElement>
    <BreakdownElement xsi:type="uma:Activity" name="Activity" id="_activity1"
presentationName="Example Activity" >
       <SuperActivity>_capability1</SuperActivity>
       <BreakdownElement xsi:type="uma:TaskDescriptor" name="example_task2"
id="_descriptortask2" >
         <SuperActivity>_activity1</SuperActivity>
         <Task>_task2</Task>
      </BreakdownElement>
      <BreakdownElement xsi:type="uma:TaskDescriptor" name="example_task3"
id="_descriptortask3" >
         <SuperActivity>_activity1</SuperActivity>
         <Predecessor linkType="finishToStart">_descriptortask2</Predecessor>
         <Task>_task3</Task>
```

```
    </BreakdownElement>
    </BreakdownElement>
  </Process>
</MethodPackage>
```

The `<MethodPackage>` tag creates a Method Package called "example_process_package."
The `<Process>` tag creates a Capability Pattern called "example_capability_pattern."
The `<BreakdownElement>` tag can create different types of elements such as a Task Descriptor and Activity. Nested within the `<BreakdownElement>` tag are `<SuperActivity>` and `<Task>` tags. You can also nest a `<Predecessor>` tag.

In the following example, a task descriptor is created that is linked to a capability pattern and a task. "_activity1" and "_task3" are id attributes.

```
  <BreakdownElement xsi:type="uma:TaskDescriptor" name="example_task3"
id="_descriptortask3" >
      <SuperActivity>_activity1</SuperActivity>
      <Predecessor linkType="finishToStart">_descriptortask2</Predecessor>
      <Task>_task3</Task>
    </BreakdownElement>
```

6 Transform Spreadsheet into XML

Suppose you want to define your process in a spreadsheet, or you have defined your process in another tool that exports to a spreadsheet format (such as CSV). You can convert any spreadsheet containing RMC information into XML that can be imported into RMC.

As an example, we will take a Excel spreadsheet containing a task name, presentation name, and GUID and create an RMC plug-in that contains those tasks.

	A	B	C
1	Name,Presentation	Name,	GUID
2	task1,Task1,task1		
3	task2,Task2,task2		
4	task3,Task3,task3		

6.1 Convert CSV File to XML

Use an online tool such as <http://www.convertcsv.com/csv-to-xml.htm>. Alternatively, you can use Excel (see article here).

The result when using the online converter looks like the following:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<data-set xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <record>
        <Name>task1</Name>
        <Presentationname>Task 1</Presentationname>
        <GUID>task1</GUID>
    </record>
    <record>
```

```
            <Name>task2</Name>
            <Presentationname>Task 2</Presentationname>
            <GUID>task2</GUID>
      </record>
      <record>
            <Name>task3</Name>
            <Presentationname>Task 3</Presentationname>
            <GUID>task3</GUID>
      </record>
</data-set>
```

6.2 Use XLST to transform XML

Once you have your data in XML form, you can use XSLT to transform into an XML format that RMC can import. Use an online transformer such as the one here, or use the XSLT transform capabilities built into the Eclipse IDE for Java EE Developers .

XML input:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<data-set xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <record>
            <Name>task1</Name>
            <Presentationname>Task 1</Presentationname>
            <GUID>task1</GUID>
      </record>
      <record>
            <Name>task2</Name>
            <Presentationname>Task 2</Presentationname>
            <GUID>task2</GUID>
      </record>
      <record>
            <Name>task3</Name>
            <Presentationname>Task 3</Presentationname>
            <GUID>task3</GUID>
      </record>
</data-set>
```

XSL input:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:output method="xml" indent="yes"/>
<xsl:template match="/">
      <uma:MethodLibrary xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:uma="http://www.eclipse.org/epf/uma/1.0.6" tool="rmc=7.5.0;epf=1.5.0">
      <MethodElementProperty name="library_synFree" value="true"/>
      <MethodPlugin name="example_plugin" userChangeable="true" id="_plugin1">
            <MethodPackage xsi:type="uma:ContentPackage" name="example_package"
id="_package1" >
     <xsl:for-each select="data-set/record">
            <ContentElement>
                  <xsl:attribute name="xsi:type">
                        <xsl:text>uma:Task</xsl:text>
```

```
                    </xsl:attribute>
                    <xsl:attribute name="name">
                          <xsl:value-of select="Name"/>
                    </xsl:attribute>
                    <xsl:attribute name="presentationName">
                          <xsl:value-of select="Presentationname"/>
                    </xsl:attribute>
                    <xsl:attribute name="id">
                          <xsl:value-of select="GUID"/>
                    </xsl:attribute>
          </ContentElement>
     </xsl:for-each>
       </MethodPackage>
       </MethodPlugin>
       </uma:MethodLibrary>
</xsl:template>
</xsl:stylesheet>
```

