TEAMCENTER

MBSE Integration Gateway for Integrating Modeling Tools

Teamcenter 2412

SIEMENS

About Siemens Digital Industries Software

Siemens Digital Industries Software is a global leader in the growing field of product lifecycle management (PLM), manufacturing operations management (MOM), and electronic design automation (EDA) software, hardware, and services. Siemens works with more than 100,000 customers, leading the digitalization of their planning and manufacturing processes. At Siemens Digital Industries Software, we blur the boundaries between industry domains by integrating the virtual and physical, hardware and software, design and manufacturing worlds. With the rapid pace of innovation, digitalization is no longer tomorrow’s idea. We take what the future promises tomorrow and make it real for our customers today. Where today meets tomorrow. Our culture encourages creativity, welcomes fresh thinking and focuses on growth, so our people, our business, and our customers can achieve their full potential.

Support Center: **support.sw.siemens.com**

Send Feedback on Documentation: **support.sw.siemens.com/doc_feedback_form**

Contents

Overview of Teamcenter MBSE Integration Gateway 1-1

Understanding the Teamcenter MBSE Integration Gateway framework 2-1

Installing Teamcenter MBSE Integration Gateway integrations 3-1

Mapping the modeling tool artifacts with Teamcenter

Integration definition file 4-1

Classify objects and attributes between your modeling tool and Teamcenter using the integration definition file 4-9

Generating classification mapping using the mbse_generate_classification_maps utility 4-13

Understanding how objects and attributes are inherited in the integration definition file 4-16

Overview of Integration Mapping Editor 4-23

Map the artifacts between Teamcenter and the authoring tool using Integration Mapping Editor 4-25

Understanding the different integration modes 5-1

Using the MBSE Integration Gateway Toolkit

Overview of MBSE Integration Gateway Toolkit 6-1

Using the sample modeling tool in the MBSE Integration Gateway Toolkit 6-3

- Overview of using the sample modeling tool 6-3
- Setting up the sample modeling tool in Eclipse 6-3
- Configuring MBSE Integration Gateway using the sample modeling tool 6-6
- Loading the saved configurations in the sample modeling tool 6-7
- Generate data and a JSON file and save the data to Teamcenter 6-8
- View the imported data in Active Workspace and download the data 6-12
- Generate the delta JSON and update the data in Teamcenter 6-12
- Remove Teamcenter data from the JSON file 6-16
- Import the JSON file to the sample modeling tool 6-16
- Export or publish data from the sample modeling tool to Teamcenter asynchronously 6-16
- Use the sample modeling tool to work with product configurator 6-18

Set up the MBSE Integration Gateway Toolkit in your modeling tool 6-22

Import com.teamcenter.rac.behaviormodeling.jar in eclipse 6-23

How the deployment diagnostic tool works 6-25

Check the status of the MBSE Integration Gateway deployment using the deployment diagnostic tool 6-26

Using Common Integration Framework SOAs

Overview of Common Integration Framework SOAs 7-1
JSON schema 7-1
Using Common Integration Framework JAVA APIs 7-8
exportCollection() 7-8
exportCollectionAsync() 7-14
syncOperationStatus() 7-16
importCollection() 7-16
compareContainers() 7-18
updateCollection() 7-19
getAllTopContainers() 7-23
getTcContainerIdentifier() 7-23
getContainerData() 7-24
Active Workspace hosting APIs 7-25
MBSE Configuration APIs 7-32
getMappedObjectMapForTcType() 7-39
getClassificationMapsForTcClassType() 7-40
getClassificationMapForToolType() 7-42
getMappedRelAttributesForToolNRelType() 7-43
getMappedAttributesForToolType() 7-47
getMappedObjectMapForToolType() 7-49
Using Common Integration Framework server SOAs 7-55
exportCollection() 7-55
updateCollection() 7-67
importCollection() 7-68

Using behavior modeling APIs and extensions

Overview of using behavior modeling APIs and extensions 8-1
Customizing Teamcenter MBSE Integration Gateway operations by using APIs 8-1
Overview of using behavior modeling APIs 8-1
Operations API 8-3
bhmLogin API 8-3
bhmOpenOperation API 8-4
bhmPreSaveOperation API 8-6
bhmSaveOperation API 8-7
bhmSaveAsOperation API 8-11
bhmOpenBlockLibrary API 8-14
bhmSaveBlockLibrary API 8-16
bhmExportOperation API 8-19
bhm0GetModelOrgData API 8-21
bhmReviseOperation API 8-24
bhmCancelCheckOutOperation API 8-26
bhmLogout API 8-28

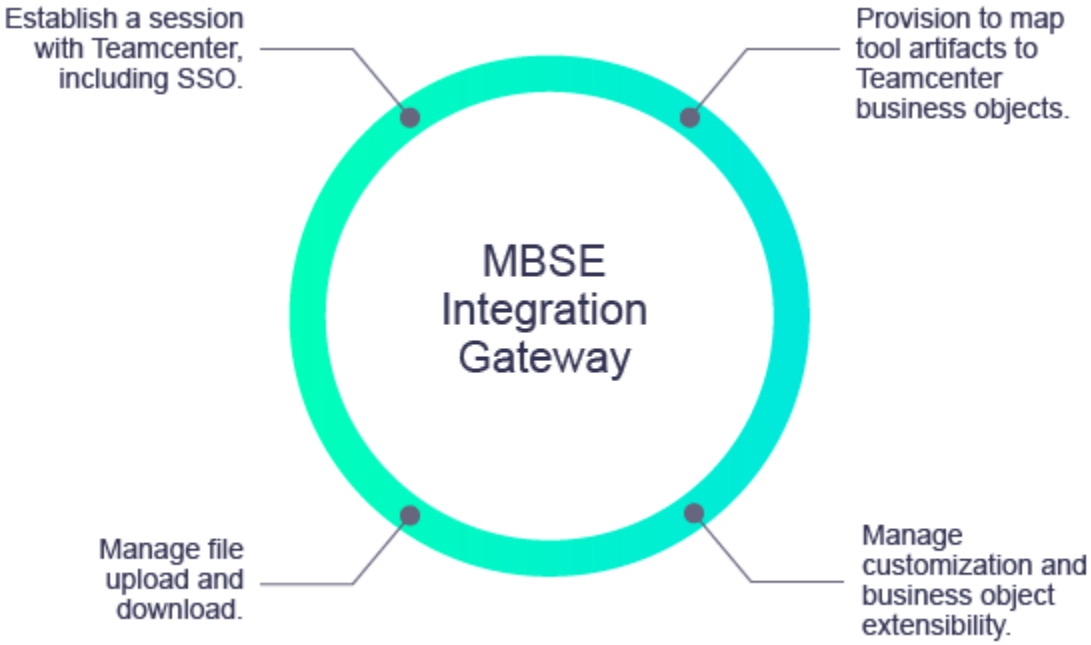
getStateOfCurrentModel API 8-28
ModelManagementFCCAPIs API 8-30
Rich client APIs 8-31
Analysis Request APIs 8-33
getAttributeData API 8-33
setAnalysisRequestData API 8-33
getAnalysisRequestData API 8-34
Customizing behavior modeling integration operations by using extensions 8-34
Configuring extensions 8-36
Configure the PRE_UI_INSERT extension 8-37

1. Overview of Teamcenter MBSE Integration Gateway

Using Teamcenter MBSE Integration Gateway, modeling tools are integrated with Teamcenter to manage the lifecycle of the models authored in the modeling tool and the associated data. The integration with Teamcenter enables modeling tools with the following functionality:

There are various ways in which a modeling tool can be integrated with Teamcenter. The integration with Teamcenter through MBSE Integration Gateway services consist of Behavior Modeling services and the Common Integration Framework.

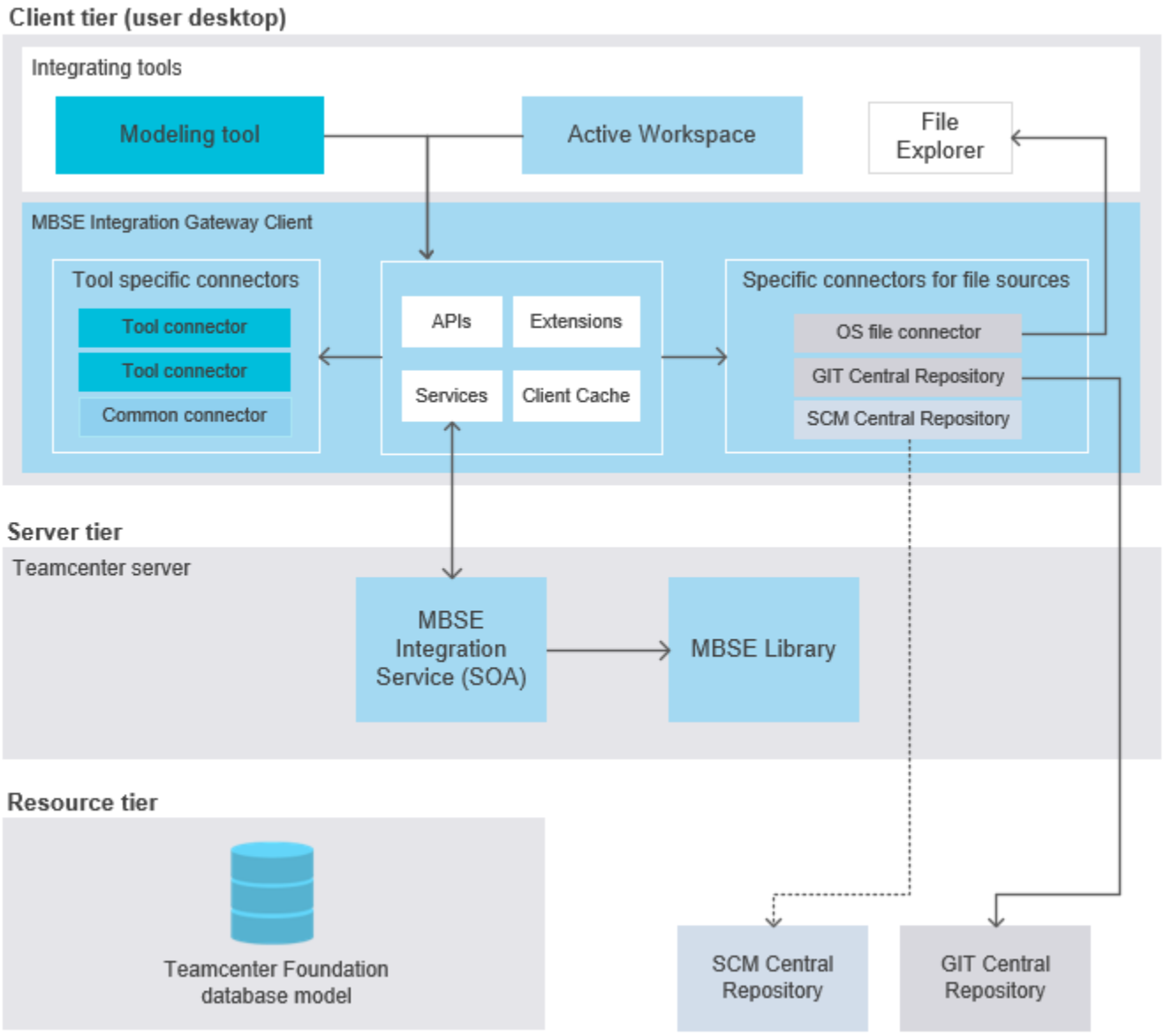
The MBSE Integration Gateway provides the following features:



Where do I go from here?

Installer	See Checklist for deploying Teamcenter MBSE Integration Gateway.
Customizer	
Understanding the different integration modes	Learn more about specific and common connector-based integration modes.
Using Common Integration Framework SOAs	See the section on Common Integration Framework SOAs to refer to the JSON schema and the client and server SOAs.

2. Understanding the Teamcenter MBSE Integration Gateway framework

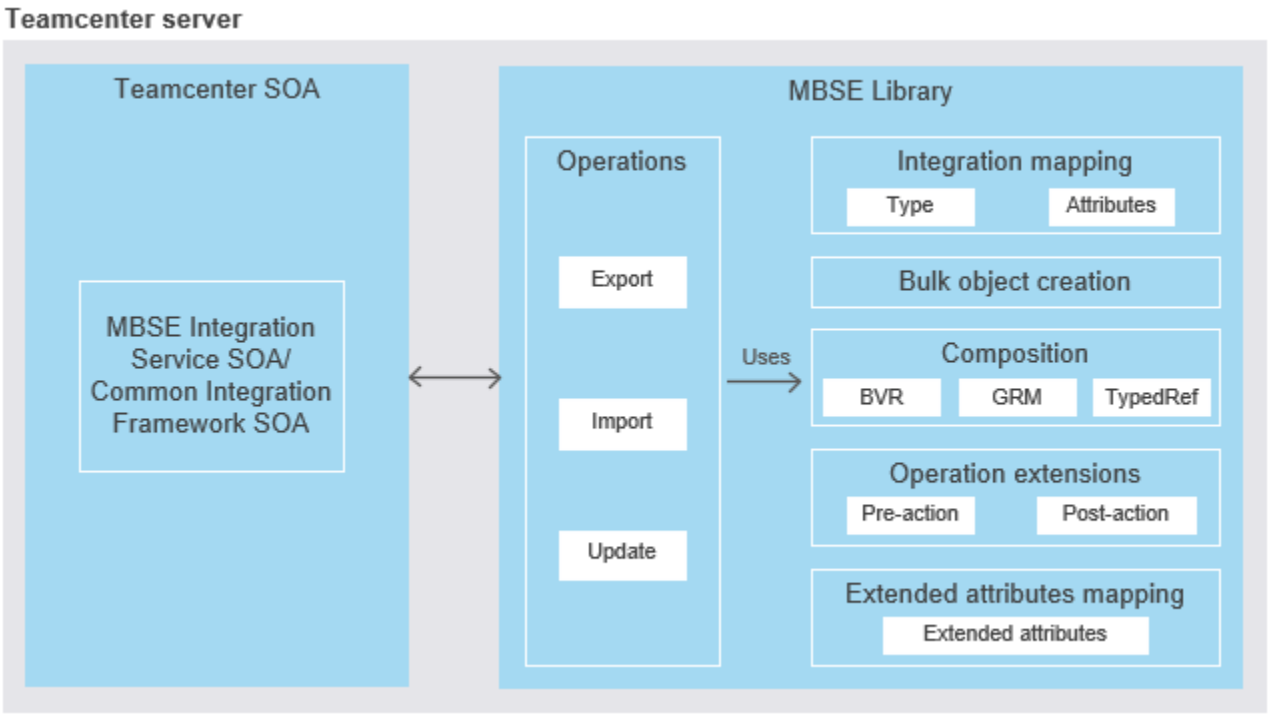


The MBSE Integration Gateway framework is the framework that supports the integration of modeling tools with Teamcenter. It consists of two main components:

- Server component
- Client component

Server component

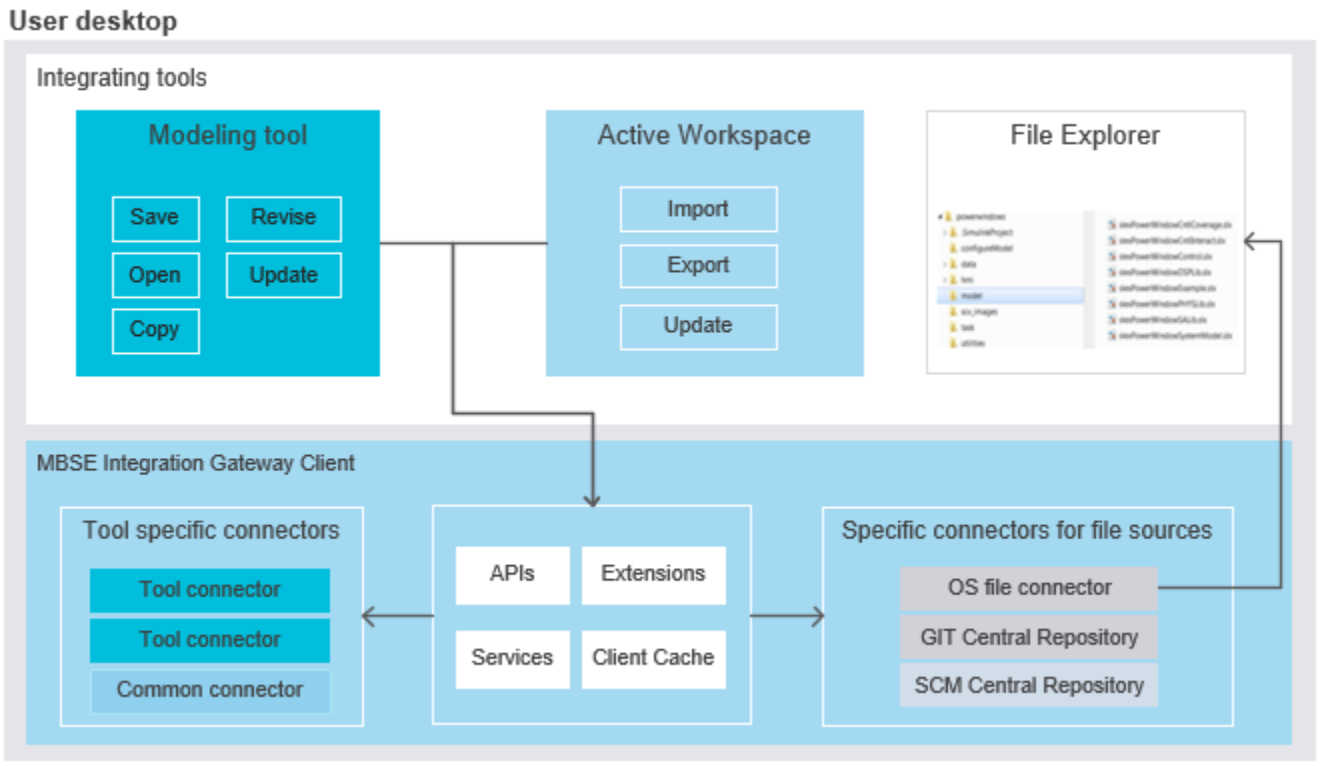
The server component resides on the Teamcenter server and contains:



- The MBSE Integration Service. This service contains MBSE Integration Gateway SOAs also called the Common Integration Framework SOAs. It takes input from the MBSE Integration Gateway client component and processes it using the MBSE library components.
- The MBSE Integration library has the operations, extensions, and other services for processing the payload. You can access these services using the Common Integration Framework SOAs.
- When you install the integrations, the data models for the same are also installed in the Teamcenter database. If you have a custom integration, ensure that you create the data model or use the data model provided by MBSE Integration Gateway.

Client component

The client component is installed on the user desktop and consists of:



1. MBSE Integration Client: This component has MBSE Integration Gateway SOAs and other components such as the client cache. This component passes the JSON payload to the Teamcenter server.

2. Connectors: The connector exchanges data between the modeling tool and Teamcenter. The data that is exchanged is mapped using a file called the integration definition file. The following types of connectors are available:

    a. Tool specific connectors: These connectors are available for specific integrations and become available when you install the integrations on the client machine. The connectors are used to process tool specific data.

    b. Common connector: This connector is used for the common or generic integration.

    c. File specific connector: This connector is used for file-based operations.

3. Modeling tool plugins: You can add plugins in your modeling tool for initiating operations with Teamcenter. You must ensure that the data you generate is in JSON format so that it can be processed by the prod-mecheng; client.

4. Active Workspace: Integration commands are available in Active Workspace.

You can also develop your client components such as connectors or plugins using the MBSE Integration Gateway Toolkit. The Toolkit is available in the Teamcenter kit.

3. Installing Teamcenter MBSE Integration Gateway integrations

For information about installing Teamcenter MBSE Integration Gateway integrations, see Deploying Teamcenter MBSE Integration Gateway.

4. Mapping the modeling tool artifacts with Teamcenter

Integration definition file

Every integration of an external tool that is done using the MBSE Integration Gateway must have an integration definition file. Configuration of the integration definition file is mandatory for the following SOAs:

- **exportCollection()**
- **importCollection()**
- **updateCollection()**

The integration definition file provides a way to configure mapping of tool artifacts and its metadata to Teamcenter business objects and properties. This integration definition file is one per tool. It is stored in a dataset with name *APPLICATION_NAME_BHM_INT_DEF_FILE*, where *APPLICATION_NAME* is a unique identifier name for a tool that is being integrated. The dataset must have administrative control.

Definition of unique application name

Every tool that gets integrated with Teamcenter is identified uniquely by the integration gateway. This information is provided by the integrators in the integration definition file as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<BHMIntegration xmlns="http://www.plmxml.org/Schemas/bhm"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.plmxml.org/Schemas/bhm"
applicationName="ECAD">
```

When various operations are performed, the integrator has to provide the value from the attribute **applicationName** as input.

Definition of the connector class

The integration gateway is executed in two modes:

- Push: In this mode, the integrating tools extract the model data and pushes this extracted data to the modeling gateway using the published APIs. This is typically done when the tool adapter or connector is an integral part of the tool.

- Pull: In this mode the integration gateway pulls the model data by invoking APIs published by the tool specific adapter or connector. The information related to the tool specific adapter or connector to be used is provided in the integration definition file. The pull mechanism is useful when bulk operations are to be performed for importing the modeling data from various modeling tools within a given modeling project.

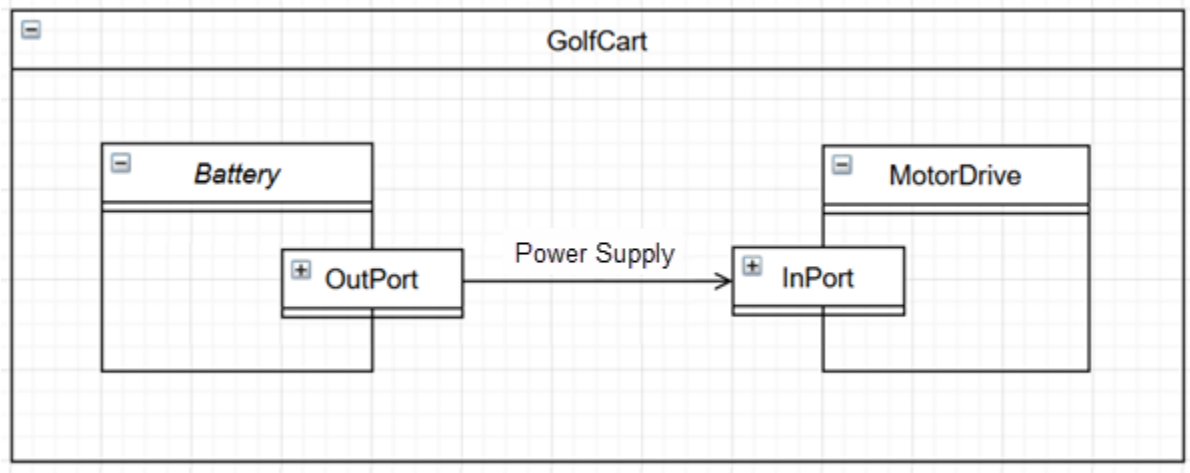
The details of the connector are given in the integration definition file as follows:

```
<ConnectorClass
fullName="com.teamcenter.behaviormodeling.matlabint.MatlabConnector"
jarFilePath="" />
```

Definition of the containers

Every modeling tool has at least two types of artifacts: a parent which acts like a container and a child which acts like a component within the container.

The container, when the model data is exported to Teamcenter from the modeling tool, is always represented in Teamcenter as an instance of some business object type. It can be a classic business object like Item or an object like dataset that holds the files or any generic workspace or POM object. In the modeling tool some containers may have children which are termed as components. When the model data is imported in Teamcenter these components are translated into instances of relationship. These relationships could be GRM, BVR or just a typed reference property on the container or a back pointer on the child container. Thus, container is considered as some business object type to be created in Teamcenter and component as some relationship. The integration definition file facilitates defining various containers from the modeling tool and its mapping to corresponding Teamcenter business object type.



In the GolfCart example GolfCart, Battery, MotorDrive, OutPort, Inport, PowerSupply all are containers.

The following code sample is a definition of a container. For each type, you must define **ObjectMapping**. Mapping for a parent type is not automatically applicable to its sub-types.

GolfCart, Battery, MotorDrive all are treated as containers by the Integration Gateway. If they are of type Model in the tool, they are mapped to the Teamcenter business object Item as follows:

```
<ObjectMapping type="Model" behaviorType="BVR" tctype="Item">
```

If the InPort and OutPort containers are of type port in the modeling tool they are mapped to Teamcenter business object GeneralDesignElement.

```
<ObjectMapping type="port" behaviorType="MIXED"
tctype="GeneralDesignElement">
```

If physical model file of GolfCart that is generated by the tool is *GolfCart.zip* then it can be stored in a Teamcenter business object of type **Zip** and is mapped as follows:

```
<ObjectMapping type="Design" behaviorType="MIXED" tctype="Zip">
```

where:

- **ObjectMapping** corresponds to the definition of a container which maps type of tool artifact to Teamcenter business object type.

ObjectMapping attributes:

- **type** is the type of container or artifact in the modeling tool.
- **tcType** is the type of business object in Teamcenter representing the container.
- **behaviorType** is the type of association between an object corresponding to the **ObjectMapping** and the object corresponding to the **BHMElement**. It can be overridden at the component level as explained in the component definition section. It may have one of the following values:
  - **BVR** If all the components are BVR occurrences.
  - **GRM** If all the components are associated with GRM relations.
  - **GBVR** If all the components are of **GBVR** type.
  - **REF** If all the components are of **REF** type.
  - **MIXED** If components are associated with more than one type of **behaviorType** such as **BVR**, **GRM**, **REF**, or **GBVR**.

Definition of all the components

Within a modeling tool a container can contain various types of components.

Each of these components are usages of some container or artifact from the tool. A component represents an association between two containers. In the integration definition file **BHMElement** is used to configure this association.

```
<ObjectMapping type="Model" behaviorType="MIXED"
tctype="Bhm0BehaviorModlItem">
  <BHMElement type="Model" tctype="Item" behaviorType="BVR">

  <BHMElement type="Design" tctype="Zip" behaviorType="GRM"
reltype="IMAN_specification" reftype ="Design" isPrimary=”false”
isPrimaryAnchor=”false” isSecondaryAnchor=”false”>

  <BHMElement type="port" tctype="GeneralDesignElement"
behaviorType="GBVR">
</ObjectMapping>
```

where:

- **BHMElement** configures the component, that is the association between two containers or **ObjectMapping**.
- **type** is the component type from the modeling tool.
- **behaviorType** is the association definition with its components.

  It may have one of the following values:

  - **BVR** If there is a parent-child (occurrence) association between the object corresponding to the **ObjectMapping** and the object corresponding to the **BHMElement**. The object corresponding to the **ObjectMapping** is parent and the object corresponding to the **BHMElement** is a child. When **behaviorType** is **BVR**, value of **relType** is not required because a Teamcenter object of type **PSOccurrence** is automatically created to associate the parent and the child object.

    For example, GolfCart and Battery containers of type Model are represented by the Teamcenter business object item. These containers have a parent-child association where GolfCart is a parent and Battery is a child.

  - **GRM** If there is a GRM relation between the object corresponding to the **ObjectMapping** and the object corresponding to the **BHMElement**. The relation object is a subtype of **ImanRelation**. Any relation object has one primary object and one secondary object. By default, the container object corresponding to the **ObjectMapping** is the primary object and the container object corresponding to the **BHMElement** is the secondary. This can be changed by configuring the **isPrimary** value.

By default, the relation is created between the revisions of the objects. You can override this behavior by configuring the **isPrimaryAnchor** and **isSecondaryAnchor** values.

- **GBVR** If the object corresponding to the **BHMElement** is a port object.
- **REF** If the object corresponding to the **BHMElement** is associated with a container object corresponding to **ObjectMapping** through a referenced property.

- **relType** defines the relation between the container corresponding to **ObjectMapping** and the container corresponding to **BHMElement**. It is a subtype of the Teamcenter business object **ImanRelation**.
- **isPrimary** This is an optional input. It is valid only if **behaviorType** is **GRM**. Every GRM relation has one primary object and one secondary object. If value of **isPrimary** is **true**, then the object mapped in **BHMElement** is primary. If value of **isPrimary** is **false**, then the object mapped in **ObjectMapping** is primary.
- **isPrimaryAnchor** This is an optional input. It is valid only if **behaviorType** is **GRM**. Every GRM relation has one primary object and one secondary object. If value of **isPrimaryAnchor** is **true**, then item of the primary object is associated with the secondary object. If value of **isPrimaryAnchor** is **false** or not set, then item revision of the primary object is associated with the secondary object.
- **isSecondaryAnchor** This is an optional input. It is valid only if **behaviorType** is **GRM**. Every GRM relation has one primary object and one secondary object. If value of **isSecondaryAnchor** is **true**, then item of the secondary object is associated with the primary object. If value of **isSecondaryAnchor** is **false** or not set, then item revision of the secondary object is associated with the primary object.

  The following are possible combinations of **isPrimaryAnchor** and **isSecondaryAnchor** values and its result on the primary object and secondary object while creating a GRM relation:

isPrimaryAnchor	isSecondaryAnchor	Container object corresponding to ObjectMapping	Container object corresponding to BHMElement
false/empty/absent	false/empty/absent	ItemRevision	ItemRevision
false/empty/absent	true	ItemRevision	Item
true	false/empty/absent	Item	ItemRevision
true	true	Item	Item

Definition of attribute mapping

Attribute mapping provides a way to map tool side metadata to properties on Teamcenter business objects. The integration definition file provides attribute mapping capability at the container as well as component level which translates to attributes on the Teamcenter business object as well as on the **Iman_relation** or **Occurrence**. For example, when an integrating tool artifact is represented by a

Teamcenter business object such as item, then the artifact name can be stored in the **object_name** property on item. The integration definition file provides a way to configure this mapping of artifact name to the object_name property of the item.

When the **AttributeMapping** is inside the **BHMElement** and if the **type** value of **BHMElement** is **RootModel** then the mapped attributes are on the Container object else they are on the association object-**GRM** relation or **BVR** occurrence.

```
 <AttributeMappings>
      <AttributeMapping name="Description" tcattr="object_desc"/>
      <AttributeMapping name="cifContainerIdentifier"
tcattr="Cif0ToolSpecificIntInfo::cif0ToolProperties"/>
      <RevisionAttributeMapping>
        <AttributeMapping name="Description" tcattr="object_desc"/>
                        <AttributeMapping name="FormDescription"
tcattr="Form::object_desc"/>
                        <AttributeMapping name="cifContainerIdentifier"
tcattr="Cif0ToolSpecificIntInfo::cif0ToolProperties"/>
                        <AttributeMapping name="ExtraMetaData1"
tcattr="Cif0ToolSpecificIntInfo::cif0ToolProperties"/>
      </RevisionAttributeMapping>
    </AttributeMappings>
```

where:

- **name** is the name of model property from the modeling tool.
- **tcattr** is the name of the attribute name on the corresponding mapped Teamcenter business object.

  Property on Form object is prefixed with **Form::**. This functionality supported only by Common Integration Framework SOAs. If there is some extra information that needs to be stored but cannot be mapped to any of the properties on a Teamcenter object it is configured as **Cif0ToolSpecificIntInfo::cif0ToolProperties**. This functionality supported only by Common Integration Framework SOAs.

  Mapping **cifContainerType** as **Cif0ToolSpecificIntInfo::cif0ToolProperties** is mandatory if multiple **ObjectMapping::type** are mapped to single **ObjectMapping::tctype**.

Integrators can map multiple attributes. The attribute mapping is supported at the Item, ItemRevision and Form level for an item type of object.

Definition of file mappings

In most of the integrations there are many different types of files that need to be imported and associated to the container objects in Teamcenter. Integration definition file supports mapping the file extension type to different dataset types and named reference in Teamcenter.

```
<FileMapping>
    <FileMap fileExt="txt" tcDatasetType="Text"
tcNameReferencedType="Text" />
    <FileMap fileExt="png" tcDatasetType="Image"
tcNameReferencedType="Image" />
    <FileMap fileExt="html" tcDatasetType="HTML"
tcNameReferencedType="HTML" />
    <FileMap fileExt="jpg" tcDatasetType="JPEG"
tcNameReferencedType="JPEG_Reference" />
    <FileMap fileExt="zip" tcDatasetType="Zip"
tcNameReferencedType="ZIPFILE" />
    <FileMap fileExt="tif" tcDatasetType="TIF"
tcNameReferencedType="TIF_Reference" />
    <FileMap fileExt="gif" tcDatasetType="GIF"
tcNameReferencedType="GIF_Reference" />
    <FileMap fileExt="*" tcDatasetType="MISC"
tcNameReferencedType="MISC_TEXT" />
</FileMapping>
```

where:

- **fileExt** is the file extension.
- **tcDatasetType** is the type of dataset that shall contain this file.
- **tcNamedReferenceType** is the type of named reference within a dataset that should be used to associate the file to the dataset.

Definition of standard folders

Many times, a system is managed as modeling project that contains various different models as a part of that project. Each project has standard folder structure and has standard output or generated files residing in some standard folders. Many times, when the models are saved to Teamcenter this derived information has to be saved and associated to the model in Teamcenter. This configuration provides a facility for the integrators to specify such standard folders so that the integration gateway automatically imports all the files in these standard folders in Teamcenter and associates it to the model object.

```
<OrganizationData>
  <Folder name="MODELFOLDER" tcRelation="TC_References"/>
</OrganizationData>
```

where:

- **name** is the name of the standard folder. **MODELFOLDER** is keyword indicating the model in which model file is residing.

- **tcRelation** specifies the type of **IMAN_Relation** to be used to associate the files from this folder to the model object

Definition of extension points

The integration operations exhibit a certain behavior. This behavior may or may not cover all the use cases of the integration. Hence the integration gateway supports extending these operations on the client side by providing extension points on all the supported operations of save, open and update. The extension points are of three types:

- **PRE-CONDITION**: Executed before the base operation begins. If this extension fails, the operation is fails.
- **PRE-ACTION**: Executed after the pre-condition but before the base operation. If this extension fails, the error is logged in log file but the operation proceeds.
- **POST-ACTION**: Executed after the base operation execution is complete. If this extension fails, the error is logged in log file but the operation proceeds.

The extension can be implemented as JAVA code, batch file or in tool native language.

```
<Extensions>
    <Extension operationName="SAVE" extensionPoint="PRE_CONDITION" >
    <Impl type="SCRIPT" location="" name="SAVE_PRE_CONDITION" />
    </Extension>
    <Extension operationName="SAVE" extensionPoint="PRE_ACTION" >
    <Impl type="BATCH" location="D:\temp" name="RunMe.bat" />
    </Extension>
    <Extension operationName="SAVE" extensionPoint="POST_ACTION" >
    <Impl type="JAVA" location=""
name="com.teamcenter.matlabcustom.SaveCustom" />
    </Extension>
</Extensions>
```

where:

- **operationName** is the name of the operation for which the extension point is being defined. It can be **SAVE**, **OPEN**, or **SAVEAS**.
- **extensionPoint** is type of extension point such as **PRE_CONDITION**, **PRE_ACTION**, or **POST_ACTION**.
- **type** is the type of implementation.
- **location** is the location on file system where the jar, script, or batch file that implements this extension resides.

- **name** is the name of the class or file that actually implements the extension.

Classify objects and attributes between your modeling tool and Teamcenter using the integration definition file

Using the integration definition file, you can define the classification for objects and attributes in your modeling tool and map them to objects and attributes in Teamcenter.

As an integrator, define the classification between the modeling tool and Teamcenter as follows:

- **Define the classification mapping in the integration definition file.**

  In this step, you specify how the objects and attributes in the modeling tool map to the Teamcenter classification objects.

- **Generate the JSON file.**

- **Use the export or import operation to classify the mapped objects.**

An example of the classification mapping in the integration definition file is as follows:

```
<ClassificationMapping>
    <AtributeCollection  toolClassId="EDABOMComp">
        <ClassAttribute toolClassAttrId="PartNumber"
tcClassAttrId="-5485" direction="TcToTool"/>
        <ClassAttribute toolClassAttrId="Mass" tcClassAttrId="-5491"
direction="ToolToTc"/>
        <ClassAttribute toolClassAttrId="OperatingTemperature"
tcClassAttrId="-5517" direction="ToolToTc"/>
    </AtributeCollection>
    <Class toolClassId="EDABOMComp" tcClassId="RNC363"
classifyAnchor="false">
        <ClassAttribute toolClassAttrId="NumberOfFans"
tcClassAttrId="-5160"/>
        <ClassAttribute toolClassAttrId="FanDiameter"
tcClassAttrId="-5159"/>
        <ClassAttribute toolClassAttrId="UnitCapacity"
tcClassAttrId="-5158"/>
        <ClassAttribute toolClassAttrId="NumbMaximumSupplyPowererOfFans"
tcClassAttrId="-5157"/>
    </Class>
    <Class toolClassId="EDABOMComp" toolClassifybyAttribute="Partition"
toolClassifybyAttributeValue="top/HEATSINK" tcClassId="RNC364"
classifyAnchor="false">
        <ClassAttribute toolClassAttrId="ThermalResistance"
tcClassAttrId="-6700" direction="TcToTool"/>
```

```
        <ClassAttribute toolClassAttrId="Height" tcClassAttrId="-6643"
direction="ToolToTc"/>
        <ClassAttribute toolClassAttrId="Length" tcClassAttrId="-6641"
direction="ToolToTc"/>
        <ClassAttribute toolClassAttrId="FinThickness"
tcClassAttrId="-5182" direction="ToolToTc"/>
    </Class>
</ClassificationMapping>
```

Define the classification mapping in the integration definition file

In the integration definition file, the classification mapping is enclosed in the **ClassificationMapping** section.