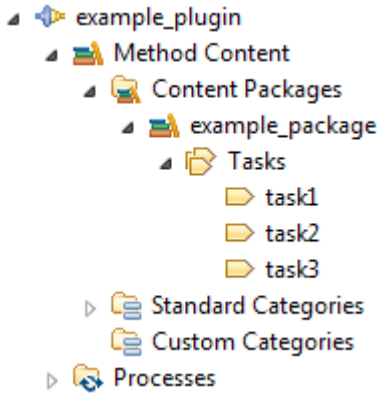
Here is an example XML transform from the previous inputs.

```
<?xml version="1.0" encoding="UTF-8"?>
<uma:MethodLibrary xmlns:uma="http://www.eclipse.org/epf/uma/1.0.6"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" tool="rmc=7.5.0;epf=1.5.0">
   <MethodElementProperty value="true" name="library_synFree" />
   <MethodPlugin id="_plugin1" userChangeable="true" name="example_plugin">
      <MethodPackage id="_package1" name="example_package"
xsi:type="uma:ContentPackage">
         <ContentElement xsi:type="uma:Task" name="task1" presentationName="Task 1"
id="task1" />
         <ContentElement xsi:type="uma:Task" name="task2" presentationName="Task 2"
id="task2" />
         <ContentElement xsi:type="uma:Task" name="task3" presentationName="Task 3"
id="task3" />
      </MethodPackage>
   </MethodPlugin>
</uma:MethodLibrary>
```

6.3 Import into RMC

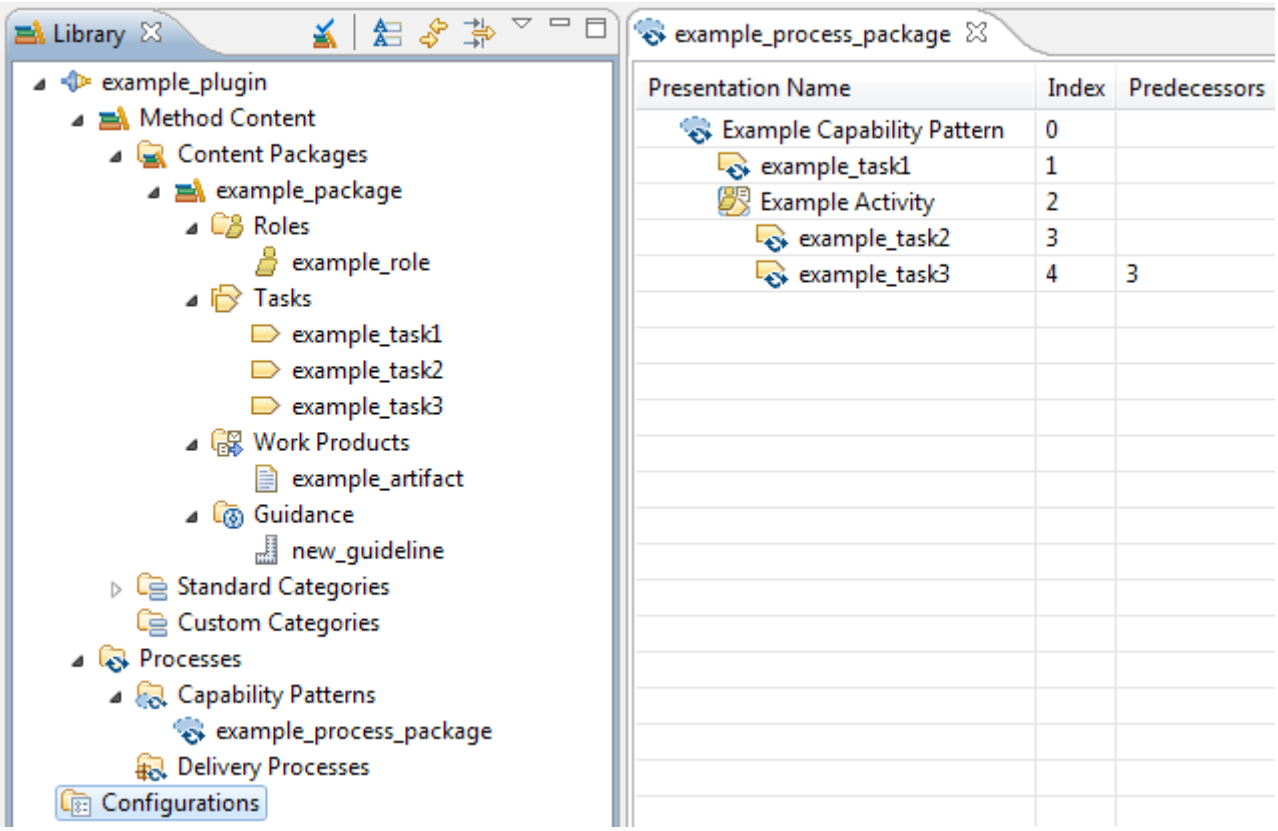
Now that you have the transformed xml, you can import it directly into RMC. **File > Import > Method Authoring > XML.**

It looks as follows:



Appendix 1: Complete Sample XML File

The XML below can be copied into a file and then imported. It demonstrates all of the examples from this article. Importing creates a library that looks like the screen shot below.



XML source:

```
<?xml version="1.0" encoding="UTF-8"?>
<uma:MethodLibrary xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:uma="http://www.eclipse.org/epf/uma/1.0.6" tool="rmc=7.5.0;epf=1.5.0">
  <MethodElementProperty name="library_synFree" value="true"/>
  <MethodPlugin name="example_plugin" userChangeable="true" id="_plugin1">
    <MethodPackage xsi:type="uma:ContentPackage" name="example_package" id="_package1" >
      <ContentElement xsi:type="uma:Artifact" name="example_artifact" id="_artifact1" />
      <ContentElement xsi:type="uma:Role" name="example_role" />
      <ContentElement xsi:type="uma:Task" name="example_task1" id="_task1" >
        <Presentation xsi:type="uma:TaskDescription" >
          <MainDescription><![CDATA[This is <strong>text</strong>.]]></MainDescription>
        </Presentation>
        <MandatoryInput>_artifact1</MandatoryInput>
             </ContentElement>
             <ContentElement xsi:type="uma:Task" name="example_task2" id="_task2" >
             </ContentElement>
             <ContentElement xsi:type="uma:Task" name="example_task3" id="_task3" >
              </ContentElement>
      <ContentElement xsi:type="uma:Guideline" name="new_guideline" />
    </MethodPackage>
    <MethodPackage xsi:type="uma:ProcessComponent" name="example_process_package"
id="_processpkg1" >
      <Process xsi:type="uma:CapabilityPattern" name="example_capability_pattern"
id="_capability1" presentationName="Example Capability Pattern" >
```

```
        <BreakdownElement xsi:type="uma:TaskDescriptor">
          <SuperActivity>_capability1</SuperActivity>
          <Task>_task1</Task>
               </BreakdownElement>
               <BreakdownElement xsi:type="uma:Activity" name="Activity" id="_activity1"
presentationName="Example Activity" >
          <SuperActivity>_capability1</SuperActivity>
          <BreakdownElement xsi:type="uma:TaskDescriptor" name="example_task2"
id="_descriptortask2" >
            <SuperActivity>_activity1</SuperActivity>
            <Task>_task2</Task>
          </BreakdownElement>
          <BreakdownElement xsi:type="uma:TaskDescriptor" name="example_task3"
id="_descriptortask3" >
            <SuperActivity>_activity1</SuperActivity>
            <Predecessor linkType="finishToStart">_descriptortask2</Predecessor>
            <Task>_task3</Task>
          </BreakdownElement>
        </BreakdownElement>
      </Process>
    </MethodPackage>
  </MethodPlugin>
</uma:MethodLibrary>
```