```
</ClassificationMapping>
  Classification mapping occurs here
</ClassificationMapping>
```

Within the **ClassificationMapping** section, you can map classification attributes from your tool to Teamcenter as follows:

- **Map the common attributes of the class**
- **Map the attributes of a class using simple mapping**
- **Map the attributes of a class using conditional mapping**

Map the common attributes of the class

You can map the common attributes of the class in the **AtributeCollection** section for the specified object.

```
<AtributeCollection toolClassId="EDABOMComp">
    <ClassAttribute toolClassAttrId="PartNumber" tcClassAttrId="-5485"
direction="TcToTool"/>
    <ClassAttribute toolClassAttrId="Mass" tcClassAttrId="-5491"
direction="ToolToTc"/>
    <ClassAttribute toolClassAttrId="OperatingTemperature"
tcClassAttrId="-5517" direction="ToolToTc"/>
</AtributeCollection>
```

In this section:

- **toolClassId** represents the ID of the object in the tool. The attributes are mapped for this object.
- **toolClassAttrId** represents the ID of the attribute that you want to map with the Teamcenter attribute.
- **tcClassAttrId** represents the classification ID of the attribute in Teamcenter. You can find the classification ID in the Classification workspace in Active Workspace.
- **direction** represents the flow of information. You can use the following values:
  - **ToolToTc**: The attribute mapping information is sent to Teamcenter, and the classification happens on the Teamcenter side.
  - **TcToTool**: The attribute mapping information is sent to the tool, and the classification happens on the tool side.
  - **Both**: Classification happens on both sides.

Map the attributes of a class using simple mapping

You can map the attributes of a class in the **Class** section using input variables.

```
<Class toolClassId="EDABOMComp" tcClassId="RNC363"
classifyAnchor="false">
    <ClassAttribute toolClassAttrId="NumberOfFans"
tcClassAttrId="-5160"/>
    <ClassAttribute toolClassAttrId="FanDiameter" tcClassAttrId="-5159"/>
    <ClassAttribute toolClassAttrId="UnitCapacity"
tcClassAttrId="-5158"/>
    <ClassAttribute toolClassAttrId="NumbMaximumSupplyPowererOfFans"
tcClassAttrId="-5157"/>
</Class>
```

In this section:

- **toolClassId** represents the ID of the object in the tool.
- **tcClassId** represents the ID of object in the Teamcenter.
- **classifyAnchor** represents whether you want to classify an item or an item revision. You can specify either value:

- **true**: An item is classified.
- **false**: An item revision is classified.

- **toolClassAttrId** represents the ID of the attribute that you want to map with the Teamcenter attribute.
- **tcClassAttrId** represents the classification ID of the attribute in Teamcenter. You can find the classification ID in the Classification workspace in Active Workspace.

Map the attributes of a class using conditional mapping

You can specify how the attribute in a class is classified based on the value of an attribute. In the following example, if the attribute **Partition** has the value **top/HEATSINK**, one set of classification mapping is followed. If the attribute **Partition** has the value **new/HEATSINK**, a different set of classification mapping is followed.

```
<Class toolClassId="EDABOMComp" toolClassifybyAttribute="Partition"
toolClassifybyAttributeValue="top/HEATSINK" tcClassId="RNC364"
classifyAnchor="false">
    <ClassAttribute toolClassAttrId="ThermalResistance"
tcClassAttrId="-6700" direction="TcToTool"/>
    <ClassAttribute toolClassAttrId="Height" tcClassAttrId="-6643"
direction="ToolToTc"/>
    <ClassAttribute toolClassAttrId="Length" tcClassAttrId="-6641"
direction="ToolToTc"/>
    <ClassAttribute toolClassAttrId="FinThickness" tcClassAttrId="-5182"
direction="ToolToTc"/>
</Class>
<Class toolClassId="EDABOMComp1" toolClassifybyAttribute="Partition"
toolClassifybyAttributeValue="new/HEATSINK" tcClassId="RNC364"
classifyAnchor="false">
    <ClassAttribute toolClassAttrId="ThermalResistance"
tcClassAttrId="-6700" direction="TcToTool"/>
    <ClassAttribute toolClassAttrId="Height" tcClassAttrId="-6644"
direction="ToolToTc"/>
    <ClassAttribute toolClassAttrId="Length" tcClassAttrId="-6642"
direction="ToolToTc"/>
    <ClassAttribute toolClassAttrId="FinThickness" tcClassAttrId="-5183"
direction="ToolToTc"/>
</Class>
```

You can map the attributes of a class in the **Class** section. In this section:

- **toolClassId** represents the ID of the object in the tool.

- **toolClassifybyAttribute** represents the attribute on which the classification will be defined.
- **toolClassifybyAttributeValue** represents the attribute value on which classification will be applied.
- **classifyAnchor** indicates whether you want to classify an item or an item revision. You can specify either values:
  - **true**: An item is classified.
  - **false**: An item revision is classified.
- **toolClassAttrId** represents the classification ID of the attribute in Teamcenter. You can find the classification ID in the Classification workspace in Active Workspace.
- **direction** represents the flow of information. You can use the following values:
  - **ToolToTc**: The attribute mapping information is sent to Teamcenter, and the classification happens on the Teamcenter side.
  - **TcToTool**: The attribute mapping information is sent to the tool, and the classification happens on the tool side.
  - **Both**: Classification happens on both sides.

Generate the JSON file

From your tool, you can generate a JSON file based on the JSON schema used by the MBSE Integration Gateway framework. For testing purposes, you can use the MBSE Integration Gateway Toolkit to generate the JSON file.

Use the export or import operation to classify the mapped objects

You can use the exportCollection and updateCollection APIs to update the classification in Teamcenter.

You can use the importCollection API to import the classification data into your modeling tool.

You can use the MBSE Integration Gateway Toolkit to test these operations.

Generating classification mapping using the mbse_generate_classification_maps utility

You can define the classification for objects and attributes in your modeling tool and map them to objects and attributes in Teamcenter using the integration definition file. However, this classification information can be lengthy and difficult to work with.

Using the **mbse_generate_classification_maps** utility, this classification information is generated and saved in an XML file. You can then paste the contents of this XML file in the **Classification mapping section of the integration definition file**.

Syntax

mbse_generate_classification_maps [**-u=***user-id* {**-p=***password* **|-pf=***password-file*} **-g=***group*] **-classid=***Classification-group-or-class-ID* **-outputpath=***path-and-file-name-of-output-file* [**-toolClassifybyAttribute=***toolClassifybyAttribute-value*] [**-attributeMode=owned | off | all**] [**-h**]

Arguments

-u

Specifies the user ID.

This is a user with Teamcenter administration privileges.

> Note:
>
> If Security Services single sign-on (SSO) is being used for your server, the **-u** and **-p** arguments are authenticated externally through SSO rather than being authenticated against the Teamcenter database. If you do not supply these arguments, the utility attempts to join an existing SSO session. If no session is found, you are prompted to enter a user ID and password.

-p

Specifies the password.

This argument is mutually exclusive with the **-pf** argument.

-pf

Specifies the password file.

This argument is mutually exclusive with the **-p** argument.

-g

Specifies the group associated with the user.

If used without a value, the user's default group is assumed.

-classid

Specifies the Classification class ID. The classification mapping is based on this argument. If you do not specify this argument, no output is generated.

-outputpath

Specifies the path and filename of the output XML file.

The file path should be in the format *pathname\filename*.

For example, *D:\Temp\File_Name.xml*.

-toolClassifybyAttribute

Specifies the attribute for which the classification mapping is to be done. This is an optional parameter that creates an XML file with **toolClassifybyAttribute** and **toolClassifybyAttributeValue**.

-attributeMode

Specifies the attribute mode that is used to generate the XML file. If the **attributeMode** is not mentioned, the default value is **off**.

You can specify one of the following values for this argument:

- **off**

  Excludes the classification attribute IDs associated with that class ID.

  This is the default value.

- **owned**

  Includes the classification attribute IDs associated with that class ID.

- **all**

  Includes the inherited classification attribute IDs associated with that class ID.

-h

Displays help for this utility.

Examples

Generate classification mapping without attributes

```
mbse_generate_classification_maps -u=Tc-admin-user -p=password -g=group
 -classid=ICM -outputpath=C:\temp\class_mapping_without_attributes.xml
-toolClassifybyAttribute=Partition
```

Generate classification mapping with associated attributes

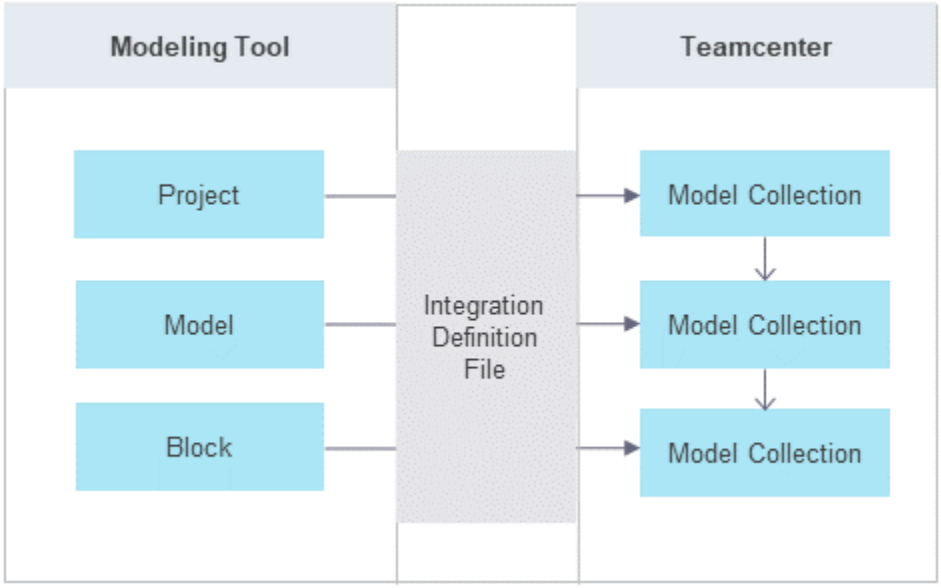
```
mbse_generate_classification_maps -u=Tc-admin-user -p=password -g=group
 -classid=ICM -outputpath=C:\temp\class_mapping_owned_attributes.xml
-toolClassifybyAttribute=Partition -attributeMode=owned
```

Generate classification mapping with inherited attributes

```
mbse_generate_classification_maps -u=Tc-admin-user -p=password -g=group
 -classid=ICM -outputpath=C:\temp\class_mapping_inherited_attributes.xml
-toolClassifybyAttribute=Partition -attributeMode=all
```

Understanding how objects and attributes are inherited in the integration definition file

To ensure that the integration between the modeling tool and Teamcenter work properly, you must correctly map the object types and attributes of the tool and Teamcenter.




```
<ObjectMapping type="PCA" behaviorType="MIXED" tctype="EDACCABase">
    <AttributeMappings>
      <AttributeMapping name="name" tcattr="object_name" includeinduplicatecheck="false"/>
      <AttributeMapping name="description" tcattr="object_desc" includeinduplicatecheck="false"/>
      <RevisionAttributeMapping>
          <AttributeMapping name="name" tcattr="object_name" includeinduplicatecheck="false"/>
          <AttributeMapping name="description" tcattr="object_desc" includeinduplicatecheck="false"/>
      </RevisionAttributeMapping>
    </AttributeMappings>
<ObjectMapping type="PCAComp" behaviorType="MIXED" tctype="EDAComPart" extends="PCA">
</ObjectMapping>
```

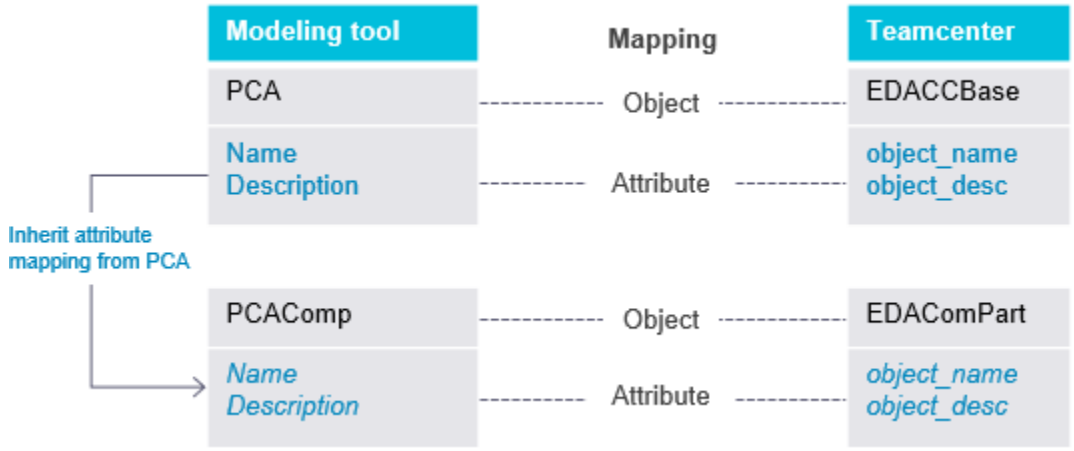
When you add new object types to your integration, you can choose to inherit the attribute mapping from an existing mapping.

For inheriting attributes, use the extends keyword.

```
<ObjectMapping type="PCA" behaviorType="MIXED" tctype="EDACCABase">
    <AttributeMappings>
        <AttributeMapping name="name" tcattr="object_name" includeinduplicatecheck="false"/>
        <AttributeMapping name="description" tcattr="object_desc" includeinduplicatecheck="false"/>
        <RevisionAttributeMapping>
            <AttributeMapping name="name" tcattr="object_name" includeinduplicatecheck="false"/>
            <AttributeMapping name="description" tcattr="object_desc" includeinduplicatecheck="false"/>
        </RevisionAttributeMapping>
    </AttributeMappings>
<ObjectMapping type="PCAComp" behaviorType="MIXED" tctype="EDAComPart" extends="PCA">
</ObjectMapping>

<ObjectMapping type="EDABOMComp" behaviorType="MIXED" tctype="" extends="PCAComp">
</ObjectMapping>
```

The following scenarios describe how the inheritance of the mapping work:

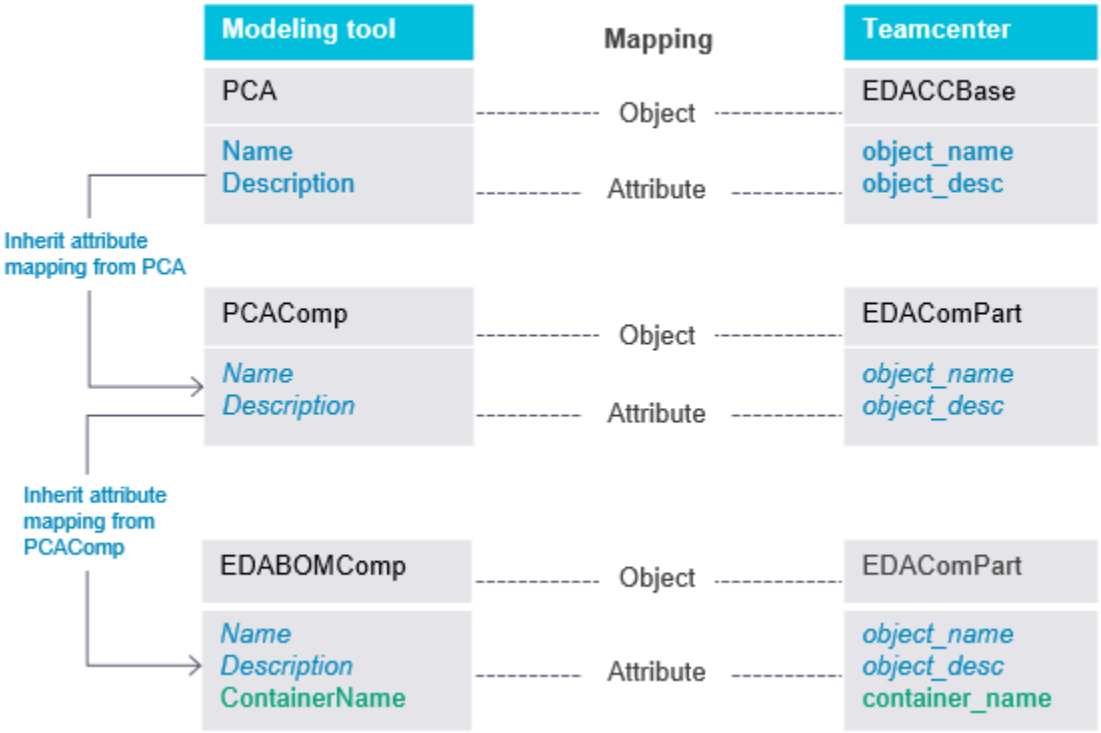
Scenario 1: Inherit mapping from the parent



In this scenario, attributes are inherited from the parent. The modeling tool type PCAComp inherits the mapping from PCA.

```
<ObjectMapping type="PCA" behaviorType="MIXED" tctype="EDACCABase">
        <AttributeMappings>
            <AttributeMapping name="name" tcattr="object_name" includeinduplicatecheck="false"/>
            <AttributeMapping name="description" tcattr="object_desc" includeinduplicatecheck="false"/>
        </AttributeMappings>
</ObjectMapping>

<ObjectMapping type="PCAComp" behaviorType="MIXED" tctype="EDAComPart" extends="PCA">
</ObjectMapping>
```

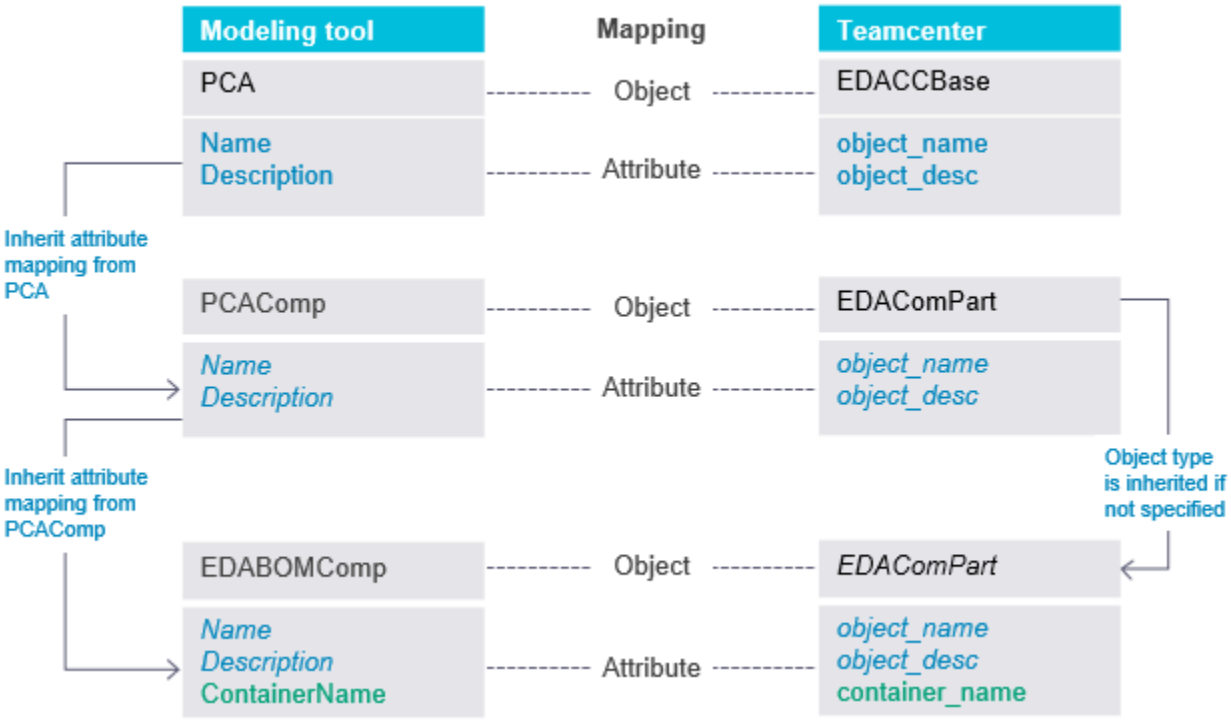
Scenario 2: Inherit attribute mapping from the parent and add new attribute mapping to the child



In this scenario, the tool type EDABOMComp inherits attributes from the parents PCAComp and PCA.

```
<ObjectMapping type="PCA" behaviorType="MIXED" tctype="EDACCABase">
        <AttributeMappings>
            <AttributeMapping name="Name" tcattr="object_name" includeinduplicatecheck="false"/>
            <AttributeMapping name="description" tcattr="object_desc" includeinduplicatecheck="false"/>
        </AttributeMappings>
</ObjectMapping>

<ObjectMapping type="PCAComp" behaviorType="MIXED" tctype="EDAComPart" extends="PCA">
</ObjectMapping>

<ObjectMapping type="EDABOMComp" behaviorType="MIXED" tctype="EDACompPart" extends="PCAComp">
        <AttributeMappings>
            <AttributeMapping name="containerName" tcattr="container_name" includeinduplicatecheck="false"/>
        </AttributeMappings>
</ObjectMapping>
```

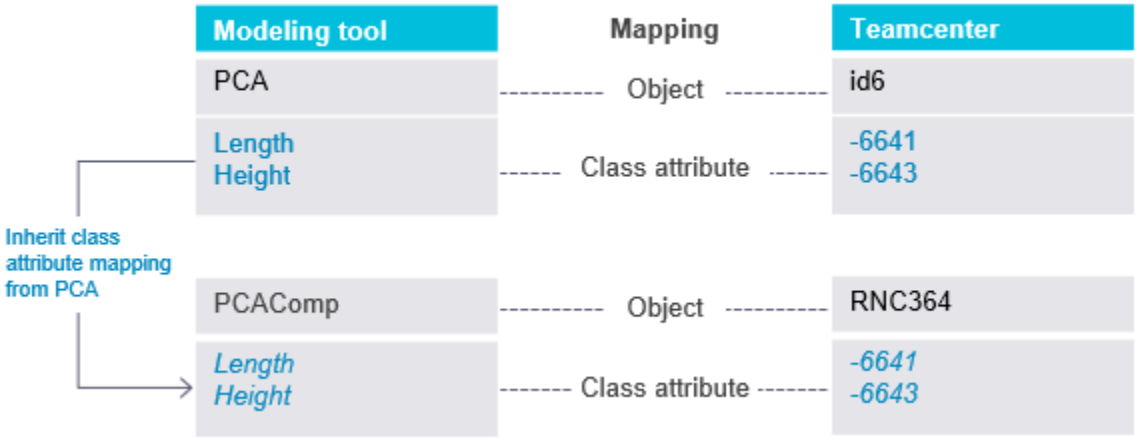
Scenario 3: Inherit Teamcenter object type when none is defined for child



This scenario only applies when you use the Integration Definition File to create the mappings. If you do not specify the Teamcenter object type for the child, it inherits the object type from the parent.

```
<ObjectMapping type="PCA" behaviorType="MIXED" tctype="EDACCABase">
        <AttributeMappings>
            <AttributeMapping name="Name" tcattr="object_name" includeinduplicatecheck="false"/>
            <AttributeMapping name="description" tcattr="object_desc" includeinduplicatecheck="false"/>
        </AttributeMappings>
</ObjectMapping>

<ObjectMapping type="PCAComp" behaviorType="MIXED" tctype="EDAComPart" extends="PCA">
</ObjectMapping>

<ObjectMapping type="EDABOMComp" behaviorType="MIXED" tctype="" extends="PCAComp">
        <AttributeMappings>
            <AttributeMapping name="containerName" tcattr="container_name" includeinduplicatecheck="false"/>
        </AttributeMappings>
</ObjectMapping>
```

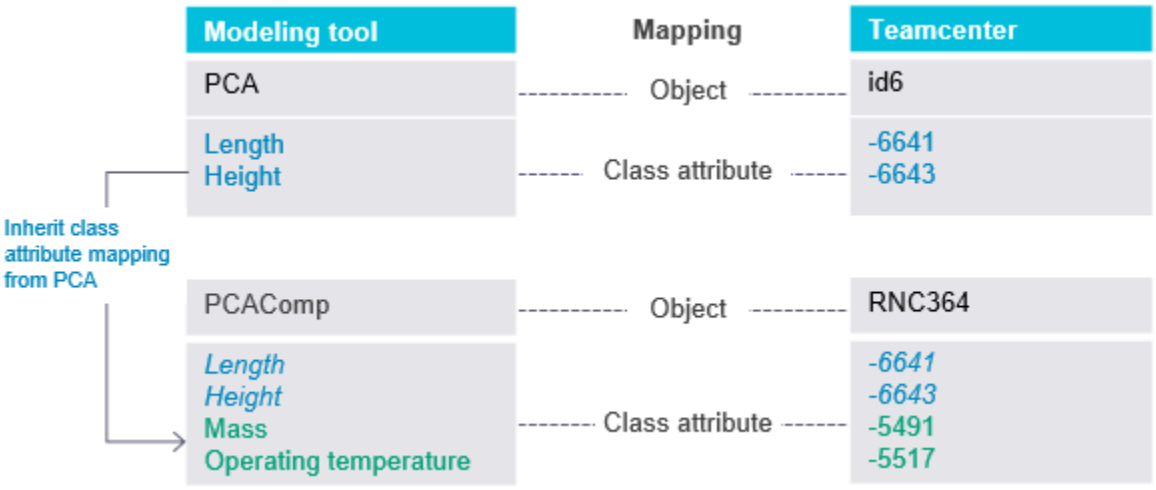
Scenario 4: Inherit classification mapping



In this scenario, the child inherits the classification attributes from the parent. The modeling tool type PCAComp inherits the mapping from PCA.

```
<ClassificationMapping>
    <Class toolClassId="PCA" tcClassId="id6" classifyAnchor="false">
        <ClassAttribute toolClassAttrId="Length" tcClassAttrId="-6641"/>
        <ClassAttribute toolClassAttrId="Height" tcClassAttrId="-6643"/>
    </Class>

    <Class toolClassId="PCAComp" tcClassId="RNC364" classifyAnchor="false" extends="PCA::id6">
    </Class>
</ClassificationMapping>
```

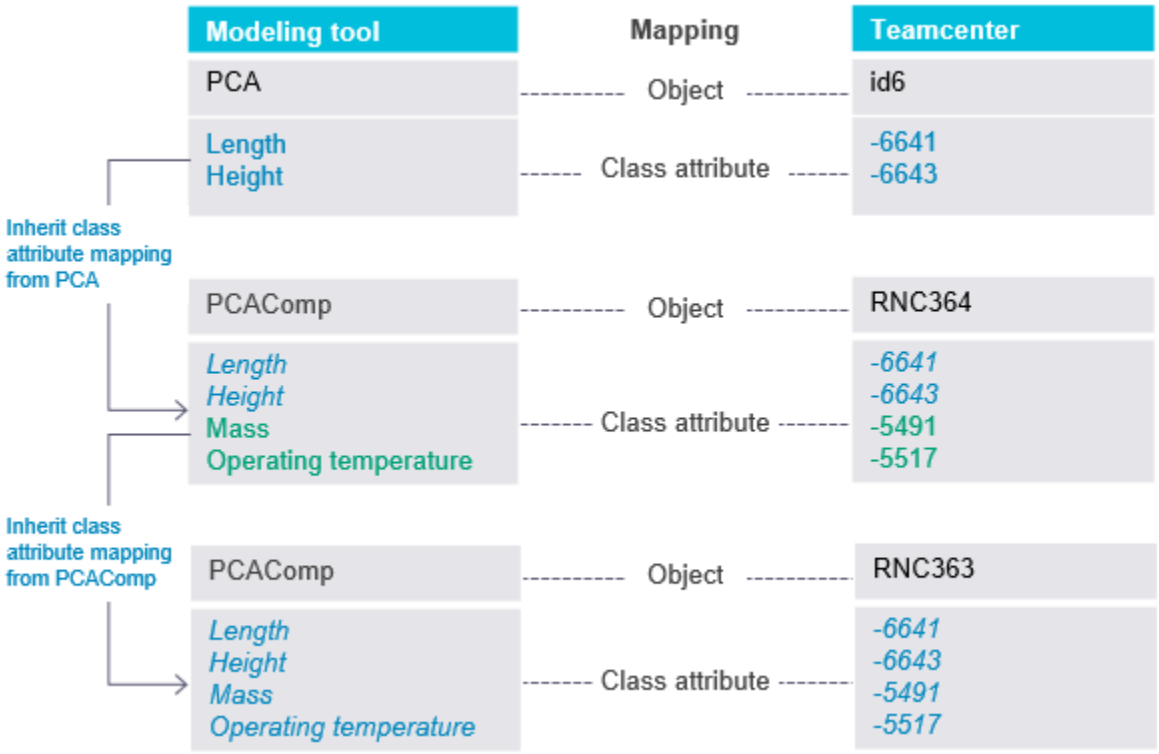
Scenario 5: Inherit classification mapping and add additional class attribute mapping



In this scenario, the child inherits the classification attributes from the parent, and you add additional classification attributes to the child.

```
<ClassificationMapping>
    <Class toolClassId="PCA" tcClassId="id6" classifyAnchor="false">
        <ClassAttribute toolClassAttrId="Length" tcClassAttrId="-6641"/>
        <ClassAttribute toolClassAttrId="Height" tcClassAttrId="-6643"/>
    </Class>

    <Class toolClassId="PCAComp" tcClassId="RNC364" classifyAnchor="false" extends="PCA::id6">
        <ClassAttribute toolClassAttrId="Mass" tcClassAttrId="-5491"/>
        <ClassAttribute toolClassAttrId="Operating Temperature" tcClassAttrId="-5517"/>
    </Class>
</ClassificationMapping>
```

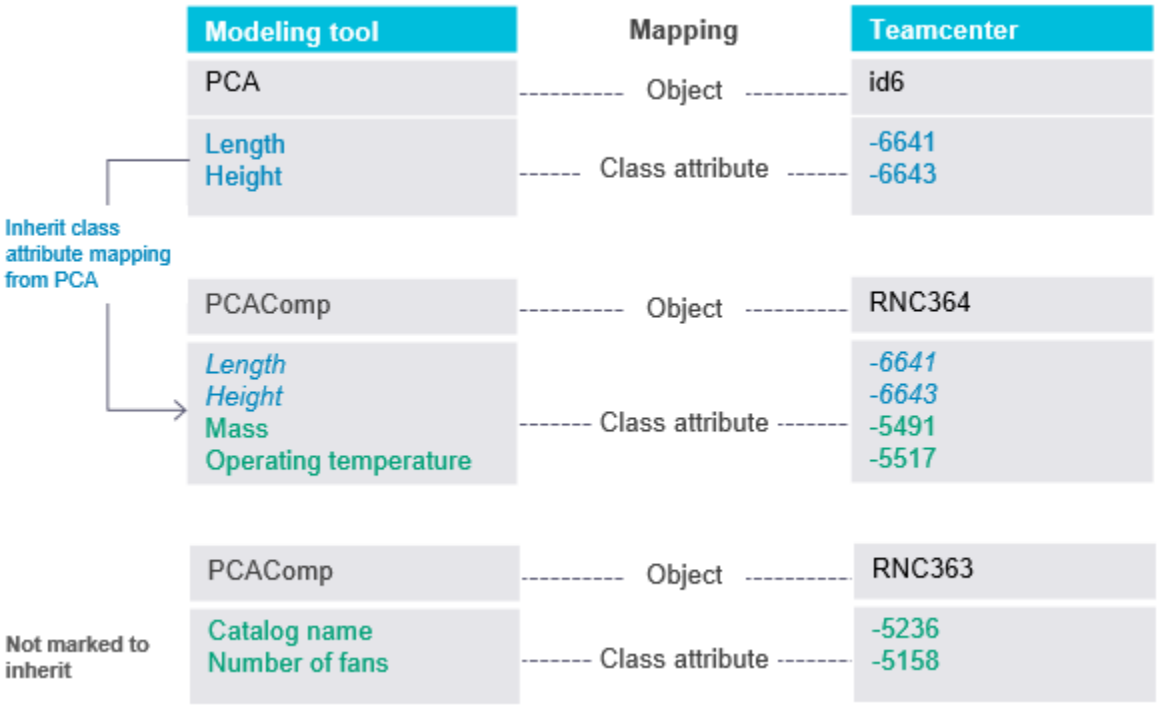
Scenario 6: Inherit classification mapping at multiple levels



In this scenario, the child inherits classification attributes from multiple parents. The tool type EDABOMComp inherits attributes from parents PCAComp and PCA.

```
<ClassificationMapping>
    <Class toolClassId="PCA" tcClassId="id6" classifyAnchor="false">
        <ClassAttribute toolClassAttrId="Length" tcClassAttrId="-6641"/>
        <ClassAttribute toolClassAttrId="Height" tcClassAttrId="-6643"/>
    </Class>

    <Class toolClassId="PCAComp" tcClassId="RNC364" classifyAnchor="false" extends="PCA::id6">
        <ClassAttribute toolClassAttrId="Mass" tcClassAttrId="-5491"/>
        <ClassAttribute toolClassAttrId="Operating Temperature" tcClassAttrId="-5517"/>
    </Class>

    <Class toolClassId="EDABOMComp" tcClassId="RNC366" classifyAnchor="false" extends="PCAComp::RNC364">
    </Class>
</ClassificationMapping>
```

Scenario 7: Inherit only classification mappings marked for inheritance



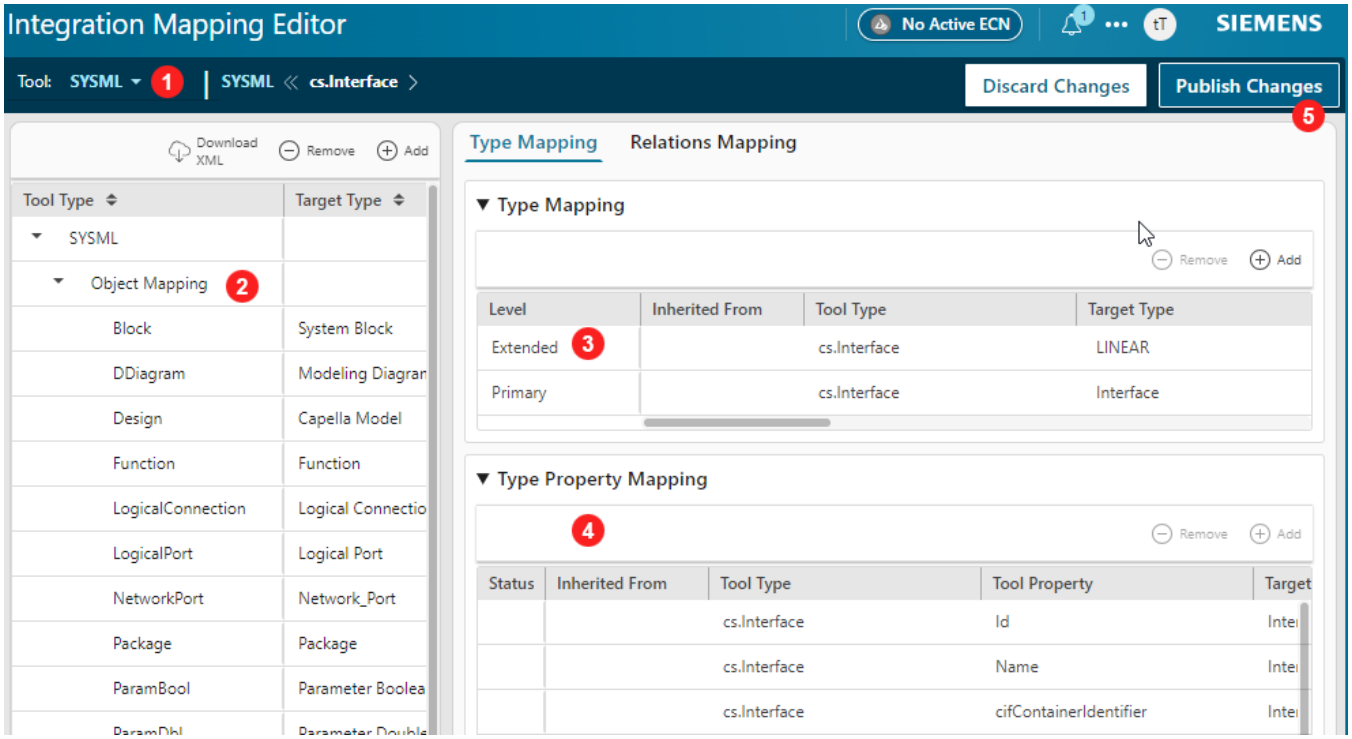
In this scenario, classification mapping exists for the same modeling tool type but the child only inherits classification mappings marked for inheritance. The rest are ignored. PCAComp inherits PCA as it is marked for inheritance. However, in the second case, PCAComp does not inherit any attributes as it is not marked for inheritance.

```
<ClassificationMapping>
    <Class toolClassId="PCA" tcClassId="id6" classifyAnchor="false">
        <ClassAttribute toolClassAttrId="Length" tcClassAttrId="-6641"/>
        <ClassAttribute toolClassAttrId="Height" tcClassAttrId="-6643"/>
    </Class>

    <Class toolClassId="PCAComp" tcClassId="RNC364" classifyAnchor="false" extends="PCA::id6">
        <ClassAttribute toolClassAttrId="Mass" tcClassAttrId="-5491"/>
        <ClassAttribute toolClassAttrId="Operating Temperature" tcClassAttrId="-5517"/>
    </Class>

    <Class toolClassId="PCAComp" tcClassId="RNC366" classifyAnchor="false">
        <ClassAttribute toolClassAttrId="CatalogName" tcClassAttrId="-5236"/>
        <ClassAttribute toolClassAttrId="NumberOfFans" tcClassAttrId="-5158"/>
    </Class>
</ClassificationMapping>
```

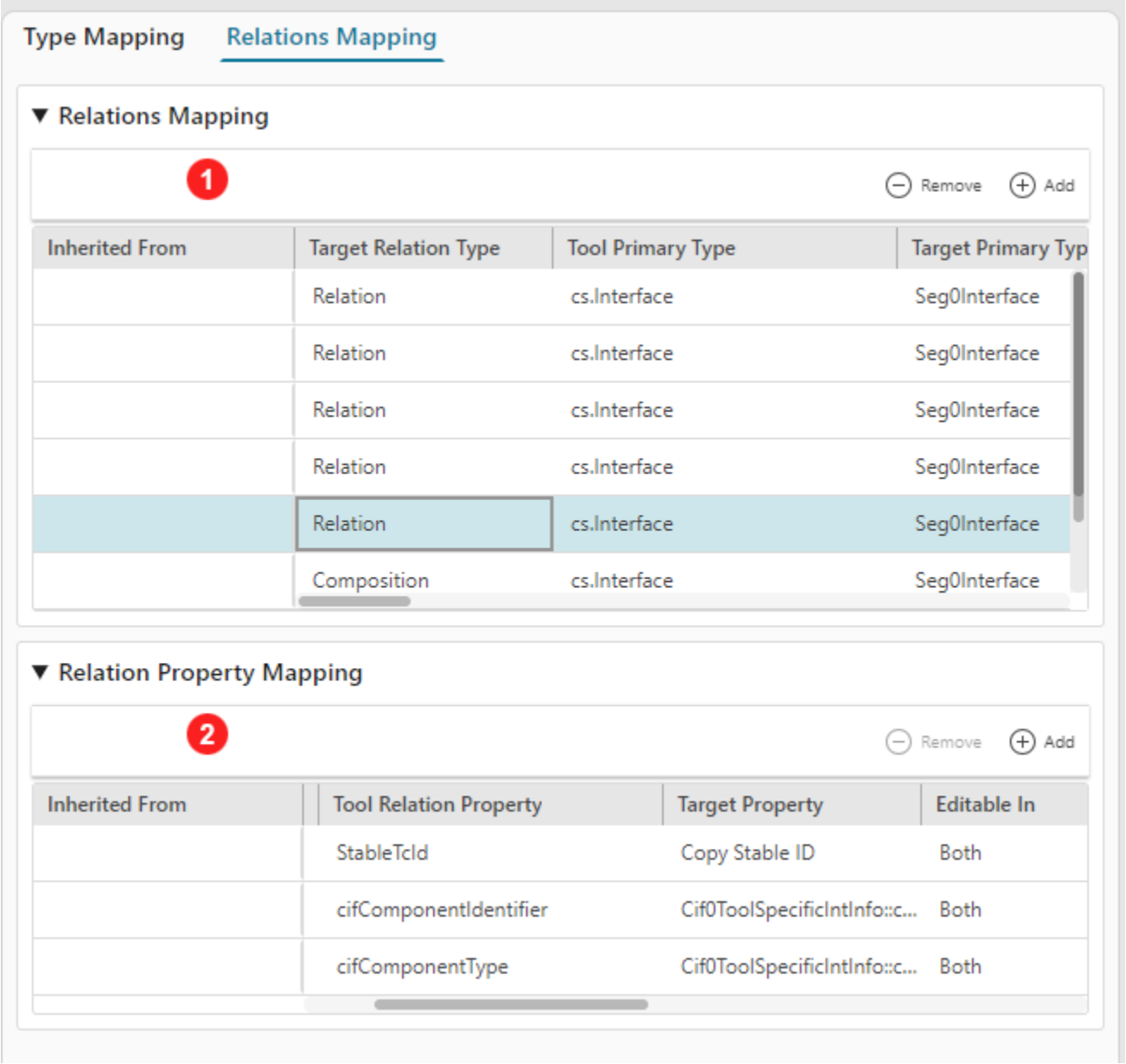
Overview of Integration Mapping Editor

Integration Mapping Editor allows you to configure the integration mapping file between the authoring tool and Teamcenter through a user interface. This editor uses the integration definition file as input and maps:

- The object types from the authoring tool with types from Teamcenter.
- The object types from the authoring tool with Teamcenter classification types. Conditional mapping of the object type from the authoring tool with multiple Teamcenter classification types is also possible.
- The properties of the object types.
- The relations from the authoring tool with relations in Teamcenter.
- The properties on relations.



1	Integration definition file	Choose the integration definition file.
2	Object Mapping	Map the object types between the authoring tool and Teamcenter.
3	Extended Mapping	Map classification classes between the authoring tool and Teamcenter.
4	Type Property Mapping	Map the relations between the authoring tool and Teamcenter.
5	Publish	Publish the changes to Teamcenter.



1	Relations Mapping	Map the relations between the authoring tool and Teamcenter.
2	Relation Property Mapping	Map the properties of the relations.

Map the artifacts between Teamcenter and the authoring tool using Integration Mapping Editor

You can do the following with Integration Mapping Editor:

- **Open Integration Mapping Editor and select the integration definition file.**

- **Create mapping between the authoring tool object types and the Teamcenter object types.**
- **Create the mapping of classification artifacts.**
- **Create mapping between properties of the authoring tool types and the Teamcenter object types.**
- **Create mapping between relations in the authoring tool and Teamcenter.**
- **Create mapping between properties of the relations in the authoring tool and Teamcenter.**
- **Publish your changes.**

Prerequisites

Ensure that the integration that you want to configure using Integration Mapping Editor has a corresponding integration definition file in Teamcenter. If not, you will not see an entry for the integration in Integration Mapping Editor.

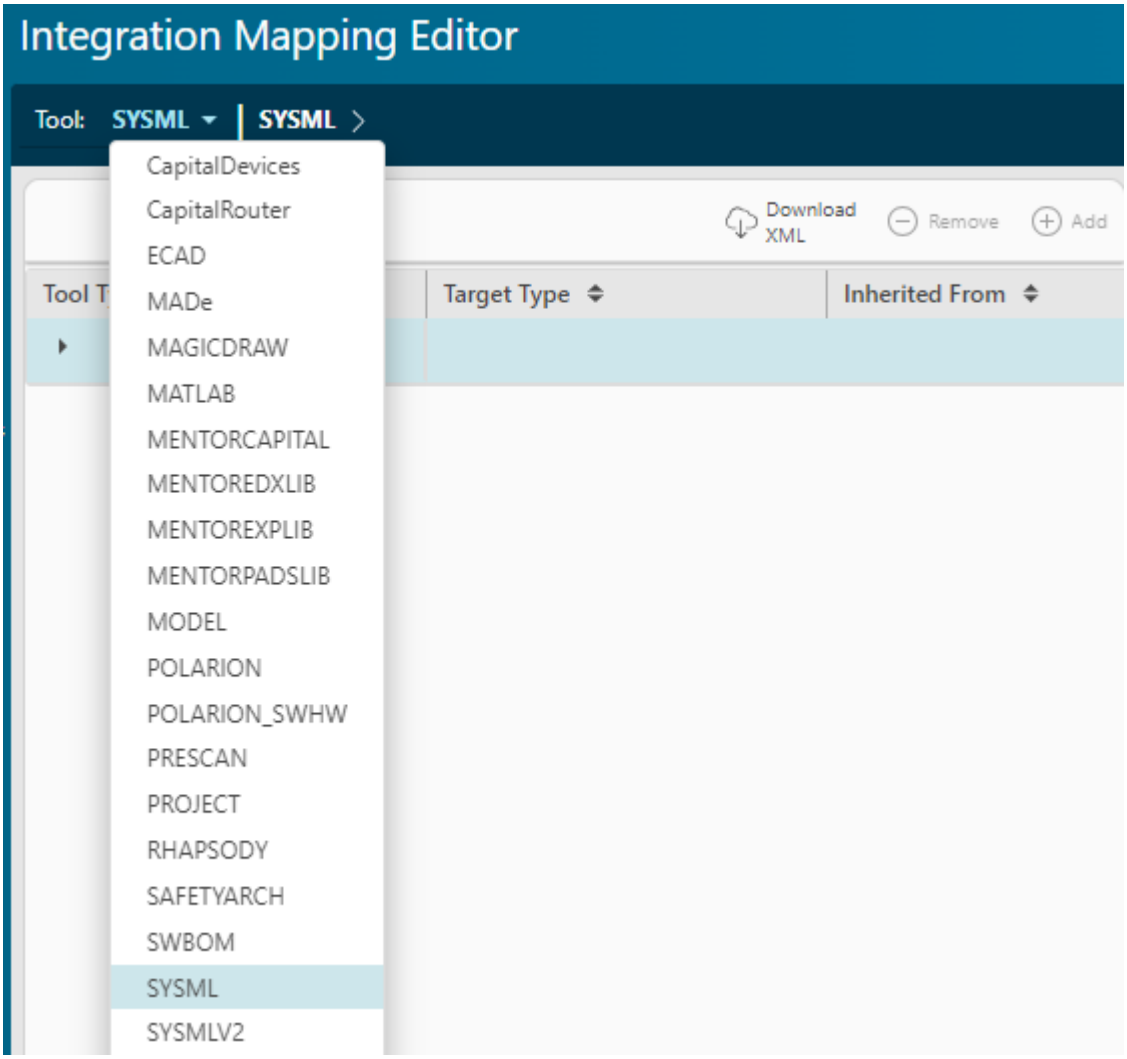
Procedure

1. *Open the Integration Mapping Editor and select the integration definition file:*

    a. In Active Workspace, switch to the Active Admin workspace.

    b. Click the **INTEGRATION MAPPING EDITOR** tile.

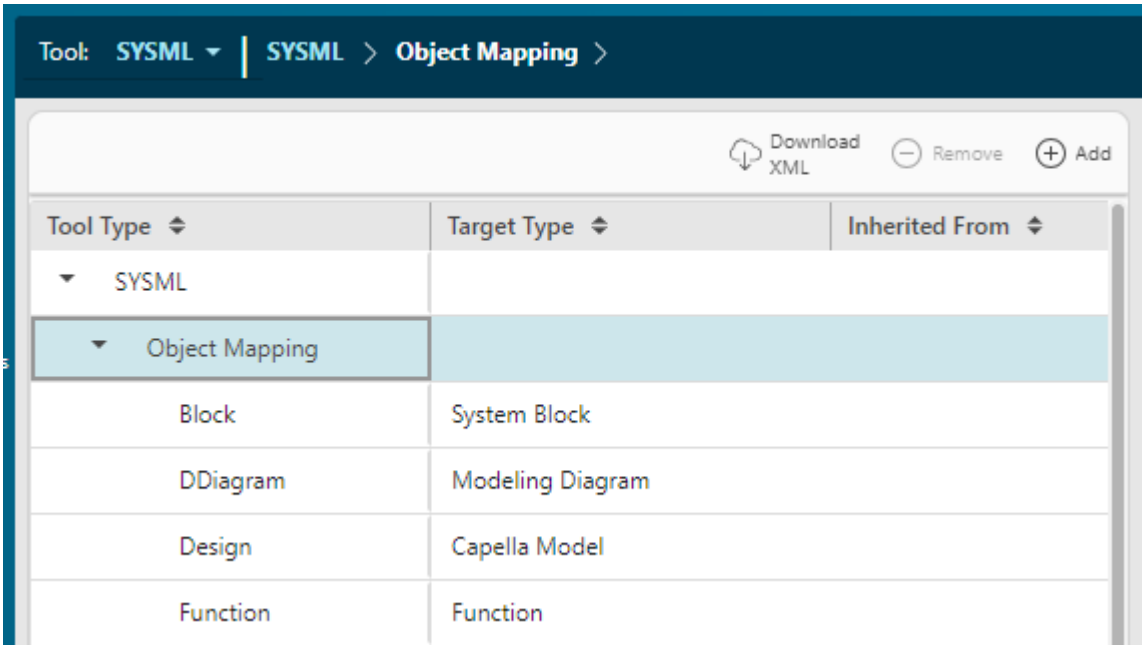
    c. From the **Tool** list, select the integration definition file.



Back to top

2. *Object Type Mapping — Create mapping between authoring tool object types and Teamcenter object types*

   a. In **Integration Mapping Editor**, select **Object Mapping** and click **Add** ⊕.



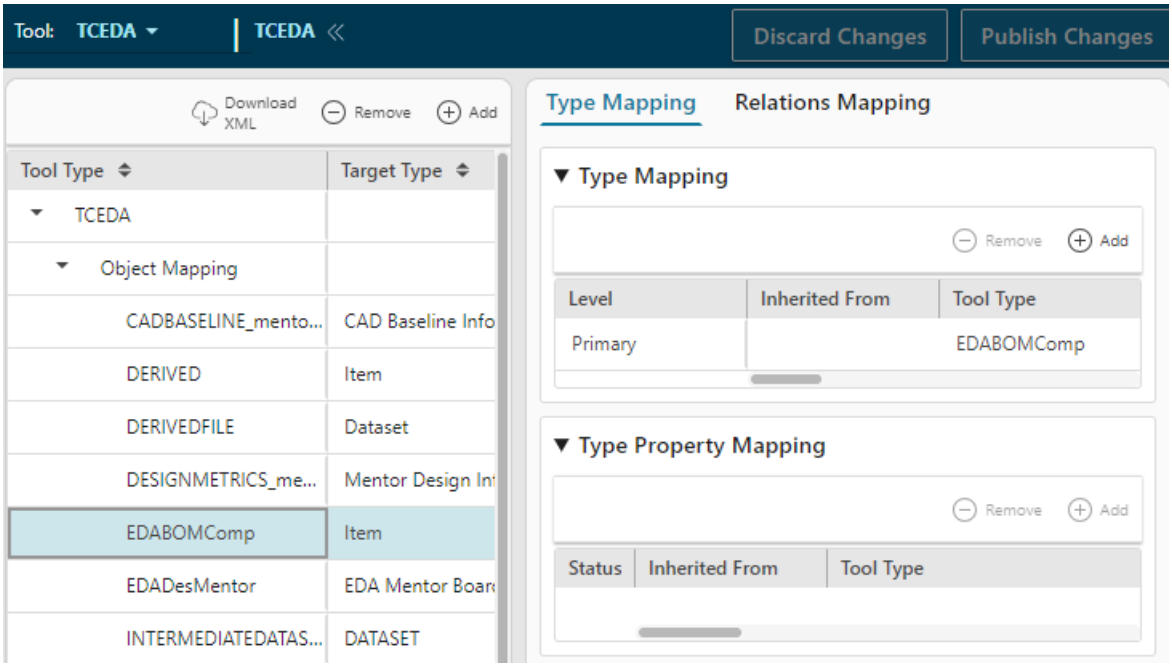
b. In the **Add Object Mapping** panel specify:

Tool Type	Type the object type from the authoring tool.
Inherited From	(Optional) If you want the object mapping to inherit properties from a parent, select a parent from the list.
Target Type	Choose the Teamcenter object type.

Back to top

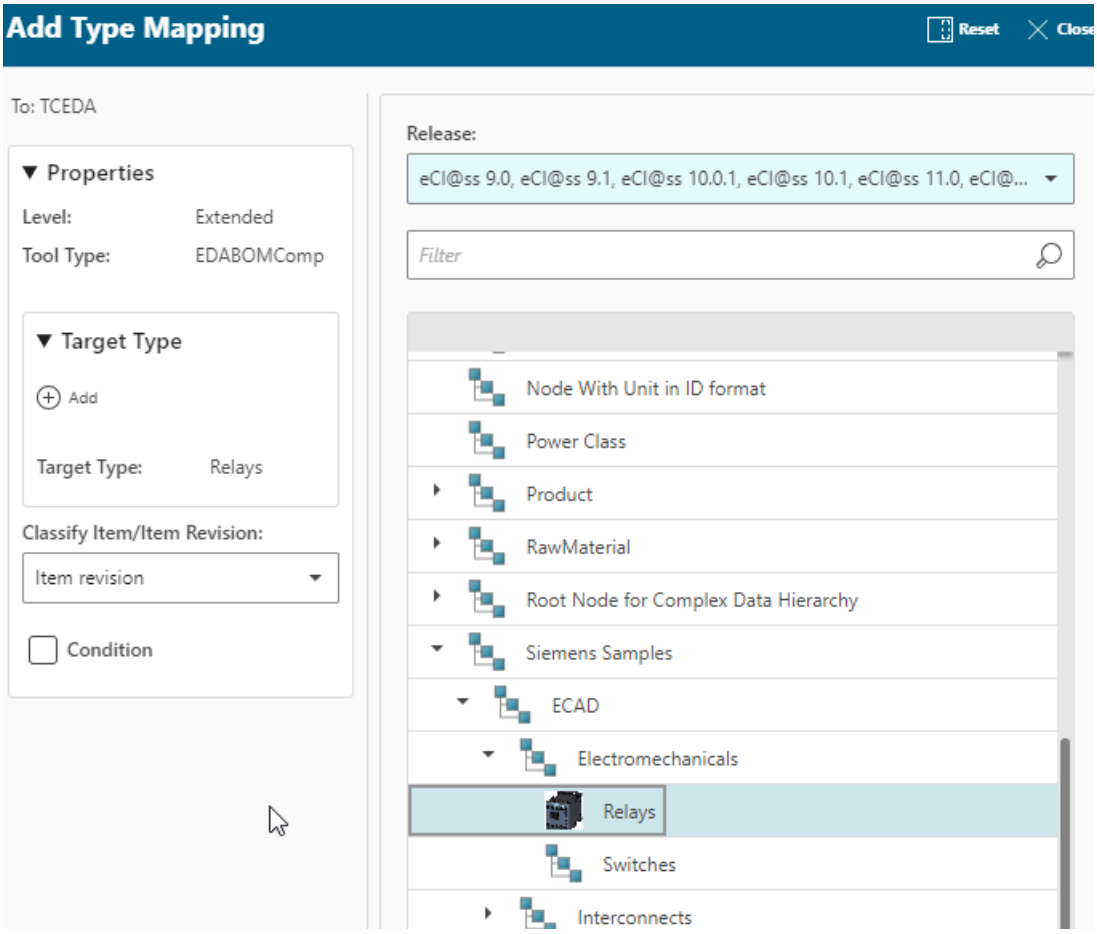
3. *Extended Type Mapping — Create mapping of classification artifacts*

a. Select an object type from the **Tool Type** column and from the **Type Mapping** section, click **Add** ⊕.



b. In the **Add Type Mapping** panel, from the **Target Type** section, click **Add** ⊕.

c. In the **Add Type Mapping** panel, from the Classification tree, select a classification type node.

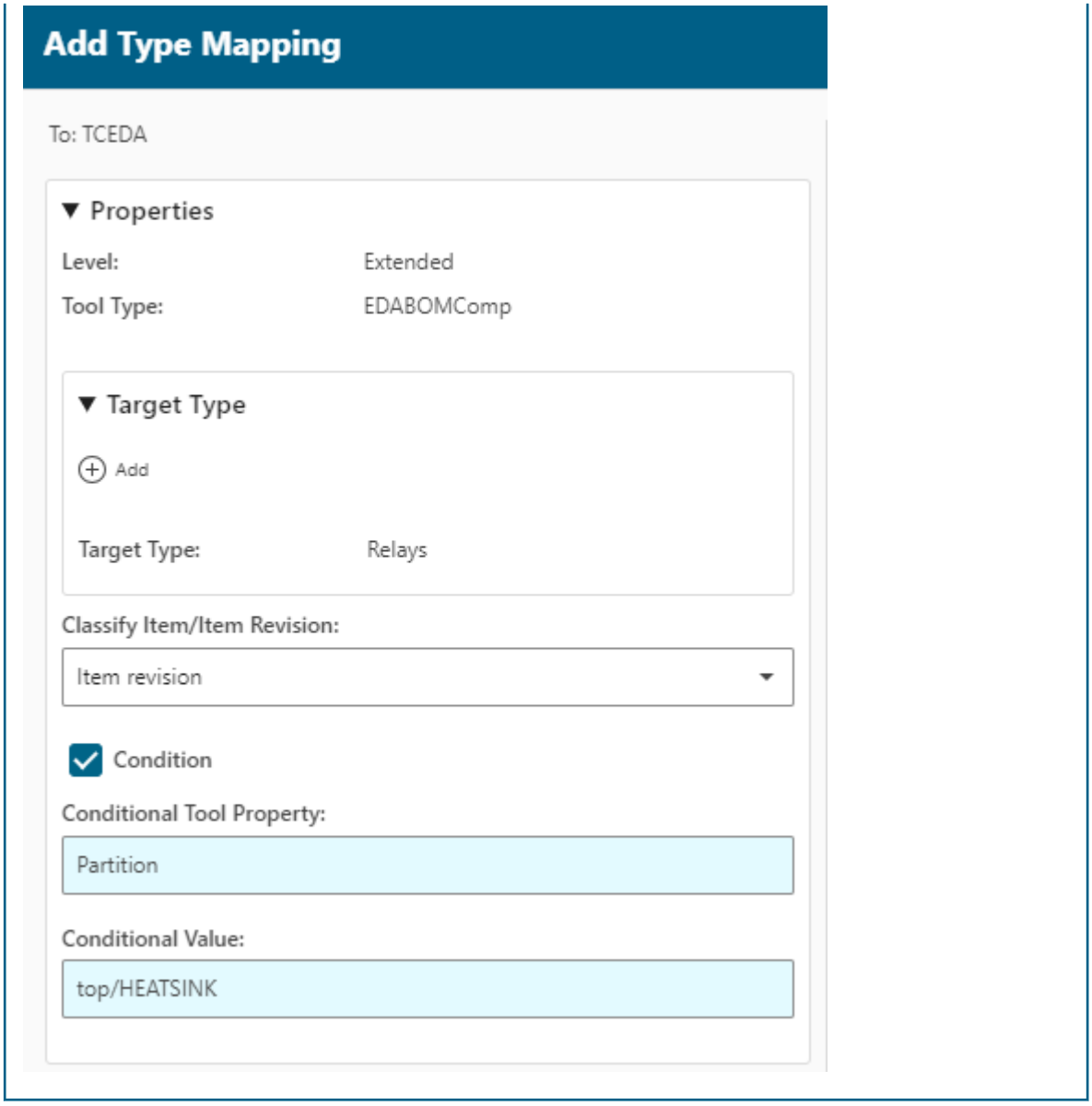


d. From the **Classify Item/Item Revision** list, choose if you want to classify the item or item revision for the object that you selected in the previous step.

e. If you want to classify the object only when certain conditions based on property and value pairs are true, select the **Condition** check box.

Specify the property and property value in the **Conditional Tool Property** and **Conditional Value** boxes, respectively.

> Example:
>
> In this example, the tool type **EDABOMComp** maps to **Relays** only when the tool property is **Partition** and its value is **top/HEATSINK**.



Back to top

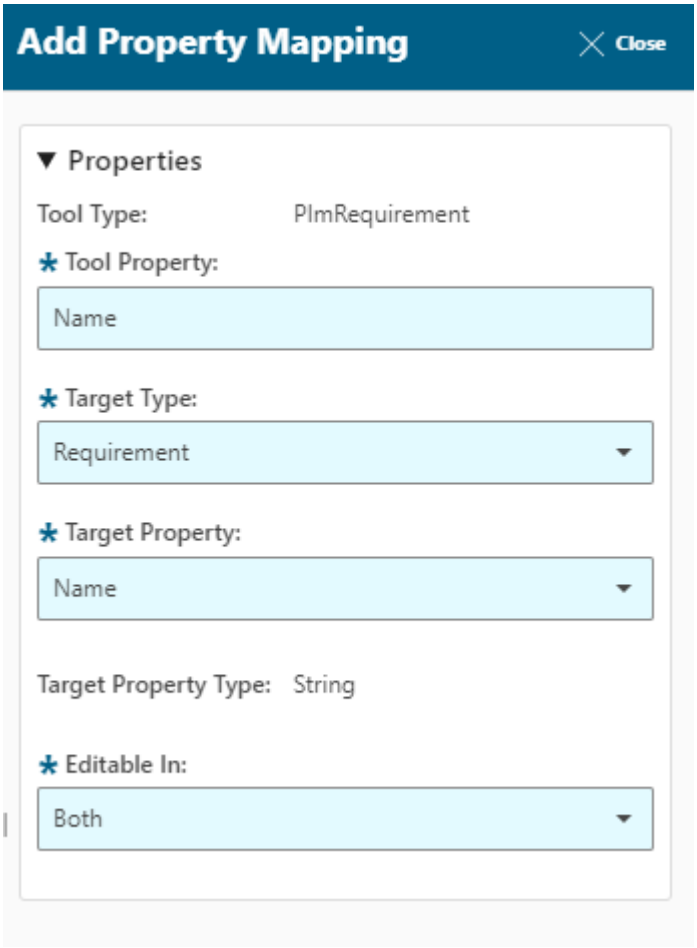
4. *Type Property Mapping — Create mapping between properties of the authoring tool and Teamcenter*

   a. From the **Type Mapping** section, choose the object.

   b. In the **Type Property Mapping** section, click **Add** ⊕.

   c. In the **Add Property Mapping** panel, add the following information:

Tool Property	This is the property in the authoring tool.
Target Type	This is the object type in Teamcenter.
Target Property	This is the Teamcenter property that you want to map with the property in the authoring tool.
Editable In	This list specifies where data can be edited. The following options are available: • **Both** Data can be edited in both the authoring tool as well as Teamcenter. • **Teamcenter** Data can be edited only in Teamcenter. • **Authoring Tool** Data can be edited only in the authoring tool.

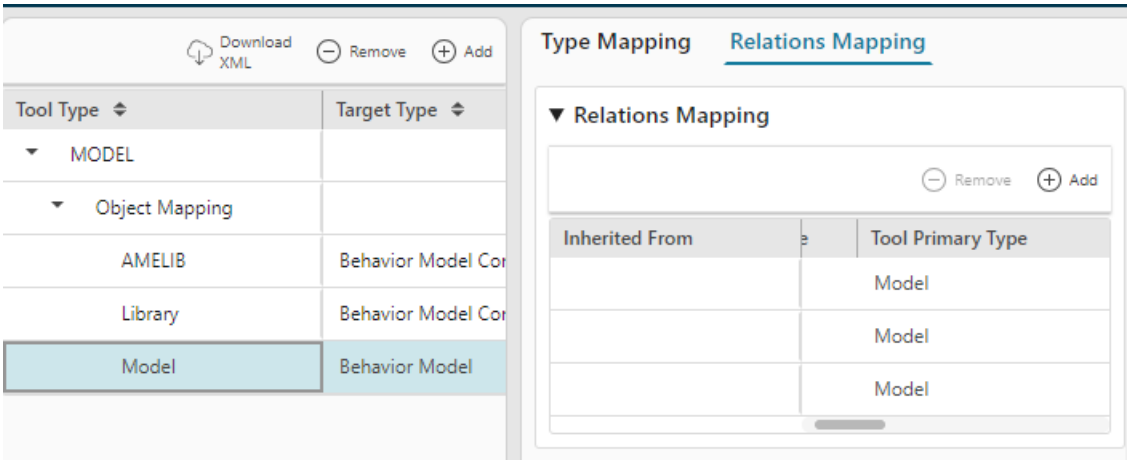


Back to top

5. *Relations Mapping — Create mapping between relations in the authoring tool and relations in Teamcenter*

   a. Select an object type from the **Tool Type** column and click the **Relations Mapping** tab.

   b. From the **Relations Mapping** section, click **Add** ⊕.



c. In the **Add Relation Mapping** panel, select the type of relation you want to use from the **Target Relation Type** list.

The following relation types are available:

- **Composition**
- **GDE Composition**
- **Relation**

  This is for mapping GRM relations.

- **Reference**

  This is used for reference type property mapping.

- **Related Reference**
- **Full Text**

  This is for requirements from Polarion.

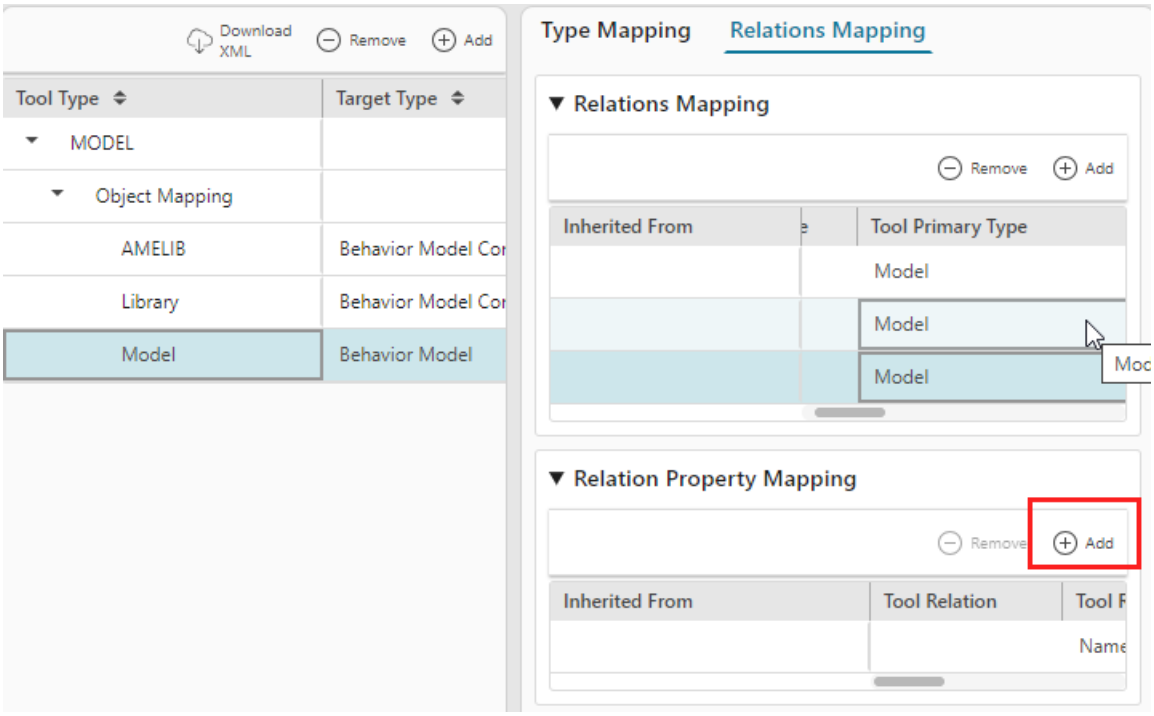
Depending on the relation type you choose, specify additional information.

Back to top

6. *Relations Type Property Mapping — Create mapping between properties of the relation in the authoring tool and Teamcenter relation*

a. Select an object from the **Tool Type** column and click the **Relations Mapping** tab.

b. From the **Relations Mapping** section, select an entry, and in the **Relation Property Mapping** section, click **Add** ⊕.



c. In the **Add Relation Property Mapping** panel, specify how you want to map the properties of the relation.

Tool Relation Property	Specify the property of the relation in the authoring tool.
Target Property	Specify the property of the relation in Teamcenter.
Editable In	This list specifies where data can be edited. The following options are available: • **Both** Data can be edited in both tools. • **Teamcenter** Data can be edited only in Teamcenter. • **Authoring Tool** Data can be edited only in the authoring tool.

d. Click **Add**.

Back to top

7. *Publish*

a. Click **Publish Changes**.

The information that you added is saved to Teamcenter.

Back to top

5. Understanding the different integration modes

Models authored in the modeling tool are saved in Teamcenter as Teamcenter business objects. You must configure each modeling tool to import model data and integrate it with Teamcenter. There are two modes of integration:

Specific connector-based integration

In this integration mode, a tool-specific connector is used to import model data into Teamcenter. This connector reads the integration definition file to decide the type of data to be imported into Teamcenter. The following data is saved to Teamcenter based on the configuration.

- A model file corresponding to the model associated to the model item revision with the **IMAN_specification** relation.
- A snapshot of the model is saved as an image and associated to the model item revision as **TC_Attaches**.
- All the files that the model is dependent on are imported as multiple datasets and each dataset corresponds to one file. Each dataset is associated to the model revision with the **Bhm0AssociatedData** relation.
- If a configuration is provided in the integration definition file for the modeling tool to import all the files from specified folders, all these files are imported as individual datasets and associated to the model item revision based on the configured relation.
- If a tool integration is configured to capture the model hierarchy and its components, all the configured components are saved in Teamcenter as separate business objects. These are associated to the model revision object either through the BOM View Revision or through the relation **Bhm0HasComponents**.
- If there are any model properties mapped to Teamcenter business object attributes in the integration definition file, the values of these model properties are stored in the respective attributes in the model object in Teamcenter.

Common connector-based integration

This integration mode provides limited integration of the modeling tool with Teamcenter. In this mode, all the model files in the operating system from a given context folder are identified based on a configuration, and these model files are saved to Teamcenter as a model object. Only the model files are associated to these objects through the **IMAN_specification** relation. No other features of the connector-based integration mode are available.

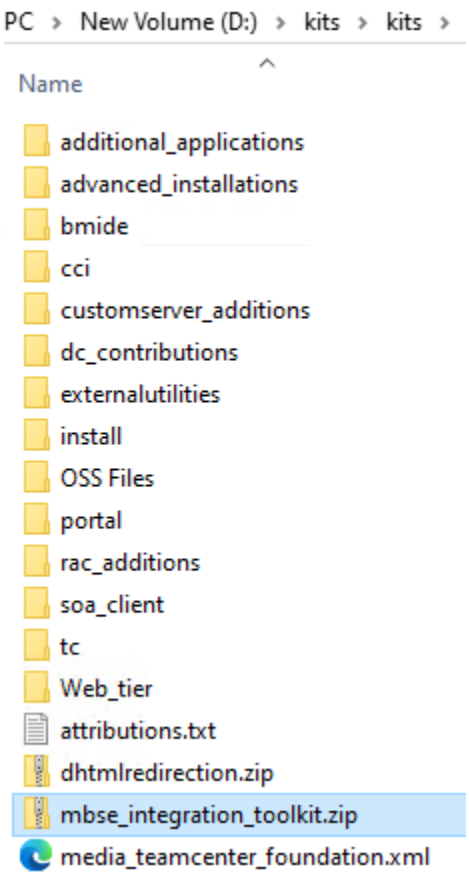
6. Using the MBSE Integration Gateway Toolkit

Overview of MBSE Integration Gateway Toolkit

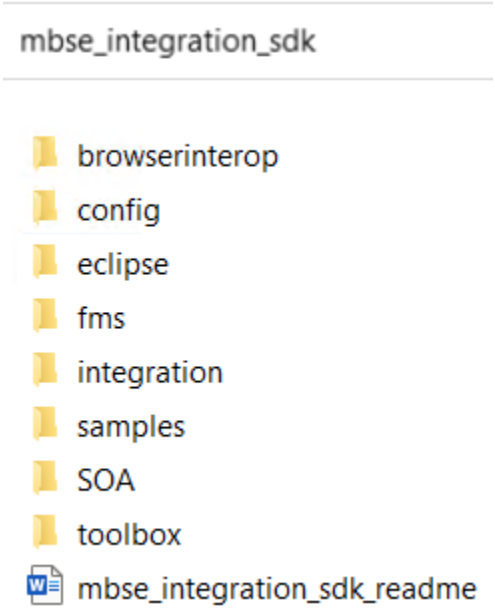
The MBSE Integration Gateway Toolkit allows you to:

- Connect to Teamcenter without installing an MBSE Integration Gateway client.
- Access all the required artifacts, that is, integration JAR files, Teamcenter SOA JAR files, and third-party JAR files that are required for integrating a tool with Teamcenter.
- Access the sample modeling tool for trying out APIs such as import, export, and compare and for generating a sample JSON file.

This Toolkit is available in the Teamcenter kit in a ZIP file called *mbse_integration_toolkit*.



Extract the contents of the ZIP file to a folder of your choice. It contains the following data:



mbse_integration_toolkit	The root folder name
browserinterop	Contains the JAR files and libraries needed for hosting the Teamcenter Active Workspace in Chromium browser.
config	Contains **mbseconfigurations.jar**. This is an independent JAR file and the APIs in this file can be called during the installation of the modeling tool to configure the MBSE Integration Gateway-related properties. Alternatively, the modeling tool can provide a user interface to set and retrieve the MBSE Integration Gateway properties using the APIs.
Eclipse	Contains the Eclipse related supporting JAR files.
fms	Contains the file management system-related JAR files needed to upload and download files from Teamcenter.
integration	Contains the MBSE Integration Gateway JAR files that provide the Common Integration Services (CIS) APIs. These APIs enable export, import, and update of tool-specific data to Teamcenter.
samples	Contains the source code of the sample mock modeling tool that depicts the use of the configuration APIs and CIS APIs for performing tool integrations. You can use this program to test tool integrations and to generate the test input JSON file.
SOA	Contains the Teamcenter SOA JAR files.
toolbox	Contains the supporting third-party JAR files.

Using the sample modeling tool in the MBSE Integration Gateway Toolkit

Overview of using the sample modeling tool

The sample modeling tool gives you a sample UI that allows you to:

- Generate and save a JSON file. The JSON file organizes the data payload.
- Save data or import data to Teamcenter.
- View the imported data in Active Workspace.
- Export the data from Teamcenter.
- Update the data.
- Remove the Teamcenter data from the JSON file.
- Generate code.

You can use the sample modeling tool to practice integration operations.

Setting up the sample modeling tool in Eclipse

Setting up the sample modeling tool in Eclipse consists of:

- Importing Toolkit sample files into Eclipse.
- Creating and running a configuration.

Before setting up the sample modeling tool in Eclipse:

- Extract the MBSE Integration Gateway Toolkit files in a location that is accessible.
- Ensure that you have an integration definition file available for use.

To set up the sample modeling tool in Eclipse:

1. Start Eclipse and click **File** > **Import**.
2. In the **Import** dialog box, expand **General** and select **Existing Projects into Workspace**.

   Click **Next**.

3. In the **Import Projects** dialog box, select the **Select root directory** option and browse to and select the folder named *samples\migjsongenerator* in the Toolkit.

   Click **Finish**.

4. To resolve the errors that result from this operation, set the SDK_HOME classpath variable as follows:

   a. Click **Window** > **Preferences**.

   b. In the **Preferences** dialog box, expand **Java** > **Build Path** and select **Classpath Variables**.

   c. Click **New** and in the **New Variable Entry** dialog box, type **SDK_HOME** in the **Name** box, and set the **Path** to the location where the Toolkit is extracted.

   > Example:
   >
   > *C:\apps\kits\aw\wntx64\mbse_integration_toolkit*

   d. Click **OK**.

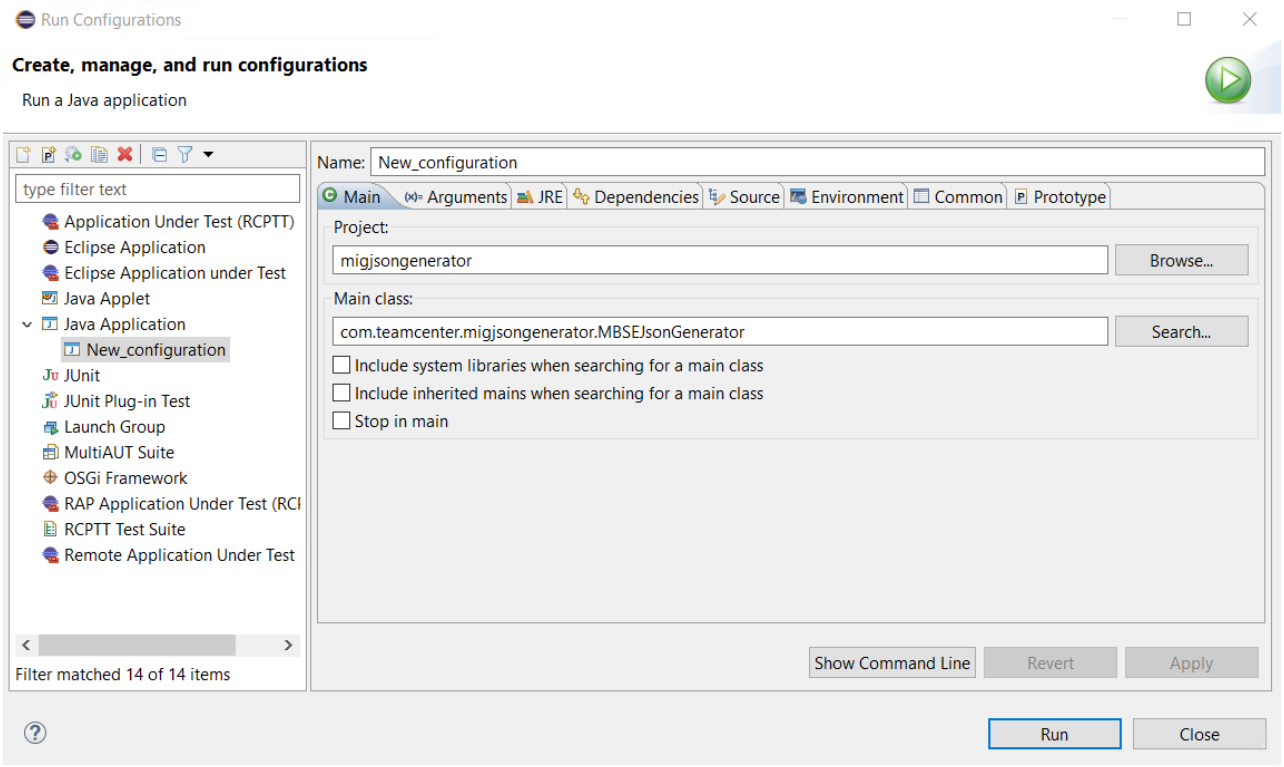
   In the **Classpath Variables** dialog box, click **Apply and Close**.

   In the **Classpath Variables Changed** dialog box, click **Yes**.

5. Click **Run** > **Run Configurations**.

6. In the **Run Configurations** dialog box, select **Java Application** and right-click and select **New Configuration** to create a Java application.

7. In the **Create, manage, and run configurations** dialog box, select the **Main** tab, go to the **Project** section, and browse to and select the **migjsongenerator** folder.



8. In the **Main class** section, search for and select **com.teamcenter.migjsongenerator.MBSEJsonGenerator**.

9. Go to the **Arguments** tab, and in the **Program arguments** box, type the path to the integration definition file of the tool.

   Example:

   C:\apps\tc\idf\POLARION_SWHW_BHMIntegrationDefinition.xml

10. Go to the **Environment** tab and click **New** to add an environment variable.

11. In the **New Environment Variable** dialog box, type **PATH** in the **Name** box and in the **Value** box, type the location of the browser interop library (*Location-of-MBSE-Integration-Gateway-Toolkit\mbse_integration_toolkit\browserinterop\org.jcef\libs*).

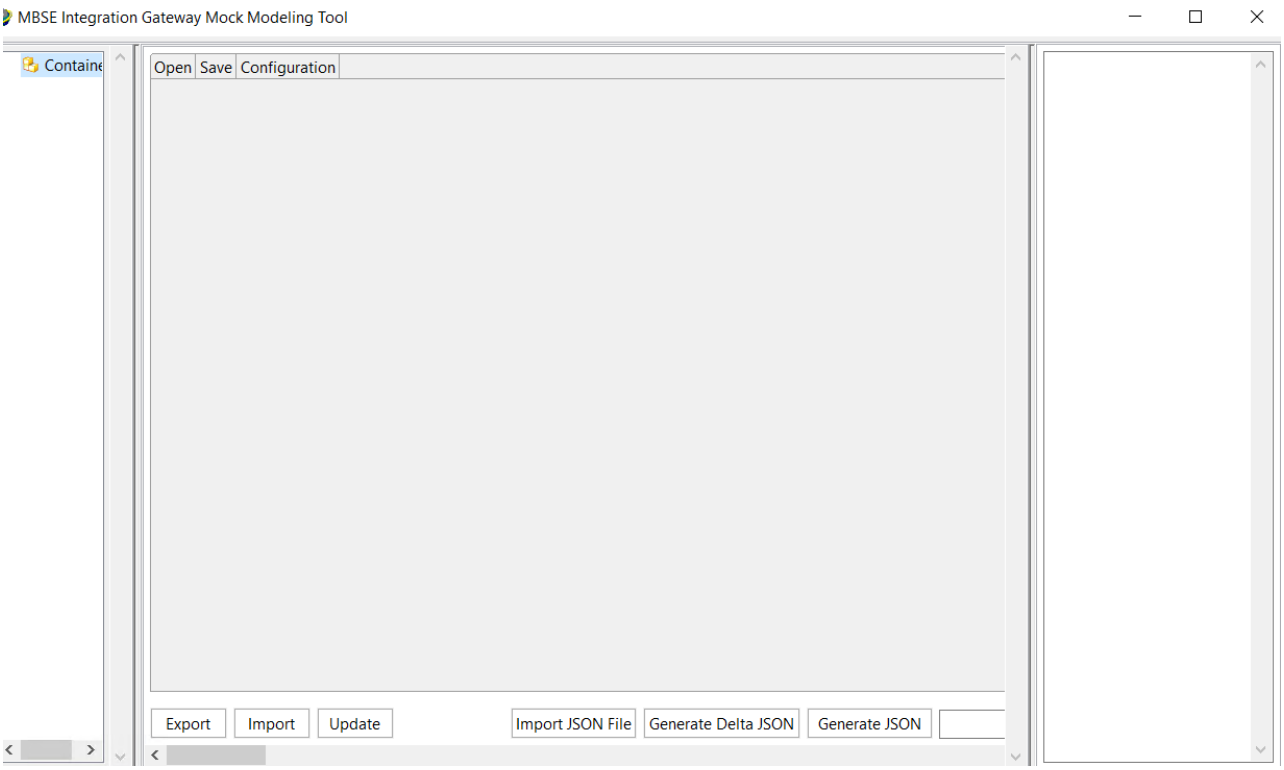
    Example:

    C:\apps\kits\aw\wntx64\mbse_integration_toolkit\browserinterop\org.jcef\libs

    This is needed for Chromium support.

    Click **OK**.

12. Click **Run** to launch the sample modeling tool.



Configuring MBSE Integration Gateway using the sample modeling tool

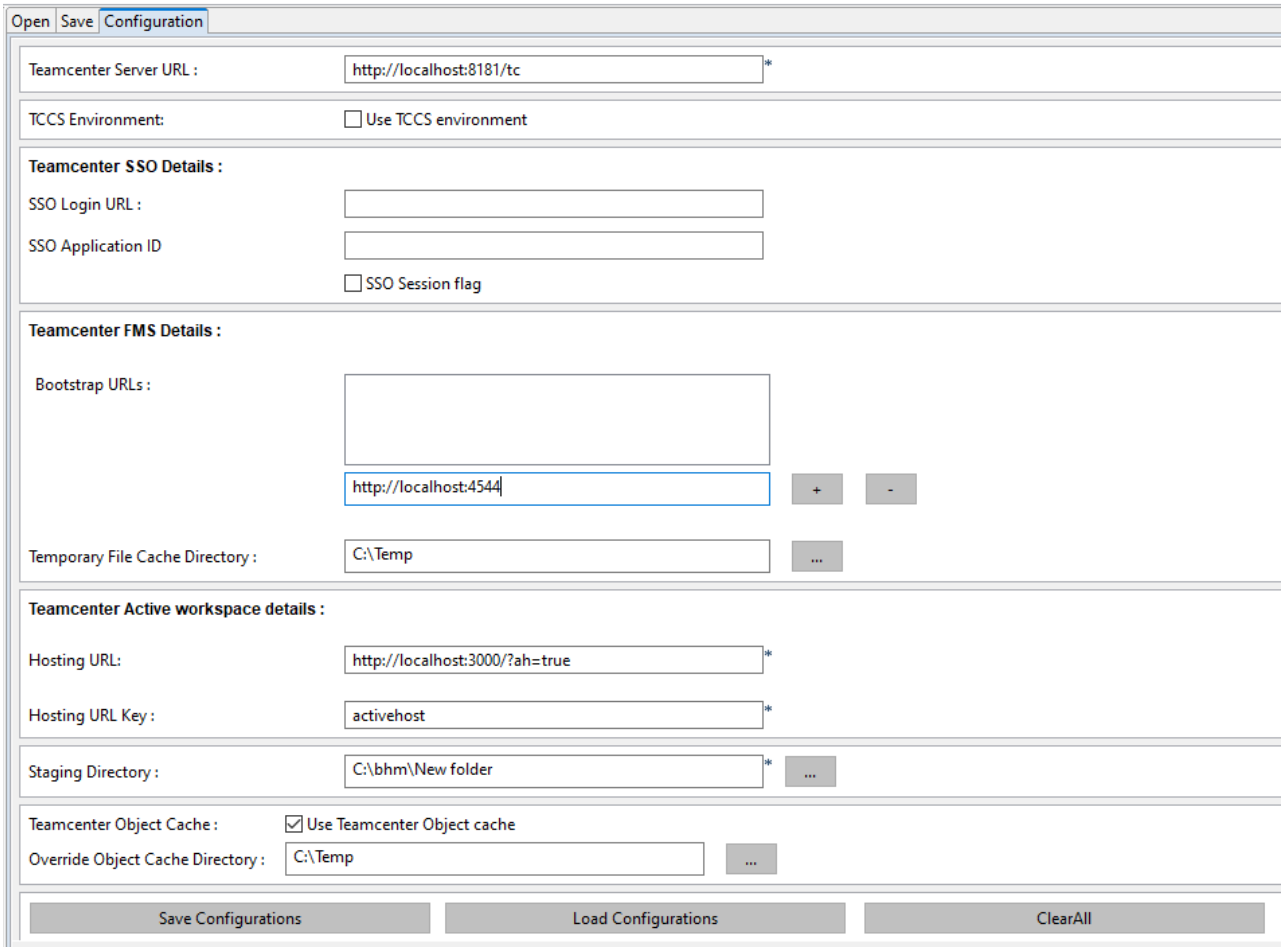
To configure MBSE Integration Gateway:

1. Start the sample modeling tool by **running the configuration you created**.

2. In the sample modeling tool, click the **Configuration** tab and add the configuration properties.

   The following properties are mandatory:

   - **Teamcenter Server URL**
   - **Bootstrap URLs**
   - **Temporary File Cache Directory**
   - **Hosting URL**
   - **Hosting URL Key**
   - **Staging Directory**



3. Click **Save Configurations**.

The configuration files are created in **%ProgramData%\Siemens\MBSE\bhm**. The following files are created:

- *CommonClient.properties*: Contains configuration information.
- *BHMClient.properties*: Contains information about the staging directory. The staging directory is specific to each tool.

Loading the saved configurations in the sample modeling tool

When you start the sample modeling tool, the configurations are not loaded. To load the configurations:

1. Start the sample modeling tool by **running the configuration you created**.
2. Go to the **Configuration** tab and click **Load Configurations**.

Generate data and a JSON file and save the data to Teamcenter

Using the sample modeling tool, you can generate data in JSON format and save the data to Teamcenter.

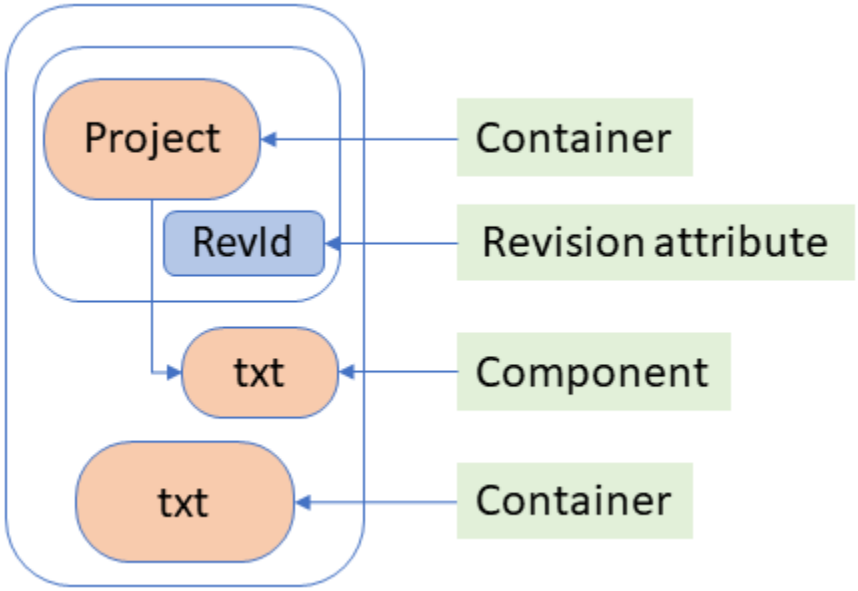
To exchange data with Teamcenter using MBSE Integration Gateway, the data must be in a JSON format and must be structured as containers and components.

Containers represent Teamcenter objects while components are the associations or relations between the containers.

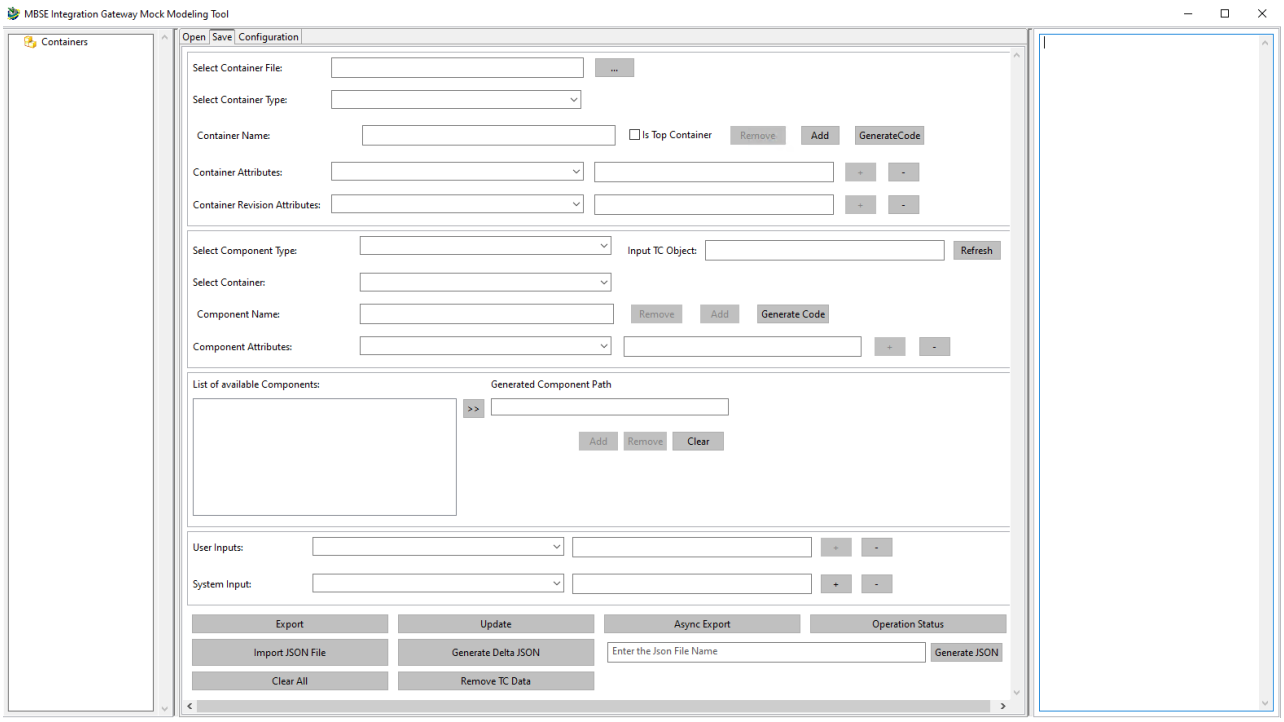
The instructions in this topic walk you through creating the data and the JSON file and saving the data to Teamcenter. To follow along with the example, ensure that:

- You can use the *MIG_Integration_Definition_File*.
- Specify this integration definition file in the **arguments tab of your project in Eclipse**.

This example walks you through creating two containers, container attributes, components, generating the JSON, and saving the data to Teamcenter.



1. Run the sample modeling tool and load the Teamcenter configurations.
2. Click the **Save** tab.



The interface consists of the following sections:

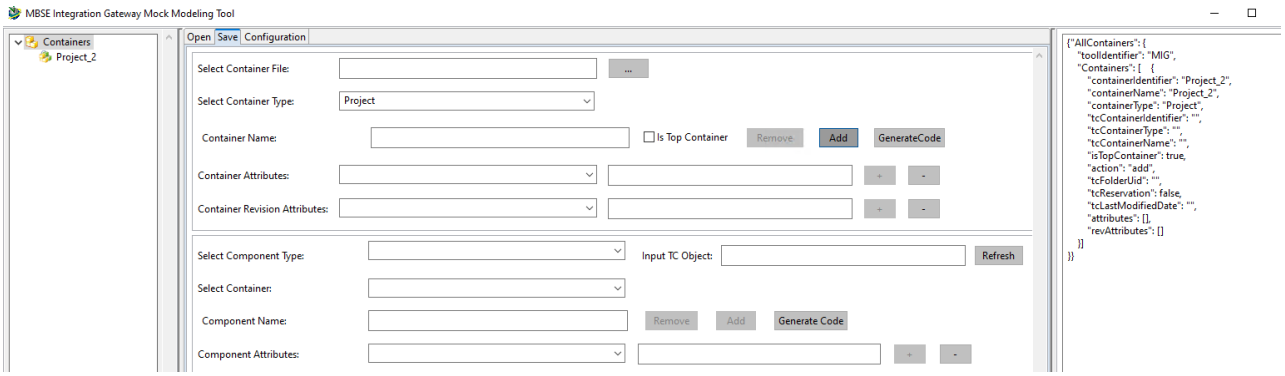
- The section on the left displays the data structure.
- The middle pane is the authoring section.
- The section on the right displays the contents of the JSON file.

3. To add a container:

   a. From the **Select Container Type** list, select a container object, for example, **Project**.

   b. If this is the top-level object, select the **Is Top Container** check box.

   c. Click **Add**.



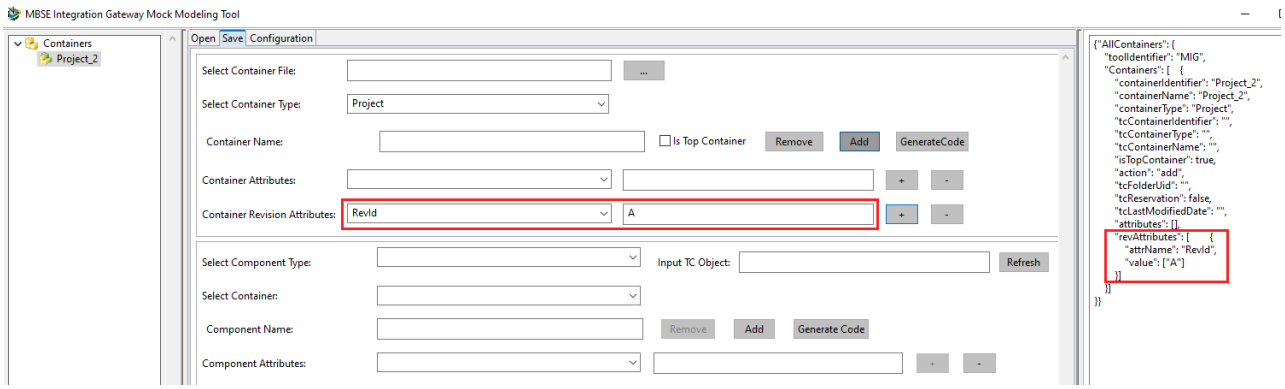
- The project is added to the section on the left while the section on the right has the JSON information.
- As this is the first time you have created the container, the value of the **action** attribute in the JSON code is **add**.
- The Teamcenter attribute information is not available in the JSON code. It is made available after you save this data to Teamcenter and then import it back into the sample modeling tool.

4. To add revision attributes to the Project container:

   a. From the section on the left, select the project.

   b. From the **Container Revision Attributes** list, select an attribute, and type the value of the attribute in the box next to it.

   c. Click the + add button.



The revision attribute is added to the JSON code.

5. To add another container containing a dataset:

   a. In the section on the left, select **Containers**.

b. From the **Select Container Type** list, select a container type. For example, **txt**.

c. From the **Select Container File** box, browse to and select the file.

d. Click **Add**.

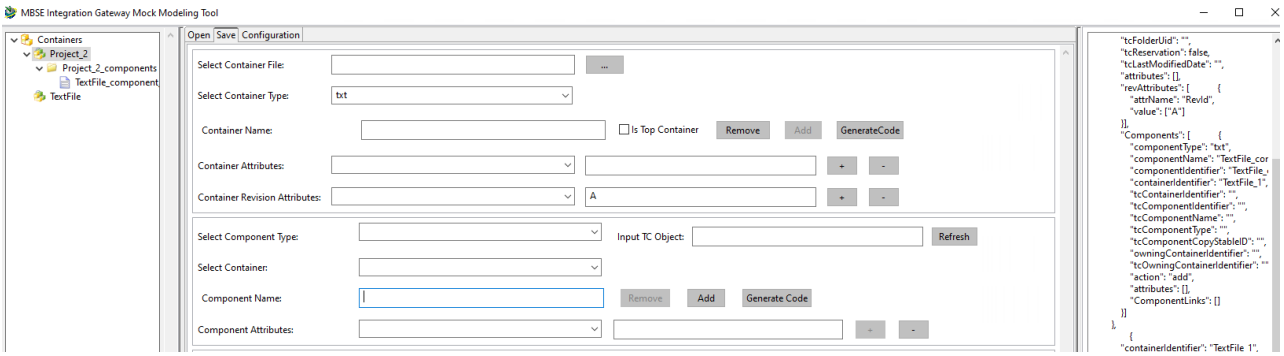
6. To add a component, that is, to create associations between containers:

a. From the section on the left, select a container as the parent, for example, **Project**.

b. From the **Select Component Type** list, select the component type that will be associated with the container that you selected in the previous step, for example, **txt**.

c. From the **Select Container** list, select the container that you created previously.

d. Click **Add**.



7. To create the JSON file that will be used to save the data to Teamcenter:

a. Type the name of the JSON file in the box that is next to the **Generate JSON** button.

b. Click **Generate JSON**.

The JSON file is saved in the same folder where the integration definition file is saved.

8. To save the data to Teamcenter, click **Export**.

If this is the first time you are logging on to Teamcenter, you see the Teamcenter logon dialog box. Type your Teamcenter credentials to save the data.

After you save the data to Teamcenter:

- The Teamcenter attribute information is filled in.

- You can view the data in Active Workspace from the sample modeling tool and import that data into the sample modeling tool.

View the imported data in Active Workspace and download the data

After you save or import the data to Teamcenter, you can view the data in Active Workspace and import it back into the sample modeling tool.

1. Click the **Open** tab in the sample modeling tool.

2. In Active Workspace, go to the **Newstuff** folder to view the data that you imported.

3. Select the object and click **Import**.

   The file attachments are downloaded to the staging directory, while the JSON file appears in the righ-side pane of the sample modeling tool.

Generate the delta JSON and update the data in Teamcenter

After you have saved the data for the first time, to update new data in Teamcenter, you must first generate the delta of the data and then save or update the data to Teamcenter.

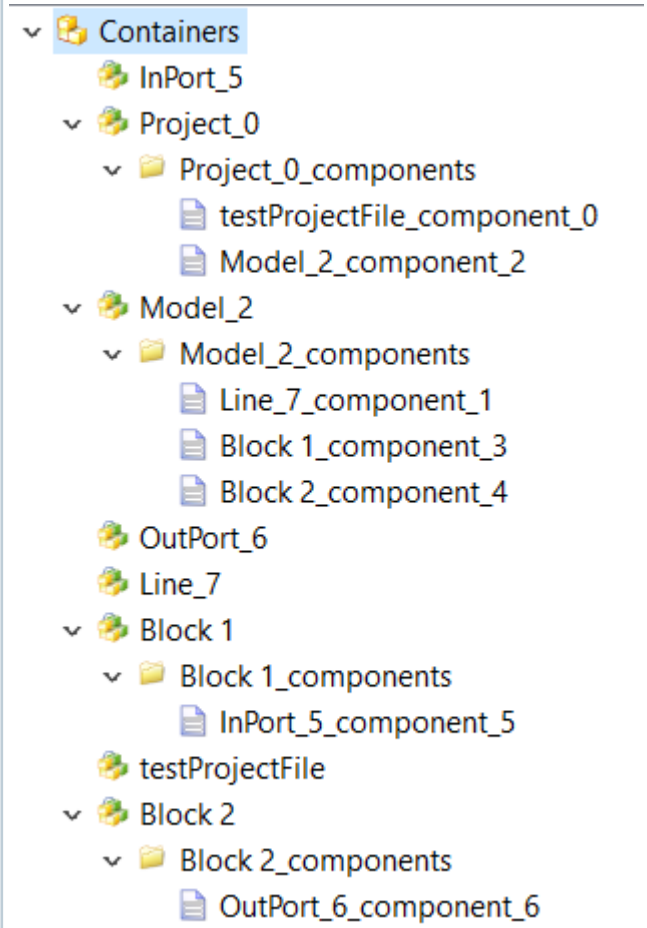
Create additional data

1. **Download data** from Teamcenter and **import the JSON file** into the sample modeling tool.

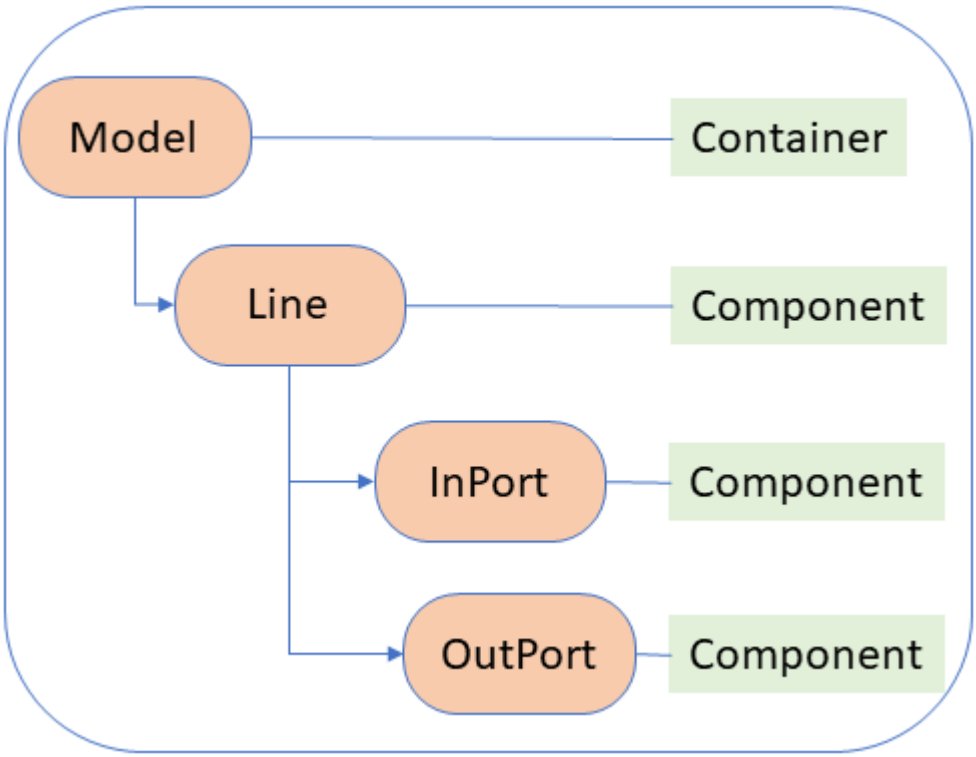
2. Create the following containers:

   - Model
   - Block 1
   - Block 2
   - Line
   - InPort
   - OutPort

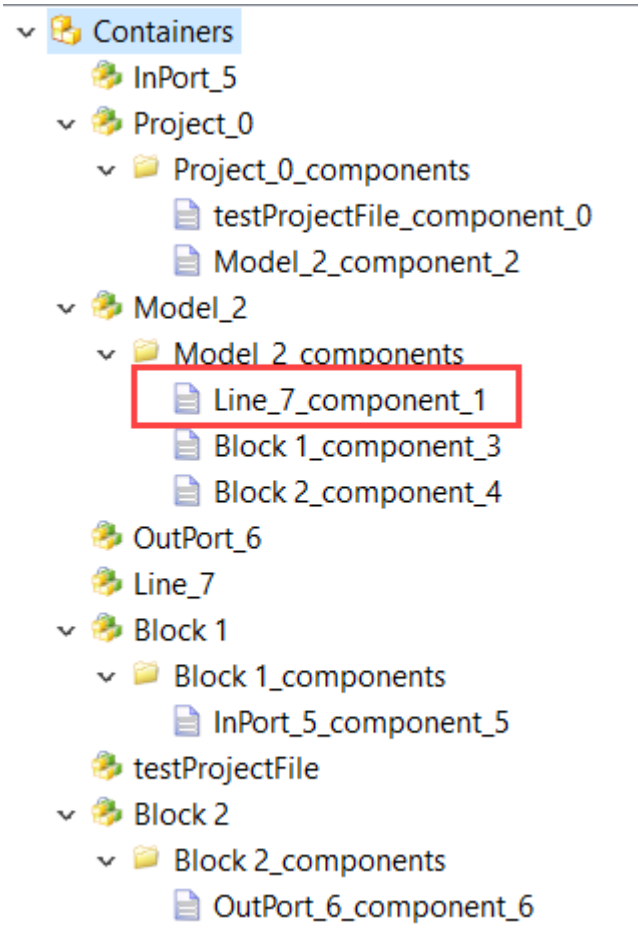
3. Create components or associations with the containers as follows:

• Model
  • Block 1
  • Block 2
  • Line
• Block 1
  • InPort
• Block 2
  • OutPort
	

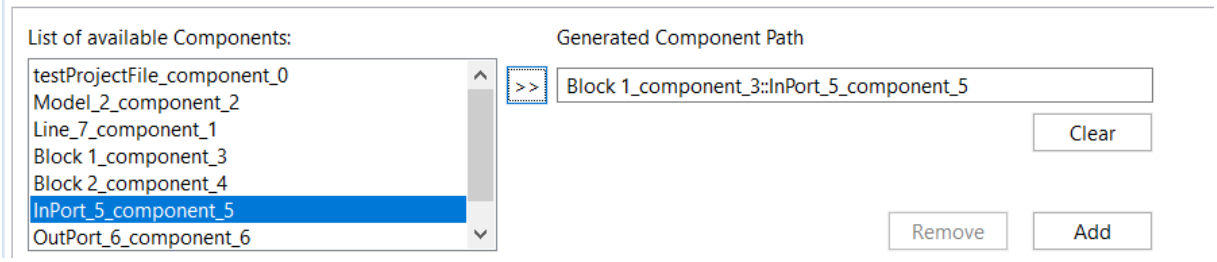
4. You can also create associations between components. For example, if you want to specify InPort and OutPort information for a line, do the following:



a. Select the **Line** component from the **Model** container.



b. From **List of available Components**, select **InPort** and click >>.



Click **Add**.

c. Similarly, select the **Line** component from the **Model** container and add **OutPort** as a component.

Generate a delta JSON and update the data in Teamcenter

1. Once you have made the changes, select the container you wish to update, for example, **Model**, and click **Generate Delta JSON**.

   Select the folder in the staging directory that has the last published version of the data in Teamcenter.

   The updated JSON file is displayed in the section on the right and the value of the **action** attribute is **add**.

2. Once you have generated the delta JSON file, click **Update.**

   Select the appropriate folder in the staging directory.

   This updates the data in Teamcenter.

Remove Teamcenter data from the JSON file

You may want to test data without Teamcenter information included in the data. To remove Teamcenter data:

- Click **Remove TC Data**.

Import the JSON file to the sample modeling tool

You can import a JSON file to check against any existing JSON files or to debug a JSON file.

1. Open the sample modeling tool and click **Import JSON File**.

2. From the staging directory, select the JSON file you need.

Export or publish data from the sample modeling tool to Teamcenter asynchronously

With asynchronous export of data to Teamcenter, the export operation happens in the background.

Prerequisites

Ensure that the following services are set up:

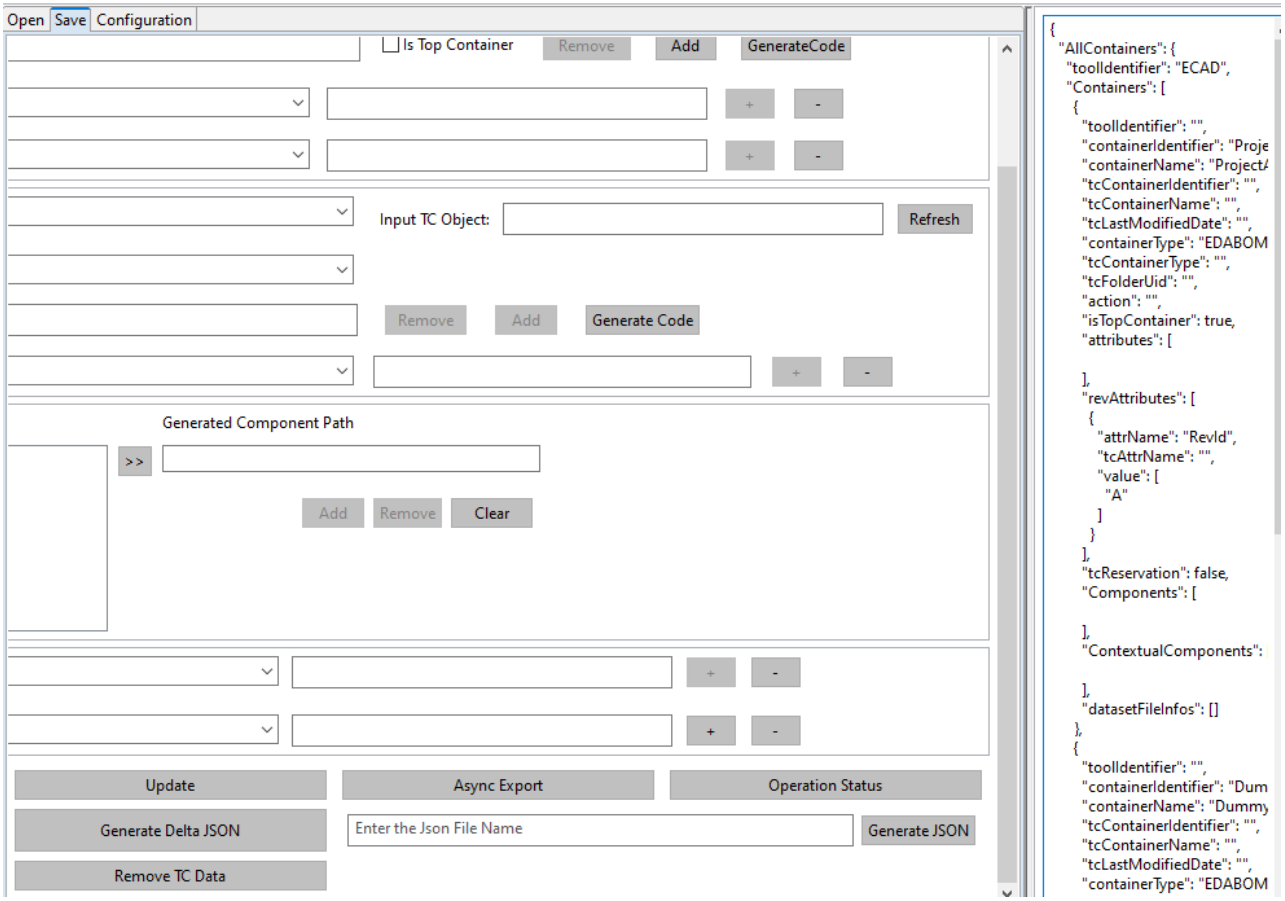
- Dispatcher, for performing the asynchronous export operation.
- Subscription Manager, to receive alerts in Active Workspace. If the Subscription Manager is not running, no notification is displayed in the **Alert** area, but the operation still occurs in the background.

Procedure

1. Run the sample modeling tool and load the Teamcenter configurations.

2. Click **Save** .

3. **Import a JSON file**, or paste the contents of the JSON file in the right-hand side pane.



4. Click **Async Export**.

   The export starts in the background. When the export is complete, you are notified in the **Alert** area in Active Workspace.

Results

You can check the status of the export in the sample modeling tool by clicking **Operation Status**.

Operation Status

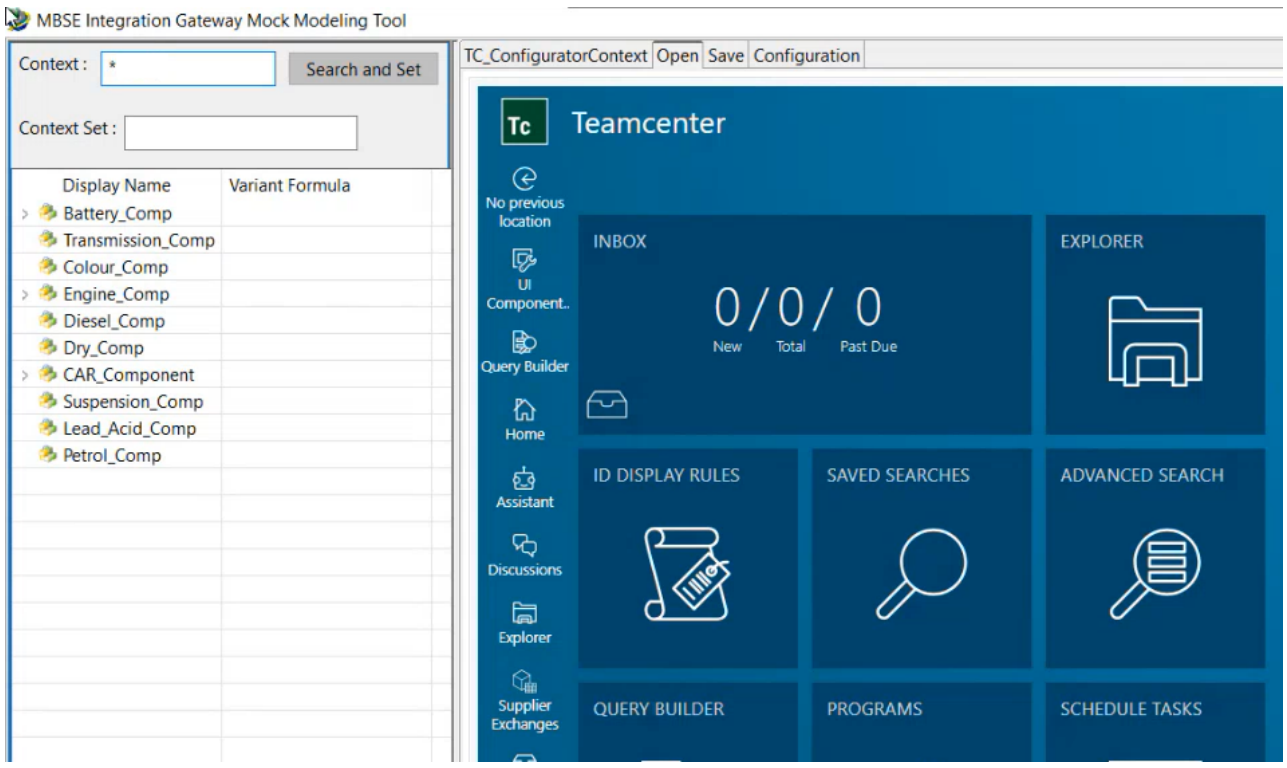
Container Name	TC Container Ide...	Operation Status	Error	Start Time	Operation ID
ProjectAsync2		Work in progress		2024-24-30-02-24-01	5c1688cb-21ef-4515-98c8-882ba865a6d3
ProjectAsync3		Work in progress		2024-31-30-02-31-56	bf4e2022-eabf-4129-a256-8649bfbadd23
ProjectAsync4		Work in progress		2024-30-30-02-30-16	afc02929-8ba5-4b5e-a9f2-2886b9073927
ProjectAsync31		Work in progress		2024-41-30-03-41-35	0f5be5c3-0f30-46c6-8245-76a8a01053bc
ProjectAsync33		Work in progress		2024-42-30-03-42-05	9d9784e9-4f42-42ba-bead-f2f42364873a
ProjectAsync34	AkqZvutR5qC5qA	Successful		2024-44-30-03-44-41	c165ee4f-dd67-400d-bdb4-b07a90a93ab1
ProjectAsync35		Work in progress		2024-50-30-03-50-11	581cf30b-b3a5-418d-ba97-98e9fbb62818

Use the sample modeling tool to work with product configurator

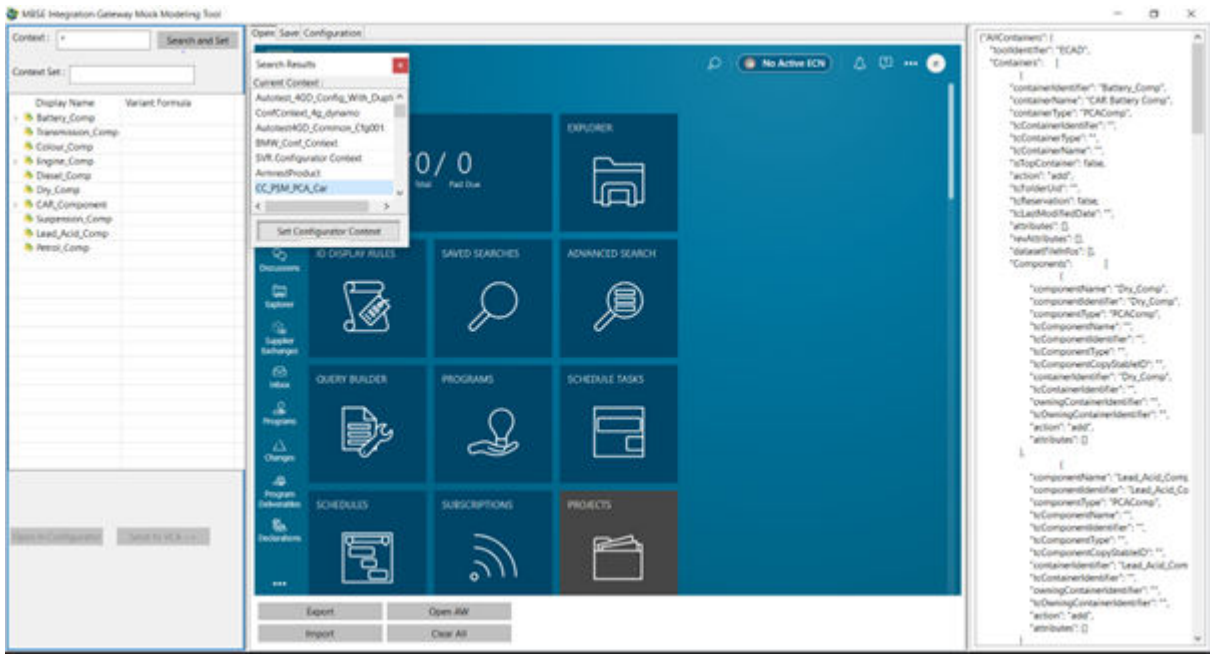
Using the sample modeling tool, you can work with Product Configurator data, and save the data to Teamcenter. You can also open the data from Teamcenter in the sample modeling tool.

Procedure

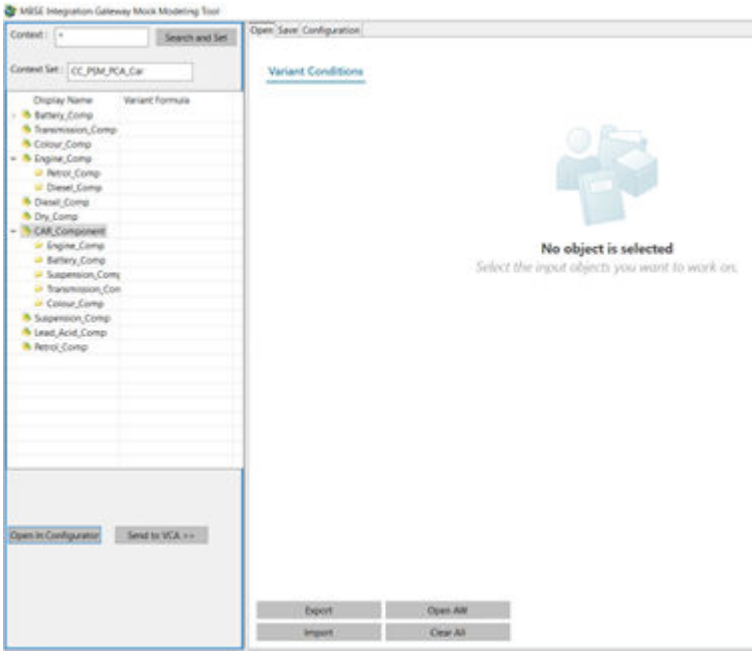
1. Run the sample modeling tool and load the Teamcenter configurations.
2. Click **Save**.
3. **Import a JSON file** or paste the contents of the JSON file in the right-hand side pane.
4. Click **Open**.
5. Search for a configurator context in the **Context** box, and click **Search and Set**.



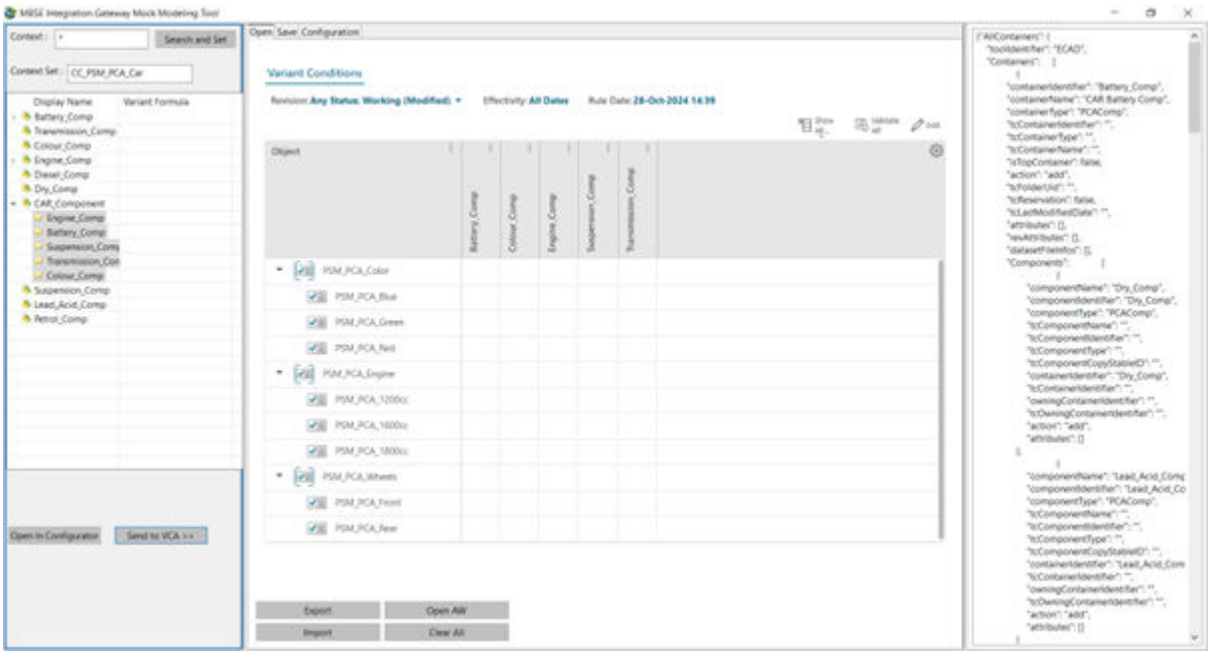
6. From the **Search Results** dialog box, select the configurator context and click **Set Configurator Context**.



7. To open the Product Configurator, click **Open in Configurator.**

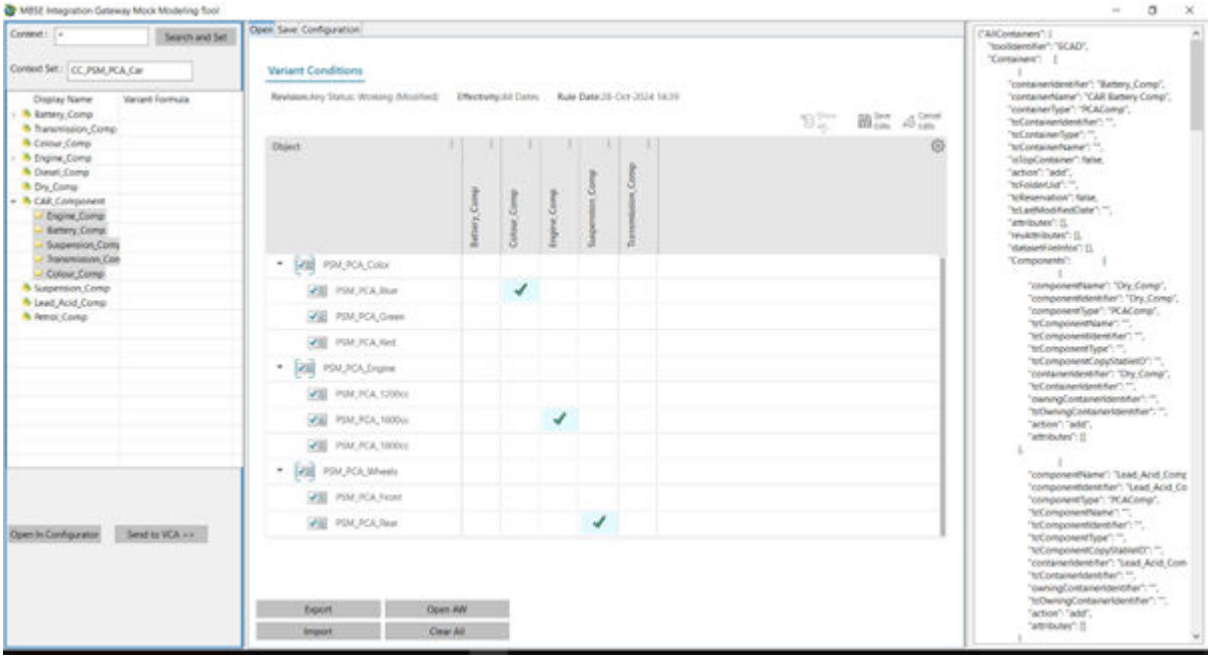


8. To work with the variant conditions view (VCA) in the sample modeling tool, select components and click **Send to VCA**.



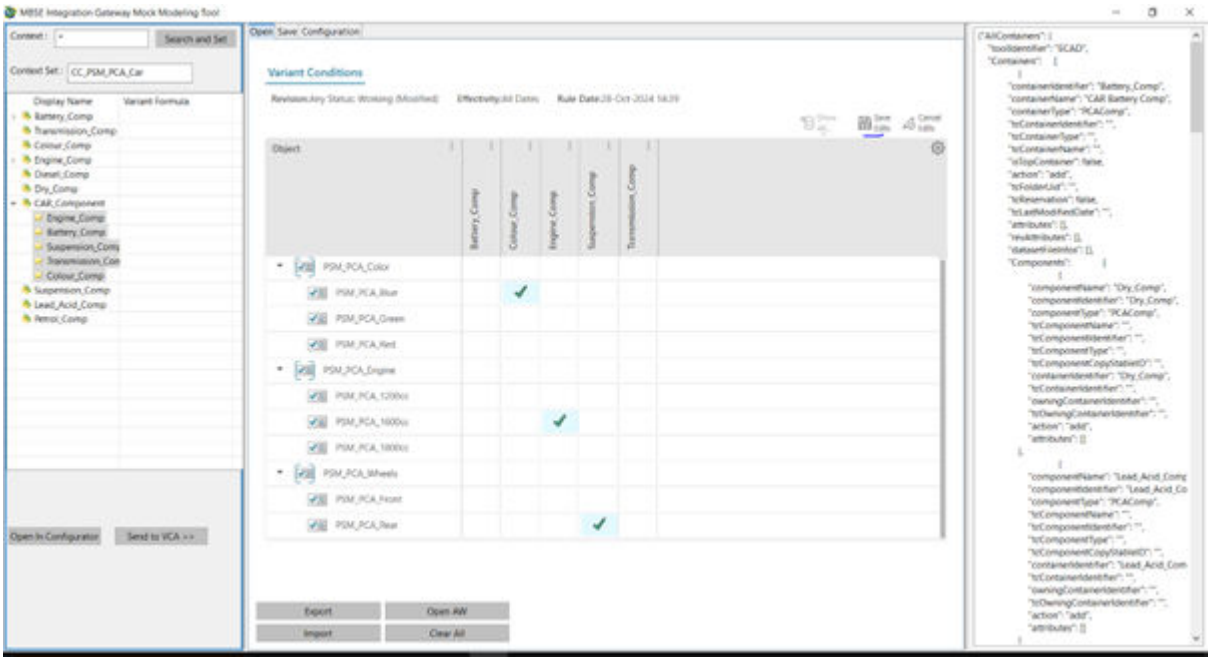
9. To edit variant conditions, in the **Variant Conditions** page, click **Edit**.

10. In the **Variant Conditions** page, select a few conditions.



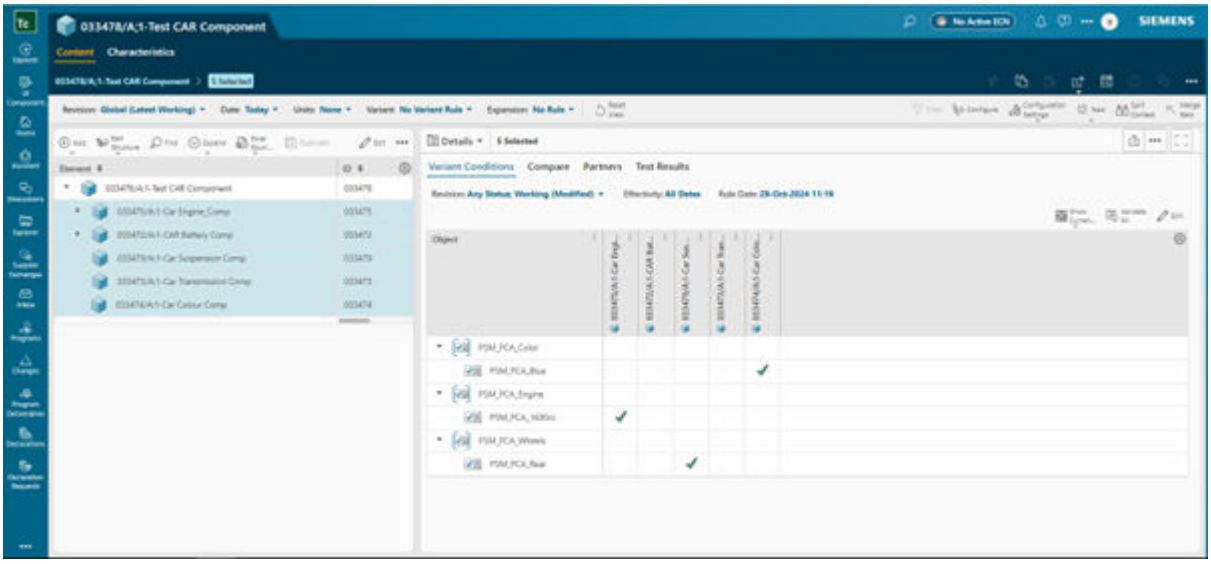
11. After you have finished editing the variant conditions, in the **Variant Conditions** page, click **Save Edits**.

    The variant conditions are now updated in the navigation section and in the JSON viewer.



12. To save the data to Teamcenter, click **Export**.

    Once the export is complete, you can view the data in Teamcenter.



Set up the MBSE Integration Gateway Toolkit in your modeling tool

1. Extract the *mbse_integration_Toolkit.zip* file to a directory of your choice.

2. If using within a modeling tool, copy the following JAR files files from the extracted location to the modeling tool.

Folder in Toolkit	Jar file name
integration	bhmcommon.jar
integration	bhmcommonclient.jar
integration	commonclient.jar
SOA	Cif0SoaIntegrationFrameworkStrong.jar
SOA	TcSoaAdministrationStrong.jar
SOA	TcSoaClient.jar
SOA	TcSoaCommon.jar
SOA	TcSoaCoreStrong.jar
SOA	TcSoaQueryStrong.jar
SOA	TcSoaStrongModel.jar
config	mbseconfigurations.jar
fms	fscclient.jar
fms	tcserverjavabinding.jar
toolbox	commons-beanutils-1.9.2.jar
toolbox	commons-collections-3.2.2.jar
toolbox	commons-httpclient-3.1.jar

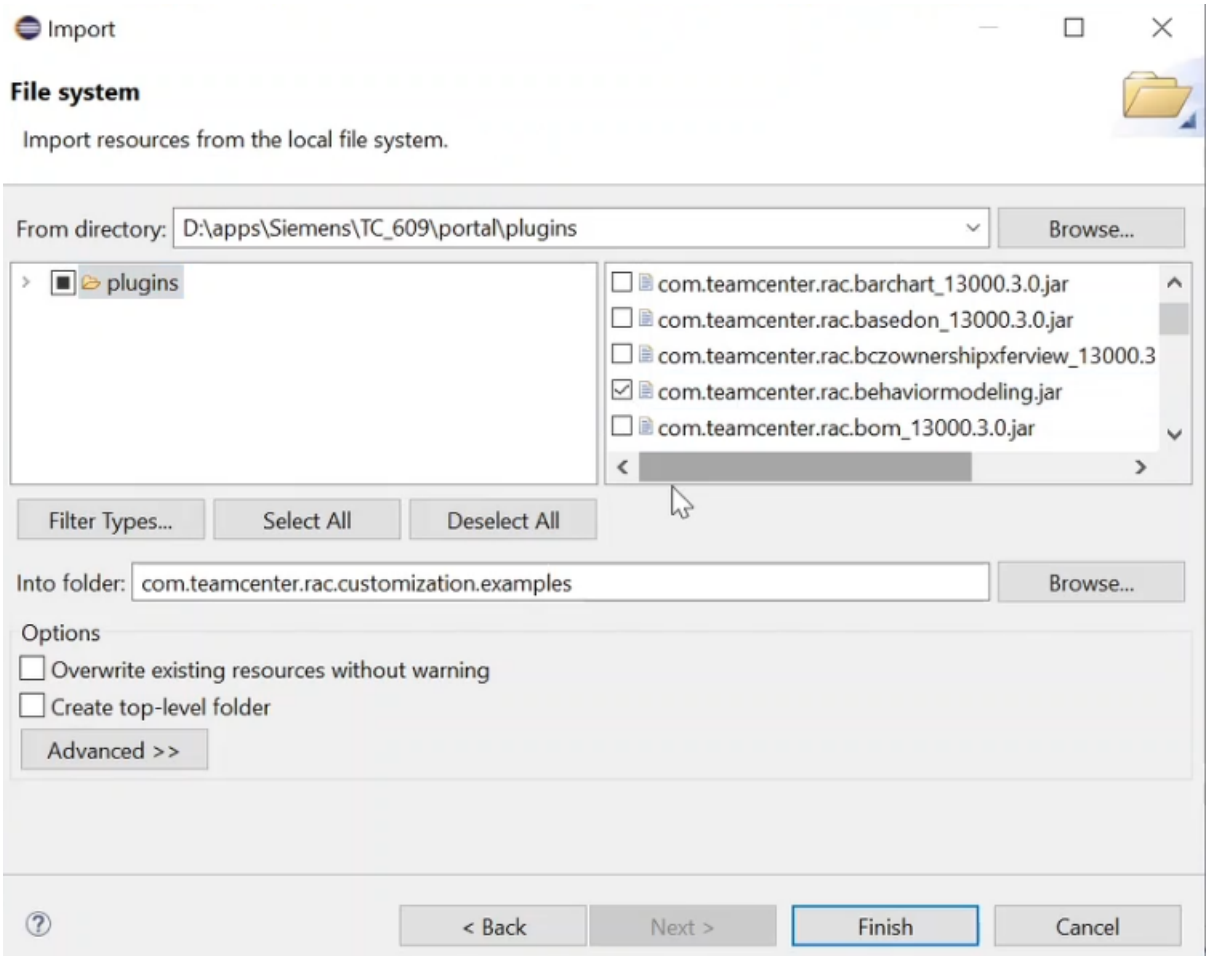
Folder in Toolkit	Jar file name
toolbox	commons-lang-2.6.jar
toolbox	commons-logging-1.1.3.jar
toolbox	derby.jar
toolbox	ezmorph-1.0.6.jar
toolbox	httpclient-4.5.2.jar
toolbox	httpcore-4.4.4.jar
toolbox	httpmime-4.5.13.jar
toolbox	jaxb-api.jar
toolbox	jaxb-impl.jar
toolbox	json-lib-2.4-jdk15.jar
toolbox	log4j-1.2-api-2.13.0.jar
toolbox	log4j-api-2.13.0.jar
toolbox	log4j-core-2.13.0.jar
toolbox	xercesImpl.jar
eclipse	org.eclipse.core.commands_3.9.200.v20180827-1727.jar
eclipse	org.eclipse.equinox.common_3.10.200.v20181021-1645.jar
eclipse	org.eclipse.jface_3.15.0.200.v20181123-1505.jar
eclipse	org.eclipse.swt.win32.win32.x86_64_3.109.0.v20181204-1801.jar

Import com.teamcenter.rac.behaviormodeling.jar in eclipse

You can load *com.teamcenter.rac.behaviormodeling.jar* in eclipse as follows:

Procedure

1. Right-click your project and choose **Import**.
2. In the **Import** dialog box, click **File System** and click **Next**.
3. In the **File System** dialog box, in the **From** directory section, click **Browse** and select the *plugin* folder. This folder is in the *TEAMCENTER _ROOT/portal* directory.
4. From the list next to the *plugins* directory, select the *com.teamcenter.rac.behaviormodeling.jar* file.



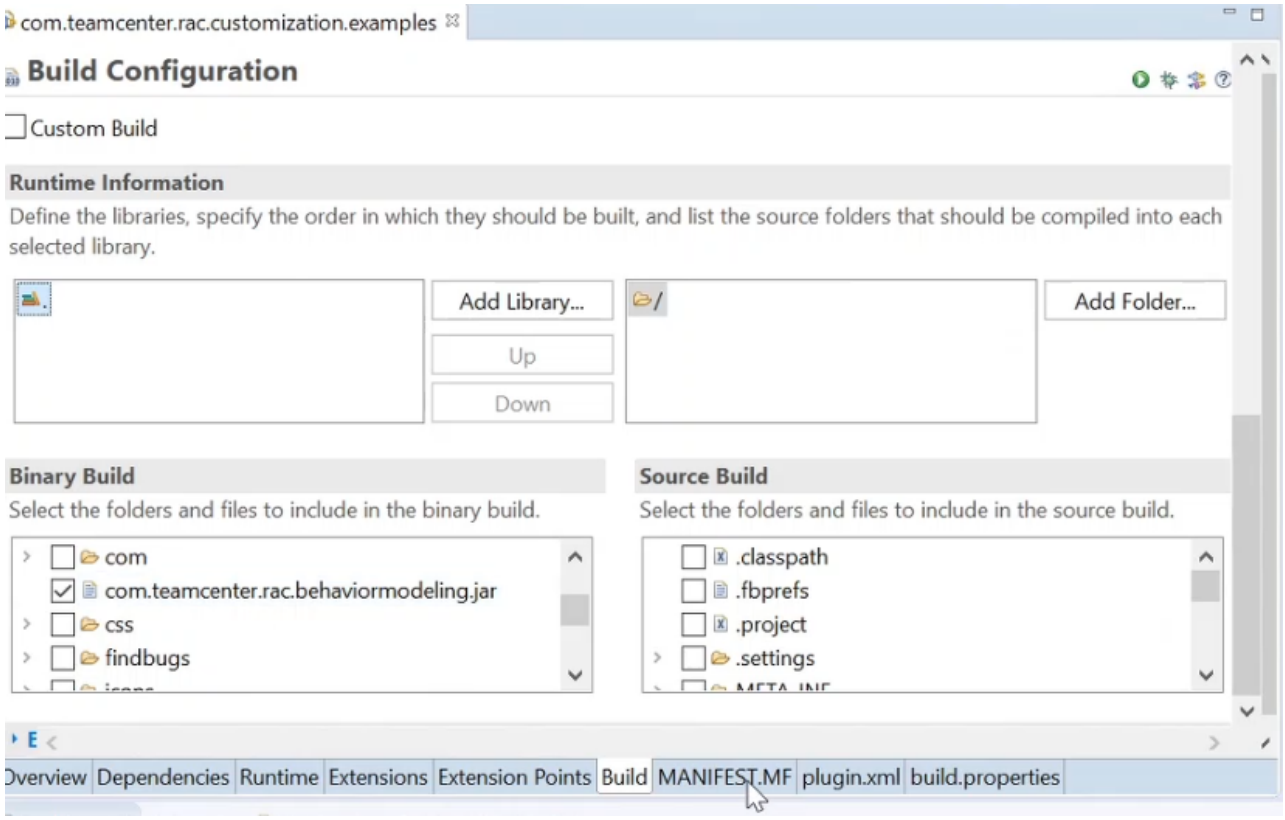
5. Click **Finish**.

6. In your project, expand the *META-INF* folder and click the *MANIFEST.MF* file.

7. Add the *com.teamcenter.rac.behaviormodeling.jar* file to the **Classpath** section of the **Runtime** tab if the JAR file is not already there.

8. Add a new library with name . to the **Classpath** section of the **Runtime** tab if the library is not already there.

9. Make sure that the **Binary Build** section of the **Build** tab contains the JAR file.



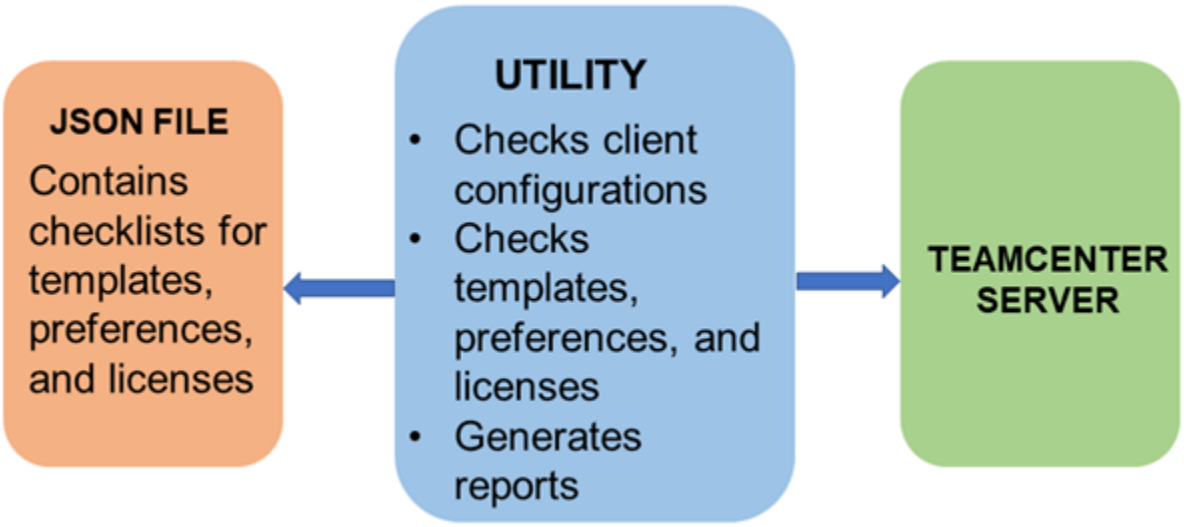
How the deployment diagnostic tool works

A deployment diagnostic tool is a utility that allows you to check the status of the MBSE Integration Gateway deployment.

The deployment diagnostic tool consists of a batch file and a JSON file.

- The batch file named *diagnostictool.bat* runs the diagnostic commands.
- The JSON file contains the list of diagnostic checks to run. A JSON file is available for each integration.

  These JSON files are available in the *TC_ROOT/bhm/utilities/json* directory of the MBSE Integration Gateway Client or in the *bhm/utilities/json* directory of the MBSE Integration Gateway Toolkit.



You can access the deployment diagnostic tool from either of the following locations:

- The *TC_ROOT/bhm/utilities* directory of the MBSE Integration Gateway Client
- The *bhm/utilities* directory of the MBSE Integration Gateway Toolkit

Check the status of the MBSE Integration Gateway deployment using the deployment diagnostic tool

A deployment diagnostic tool is a utility that allows you to check the status of the MBSE Integration Gateway deployment.

You can access the deployment diagnostic tool from either of the following locations:

- The *TC_ROOT/bhm/utilities* directory of the MBSE Integration Gateway Client
- The *bhm/utilities* directory of the MBSE Integration Gateway Toolkit

Prerequisites

If you plan to use the deployment diagnostic tool from the MBSE Integration Gateway Toolkit, ensure that you have **configured MBSE Integration Gateway**.

Procedure

You can use the deployment diagnostic tool from either the MBSE Integration Gateway Client or the MBSE Integration Gateway Toolkit.

1. Using the deployment diagnostic tool from the MBSE Integration Gateway client

From the *TC_ROOT/bhm/utilities* directory, run the *diagnostictool.bat* batch file as follows:

```
diagnostictool.bat -u=username -p=password -rootDirectory=Teamcenter-root-
directory -toolType=integrating-tool

diagnostictool.bat -u=demo -p=demo
-rootDirectory=D:\Siemens\Teamcenter -toolType=AMESIM
```

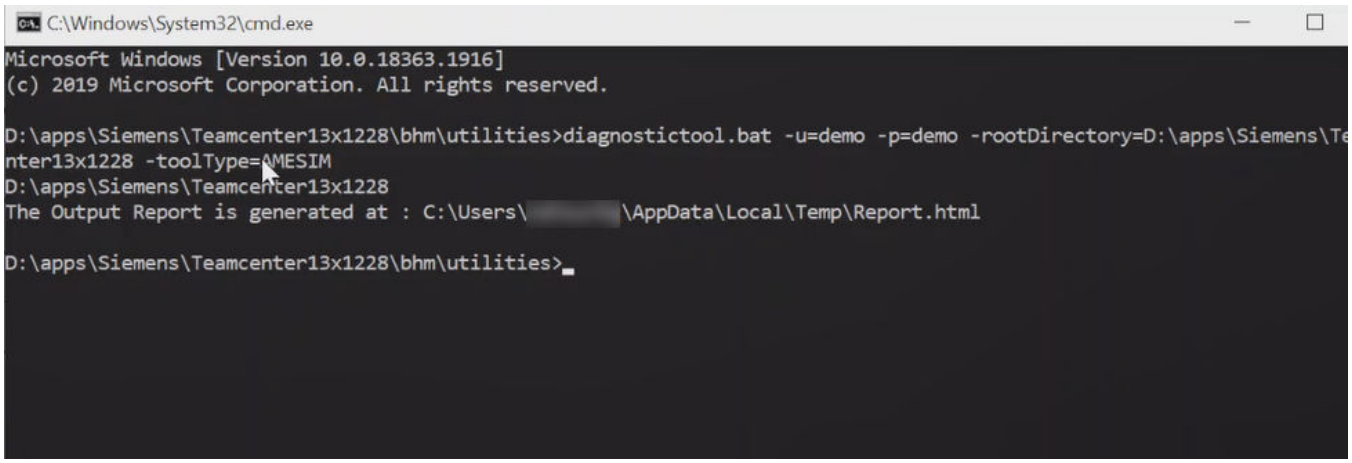
2. Using the deployment diagnostic tool from the MBSE Integration Gateway Toolkit

   From the *bhm/utilities* directory of the MBSE Integration Gateway Toolkit, run the *diagnostictool.bat* batch file as follows:

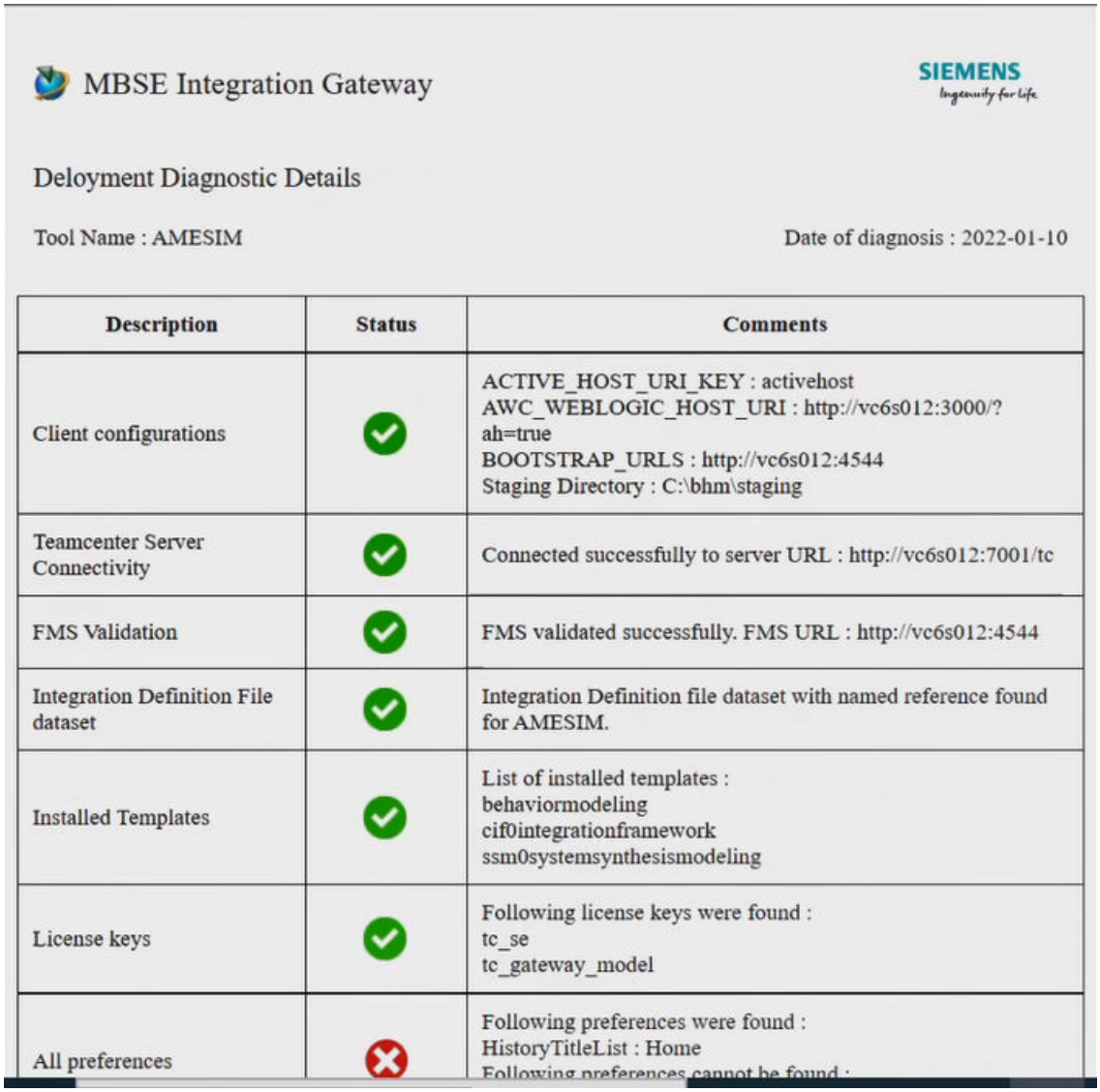
```
diagnostictool.bat -u=username -p=password -rootDirectory=Toolkit-path
-toolType=integrating-tool

diagnostictool.bat -u=demo -p=demo
-rootDirectory=C:\apps\kits\me\wntx64\mbse_integration_toolkit
-toolType=AMESIM
```

Results

The diagnostic report is generated, and the location of the diagnostic tool is displayed in the utility window.



The report shows which diagnostic tests were run and whether the tests passed, failed, or were not performed.



MBSE Integration Gateway

SIEMENS
Ingenuity for life

Deloyment Diagnostic Details

Tool Name : AMESIM

Date of diagnosis : 2022-01-10

Description	Status	Comments
Client configurations	✓	ACTIVE_HOST_URI_KEY : activehost AWC_WEBLOGIC_HOST_URI : http://vc6s012:3000/?ah=true BOOTSTRAP_URLS : http://vc6s012:4544 Staging Directory : C:\bhm\staging
Teamcenter Server Connectivity	✓	Connected successfully to server URL : http://vc6s012:7001/tc
FMS Validation	✓	FMS validated successfully. FMS URL : http://vc6s012:4544
Integration Definition File dataset	✓	Integration Definition file dataset with named reference found for AMESIM.
Installed Templates	✓	List of installed templates : behaviormodeling cif0integrationframework ssm0systemsynthesismodeling
License keys	✓	Following license keys were found : tc_se tc_gateway_model
All preferences	✗	Following preferences were found : HistoryTitleList : Home Following preferences cannot be found :

7. Using Common Integration Framework SOAs

Overview of Common Integration Framework SOAs

These are similar to Teamcenter SOAs and provide Java, .Net, and C++ bindings. However, these SOAs, unlike other Teamcenter SOAs, take JSON strings as input and output. Similar to Teamcenter SOAs, **serviceData** containing errors and warnings is returned through an SOA response structure.

Before calling the SOAs, make sure that the integration definition file is configured and uploaded to Teamcenter for the integrating tool.

JSON schema

JSON key	Description
AllContainers	Holds all the containers to be created, updated or imported from Teamcenter and information common to containers.
toolIdentifier	Application name to identify the integrating tool, for example MATLAB, ECAD, SAMPLE. This value is used by the framework to identify the integration definition file where integration specific configuration is done. This value and corresponding integration definition file are unique for different integrations. This is a required field.
Containers	This array holds containers to be created, updated or imported from Teamcenter. Each container is represented by a separate Teamcenter business object.
containerIdentifier	A unique identifier for the container assigned by the integrating modeling tool. This is required in all operations.
tcContainerIdentifier	A unique identifier of the Teamcenter business object corresponding to the container. This field is empty for **exportCollection** as no corresponding Teamcenter business object is created. It is a required filed for **updateCollection()** and **importCollection()** SOAs.
containerName	Name of container in the tool.
tcContainerName	Name of the Teamcenter business object corresponding to the container. If this field is empty then **containerName** is given to the Teamcenter object.
containerType	The object type of a container in the tool. This is used to process the corresponding Teamcenter object mapped in the integration definition file.

JSON key	Description
tcContainerType	The type of object in Teamcenter such as Item, Dataset. This maps to the **containerType** in the tool. If this value is empty, then the mapped Teamcenter object type is picked from the integration definition file.
isTopContainer	If value is **true** and **tcFolderUid** value is empty then the container is copied under **Newstuff** folder in Teamcenter.
action	Action to be performed on this container. **add** creates new container object in Teamcenter. **update** updates the relationship or metadata of existing Teamcenter component.
tcFolderUid	The UID of Teamcenter folder where the new container can be pasted. It is optional. If value of **isTopContainer** is **true** and **tcFolderUid** value is empty then the container is copied under **Newstuff** folder in Teamcenter.
tcLastModifiedDate	The last modified data of the container saved to Teamcenter. This is populated in the output JSON.
attributes	An array of attributes of model object in tool mapped to the Teamcenter object properties and their values.
attrName	Name of attribute in the tool.
tcAttrName	Name of the Teamcenter object property mapped with the name of attribute in tool.
value	Array of attribute values. If the Teamcenter object property accepts single value, then the array will have only one value.
revAttributes	An array of attributes of model object in tool mapped to the Teamcenter revision object properties and their values.
attrName	Name of attribute in tool.
tcAttrName	Name of the Teamcenter object property mapped with the name of attribute in tool.
value	Array of attribute values. If the Teamcenter object property accepts single value, then the array will have only one value.
datasetFileInfos	An array of attributes of tool files to be associated with a Teamcenter dataset.
fileName	Name of the file to be uploaded to Teamcenter without extension.
fileExtension	Extension of the file to be uploaded to Teamcenter without extension.
filePath	Absolute path of the folder where the file resides on the user desktop.

JSON key	Description
namedReferenceName	Name of the file reference in the dataset business object.
allowReplace	Allows to override an existing named reference of a dataset. **true** will override, **false** will add a new named reference to the dataset.
isText	The type of Named Reference file. **true** represents a text file, **false** represents a binary file.
writeTicket	This is an FMS write ticket to upload a file to Teamcenter dataset. This will be empty in case of saving a new dataset to Teamcenter. This will be populated in the output JSON returned by the Common Integration Framework service operations **exportCollection()** and **updateCollection()**.
readTicket	The FMS read ticked to download a file from Teamcenter dataset. This will be empty in case of saving data to Teamcenter. This is populated in the output JSON of **importCollection()**.
Components	Array of components associated to the container. The association can be of following types • a relation (GRM) • the product structure (BVR) of the owning container • referenced container (REF)
componentName	This is name of the occurrence if the this is a BVR association.
componentIdentifier	Unique identifier of a component in the tool. This is required field if component is added in the **Container**.
componentType	Type of component in the tool. For example a Model, Referenced Model or a Referenced library in Matlab. This is a required field.
tcComponentType	Type of Teamcenter object. If this is empty then the default value is picked from the mapping file of the tool. This is the Teamcenter type of a relation object or the occurrence object. In case of referenced component it is empty.
containerIdentifier	The container identifier of this component in the JSON document. This is required field.
tcContainerIdentifier	If this container exists in Teamcenter, then this is UID of the Teamcenter object of this component. This is empty in input JSON document of **exportCollection** and **updateCollection**.
action	Action to be performed for this component. **add** adds the component to the owning container. **update** updates the relationship or metadata of existing

JSON key	Description
	Teamcenter component. **remove** removes the association of the component with it's owning container. This is a required field.
attributes	An array of attributes of model object in tool mapped to the Teamcenter object properties and their values.
ComponentLinks	This array in **Component** is to establish an association between two components of a container.
componentIdentifier	This is the tool assigned unique identifier of component to be linked.
owningContainerIdentifier	This is the tool assigned unique identifier of container in which the component resides.
ContextualComponents	An array of contextual components, that is, associations or relations between two components in the context of a specific container, not necessarily the immediate parent container. For example, a tracelink relation between two occurrences of a wheel (Container) such as front wheel (component) and rear wheel (component) in the context of bicycle (container).
componentIdentifier	This is a tool or client assigned unique identifier of the component to be linked. This is a required field.
tcComponentIdentifier	This is a Teamcenter assigned unique identifier of the associated or related object. This is required field when the action is **UPDATE** or **REMOVE**.
componentType	It is the client or tool assigned type of the association or relation. In the integration definition file **componentType** is represented by type attribute in **<BHMElement>**. In integration definition file it is mapped to **reltype** in **<BHMElement>**. This is a required field.
tcComponentType	This is Teamcenter business object type corresponding to **componentType**. This is not a required input as **componentType** is mapped to **tcComponentType** in the integration definition file.
action	It can take following values: • **add**: Creates an association between two components. • **update**: Updates the metadata of existing Teamcenter relation between two components. The metadata to be updated is passed through attributes[] • **remove**: removes the association between two components. This is a required field.

JSON key	Description
Attributes	An array of attributes of the associated or related object.
attrName	Name of attribute in tool. In the integration definition file **attrName** corresponds to the name property in **<AttributeMapping>**. This is a required field if **tcAttrName** is not populated.
tcAttrName	Name of the Teamcenter object property. In the integration definition file **tcAttrName** corresponds to **tcattr** property in **<AttributeMapping>**. This is not required if the client name of attribute in tool or client is mapped to the Teamcenter object property in the integration definition file.
Value	These are the values of the attribute corresponding to **attrName** or **tcAttrName**. Array of attribute values. If the Teamcenter object property accepts single value, then the array will have only one value.
ComponentLinks	An array containing two ends of a contextual association or relation. For example, one end may represent a front wheel (component), an occurrence of wheel (container) and a second end may represent a rear wheel (component), an another occurrence of the same wheel (container) in the context of a bicycle (container). In this example both the components are in the same container Bicycle. In this case first component link is the defining end of a tracelink or a primary end of a relation and the second end is the complying end of a tracelink of secondary end of a relation in Teamcenter. If the components to be linked are from two different structures then one of the components must be from the container in which the contextual components are added. In this case the component in this container is the defining end of the tracelink or primary end of a relation object in Teamcenter.
componentIdentifier	This is a tool or client assigned unique identifier of a hierarchical component (BVR component). This represents one end of a contextual association. For example, front wheel in the context of a Bicycle. This is a required field.
tcComponentIdentifier	This is a Teamcenter assigned unique identifier of associated or related objects. This is required field when action is **UPDATE** or **REMOVE**.

owningComponentIdentifier	This is a :: (double colon) separated list of all parent componentIdentifiers
	For example, consider following example:
	`Bicycle (Container)`
	`|-- Front Wheel (Component)`

JSON key	Description

	<pre>|-- Rear Wheel (Component)
	
	Wheel (Container)
	|--Rim (Component)
	
	Rim (Container)
	|--Valve (Component)
	
	Valve (Container)</pre>
	To uniquely identify a component valve in front wheel of a bicycle the value of **owningComponentIdentifier** is componentIdentifier of Front Wheel component :: componentIdentifier of Rim component.
	If few of the parent components are already saved in Teamcenter then their componentIdentifiers are not added in **owningComponentIdentifier**. Their tcComponentIdentifiers are added in **tcOwningComponentIdentifier**.
	This is required if the parent components are not saved into Teamcenter.

tcOwningComponentIdentifier	This is a :: (double colon) separated list of all parent tcComponentIdentifiers.
	For example, consider following example:
	<pre>Bicycle (Container)
	|--Front Wheel (Component)
	|--Rear Wheel (Component)
	
	Wheel (Container)
	|--Rim (Component)
	
	Rim (Container)
	|--Valve (Component)
	
	Valve (Container)</pre>
	To uniquely identify a component valve in front wheel of a bicycle the value of **tcOwningComponentIdentifier** is **tcComponentIdentifier** of Front Wheel component :: tcComponentIdentifier of Rim component.
	If few of the parent components are not saved in Teamcenter then their componentIdentifiers are added in **owningComponentIdentifier**.
	This is required if the parent components are not saved into Teamcenter.

rootContainerIdentifier	This is **containerIdentifier** of a container in the context of which the association or relation is created.

JSON key	Description
tcRootContainerIdentifier	This is **tcContainerIdentifier** of a container in the context of which the association or relation is created.
Action	Value of action is **add** or **remove**. If the value is **add**, then the component is added to one of the ends of a relation. If the value is **remove**, then the component is removed from the relation. In this case the relation becomes a dangling relation.
userSpecificInputs	User actions and operation specific input.
inputKey	User action key name **CHECKOUT**: If its corresponding value is **true** the container is checked out in **exportCollection** and **updateCollection**. **CHECKIN**: If its corresponding value is **true** the container is checked in **exportCollection** and **updateCollection**. **REVISION_RULE**: This input key is send as an input to **importCollection**. Its corresponding value is a Teamcenter UID of a revision rule. The output of **importCollection** returns containers and components corresponding to this revision rule. **VARIANT_RULE**: This input key is send as an input to **importCollection**. Its corresponding value is a Teamcenter UID of a variant rule. The output of **importCollection** returns containers and components corresponding to this variant rule. This **inputKey** can also be used by customization team for passing information to extensions.
Values	Array of user action values, for example, **true**.
systemInputs	Special inputs.
inputKey	• **RETURN_MAPPED_ATTRIBUTES** If value is **true**, then output of **updateCollection** and **exportCollection** contains information of attributes mapped in the integration definition file. If value is **false** or not mentioned, then output of **updateCollection** and **exportCollection** contains information of attributes passed through the input JSON • **BOM_VIEW_TYPE_NAME**

JSON key	Description

	If a value corresponding to **BOM_VIEW_TYPE_NAME** is populated , then **exportCollection** creates a BOM view of the input value else default bom view is created.
	Example:
	`"systemInputs" : [ {`
	`"values" : [ "true" ],`
	`"inputKey" :`
	`"RETURN_MAPPED_ATTRIBUTES"`
	`} ],`

Values	Array of user action values, for example, **true**.

Using Common Integration Framework JAVA APIs

exportCollection()

This API saves or exports data to Teamcenter from the modeling tool. This API also uploads the files referenced by datasets to Teamcenter. Note that files have to be uploaded explicitly in the case of **exportCollection SOA**.

It takes a JSON file payload as input, creates all the defined objects, GRM and BVR associations, and related files as datasets in Teamcenter. The output returned by the API contains the Teamcenter identifiers and metadata of the objects and associations created in Teamcenter.

```
ResponseData exportCollection(String applicationName, String
containerIdentifier, String userSpecifiedSourceContextFolder, String
userSpecifiedTargetContextFolder, String inputJSONData )
```

Input Parameters

- **applicationName**

  The application name to identify the integrating tool, for example, MATLAB, ECAD, or Amesim. This value is used to identify the mapping file of the tool.

- **containerIdentifier**

  A unique identifier for the container in the tool. This is used to identify a container which is used as a component in another container.

- **userSpecifiedSourceContextFolder**

  This is not yet implemented.

- **userSpecifiedTargetContextFolder**

  This is not yet implemented.

- **inputData**

  The input payload in the form of a JSON string. The JSON payload is categorized in two types — Containers and Components.

- **Container**

  All objects that must be created in Teamcenter must have their input information as Container. This applies to Teamcenter business objects of the types Item and Dataset.

- **Component**

  All objects that are associated with the newly created Container in Teamcenter with a certain GRM relation or as an instance to the product structure of the Container.

Returns

ResponseData containing the JSON document with the Teamcenter UIDs of the objects created. The format of the output JSON file is the same as that of the input JSON file.

Use Case 1- Create a business object in Teamcenter representing a model in the integrating tool

Pass the following JSON string as an input to the **exportCollection()** API. The output returns the **tcContainerIdentifier** value in a similar output JSON. This is the Teamcenter assigned unique identifier of the business object representing the model. The **tcContainerIdentifier** can subsequently be used to import or update the model in Teamcenter.

```
{
  "AllContainers": {
    "toolIdentifier": "MIG",
    "Containers": [
      {
        "containerIdentifier": "Battery_Identifier",
        "containerName": "Battery",
        "containerType": "Assembly",
        "tcContainerName": "Battery",
        "isTopContainer": false,
```

```
        "action": "add",
        "attributes": [
          {
            "attrName": "Item_Description",
            "tcAttrName": "",
            "value": [ "This is a cart battery." ]
          }
        ],
        "revAttributes": [
          {
            "attrName": "Revision_Description",
            "tcAttrName": "object_desc",
            "value": [ "This is a cart battery, revision A." ]
          }
        ],
        "userSpecificInputs": [
          {
            "inputKey": "CHECKOUT",
            "values": [ "true" ]
          }
        ]
      }
    ]
  }
}
```

Use Case 2 – Create a business object in Teamcenter to store the model file authored in the integrating tool

Pass the following JSON string as an input to the **exportCollection()**. The output returns **tcContainerIdentifier** value of the dataset under which the file is to be uploaded. The output JSON also returns a writeTicket that can be used to upload files to Teamcenter using FMS SOAs.

```
{
  "AllContainers": {
    "toolIdentifier": "MIG",
    "Containers": [
      {
        "containerIdentifier": "GolfCart_Dataset_identifier",
        "containerName": "GolfCart_Dataset",
        "containerType": "DesignData",
        "tcContainerIdentifier": "",
        "tcContainerType": "Zip",
        "tcContainerName": "GolfCart_Dataset",
        "isTopContainer": false,
        "action": "add",
```

```
        "datasetFileInfos": [
          {
            "fileName": "GolfCartModelFile",
            "fileExtension": "zip",
            "filePath": "D:\\GolfCart",
            "namedReferenceName": "ZIPFILE",
            "allowReplace": true,
            "isText": false
          }
        ]
      }
    ]
  }
}
```

Use Case 3 - Associate or organize models in Teamcenter as a BVR structure

Pass the following JSON string as an input to the **exportCollection()** API to associate **Battery** in the code example as a component (BVR Structure) to **Golf Cart**.

Note that the object mapping for the containerType of **Battery**, that is, **Assembly**, must have **behaviorType="BVR"** in the integration definition file.

```
{
  "AllContainers": {
    "toolIdentifier": "MIG",
    "Containers": [
      {
        "containerIdentifier": "Battery_Identifier",
        "containerName": "Battery",
        "containerType": "Assembly",
        "tcContainerName": "Battery",
        "isTopContainer": false,
        "action": "add",
        "attributes": [
          {
            "attrName": "Item_Description",
            "tcAttrName": "",
            "value": [ "This is a cart battery." ]
          }
        ],
        "revAttributes": [
          {
            "attrName": "Revision_Description",
            "tcAttrName": "object_desc",
            "value": [ "This is a cart battery, revision A." ]
```

```
            }
          ],
          "components": []
        },
        {
          "containerIdentifier": "GolfCart_identifier",
          "containerName": "GolfCart",
          "containerType": "Project",
          "tcContainerName": "GolfCart ",
          "isTopContainer": true,
          "action": "add",
                  "attributes": [
            {
              "attrName": "Item_Description",
              "tcAttrName": "",
              "value": [ "This is a Golf Cart Assembly." ]
            }
          ],
          "revAttributes": [
            {
              "attrName": "Revision_Description",
              "tcAttrName": "object_desc",
              "value": [ "This is a golf cart, revision A." ]
            }
          ],
          "components": [
            {
                  "componentType": " Battery",
                  "componentName": " Battery_component_1",
                  "componentIdentifier": "Battery_component_1_identifier",
                  "containerIdentifier": "Battery_Identifier",
                  "tcContainerIdentifier": "",
                  "tcComponentIdentifier": "",
                  "tcComponentName": "",
                  "tcComponentType": "",
                  "tcComponentCopyStableID": "",
                  "owningContainerIdentifier": "",
                  "tcOwningContainerIdentifier": "",
                  "action": "add",
                  "attributes": [],
                  "ComponentLinks": []
            }
          ]
        }
      ]
    }
}
```

Use Case 4 - Associate or organize models in Teamcenter as GRM

Pass the following JSON string as an input to the **exportCollection()** API to associate the **OutputPort** as a GRM component to **Battery**.

Note that the object mapping for the **containerType** of **OutputPort**, that is, **Port**, must have **behaviorType="GRM"** in the integration definition file

```
{
  "AllContainers": {
    "toolIdentifier": "MIG",
    "Containers": [
      {
        "containerIdentifier": "OutputPort_Identifier",
        "containerName": "OutputPort",
        "containerType": "Port",
        "tcContainerName": "OutputPort",
        "isTopContainer": false,
        "action": "add",
        "attributes": [
          {
            "attrName": "Item_Description",
            "tcAttrName": "",
            "value": [ "This is an Output Port." ]
          }
        ],
        "revAttributes": [],
        "components": []
      },
      {
        "containerIdentifier": "Battery_identifier",
        "containerName": "Battery",
        "containerType": "Assembly",
        "tcContainerName": "Battery ",
        "isTopContainer": true,
        "action": "add",
                "attributes": [
          {
            "attrName": "Item_Description",
            "tcAttrName": "",
            "value": [ "This is a battery on the cart." ]
          }
        ],
        "revAttributes": [
          {
            "attrName": "Revision_Description",
            "tcAttrName": "object_desc",
            "value": [ "This is a battery, revision A." ]
```

```
            }
          ],
          "components": [
            {
                  "componentType": "OutputPort",
                  "componentName": " OutputPort_component_1",
                  "componentIdentifier":
"OutputPort_component_1_identifier",
                  "containerIdentifier": "OutputPort_Identifier",
                  "tcContainerIdentifier": "",
                  "tcComponentIdentifier": "",
                  "tcComponentName": "",
                  "tcComponentType": "",
                  "tcComponentCopyStableID": "",
                  "owningContainerIdentifier": "",
                  "tcOwningContainerIdentifier": "",
                  "action": "add",
                  "attributes": [],
                  "ComponentLinks": []
            }
          ]
        }
      ]
    }
}
```

You can use the **MBSE Integration Gateway Toolkit** to try out this API.

exportCollectionAsync()

> Note:
>
> Contact your Siemens representative before using this API to check for compatibility with future enhancements.

This API exports data to Teamcenter asynchronously from the modeling tool.

The API takes a JSON file payload as input, and then it creates all the defined objects, GRM and BVR associations, and related files as datasets in Teamcenter.

```
ResponseData exportCollectionAsync(String applicationName, String
rootContainerIdentifier, String userSpecifiedSourceContextFolder, String
userSpecifiedTargetContextFolder, String inputData)
```

Input Parameters

- **applicationName**

  The application name that identifies the integrating tool, for example, MATLAB, ECAD, or Amesim. This value is used to identify the mapping file of the tool.

- **rootContainerIdentifier**

  This is not yet implemented.

- **userSpecifiedSourceContextFolder**

  This is not yet implemented.

- **userSpecifiedTargetContextFolder**

  This is not yet implemented.

- **inputData**

  The input payload in the form of a JSON string. The JSON payload is categorized in two types - Containers and Components. The format and the construct use are the same as that of the **exportCollection** API.

- **Container**

  All objects that must be created in Teamcenter must have their input information set as Container. This requirement applies to Teamcenter business objects with the following types: Item and Dataset.

- **Component**

  All objects that are associated with the newly created Container in Teamcenter with a certain GRM relation, or as an instance to the product structure of the Container.

Returns

Because this is asynchronous operation, it does not return output JSON.

You can use the **MBSE Integration Gateway Toolkit** to try out this API.

syncOperationStatus()

> Note:
>
> Contact your Siemens representative before using this API to check for compatibility with future enhancements.

This API retrieves the latest status of the asynchronously exported project based on the provided container identifier. If the container identifier is empty, it processes all entities in the cache.

```
String syncOperationStatus (String ContainerIdentifier)
```

Input Parameters

- **ContainerIdentifier**

  The identifier of the container from which the operation status is to be retrieved. If this parameter is empty, all container statuses are checked.

Returns

A JSON stringis returned, representing the status of the requested containers.

You can use the **MBSE Integration Gateway Toolkit** to try out this API.

importCollection()

This API imports the model data that is saved in Teamcenter. This API also downloads the files referenced by datasets to Teamcenter. Note that the files must be downloaded explicitly in case of the **importCollection** SOA.

It takes a JSON file payload as input containing the **tcContainerIdentifier** of the model to be imported. It returns all the associated models and datasets and their properties as configured in the integration definition file.

```
ResponseData importCollection( String  applicationName, String
userSpecifiedDownloadFolder, boolean shouldDownloadFiles, String
inputData )
```

Input Parameters

- **applicationName**

  The name that identifies the integrating tool, for example, MATLAB, ECAD, or Amesim. This value is used to identify the mapping file of the tool.

- **userSpecifiedSourceContextFolder**

  This is not yet implemented.

- **userSpecifiedTargetContextFolder**

  This is not yet implemented.

- **inputData**

  The input payload in the form of a JSON string. The JSON payload contains the **tcContainerIdentifier** of the model to be imported.

Returns

ResponseData containing the JSON document with the properties and Teamcenter UID of the model specified in the input along with the properties and Teamcenter UIDs of all the associated models and datasets.

Use case 1 - Import the model from Teamcenter

In the following code example, the element **tcContainerIdentifier** with the value **wJa5Z1Ghr8jZAC** is the Teamcenter UID of the existing model object.

```
{
  "AllContainers": {
    "toolIdentifier": "MIG",
    "Containers": [
      {
        "tcContainerIdentifier": " wJa5Z1Ghr8jZAC ",
        "isTopContainer": true
      }
    ]
  }
}
```

Use case 2 - Apply filter on data that is extracted from Teamcenter using the importCollection API

Integrators can filter the output of the **importCollection** API. You can achieve this by specifying the filter expression for the type in the **importCollection** API. If no filter expression is specified for the type, then all types of objects are returned. Use the filters in the input payload of the **importCollection** API.

Supported boolean operators are **&&**(AND), **||**(OR), **!**(NOT). String comparisons with **==** (equals) and **!=** (not equals) are supported.

```
{
    "AllContainers":    {
        "toolIdentifier": "SYSML",
        "Containers": [
            {
                "tcContainerIdentifier": "FXrJw25LDpZ$lA",
                "isTopContainer": true
            }
        ],
        "Filters": {
            "Expressions": {
                "Block": "attr[Kind] != 'UNSET'",
                "pa.PhysicalComponent": "ignore_inheritance; true",
                "Function": "attr[Kind] == 'FUNCTION'"
            }
        }
    }
}
```

The previous example code returns the following based on the filters that are set:

- Returns objects of the **Block** type that have a property named **KIND** whose value is not equal to **UNSET**.
- Returns objects of the **Function** type that have a property named **KIND** whose value is equal to **FUNCTION**.
- The object **pa.PhysicalComponent** does not inherit the attributes of its parents when **ignore_inheritance; true** is specified. If you do not specify this parameter, the attributes are inherited from the parent.

  Inheritance is defined in the integration definition file using the **extends** keyword. You can override this inheritance using the **ignore_inheritance; true** value.

You can use the **MBSE Integration Gateway Toolkit** to try out this API.

compareContainers()

This API specifies the difference between the state of the model as modified in the modeling tool versus that state of the model when it was last published in Teamcenter.

```
ResponseData compareContainers(String applicationName, String
toolGeneratedJson, String contextDirectory )
```

Input Parameters

- **applicationName**

  The name that identifies the integrating tool, for example, MATLAB, ECAD, or Amesim. This value is used to identify the mapping file of the tool.

- **toolGeneratedJson**

  The input payload of the modified model in the form of a JSON string.

- **contextDirectory**

  The context folder of the model data from the staging directory.

Returns

ResponseData identifying the delta JSON document containing the modified models, datasets, and their properties. The format of output JSON is the same as that of the input JSON.

updateCollection()

This API takes the current state of the model in the form of input JSON and updates the corresponding Teamcenter objects. It returns the updated state of the model in a JSON format.

This API also updates the model properties, model composition (GRM or BVR), and associated files.

```
ResponseData  updateCollection( String  applicationName, String
inputData , String  contextFolder )
```

Input Parameters

- **applicationName**

  The name that identifies the integrating tool, for example, MATLAB, ECAD, or Amesim. This value is used to identify the mapping file of the tool.

- **inputData**

  The delta input payload in the form of a JSON string. This is the JSON file that is created after the **compareContainers()** API is executed.

- **contextDirectory**

  The context folder of the model data from the staging directory.

Returns

ResponseData containing the JSON document with the properties and Teamcenter UID of the model specified in the input along with the properties and Teamcenter UIDs of all the associated models and datasets.

Use case 1 – Update properties on a model

The following JSON input string is sent to **updateCollection** to modify the description of the model. In the example, the element **tcContainerIdentifier** with the value **wJa5Z1Ghr8jZAC** is the Teamcenter UID of the existing model object, **Battery**.

```
{
  "AllContainers": {
    "toolIdentifier": "MIG",
    "Containers": [
      {
        "containerIdentifier": "Battery_Identifier",
        "containerName": "Battery",
        "containerType": "Assembly",
        "tcContainerIdentifier": " wJa5Z1Ghr8jZAC ",
        "action": "update",
        "revAttributes": [
          {
            "attrName": "Revision_Description",
            "tcAttrName": "object_desc",
            "value": [ "This is a updated description" ]
          }
        ]
      }
    ]
  }
}
```

Use case 2 – Update model composition

The following JSON input string is sent to **updateCollection** to add a submodel **MotorDrive** to an existing model, **GolfCart**, in Teamcenter. In the example, the element **tcContainerIdentifier** with the value **wJa5Z1Ghr8jZFD** is the Teamcenter UID of the existing model object, **GolfCart**.

```
{"AllContainers":
 {
   "toolIdentifier": "MIG",
   "Containers": [
    {
```

```
    "containerIdentifier": "MotorDrive_Identifier",
    "containerName": " MotorDrive",
    "containerType": "Assembly",
    "tcContainerIdentifier": "",
    "tcContainerType": "",
    "tcContainerName": "",
    "isTopContainer": false,
    "action": "add",
    "tcFolderUid": "",
    "tcReservation": false,
    "tcLastModifiedDate": "",
    "attributes": [],
    "revAttributes": [],
},
{
     "containerIdentifier": "GolfCart_Identifier",
     "containerName": " GolfCart",
     "containerType": "Project",
     "tcContainerIdentifier": " wJa5Z1Ghr8jZFD ",
     "tcContainerType": "Assembly",
     "tcContainerName": " GolfCart ",
     "isTopContainer": true,
     "tcReservation": "false",
     "tcLastModifiedDate": "",
     "attributes": [],
     "revAttributes": [],
     "Components": [
      {
         "componentType": "jpg",
         "componentName": "MotorDrive_component",
         "componentIdentifier": "MotorDrive_component_Identifier”,
         "containerIdentifier": " MotorDrive_Identifier",
         "tcContainerIdentifier": "",
         "tcComponentIdentifier": "",
         "tcComponentName": "",
         "tcComponentType": "",
         "tcComponentCopyStableID": "",
         "owningContainerIdentifier": "",
         "tcOwningContainerIdentifier": "",
         "action": "add",
         "attributes": [],
         "ComponentLinks": []
      }],
      "datasetFileInfos": [],
      "action": "update"
     }
    ]
}
```

```
}
```

Use case 3 – Update associated files

The following JSON input string is sent to **updateCollection** to modify the file associated to the existing dataset. In the example, the element **tcContainerIdentifier** with the value **wJa5Z1Ghr8jMAT** is the Teamcenter UID of the **GolfCart_Dataset** in Teamcenter.

```
{
"AllContainers":{
"toolIdentifier": "MIG",
"Containers": [
 {
  "containerIdentifier": " GolfCart_Dataset_Identifier",
  "containerName": " GolfCart_Dataset",
  "containerType": "DesignData",
  "tcContainerIdentifier": “wJa5Z1Ghr8jMAT”
  "tcContainerType": "zip",
  "tcContainerName": " GolfCart_Dataset",
  "isTopContainer": false,
  "tcReservation": "false",
  "tcLastModifiedDate": "",
  "attributes": [],
  "revAttributes": [],
  "Components": [],
  "datasetFileInfos": [
   {
     "clientId": "",
     "fileName": " GolfCartModelFile",
     "filePath": "C:\\bhm\\staging\\GolfCart",
      "namedReferenceName": "ZIPFILE",
      "allowReplace": true,
      "isText": true,
      "writeTicket": "",

"readTicket":"66cf12caa7a82d10d91a544d225724207add97b7c0507ec06528a55bf32
4388bv1004F00000000000004869603e7ee794a489282021%2f03%2f03+00%3a00%3a00-18
01156312++++++++++++++++++++kalyani++++++++++++++++++++++++
+00f65fe0389894a48928++++++++++++
%5cengineering_5fdfeef3%5cgolfcart_7ie0dpw8yz3tp.zip",
"fileExtension": "zip",
       "fileHash": "DA39A3EE5E6B4B0D3255BFEF95601890AFD80709",
        "action": "update"
     }]
  }]
 }
```

```
}
```

You can use the **MBSE Integration Gateway Toolkit** to try out this API.

getAllTopContainers()

This API gets a list of all the top containers in the specified input JSON file.

```
ResponseData getAllTopContainers(String inputJson, List<String>
allSeperateJsonArray)
```

Input Parameters

- **inputJson**

  The JSON file that contains a mix of top containers and child containers.

- **downloadDirectory**

  The download folder in the staging directory where the tool artifact data is imported.

Output Parameters

- **Returns**

  ResponseData.OutputData allSeperateJsonArray containing the list of all the top containers.

Returns

ResponseData containing the operation status and errors or warnings if any.

getTcContainerIdentifier()

This is a Common Integration Services cache API to fetch the Teamcenter identifier (TcUID) for the specified tool artifact. It can be used to find if a tool artifact exists in Teamcenter.

```
ResponseData getTcContainerIdentifier(String containerIdentifier, String
downloadDirectory,  Map<String,String> outputMap )
```

Input Parameters

- **containerIdentifier**

A unique identifier for the artifact in the tool.

- **downloadDirectory**

  The download folder in the staging directory where the tool artifact data is imported.

Output Parameters

- **outputMap**

  Map of the download directory and **tcContainerIdentifier**.

Returns

ResponseData containing the operation status and errors or warnings if any.

getContainerData()

This is a Common Integration Services cache API to fetch the entire artifact data for the specified tool artifact.

```
ResponseData getTcContainerIdentifier(String containerIdentifier, String
downloadDirectory,  Map<String,String> outputMap )
```

Input Parameters

- **containerIdentifier**

  A unique identifier for the artifact in the tool.

- **downloadDirectory**

  The download folder in the staging directory where the tool artifact data is imported.

Output Parameters

- **outputMap**

  Map of the download directory and **tcContainerIdentifier**.

Returns

ResponseData containing the operation status and errors or warnings if any.

Active Workspace hosting APIs

The Active Workspace hosting APIs provide a set of tools to integrate the Active Workspace client with various modeling tools. These APIs support hosting environments such as SWT, JCEF, and Swing.

To use these APIs, ensure that you complete your configuration using one of the following options:

- Update following entries in the *CommonClient.properties* file:
  - **ACTIVE_HOST_URI_KEY=activehost**
  - **AWC_WEBLOGIC_HOST_URI=***Active-Workspace-URL*

    Example:

    AWC_WEBLOGIC_HOST_URI=http://vc6s015:3000/?ah=true

- Update the preference **ActiveWorkspaceHosting.URL** with the value of the Active Workspace hosting URL.

  Example:

  ActiveWorkspaceHosting.URL=http://vc6s015:3000/?ah=true

The following are the APIs:

- **tcmeActiveWorkspaceHostLogin()**

  This API establishes a Teamcenter session for the integration of the model management tool with Teamcenter. After a successful login, you can perform the integration operations.

  ```
  public TCMELoginOutput tcmeActiveWorkspaceHostLogin()
  ```

  Returns

  TCMEOperationOutput containing the operation status and error or warning message.

- **tcmeAddActiveWorkspaceHostControl()**

  This API adds a hosted Active Workspace client to a parent composite, which is an SWT widget.

  ```
  public void tcmeAddActiveWorkspaceHostControl(Composite parent, String
  strTOOLTYPE)
  ```

  Input parameters

- **parent**

  The parent composite of the modeling tool.

- **strTOOLTYPE**

  The tool type sent by the modeling tool.

- **tcmeAddActiveWorkspaceHostControlJCEF()**

  This API adds or shows a hosted Active Workspace client to a parent composite using a JCEF-based browser.

  ```
  public void tcmeAddActiveWorkspaceHostControlJCEF(Composite parent, String strTOOLTYPE)
  ```

  Input parameters

  - **parent**

    The parent composite of the modeling tool.

  - **strTOOLTYPE**

    The tool type sent by the modeling tool.

- **tcmeAddActiveWorkspaceHostControlJCEFSwing()**

  This API shows the Active Workspace client in a hosted environment with a JCEF-based browser.

  ```
  public JCEFBrowserPanel tcmeAddActiveWorkspaceHostControlJCEFSwing(String strTOOLTYPE)
  ```

  Input parameters

  - **strTOOLTYPE**

    Tool type sent by the modeling tool.

  Returns

  JCEFBrowserPanel: The Swing JPanel in which the Active Workspace client is hosted.

- **tcmeAddActiveWorkspaceHostControlSwing()**

  This API shows the Active Workspace client in a hosted environment using the Swing API.

```
public JCEFBrowserPanel tcmeAddActiveWorkspaceHostControlSwing(JPanel
parentPanel, String strTOOLTYPE)
```

Input parameters

- **parentPanel**

  The parent JPanel of the modeling tool.

- **strTOOLTYPE**

  The tool type sent by the modeling tool.

- **tcmeAddDragDropListener()**

  This API adds a drag-and-drop listener to the hosted Active Workspace client.

  ```
  public void tcmeAddDragDropListener(final IDragAndDropListener
  dragAndDropListener)
  ```

  Input parameters

  - **dragAndDropListener**

    Instance of the **browserinterop.IDragAndDropListener** service to be added.

- **tcmeRemoveDragDropListener()**

  This API removes the instance of the drag-and-drop listener from the hosted Active Workspace client.

  ```
  public void tcmeRemoveDragDropListener(final IDragAndDropListener
  dragAndDropListener)
  ```

  Input parameters

  - **dragAndDropListener**

    The instance of the **browserinterop.IDragAndDropListener** service to be removed.

- **tcmeShowObjectInActiveWorkspaceHostControl()**

  This API shows a desired Teamcenter object in the hosted Active Workspace client from the modeling tool. Use this API with **tcmeAddActiveWorkspaceHostControl** and **tcmeAddActiveWorkspaceHostControlJCEF**.

```
public void tcmeShowObjectInActiveWorkspaceHostControl(Composite parent,
String strTOOLTYPE, String showObjectTCUID)
```

Input parameters

- **parent**

  The parent composite of the modeling tool.

- **strTOOLTYPE**

  The tool type sent by the modeling tool.

- **showObjectTCUID**

  The Teamcenter UID of the object to be shown in the hosted Active Workspace.

- **tcmeShowObjectInActiveWorkspaceHostControl()**

  This API shows a desired Teamcenter object in the hosted Active Workspace client from the modeling tool. Use this API with **tcmeAddActiveWorkspaceHostControlJCEFSwing**.

  ```
  public JCEFBrowserPanel tcmeShowObjectInActiveWorkspaceHostControl(String
  showObjectTCUID)
  ```

  Input parameters

  - **showObjectTCUID**

    Teamcenter UID of the object to be shown in the hosted Active Workspace.

  Returns

  JCEFBrowserPanel: The Swing JPanel in which the Active Workspace client is hosted.

- **tcmeShowObjectInActiveWorkspaceHostControlSwing()**

  This API shows a desired Teamcenter object in the hosted Active Workspace client from the modeling tool. Use this API with **tcmeAddActiveWorkspaceHostControlSwing**.

  ```
  public JCEFBrowserPanel
  tcmeShowObjectInActiveWorkspaceHostControlSwing(JPanel parentPanel, String
  showObjectTCUID, boolean isWindowOpen)
  ```

  Input parameters

- **parentPanel**

  The parent JPanel of the modeling tool.

- **showObjectTCUID**

  The Teamcenter UID of the object to be shown in the hosted Active Workspace.

- **isWindowOpen**

  The parameter is set to **true** if the hosted Active Workspace client window or parent panel is already open. The parameter is set to **false** if the parent panel is closed and the user opens it.

- **Returns**

  JCEFBrowserPanel: The Swing JPanel in which the Active Workspace client is hosted.

- **tcmeShowObjectURIInActiveWorkspaceHostControl()**

  This API shows a desired Teamcenter object in the hosted Active Workspace client from the modeling tool. Use this API with **tcmeAddActiveWorkspaceHostControl** and **tcmeAddActiveWorkspaceHostControlJCEF**.

  ```
  public void tcmeShowObjectURIInActiveWorkspaceHostControl(Composite
  parent, String strTOOLTYPE, String showObjectURI)
  ```

  Input parameters

  - **parent**

    The parent composite of the modeling tool.

  - **strTOOLTYPE**

    The tool type sent by the modeling tool.

  - **showObjectURI**

    The URL of the Teamcenter object to be shown in the hosted Active Workspace.

- **tcmeShowObjectURIInActiveWorkspaceHostControl()**

  This API shows a desired Teamcenter object in the hosted Active Workspace client from the modeling tool. Use this API with **tcmeAddActiveWorkspaceHostControlJCEFSwing**.

```
public JCEFBrowserPanel
tcmeShowObjectURIInActiveWorkspaceHostControl(String showObjectURI)
```

Input parameters

- **showObjectURI**

  The URI of the Teamcenter object to be shown in the hosted Active Workspace.

- **tcmeShowHomePageInActiveWorkspaceHostControl()**

  This API shows the Active Workspace home page in the hosted Active Workspace client from the modeling tool. Use this API with **tcmeAddActiveWorkspaceHostControlJCEFSwing**.

  ```
  public JCEFBrowserPanel tcmeShowHomePageInActiveWorkspaceHostControl()
  ```

  Returns

  JCEFBrowserPanel: The Swing JPanel in which the Active Workspace client is hosted.

- **tcmeShowHomePageInActiveWorkspaceHostControlSwing()**

  This API shows the Active Workspace home page in the hosted Active Workspace client from the modeling tool. Use this API with **tcmeAddActiveWorkspaceHostControlSwing**.

  ```
  public JCEFBrowserPanel
  tcmeShowHomePageInActiveWorkspaceHostControlSwing(JPanel parentPanel)
  ```

  Input parameters

  - **parentPanel**

    The parent JPanel of the modeling tool.

  Returns

  JCEFBrowserPanel: The Swing JPanel in which the Active Workspace client is hosted.

- **tcmeSetShowHomePageOnCreation()**

  This API sets a flag to show the home page in the hosted Active Workspace client when it is open.

  ```
  public void tcmeSetShowHomePageOnCreation(boolean showHomePage)
  ```

  Input parameters

- **showHomePage**

  Set this parameter to **true** to show the home page, or **false** if you do not want to show the home page.

- **tcmeSetShowHomePageOnCreation()**

  This API sets a flag to show the home page in the hosted Active Workspace client when it is open.

  ```
  public void tcmeSetShowHomePageOnCreation(Composite parent, boolean showHomePage)
  ```

  Input parameters

  - **parent**

    The parent composite of the modeling tool.

  - **showHomePage**

    Set this parameter to **true** to show the home page, or **false** if you do not want to show the home page.

- **tcmeShowActiveWorkspaceComponent()**

  This API opens any Active Workspace component URL in the hosted Active Workspace.

  Initialize the Active Workspace client hosting using the existing APIs, such as **tcmeAddActiveWorkspaceHostControl**, **tcmeAddActiveWorkspaceHostControlJCEF**, **tcmeAddActiveWorkspaceHostControlJCEFSwing**, or **tcmeAddActiveWorkspaceHostControlSwing** before calling this API.

  ```
  public void tcmeShowActiveWorkspaceComponent(String awcComponentUrlWithParameters) throws TCMEException
  ```

  Input parameters

  - **awcComponentUrlWithParameters**

    The Active Workspace client component URL to open in the hosted Active Workspace. The value should not be null or empty. If you pass a null or empty value, an exception occurs.

  Exceptions

  TCMEException: This exception occurs in case of errors. Any errors are logged in the MBSE Integration Gateway client log.

- **tcmeGetHostControlInstance()**

  This API returns an instance of **IHostControlInstance**. Use this host control instance to register browser interoperability (browserinterop) services and handlers.

  Initialize the Active Workspace client hosting using the existing APIs, such as **tcmeAddActiveWorkspaceHostControl**, **tcmeAddActiveWorkspaceHostControlJCEF**, **tcmeAddActiveWorkspaceHostControlJCEFSwing**, or **tcmeAddActiveWorkspaceHostControlSwing** before calling this API.

  ```
  public IHostControlInstance tcmeGetHostControlInstance() throws
  TCMEException
  ```

  Returns

  IHostControlInstance: This instance is used for registering browserinterop services and handlers.

  Exception

  TCMEException: This exception occurs in case of errors. Any errors are logged in the MBSE Integration Gateway client log.

MBSE Configuration APIs

MBSEConfigurations class

The **MBSEConfigurations** class contains data members and some exposed APIs that can be used to generate the configuration files needed for the integration tools to work. All the exposed APIs are non-static. You must create objects of this class to access the APIs. For data members, getter and setter methods are also available.

Constructor: MBSEConfigurations(String toolType)

This is the **MBSEConfigurations** class constructor. It checks if the configuration files are created. If the configuration files exist, the data members are assigned the appropriate values that are read from the configuration files.

```
MBSEConfigurations(String toolType)
```

Inputs

- **toolType**

  The application name to identify the integrating tool. For example, MATLAB, ECAD, MIG, Amesim. This value is used to identify the staging directory mentioned in the configuration file for the tool.

Returns

MBSEConfigurations instance containing all the data members.

Methods

Method name	Description
setServerUrl(String serverUrl)	Sets the server URL.
getServerUrl()	Gets the server URL.
setFMSConfiguration(String bootStrpUrls, String tempFMSDir)	Sets the FMS configurations.
getFMSConfiguration()	Gets the FMS configurations.
setHostedActiveWorkspaceConfiguration(String awcHostURI, String awcHostUriKey)	Specifies the host URI and Host URI key for hosted Active Workspace.
getHostedActiveWorkspaceConfiguration()	Gets the host URI and Host URI key for hosted Active Workspace.
setTCCS(boolean isTCCS)	Sets the TCCS flag value.
getTCCS()	Gets the TCCS flag value.
setSSOConfiguration(String ssoLoginUrl, String ssoAppId, boolean ssoSessionFlag)	Sets the SSO environment configurations.
getSSOConfiguration()	Gets the SSO environment configurations.
setStagingDirectory(String tool, String toolStagingDir)	Configures the staging directory.
getToolStagingDir()	Gets the staging directory location.
setTeamcenterObjectCacheDirectory(String derbyCacheDir, boolean useDerbyCache)	Sets the Teamcenter Object Cache configurations.
getTeamcenterObjectCacheDirectory	Gets the Teamcenter Object Cache configurations.
getTCRootEnv()	Gets the *TC_ROOT* environment variable.
saveConfigurations()	Creates the configuration files *CommonClient.properties* and *BHMClient.properties*.

setServerUrl(String serverUrl)

This method sets the server URL.

```
setServerUrl(String serverUrl)
```

Inputs

- **serverUrl**

  The server URL.

getServerUrl()

This method gets the server URL.

```
setServerUrl(String serverUrl)
```

Returns

- **serverUrl**

  String containing the server URL.

setFMSConfiguration(String bootStrpUrls, String tempFMSDir)

This method is used to set the FMS configurations.

```
setFMSConfiguration(String bootStrpUrls, String tempFMSDir)
```

Inputs

- **bootStrpUrls**

  You can specify multiple bootstrap URLs. These URLs can point to different servers.

- **tempFMSDir**

  Temporary FMS directory where the files are downloaded during the FMS download.

getFMSConfiguration(String bootStrpUrls, String tempFMSDir)

This method is used to get the FMS configurations consisting of bootstrap URLs and temporary FMS directory.

```
Map<String,String> getFMSConfiguration()
```

setHostedActiveWorkspaceConfiguration(String awcHostURI, String awcHostUriKey)

This method specifies the host URI and Host URI key for hosted Active Workspace.

```
setHostedActiveWorkspaceConfiguration(String awcHostURI, String
awcHostUriKey)
```

Inputs

- **awcHostURI**

  The URL required to host Active Workspace in any tool.

- **awcHostUriKey**

  The key required to configure hosted Active Workspace.

getHostedActiveWorkspaceConfiguration()

This method gets the host URI and host URI key for hosted Active Workspace.

```
Map<String,String> getHostedActiveWorkspaceConfiguration()
```

Returns

- **awcConfiguration**

  The map contains the following keys and the corresponding values:

  - **AWC_WEBLOGIC_HOST_URI**

    The URL required to host Active Workspace in any tool.

  - **ACTIVE_HOST_URI_KEY**

    The key required to configure Hosted Active Workspace.

setTCCS(boolean isTCCS)

This method sets the TCCS flag value.

```
setTCCS(boolean isTCCS)
```

Inputs

- **isTCCS**

  Flag is set to **true** if TCCS is configured.

getTCCS()

This method gets the TCCS flag value.

```
boolean getTCCS()
```

Returns

- **isTCCS**

  Flag is **true** if TCCS is configured.

setSSOConfiguration(String ssoLoginUrl, String ssoAppId, boolean ssoSessionFlag)

This method sets the SSO environment configurations.

```
setSSOConfiguration(String ssoLoginUrl, String ssoAppId, boolean
ssoSessionFlag)
```

Inputs

- **ssoLoginUrl**

  The server URL of SSO environment.

- **ssoAppId**

  The application ID of the SSO environment.

- **ssoSessionFlag**

  The flag indicating whether to use the SSO environment.

getSSOConfiguration()

This method gets the SSO environment configurations.

```
Map<String,String> getSSOConfiguration()
```

Returns

- The map contains following keys and their corresponding values:
  - **ssoLoginUrl**

    The server URL of the SSO environment.
  - **ssoAppId**

    The application ID of the SSO environment.
  - **ssoSessionFlag**

    The flag indicating whether to use SSO environment.

setStagingDirectory(String tool, String toolStagingDir)

This method configures the staging directory.

```
setStagingDirectory(String tool, String toolStagingDir)
```

Inputs

- **tool**

  The application name.
- **toolStagingDir**

  The staging directory where the tool artifact data is imported.

getToolStagingDir()

This method gets the staging directory location.

```
String getToolStagingDir()
```

Returns

- **toolStagingDir**

  String containing the staging directory location.

setTeamcenterObjectCacheDirectory(String derbyCacheDir, boolean useDerbyCache)

This method sets the Teamcenter Object cache configurations.

```
setTeamcenterObjectCacheDirectory(String derbyCacheDir, boolean
useDerbyCache)
```

Inputs

- **derbyCacheDir**

  Client Cache location.

- **useDerbyCache**

  Flag to determine if Derby should be used.

getTeamcenterObjectCacheDirectory()

This method gets the Teamcenter Object Cache configurations.

```
setTeamcenterObjectCacheDirectory(String derbyCacheDir, boolean
useDerbyCache)
```

Returns

- **mapTcObjectCacheDir**

  The map contains following keys and their corresponding values:

- • **TC_CLIENT_CACHE**

  Flag is true if Derby is used.

  - **CacheDir**

    Client cache location.

getTCRootEnv()

This method gets the *TC_ROOT* environment variable. This method returns the *TC_ROOT* environment variable if set. If *TC_ROOT* location does not exist, the variable is set in the following directories:

- Windows: *C:\ProgramData\Siemens\MBSE\bhm*

- Linux: */etc/Siemens/MBSE/bhm*

```
String getTCRootEnv()
```

Returns

- **tcRoot**

  The location of *TC_ROOT*.

saveConfigurations()

This API creates the configuration files *CommonClient.properties* and *BHMClient.properties*. These files contain the necessary configurations needed for the integrations to work. These files are copied to *TC_ROOT\bhm* location if *TC_ROOT* is set. If *TC_ROOT* is not set then they are copied to *C:\ProgramData\Siemens\MBSE\bhm* for Windows and to */etc/Siemens/MBSE/bhm* for Linux.

The exceptions and status are logged in *ConfigurationStatus.log*. The APIs also copy the *log4j2.xml*, *bhmcache_create_table.sql* to the *TC_ROOT\bhm* location if *TC_ROOT* is set in that location otherwise it is copied in the configured location.

```
String saveConfigurations( )
```

Returns

- **tcRoot**

  The location where the configuration files are created.

getMappedObjectMapForTcType()

This API retrieves all object mappings that correspond to a specific Teamcenter type (**tcType**).

Th API iterates over all the object mappings. It populates a list of **ObjectMappingType** objects that correspond to the specified **tcType**. Once the list is populated, the function returns it.

```
public List<ObjectMappingType> getMappedObjectMapForTcType(String
tcType, String applicationName) throws Exception
```

Input Parameters

- **tcType**

  The Teamcenter type for which the object mappings need to be retrieved.

- **applicationName**

The application name to identify the integrating tool, such as MATLAB, ECAD, or Amesim. This value is used to identify the mapping file of the tool.

Returns

List<ObjectMappingType>: A list of **ObjectMappingType** objects containing the relevant mappings for the given **tcType**.

Exceptions

- **EXCEPTION_OBJECT_MAPPING_NOT_FOUND**: Thrown if object mapping is not found for some tool type.
- **EXCEPTION_CIRCULAR_INHERITANCE**: Thrown if circular inheritance is found.

Example

Consider the following object mappings:

```
<ObjectMapping type="PCA" behaviorType="MIXED" tctype="EDACCABase"
extends="Design">
<ObjectMapping type="Design" tctype="EDACCABase" extends="PCAComp">
<ObjectMapping type="PCAComp" behaviorType="MIXED" tctype="Item">
<ObjectMapping type="EDABOMComp" behaviorType="MIXED"
tctype="EDAComPart" extends="PCAComp">
```

For the given **tcType** of **EDACCABase**, the function will return the following object mappings along with their corresponding BHMElements:

```
<ObjectMapping type="PCA" behaviorType="MIXED" tctype="EDACCABase"
extends="Design">
<ObjectMapping type="Design" tctype="EDACCABase" extends="PCAComp">
```

getClassificationMapsForTcClassType()

This API retrieves the classes within the classification mapping for a specified Teamcenter class ID (**tcClassId**) from the integration definition file.

This API checks the tool type associated with these classes and identifies any inherited tool types. The classes from both the inherited tool types and the original tool type are added to the result. Finally, the function returns a comprehensive list of class types.

```
public List<ClassType> getClassificationMapsForTcClassType(String
tcClassId, String applicationName) throws ServiceException,
BHMException, NotLoadedException, BHMCommonException, Exception
```

Input Parameters

- **tcClassId**

  The Teamcenter class ID for which a comprehensive list of classes, including inherited classes, is to be retrieved.

- **applicationName**

  The application name to identify the integrating tool, such as MATLAB, ECAD, or Amesim. This value is used to identify the mapping file of the tool.

Returns

A list of class types.

Example

Consider the following object mappings:

```
<ObjectMapping type="PCAComp" behaviorType="MIXED" tctype="Item">
<ObjectMapping type="EDABOMComp" behaviorType="MIXED"
tctype="EDAComPart" extends="PCAComp">
<Class toolClassId="EDABOMComp" tcClassId="RNC363"
classifyAnchor="false">
<Class toolClassId="EDABOMComp" toolClassifybyAttribute="Partition"
toolClassifybyAttributeValue="top/HEATSINK" tcClassId="RNC364"
classifyAnchor="false">
<Class toolClassId="EDABOMComp" toolClassifybyAttribute="Partition"
toolClassifybyAttributeValue="new/HEATSINK" tcClassId="RNC364"
classifyAnchor="false">
<Class toolClassId="EDABOMComp" toolClassifybyAttribute="Partition"
toolClassifybyAttributeValue="top/HEATER" tcClassId="RNC366"
classifyAnchor="false">
<Class toolClassId="PCAComp" toolClassifybyAttribute="Partition"
toolClassifybyAttributeValue="new/HEATER" tcClassId="RNC366"
classifyAnchor="false">
```

For the given **tcClassId**, the function will return the following classes:

```
<Class toolClassId="EDABOMComp" tcClassId="RNC363"
```

```
classifyAnchor="false">
<Class toolClassId="EDABOMComp" toolClassifybyAttribute="Partition"
toolClassifybyAttributeValue="top/HEATSINK" tcClassId="RNC364"
classifyAnchor="false">
<Class toolClassId="EDABOMComp" toolClassifybyAttribute="Partition"
toolClassifybyAttributeValue="new/HEATSINK" tcClassId="RNC364"
classifyAnchor="false">
<Class toolClassId="EDABOMComp" toolClassifybyAttribute="Partition"
toolClassifybyAttributeValue="top/HEATER" tcClassId="RNC366"
classifyAnchor="false">
<Class toolClassId="PCAComp" toolClassifybyAttribute="Partition"
toolClassifybyAttributeValue="new/HEATER" tcClassId="RNC366"
classifyAnchor="false">
```

getClassificationMapForToolType()

This API retrieves the classes for a given tool type, including classes obtained through inheritance.

The API iterates through all classification mappings and populates a list of classes corresponding to the specified tool type. It checks for inheritance and appends any parent mappings to the list. Once the list is fully populated, the function returns it.

```
public List<ClassType> getClassificationMapForToolType(String toolType,
String applicationName) throws ServiceException, BHMException,
NotLoadedException, BHMCommonException, Exception
```

Input Parameters

- **toolType**

  The tool type for which classes are to be fetched.

- **applicationName**

  The application name to identify the integrating tool, such as MATLAB, ECAD, or Amesim. This value is used to identify the mapping file of the tool.

Returns

A list of **ClassType** objects containing the relevant classes for the given **toolType**.

Exceptions

- **EXCEPTION_OBJECT_MAPPING_NOT_FOUND**: Thrown if object mapping is not found for some tool type.

- **EXCEPTION_CIRCULAR_INHERITANCE**: Thrown if circular inheritance is found.

Example

Consider the following object mappings and classification mappings. In this example, the object mapping of EDABOMComp extends PCAComp.

```
<ObjectMapping type="PCA" behaviorType="MIXED" tctype="EDACCABase"
extends="Design">
<ObjectMapping type="PCAComp" behaviorType="MIXED" tctype="Item">
<ObjectMapping type="EDABOMComp" behaviorType="MIXED"
tctype="EDAComPart" extends="PCAComp">
<Class toolClassId="EDABOMComp" tcClassId="RNC363"
classifyAnchor="false">
<Class toolClassId="EDABOMComp" toolClassifybyAttribute="Partition"
toolClassifybyAttributeValue="top/HEATSINK" tcClassId="RNC364"
classifyAnchor="false">
<Class toolClassId="PCAComp" toolClassifybyAttribute="Partition"
toolClassifybyAttributeValue="new/HEATSINK" tcClassId="RNC364"
classifyAnchor="false">
<Class toolClassId="EDABOMComp" toolClassifybyAttribute="Partition"
toolClassifybyAttributeValue="top/HEATER" tcClassId="RNC366"
classifyAnchor="false">
<Class toolClassId="EDABOMComp1" toolClassifybyAttribute="Partition"
toolClassifybyAttributeValue="new/HEATER" tcClassId="RNC366"
classifyAnchor="false">
```

For the given **toolType**, the function will return the following classes:

```
<Class toolClassId="EDABOMComp" tcClassId="RNC363"
classifyAnchor="false">
<Class toolClassId="EDABOMComp" toolClassifybyAttribute="Partition"
toolClassifybyAttributeValue="top/HEATSINK" tcClassId="RNC364"
classifyAnchor="false">
<Class toolClassId="PCAComp" toolClassifybyAttribute="Partition"
toolClassifybyAttributeValue="new/HEATSINK" tcClassId="RNC364"
classifyAnchor="false">
<Class toolClassId="EDABOMComp" toolClassifybyAttribute="Partition"
toolClassifybyAttributeValue="top/HEATER" tcClassId="RNC366"
classifyAnchor="false">
```

getMappedRelAttributesForToolNRelType()

This API retrieves all attributes associated with the specified **toolRelType** of the **BHMElement** mapped under the object mappings of the provided **toolType** from the integration definition file.

It collects the attributes for each **BHMElement**, storing them in a map where the key is the **BHMElement** type and the value is the list of attributes. This process continues recursively through the inheritance chain, from child to parent object mappings, ensuring that attributes from both child and parent mappings are included. Finally, the function returns the complete map of attributes.

```
public Map<String, List<AttributeMappingType>>
getMappedRelAttributesForToolNRelType(String toolType, String
toolRelType, String applicationName) throws ServiceException,
BHMException, NotLoadedException, BHMCommonException, Exception
```

Input Parameters

- **toolType**

  The tool type for which the object mappings are to be fetched.

- **toolRelType**

  The **toolRelType** of the **BHMElement** for which the attributes are to be fetched.

- **applicationName**

  The application name to identify the integrating tool, such as MATLAB, ECAD, or Amesim. This value is used to identify the mapping file of the tool.

Returns

Map<String, List<AttributeMappingType>>: A map where each key is a **BHMElement** type, and the corresponding value is a list of attributes for that element, covering both the child and parent object mappings.

Exceptions

- **EXCEPTION_OBJECT_MAPPING_NOT_FOUND**: Thrown if object mapping is not found for some tool type.
- **EXCEPTION_CIRCULAR_INHERITANCE**: Thrown if circular inheritance is found.

Example

Consider the following object mappings, where **Design** extends **PCAComp** and the **toolRelType** under consideration is **forExample2**:

```
<ObjectMapping type="Design" tctype="EDACCABase" extends="PCAComp">
    <BHMElement type="RootModel" tctype="EDADesExpBrd">
```

```
        <AttributeMappings>
            <AttributeMapping name="name" tcattr="object_name"
includeinduplicatecheck="false"/>
            <AttributeMapping name="description" tcattr="object_desc"
includeinduplicatecheck="false"/>
            <AttributeMapping name="cifContainerIdentifier"
tcattr="Cif0ToolSpecificIntInfo::cif0ToolProperties"/>
            <AttributeMapping name="cifContainerType"
tcattr="Cif0ToolSpecificIntInfo::cif0ToolProperties"/>
        </AttributeMappings>
    </BHMElement>
    <BHMElement type="Design" tctype="EDADesExpBrd" behaviorType="GRM"
reltype="FND_tracelink" toolReltype="forExample2" reftype ="Design">
        <AttributeMappings>
            <AttributeMapping name="name" tcattr="object_name"
includeinduplicatecheck="false"/>
            <AttributeMapping name="description" tcattr="object_desc"
includeinduplicatecheck="false"/>
            <RevisionAttributeMapping>
                <AttributeMapping name="name" tcattr="object_name"
includeinduplicatecheck="false"/>
                <AttributeMapping name="description"
tcattr="object_desc" includeinduplicatecheck="false"/>
                <AttributeMapping name="cifContainerIdentifier"
tcattr="Cif0ToolSpecificIntInfo::cif0ToolProperties"/>
                <AttributeMapping name="cifContainerType"
tcattr="Cif0ToolSpecificIntInfo::cif0ToolProperties"/>
            </RevisionAttributeMapping>
        </AttributeMappings>
    </BHMElement>
    <FileMapping>
        <FileMap fileExt="zip" tcNameReferencedType="ZIPFILE"  />
    </FileMapping>
</ObjectMapping>

<ObjectMapping type="PCAComp" behaviorType="MIXED" tctype="Item">
    <BHMElement type="RootModel" tctype="Item">
        <AttributeMappings>
            <AttributeMapping name="name" tcattr="object_name"
includeinduplicatecheck="false"/>
            <AttributeMapping name="description" tcattr="object_desc"
includeinduplicatecheck="false"/>
            <RevisionAttributeMapping>
                <AttributeMapping name="name" tcattr="object_name"
includeinduplicatecheck="false"/>
                <AttributeMapping name="description"
tcattr="object_desc" includeinduplicatecheck="false"/>
                <AttributeMapping name="cifContainerIdentifier"
tcattr="Cif0ToolSpecificIntInfo::cif0ToolProperties"/>
```

```
                <AttributeMapping name="cifContainerType"
tcattr="Cif0ToolSpecificIntInfo::cif0ToolProperties"/>
            </RevisionAttributeMapping>
        </AttributeMappings>
    </BHMElement>
    <BHMElement type="PCAComp" tctype="Item" behaviorType="GRM"
reltype="FND_tracelink" toolReltype="forExample2" reftype ="PCAComp">
        <AttributeMappings>
            <AttributeMapping name="name" tcattr="object_name"
includeinduplicatecheck="false"/>
            <AttributeMapping name="description" tcattr="object_desc"
includeinduplicatecheck="false"/>
            <RevisionAttributeMapping>
                <AttributeMapping name="name" tcattr="object_name"
includeinduplicatecheck="false"/>
                <AttributeMapping name="description"
tcattr="object_desc" includeinduplicatecheck="false"/>
                <AttributeMapping name="cifContainerIdentifier"
tcattr="Cif0ToolSpecificIntInfo::cif0ToolProperties"/>
                <AttributeMapping name="cifContainerType"
tcattr="Cif0ToolSpecificIntInfo::cif0ToolProperties"/>
            </RevisionAttributeMapping>
        </AttributeMappings>
    </BHMElement>
    <OrganizationData/>
</ObjectMapping>
```

For the given use case, the output map will be as follows:

Key: **BHMElement type: Design**

Value: **Attributes:**

```
<AttributeMapping name="name" tcattr="object_name"
includeinduplicatecheck="false"/>
<AttributeMapping name="description" tcattr="object_desc"
includeinduplicatecheck="false"/>
<AttributeMapping name="cifContainerIdentifier"
tcattr="Cif0ToolSpecificIntInfo::cif0ToolProperties"/>
<AttributeMapping name="cifContainerType"
tcattr="Cif0ToolSpecificIntInfo::cif0ToolProperties"/>
```

Key: **BHMElement type: PCAComp**

Value: **Attributes:**

```
<AttributeMapping name="name" tcattr="object_name"
includeinduplicatecheck="false"/>
<AttributeMapping name="description" tcattr="object_desc"
includeinduplicatecheck="false"/>
<AttributeMapping name="cifContainerIdentifier"
tcattr="Cif0ToolSpecificIntInfo::cif0ToolProperties"/>
<AttributeMapping name="cifContainerType"
tcattr="Cif0ToolSpecificIntInfo::cif0ToolProperties"/>
```

getMappedAttributesForToolType()

This API retrieves all the attributes for a given object mapping of the specified toolType, including attributes from both extended mappings and classification mappings.

The API traverses through all parent object mappings of the given **toolType**, as well as the classification mappings, to gather attributes. Since attributes from object mappings and classification mappings are structured differently, the function uses an **AttributeInfo** structure to standardize and store them. After collecting all the attributes, they are stored in an **AttributeInfo** list, which is then returned.

```
public List<AttributeInfo> getMappedAttributesForToolType(String
toolType, String applicationName) throws ServiceException, BHMException,
NotLoadedException, BHMCommonException, Exception
```

Input Parameters

- **toolType**

  The tool type for which the attributes are to be fetched.

- **applicationName**

  The application name to identify the integrating tool, such as MATLAB, ECAD, or Amesim. This value is used to identify the mapping file of the tool.

Returns

List<AttributeInfo>: A list of **AttributeInfo** objects. Each **AttributeInfo** contains fields, such as **direction**, **tcAttribute**, **toolAttribute**, **isInherited**, and **attributeType**, to indicate the nature of the attribute.

The **attributeType** field specifies whether an attribute is a **revisionAttribute**, a **generalAttribute**, or an attribute from **classAttribute**.

Exceptions

- **EXCEPTION_OBJECT_MAPPING_NOT_FOUND**: Thrown if object mapping is not found for some tool type.
- **EXCEPTION_CIRCULAR_INHERITANCE**: Thrown if circular inheritance is found.

Example

Consider the following object mappings, where **Design** extends **PCAComp**:

```
<ObjectMapping type="Design" tctype="EDACCABase" extends="PCAComp">
    <BHMElement type="RootModel" tctype="EDADesExpBrd">
        <AttributeMappings>
            <AttributeMapping name="name" tcattr="object_name"
includeinduplicatecheck="false"/>
            <AttributeMapping name="description" tcattr="object_desc"
includeinduplicatecheck="false"/>
            <AttributeMapping name="cifContainerIdentifier"
tcattr="Cif0ToolSpecificIntInfo::cif0ToolProperties"/>
            <AttributeMapping name="cifContainerType"
tcattr="Cif0ToolSpecificIntInfo::cif0ToolProperties"/>
        </AttributeMappings>
    </BHMElement>
    <FileMapping>
        <FileMap fileExt="zip" tcNameReferencedType="ZIPFILE"  />
    </FileMapping>
</ObjectMapping>

<ObjectMapping type="PCAComp" behaviorType="MIXED" tctype="Item">
    <BHMElement type="RootModel" tctype="Item">
        <AttributeMappings>
            <AttributeMapping name="name" tcattr="object_name"
includeinduplicatecheck="false"/>
            <AttributeMapping name="description" tcattr="object_desc"
includeinduplicatecheck="false"/>
            <RevisionAttributeMapping>
                <AttributeMapping name="name" tcattr="object_name"
includeinduplicatecheck="false"/>
                <AttributeMapping name="description"
tcattr="object_desc" includeinduplicatecheck="false"/>
                <AttributeMapping name="cifContainerIdentifier"
tcattr="Cif0ToolSpecificIntInfo::cif0ToolProperties"/>
                <AttributeMapping name="cifContainerType"
tcattr="Cif0ToolSpecificIntInfo::cif0ToolProperties"/>
            </RevisionAttributeMapping>
        </AttributeMappings>
```

```
        </BHMElement>
        <OrganizationData/>
    </ObjectMapping>
```

For the given use case, the output will be as follows:

Each row in the table represents an **AttributeInfo** object with fields, such as **direction**, **isInherited**, and other characteristics, provided in the corresponding column headers.

	TcAttribute	Direction	Tool attribute	IsInherited	Attribute type
1	object_name	null	name	FALSE	AttributeMapping
2	object_desc	null	description	FALSE	AttributeMapping
3	object_name	null	name	FALSE	RevisionAttributeMappin
4	object_desc	null	description	FALSE	RevisionAttributeMappin
5	Cif0ToolSpecificIntInfo::cif0ToolProperties	null	cifContainerIdentifier	FALSE	RevisionAttributeMappin
6	Cif0ToolSpecificIntInfo::cif0ToolProperties	null	cifContainerType	FALSE	RevisionAttributeMappin
7	Cif0ToolSpecificIntInfo::cif0ToolProperties	null	cifContainerIdentifier	TRUE	AttributeMapping
8	Cif0ToolSpecificIntInfo::cif0ToolProperties	null	cifContainerType	TRUE	AttributeMapping

getMappedObjectMapForToolType()

This API returns the object mapping, along with all attributes inherited from its parent mappings.

The API first retrieves the object mapping for the specified **toolType**. Once the object mapping is found, the function iterates through its parent mappings, collecting attributes from each parent. After traversing all parent mappings and aggregating their attributes, the function returns the complete object mapping with the combined attributes.

```
public ObjectMappingType getMappedObjectMapForToolType(String toolType,
String applicationName) throws ServiceException, BHMException,
NotLoadedException, BHMCommonException, Exception
```

Input Parameters

- **toolType**

  The tool type for which the object mapping is to be fetched.

- **applicationName**

  The application name to identify the integrating tool, such as MATLAB, ECAD, or Amesim. This value is used to identify the mapping file of the tool.

Returns

ObjectMappingType: The object mapping, with attributes inherited from all its parent mappings.

Exceptions

- **EXCEPTION_OBJECT_MAPPING_NOT_FOUND**: Thrown if object mapping is not found for some tool type.
- **EXCEPTION_CIRCULAR_INHERITANCE**: Thrown if circular inheritance is found.

Example 1: Single-level inheritance

Consider the following object mappings, where **PCA** extends **Design** and the toolType is **PCA**:

```
<ObjectMapping type="PCA" behaviorType="MIXED" tctype="EDACCABase"
extends="Design">
</ObjectMapping>

<ObjectMapping type="Design" tctype="EDACCABase">
    <BHMElement type="RootModel" tctype="EDADesExpBrd">
        <AttributeMappings>
            <AttributeMapping name="name" tcattr="object_name"
includeinduplicatecheck="false"/>
            <AttributeMapping name="description" tcattr="object_desc"
includeinduplicatecheck="false"/>
            <AttributeMapping name="cifContainerIdentifier"
tcattr="Cif0ToolSpecificIntInfo::cif0ToolProperties"/>
            <AttributeMapping name="cifContainerType"
tcattr="Cif0ToolSpecificIntInfo::cif0ToolProperties"/>
        </AttributeMappings>
    </BHMElement>
    <FileMapping>
        <FileMap fileExt="zip" tcNameReferencedType="ZIPFILE"  />
    </FileMapping>
</ObjectMapping>
```

The output in this case will be the following object mapping where the attributes for the **ObjectMapping PCA** are updated because of the inheritance:

```
<ObjectMapping type="PCA" behaviorType="MIXED" tctype="EDACCABase"
extends="Design">
    <BHMElement type="RootModel" tctype="EDADesExpBrd">
        <AttributeMappings>
            <AttributeMapping name="name" tcattr="object_name"
```

```
includeinduplicatecheck="false"/>
            <AttributeMapping name="description" tcattr="object_desc"
includeinduplicatecheck="false"/>
            <AttributeMapping name="cifContainerIdentifier"
tcattr="Cif0ToolSpecificIntInfo::cif0ToolProperties"/>
            <AttributeMapping name="cifContainerType"
tcattr="Cif0ToolSpecificIntInfo::cif0ToolProperties"/>
        </AttributeMappings>
    </BHMElement>
</ObjectMapping>
```

Example 2: Multi-level inheritance

Consider the following object mappings, where **PCA** extends **Design**, **Design** extends **PCAComp**, and the **toolType** is **PCA**:

```
<ObjectMapping type="PCA" behaviorType="MIXED" tctype="EDACCABase"
extends="Design">
    <BHMElement type="RootModel" tctype="EDACCABase">
        <AttributeMappings>
            <AttributeMapping name="name" tcattr="object_name"
includeinduplicatecheck="false"/>
            <AttributeMapping name="description" tcattr="object_desc"
includeinduplicatecheck="false"/>
            <RevisionAttributeMapping>
                <AttributeMapping name="name" tcattr="object_name"
includeinduplicatecheck="false"/>
                <AttributeMapping name="description"
tcattr="object_desc" includeinduplicatecheck="false"/>
                <AttributeMapping name="cifContainerIdentifier"
tcattr="Cif0ToolSpecificIntInfo::cif0ToolProperties"/>
                <AttributeMapping name="cifContainerType"
tcattr="Cif0ToolSpecificIntInfo::cif0ToolProperties"/>
            </RevisionAttributeMapping>
        </AttributeMappings>
    </BHMElement>
    <BHMElement type="Design" tctype="EDADesExpBrd" behaviorType="GRM"
reltype="IMAN_specification" toolReltype="forExample" reftype ="Design">
        <AttributeMappings>
            <AttributeMapping name="name" tcattr="object_name"
includeinduplicatecheck="false"/>
            <RevisionAttributeMapping/>
        </AttributeMappings>
    </BHMElement>
    <BHMElement type="Design" tctype="EDADesExpBrd" behaviorType="GRM"
reltype="FND_tracelink" toolReltype="forExample2" reftype ="Design">
        <AttributeMappings>
            <AttributeMapping name="name" tcattr="object_name"
```

```
includeinduplicatecheck="false"/>
            <RevisionAttributeMapping/>
        </AttributeMappings>
    </BHMElement>
    <BHMElement type="Design" tctype="EDADesExpBrd" behaviorType="BVR"
reltype="" toolReltype="forExample3" reftype ="Design">
        <AttributeMappings>
            <AttributeMapping name="name" tcattr="object_name"
includeinduplicatecheck="false"/>
            <RevisionAttributeMapping/>
        </AttributeMappings>
    </BHMElement>
    <BHMElement type="PCAComp" tctype="Item" behaviorType="GRM"
reltype="FND_tracelink" toolReltype="forExample2" reftype ="PCAComp">
        <AttributeMappings>
            <RevisionAttributeMapping/>
        </AttributeMappings>
    </BHMElement>
    <BHMElement type="PCAComp" tctype="Item" behaviorType="BVR"
reltype="" reftype ="PCAComp">
        <AttributeMappings>
            <RevisionAttributeMapping/>
        </AttributeMappings>
    </BHMElement>
    <BHMElement type="EDABOMComp" tctype="EDAComPart" behaviorType="BVR"
reftype ="EDABOMComp">
    </BHMElement>
</ObjectMapping>

<ObjectMapping type="Design" tctype="EDACCABase" extends="PCAComp">
    <BHMElement type="RootModel" tctype="EDADesExpBrd">
        <AttributeMappings>
            <AttributeMapping name="name" tcattr="object_name"
includeinduplicatecheck="false"/>
            <AttributeMapping name="description" tcattr="object_desc"
includeinduplicatecheck="false"/>
            <AttributeMapping name="cifContainerIdentifier"
tcattr="Cif0ToolSpecificIntInfo::cif0ToolProperties"/>
            <AttributeMapping name="cifContainerType"
tcattr="Cif0ToolSpecificIntInfo::cif0ToolProperties"/>
        </AttributeMappings>
    </BHMElement>
    <BHMElement type="PCAComp" tctype="Item" behaviorType="GRM"
reltype="FND_tracelink" toolReltype="forExample2" reftype ="PCAComp">
        <AttributeMappings>
            <AttributeMapping name="name" tcattr="object_name"
includeinduplicatecheck="false"/>
            <AttributeMapping name="description" tcattr="object_desc"
includeinduplicatecheck="false"/>
```

```
            <AttributeMapping name="cifContainerIdentifier"
tcattr="Cif0ToolSpecificIntInfo::cif0ToolProperties"/>
            <AttributeMapping name="cifContainerType"
tcattr="Cif0ToolSpecificIntInfo::cif0ToolProperties"/>
        </AttributeMappings>
    </BHMElement>
    <BHMElement type="EDABOMComp" tctype="EDAComPart" behaviorType="BVR"
reftype ="EDABOMComp">
        <AttributeMappings>
            <AttributeMapping name="name" tcattr="object_name"
includeinduplicatecheck="false"/>
            <RevisionAttributeMapping/>
        </AttributeMappings>
    </BHMElement>
    <FileMapping>
        <FileMap fileExt="zip" tcNameReferencedType="ZIPFILE"  />
    </FileMapping>
</ObjectMapping>

<ObjectMapping type="PCAComp" behaviorType="MIXED" tctype="Item">
    <BHMElement type="RootModel" tctype="Item">
        <AttributeMappings>
            <AttributeMapping name="name" tcattr="object_name"
includeinduplicatecheck="false"/>
            <AttributeMapping name="description" tcattr="object_desc"
includeinduplicatecheck="false"/>
            <RevisionAttributeMapping>
                <AttributeMapping name="name" tcattr="object_name"
includeinduplicatecheck="false"/>
                <AttributeMapping name="description"
tcattr="object_desc" includeinduplicatecheck="false"/>
                <AttributeMapping name="cifContainerIdentifier"
tcattr="Cif0ToolSpecificIntInfo::cif0ToolProperties"/>
                <AttributeMapping name="cifContainerType"
tcattr="Cif0ToolSpecificIntInfo::cif0ToolProperties"/>
            </RevisionAttributeMapping>
        </AttributeMappings>
    </BHMElement>
    <BHMElement type="EDABOMComp" tctype="EDAComPart" behaviorType="BVR"
reftype ="EDABOMComp">
        <AttributeMappings>
            <AttributeMapping name="name" tcattr="object_name"
includeinduplicatecheck="false"/>
            <AttributeMapping name="description" tcattr="object_desc"
includeinduplicatecheck="false"/>
            <RevisionAttributeMapping>
                <AttributeMapping name="name" tcattr="object_name"
includeinduplicatecheck="false"/>
                <AttributeMapping name="description"
```

```
tcattr="object_desc" includeinduplicatecheck="false"/>
                <AttributeMapping name="cifContainerIdentifier"
tcattr="Cif0ToolSpecificIntInfo::cif0ToolProperties"/>
                <AttributeMapping name="cifContainerType"
tcattr="Cif0ToolSpecificIntInfo::cif0ToolProperties"/>
            </RevisionAttributeMapping>
        </AttributeMappings>
    </BHMElement>
    <OrganizationData/>
</ObjectMapping>
```

The output in this case will be the following object mapping, where the attributes for the **BHMElement** (**PCAComp**) of the object mapping **PCA** are updated because of the inheritance:

```
<ObjectMapping type="PCA" behaviorType="MIXED" tctype="EDACCABase"
extends="Design">
    <BHMElement type="RootModel" tctype="EDACCABase">
        <AttributeMappings>
            <AttributeMapping name="name" tcattr="object_name"
includeinduplicatecheck="false"/>
            <AttributeMapping name="description" tcattr="object_desc"
includeinduplicatecheck="false"/>
            <RevisionAttributeMapping>
                <AttributeMapping name="name" tcattr="object_name"
includeinduplicatecheck="false"/>
                <AttributeMapping name="description"
tcattr="object_desc" includeinduplicatecheck="false"/>
                <AttributeMapping name="cifContainerIdentifier"
tcattr="Cif0ToolSpecificIntInfo::cif0ToolProperties"/>
                <AttributeMapping name="cifContainerType"
tcattr="Cif0ToolSpecificIntInfo::cif0ToolProperties"/>
            </RevisionAttributeMapping>
        </AttributeMappings>
    </BHMElement>
    <BHMElement type="Design" tctype="EDADesExpBrd" behaviorType="GRM"
reltype="IMAN_specification" toolReltype="forExample" reftype ="Design">
        <AttributeMappings>
            <AttributeMapping name="name" tcattr="object_name"
includeinduplicatecheck="false"/>
            <RevisionAttributeMapping/>
        </AttributeMappings>
    </BHMElement>
    <BHMElement type="Design" tctype="EDADesExpBrd" behaviorType="GRM"
reltype="FND_tracelink" toolReltype="forExample2" reftype ="Design">
        <AttributeMappings>
            <AttributeMapping name="name" tcattr="object_name"
includeinduplicatecheck="false"/>
            <RevisionAttributeMapping/>
```

```
        </AttributeMappings>
    </BHMElement>
    <BHMElement type="Design" tctype="EDADesExpBrd" behaviorType="BVR"
reltype="" toolReltype="forExample3" reftype ="Design">
        <AttributeMappings>
            <AttributeMapping name="name" tcattr="object_name"
includeinduplicatecheck="false"/>
            <RevisionAttributeMapping/>
        </AttributeMappings>
    </BHMElement>
    <BHMElement type="PCAComp" tctype="Item" behaviorType="GRM"
reltype="FND_tracelink" toolReltype="forExample2" reftype ="PCAComp">
        <AttributeMappings>
            <AttributeMapping name="name" tcattr="object_name"
includeinduplicatecheck="false"/>
            <AttributeMapping name="description" tcattr="object_desc"
includeinduplicatecheck="false"/>
            <AttributeMapping name="cifContainerIdentifier"
tcattr="Cif0ToolSpecificIntInfo::cif0ToolProperties"/>
            <AttributeMapping name="cifContainerType"
tcattr="Cif0ToolSpecificIntInfo::cif0ToolProperties"/>
            <RevisionAttributeMapping/>
        </AttributeMappings>
    </BHMElement>
    <BHMElement type="PCAComp" tctype="Item" behaviorType="BVR"
reltype="" reftype ="PCAComp">
        <AttributeMappings>
            <RevisionAttributeMapping/>
        </AttributeMappings>
    </BHMElement>
    <BHMElement type="EDABOMComp" tctype="EDAComPart" behaviorType="BVR"
reftype ="EDABOMComp">
    </BHMElement>
</ObjectMapping>
```

Using Common Integration Framework server SOAs

exportCollection()

```
DataManagementImpl::CollectionOutput exportCollection ( const
std::string inputData )
```

Use Case 1 – Create a business object in Teamcenter representing the integrating tool authored model

This example shows how to create an Item object representing a GolfCart model. Also after creating the object in Teamcenter, it is checked out using userSpecificInputs as shown in the JSON example.

The following entry in the Integration Definition File is required for this to work:

```
<ObjectMapping type="Model" behaviorType="MIXED" tctype="Item">
    <AttributeMappings>
        <AttributeMapping name="Description" tcattr="object_desc"/>
        <RevisionAttributeMapping>
            <AttributeMapping name="Description" tcattr="object_desc"/>
        </RevisionAttributeMapping>
    </AttributeMappings>
</ObjectMapping>
```

Pass the following JSON string as an input to **exportCollection()**. The output returns the **tcContainerIdentifier** value in a similar output JSON, which is, a Teamcenter-assigned unique identifier of the business object representing the model. The **tcContainerIdentifier** can subsequently be used to import or update the model in Teamcenter.

```
{
  "AllContainers": {
    "toolIdentifier": "SAMPLE",
    "Containers": [
      {
        "containerIdentifier": "GolfCart_Identifier",
        "containerName": "GolfCart",
        "containerType": "Model",
        "tcContainerName": "GolfCart",
        "isTopContainer": true,
        "action": "add",
        "attributes": [
          {
            "attrName": "Description",
            "tcAttrName": "",
            "value": [
              "This is a GolfCart."
            ]
          }
        ],
        "revAttributes": [
          {
            "attrName": "Description",
            "tcAttrName": "object_desc",
```

```
            "value": [
              "This is a cart GolfCart, revision A."
            ]
          }
        ],
        "Components": [],
        "userSpecificInputs": [
          {
            "inputKey": "CHECKOUT",
            "values": [
              "true"
            ]
          }
        ]
      }
    ]
  }
}
```

Use Case 2 – Create a business object in Teamcenter to store a tool-authored model file

The following entry in the integration definition file is required for this to work. Note that **RevisionAttributeMapping** is not required for a non-Item Teamcenter object.

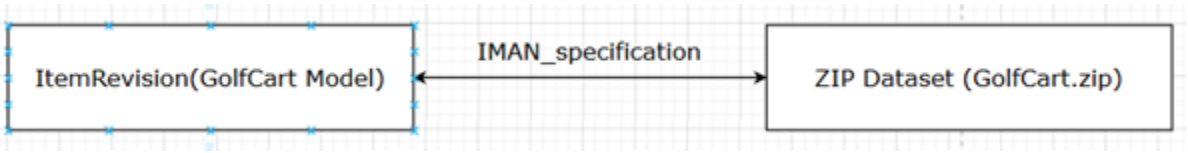
```
<ObjectMapping type=" Design" behaviorType="MIXED" tctype=" Zip">
    <AttributeMappings>
        <AttributeMapping name="Description" tcattr="object_desc"/>
    </AttributeMappings>
</ObjectMapping>
```

Pass the following JSON string as an input to **exportCollection()**. The output returns the **tcContainerIdentifier** value of the dataset under which the file is to be uploaded. The output JSON also returns a **writeTicket** that can be used to upload files to Teamcenter using FMS SOAs.

```
{
  "AllContainers": {
    "toolIdentifier": "SAMPLE",
    "Containers": [
      {
        "containerIdentifier": "GolfCart_Dataset_identifier",
        "containerName": "GolfCart_Dataset",
        "containerType": "DesignData",
        "tcContainerIdentifier": "",
        "tcContainerType": "Zip",
        "tcContainerName": "GolfCart_Dataset",
        "isTopContainer": false,
```

```
        "action": "add",
        "datasetFileInfos": [
          {
            "fileName": "GolfCartModelFile",
            "fileExtension": "zip",
            "filePath": "D:\\GolfCart",
            "namedReferenceName": "ZIPFILE",
            "allowReplace": true,
            "isText": false
          }
        ]
      }
    ]
  }
}
```

Use Case 3 - Associate two business objects in Teamcenter through a GRM relation



Use case 1 shows how to create a business object representing a GolfCart Model. Use Case 2 shows how to create a business object that stores a GolfCart Model File. This use case shows how to associate these two objects. The type of association that is BVR, GRM, GBVR or REF, in this case GRM is decided by the integration definition file. The GRM relation type to be created between these two objects is also decided by **reltype** in **BHMElement**the integration definition file.

The following entry in the integration definition file is required for this to work. Attribute mapping is omitted from the following integration definition file snippet to keep it concise.

```
<ObjectMapping type="Model" behaviorType="MIXED" tctype="Item">
        <BHMElement type="Design" tctype="Zip" behaviorType="GRM"
reltype="IMAN_specification" isPrimary=”false” isPrimaryAnchor=”false”
isSecondaryAnchor=”false”> </BHMElement>
</ObjectMapping>
```

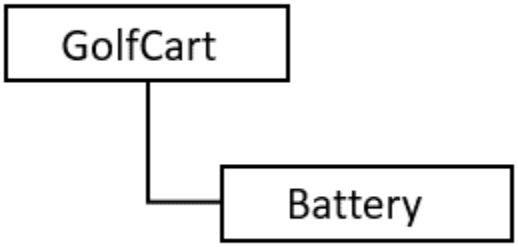
Pass the following JSON string as an input to **exportCollection()**. The output JSON contains the **tcComponentIdentifier** value of the newly created GRM relation along with the **tcContainerIdentifier** values of the container objects. Note that **xHAHH_KPNqBkC** that is the value of **tcContainerIdentifier** of an already created GolfCart_Dataset is populated in the **Components** section. Container information of an already created object is not required in the input JSON.

```
{
```

```
  "AllContainers": {
    "toolIdentifier": "SAMPLE",
    "Containers": [
      {
        "containerIdentifier": "GolfCart_Identifier",
        "containerName": "GolfCart",
        "containerType": "Model",
        "tcContainerName": "GolfCart",
        "isTopContainer": true,
        "action": "add",
        "attributes": [],
        "revAttributes": [],
        "Components": [
          {
            "componentName": "",
            "componentIdentifier": "GolfcartDesignData",
            "componentType": "Design",
            "tcComponentName": "",
            "tcComponentIdentifier": "",
            "tcComponentType": "",
            "tcComponentCopyStableID": "",
            "containerIdentifier": "GolfCart_Dataset_identifier",
            "tcContainerIdentifier": "","xHAHH_KPNqBkC",
            "action": "add",
            "attributes": []
          }
        ]
      }
    ]
  }
}
```


Use Case 4 – Associate two business objects in Teamcenter through a BVR relation

Use case 1 shows how to create a business object representing a GolfCart Model. Similarly, an object representing Battery model is created. This use case shows how to associate these two objects through a parent-child relation. The type of association that is BVR, GRM, GBVR or REF, in this case BVR is decided by the **behaviorType** in **BHMElement** in the integration definition file.



The following entry in the integration definition file is required:

```
<ObjectMapping type="Model" behaviorType="MIXED" tctype="Item">
    <BHMElement type="Model" tctype="Item"  behaviorType="BVR">
            <AttributeMappings>
                <AttributeMapping name=" occurrence_name"
tcattr="bl_occurrence_name"/>
                <AttributeMapping name="cifComponentIdentifier"
tcattr="Cif0ToolSpecificIntInfo::cif0ToolProperties"/>
                <RevisionAttributeMapping/>
            </AttributeMappings>
    </BHMElement>
</ObjectMapping>
```

Pass the following JSON string as an input to **exportCollection()**. The output JSON contains the **tcComponentIdentifier** value of the newly created occurrence along with the **tcContainerIdentifier** values of the objects.

```
{
  "AllContainers": {
    "toolIdentifier": "SAMPLE",
    "Containers": [
      {
        "containerIdentifier": "GolfCart_Identifier",
        "containerName": "GolfCart",
        "containerType": "Model",
        "tcContainerName": "GolfCart",
        "isTopContainer": true,
        "action": "add",
        "attributes": [],
        "revAttributes": [],
        "Components": [
          {
            "componentName": "",
```

```
            "componentIdentifier": "Battery",
            "componentType": "Model",
            "tcComponentName": "",
            "tcComponentIdentifier": "",
            "tcComponentType": "",
            "tcComponentCopyStableID": "",
            "containerIdentifier": "Battery_Identifier",
            "tcContainerIdentifier": "",
            "action": "add",
            "attributes": [
              {
                "attrName": "occurrence_name",
                "tcAttrName": "bl_occurrence_name",
                "value": [
                  "Battery"
                ]
              }
            ]
          }
        ]
      },
      {
        "containerIdentifier": "Battery_Identifier",
        "containerName": "Battery",
        "containerType": "Model",
        "tcContainerName": "Battery",
        "isTopContainer": false,
        "action": "add",
        "attributes": [],
        "revAttributes": [],
        "Components": []
      }
    ]
  }
}
```


Use Case 5 – Associate two port objects in Teamcenter through a connection

This is a special case in which a port object is associated with the parent container object through a **GBVR** association. The PowerSupply connection object is associated with its parent GolfCart object with a **BVR** association.

The following entry in the integration definition file is required. Attribute mapping is omitted in the integration definition file snippet to keep it concise.

```
<ObjectMapping type="Model" behaviorType="MIXED" tctype="Item">
    <BHMElement type="Model" tctype="Item"  behaviorType="BVR"></
```

```
BHMElement>
    <BHMElement type="Port" tctype="GeneralDesignElement"
behaviorType="GBVR"></BHMElement>
</ObjectMapping>

<ObjectMapping type="Connection" behaviorType="MIXED"
tctype="Connection">
</ObjectMapping>

<ObjectMapping type="Port" behaviorType="MIXED"
tctype="GeneralDesignElement">
</ObjectMapping>
```

Pass the following JSON string as an input to **exportCollection()**:

```
{
  "AllContainers": {
    "toolIdentifier": "SAMPLE",
    "Containers": [
      {
        "containerIdentifier": "GolfCart_Identifier",
        "containerName": "GolfCart",
        "containerType": "Model",
        "tcContainerName": "GolfCart",
        "isTopContainer": true,
        "action": "add",
        "attributes": [
          {
            "attrName": "Item_Description",
            "tcAttrName": "",
            "value": [
              "This is a cart GolfCart."
            ]
          }
        ],
        "revAttributes": [
          {
            "attrName": "Revision_Description",
            "tcAttrName": "object_desc",
            "value": [
              "This is a cart GolfCart, revision A."
            ]
          }
        ],
        "Components": [
          {
            "componentName": "",
            "componentIdentifier": "Battery",
```

```
    "componentType": "Model",
    "tcComponentName": "",
    "tcComponentIdentifier": "",
    "tcComponentType": "",
    "tcComponentCopyStableID": "",
    "containerIdentifier": "Battery_Identifier",
    "tcContainerIdentifier": "",
    "action": "add",
    "attributes": [
      {
        "attrName": "occurrence_name",
        "tcAttrName": "bl_occurrence_name",
        "value": [
          "Battery"
        ]
      }
    ]
  },
  {
    "componentName": "",
    "componentIdentifier": "MotorDrive",
    "componentType": "Model",
    "tcComponentName": "",
    "tcComponentIdentifier": "",
    "tcComponentType": "",
    "tcComponentCopyStableID": "",
    "containerIdentifier": "MotorDrive_Identifier",
    "tcContainerIdentifier": "",
    "action": "add",
    "attributes": [
      {
        "attrName": "occurrence_name",
        "tcAttrName": "bl_occurrence_name",
        "value": [
          "MotorDrive"
        ]
      }
    ]
  },
  {
    "componentName": "",
    "componentIdentifier": "Connection1",
    "componentType": "Connection",
    "tcComponentName": "",
    "tcComponentIdentifier": "",
    "tcComponentType": "",
    "tcComponentCopyStableID": "",
    "containerIdentifier": "PowerSupply",
    "tcContainerIdentifier": "",
```

```
        "owningContainerIdentifier": "",
        "tcOwningContainerIdentifier": "",
        "action": "add",
        "attributes": [
          {
            "attrName": "occurrence_name",
            "tcAttrName": "bl_occurrence_name",
            "value": [
              "Connection1"
            ]
          }
        ],
        "ComponentLinks": [
          {
            "componentIdentifier": "OutPort",
            "tcComponentIdentifier": "",
            "owningComponentIdentifier": "Battery",
            "tcOwningComponentIdentifier": "",
            "action": "add"
          },
          {
            "componentIdentifier": "InPort",
            "tcComponentIdentifier": "",
            "owningComponentIdentifier": "MotorDrive",
            "tcOwningComponentIdentifier": "",
            "action": "add"
          }
        ]
      }
    ],
    "userSpecificInputs": [
      {
        "inputKey": "CHECKOUT",
        "values": [
          "true"
        ]
      }
    ]
  },
  {
    "containerIdentifier": "Battery_Identifier",
    "containerName": "Battery",
    "containerType": "Model",
    "tcContainerName": "Battery",
    "isTopContainer": false,
    "action": "add",
    "attributes": [
      {
        "attrName": "Item_Description",
```

```
        "tcAttrName": "",
        "value": [
          "This is a cart battery."
        ]
      }
    ],
    "revAttributes": [
      {
        "attrName": "Revision_Description",
        "tcAttrName": "object_desc",
        "value": [
          "This is a cart battery, revision A."
        ]
      }
    ],
    "Components": [
      {
        "componentName": "",
        "componentIdentifier": "OutPort",
        "componentType": "Port",
        "tcComponentName": "",
        "tcComponentIdentifier": "",
        "tcComponentType": "",
        "tcComponentCopyStableID": "",
        "containerIdentifier": "TestPort",
        "tcContainerIdentifier": "",
        "owningContainerIdentifier": "",
        "tcOwningContainerIdentifier": "",
        "action": "add",
        "attributes": [
          {
            "attrName": "occurrence_name",
            "tcAttrName": "bl_occurrence_name",
            "value": [
              "OutPort"
            ]
          }
        ]
      }
    ]
  },
  {
    "containerIdentifier": "MotorDrive_Identifier",
    "containerName": "MotorDrive",
    "containerType": "Model",
    "tcContainerName": "MotorDrive",
    "isTopContainer": false,
    "action": "add",
    "attributes": [
```

```
      {
        "attrName": "Item_Description",
        "tcAttrName": "",
        "value": [
          "This is a cart MotorDrive."
        ]
      }
    ],
    "revAttributes": [
      {
        "attrName": "Revision_Description",
        "tcAttrName": "object_desc",
        "value": [
          "This is a cart MotorDrive, revision A."
        ]
      }
    ],
    "Components": [
      {
        "componentName": "",
        "componentIdentifier": "InPort",
        "componentType": "Port",
        "tcComponentName": "",
        "tcComponentIdentifier": "",
        "tcComponentType": "",
        "tcComponentCopyStableID": "",
        "containerIdentifier": "TestPort",
        "tcContainerIdentifier": "",
        "owningContainerIdentifier": "",
        "tcOwningContainerIdentifier": "",
        "action": "add",
        "attributes": [
          {
            "attrName": "occurrence_name",
            "tcAttrName": "bl_occurrence_name",
            "value": [
              "InPort"
            ]
          }
        ]
      }
    ]
  },
  {
    "containerIdentifier": "PowerSupply",
    "containerName": "PowerSupply",
    "containerType": "Connection",
    "tcContainerIdentifier": "",
    "tcContainerType": "",
```

```
        "tcContainerName": "",
        "isTopContainer": false,
        "action": "add",
        "tcFolderUid": "",
        "tcLastModifiedDate": "",
        "attributes": [
          {
            "attrName": "name",
            "tcAttrName": "object_name",
            "value": [
              "PowerSupply"
            ]
          }
        ],
        "revAttributes": [],
        "datasetFileInfos": [],
        "Components": []
      },
      {
        "containerIdentifier": "TestPort",
        "containerName": "Test Port",
        "containerType": "Port",
        "tcContainerIdentifier": "",
        "tcContainerType": "",
        "tcContainerName": "",
        "isTopContainer": false,
        "action": "add",
        "tcFolderUid": "",
        "tcLastModifiedDate": "",
        "attributes": [
          {
            "attrName": "name",
            "tcAttrName": "object_name",
            "value": [
              "TestPort"
            ]
          }
        ]
      }
    ]
  }
}
```

updateCollection()

```
DataManagementImpl::CollectionOutput updateCollection ( const
std::string inputData )
```

updateCollection() is used to:

- Update properties on the models
- Add or remove the association of models with other tool-authored artifacts
- Update the properties of association between models
- Update the model, submodel structure hierarchy

Use case 1 – Update properties on a model

The following JSON input string is sent to **updateCollection** to modify the description of the model. In the following example, **"tcContainerIdentifier" - "wJa5Z1Ghr8jZAC"** is the Teamcenter UID of the already existing item revision object.

```
DataManagementImpl::CollectionOutput updateCollection ( const
std::string inputData )
{
  "AllContainers": {
    "toolIdentifier": "SAMPLE",
    "Containers": [
      {
        "containerIdentifier": "Battery_Identifier",
        "containerName": "Battery",
        "containerType": "Assembly",
        "tcContainerIdentifier": " wJa5Z1Ghr8jZAC ",
        "action": "update",
        "revAttributes": [
          {
            "attrName": "Revision_Description",
            "tcAttrName": "object_desc",
            "value": [ "This is a updated description" ]
          }
        ]
      }
    ]
  }
}
```

importCollection()

```
DataManagementImpl::CollectionOutput importCollection ( const
std::string inputData )
```

Use case 1 – Import model from Teamcenter

The following JSON input string is sent to **importCollection**. It returns all the associated models and their properties as configured in the integration definition file. It also returns readTickets that then can be used to download files from Teamcenter using FMS SOAs. In the following example, **"tcContainerIdentifier" - "wJa5Z1Ghr8jZAC"** is the Teamcenter UID of the already existing item revision object.

```
DataManagementImpl::CollectionOutput updateCollection ( const
std::string inputData )
{
  "AllContainers": {
    "toolIdentifier": "SAMPLE",
    "Containers": [
      {
        "containerIdentifier": "Battery_Identifier",
        "containerName": "Battery",
        "containerType": "Assembly",
        "tcContainerIdentifier": " wJa5Z1Ghr8jZAC ",
      }
    ]
  }
}
```

8. Using behavior modeling APIs and extensions

Overview of using behavior modeling APIs and extensions

Teamcenter provides Java APIs and extensions to help you customize behavior modeling operations.

To use these APIs and extensions, you must install the following components:

- Behavior modeling common client
- Behavior modeling tool connector, for example, MATLAB connector.

> Note:
>
> Siemens recommends that you use the **Common Integration Framework APIs** for your integration operations instead of the behavior modeling APIs.

Customizing Teamcenter MBSE Integration Gateway operations by using APIs

Overview of using behavior modeling APIs

Teamcenter provides Java APIs for the following behavior modeling operations:

- Open
- Pre-save
- Save
- Export
- Login
- Logout
- Derby cache status
- FCC status

You can invoke these APIs from any Java program or from the behavior modeling tool-specific script.

To use the APIs:

- Implement an external Java program or tool-specific script using the published APIs.
- Add all the .jar files in the *Teamcenter_ROOT***bhm** directory to the Java build path libraries.
- If you are using MATLAB, ensure that you add the following entries to the PATH variable:
  - *MATLAB_ROOT***bin\win64**
  - *MATLAB_ROOT***sys\jxbrowser\win64\lib**
- Add **xercesImpl.jar** to the Java build path libraries.
- The class **OperationsAPIs** in the **com.teamcenter.behaviormodeling.commonclient.operations.api** package contains the behavior modeling integration APIs.
- Before using the APIs, you must establish a Teamcenter session using the login API.
- Use the utility functions to generate the xml input strings for the Save and the Open APIs. This utility is available in the **BHMOperationData** and the **BHMDesign** classes located in the **com.teamcenter.behaviormodeling.common.xml** package.

  You can use the Pre-save API to generate input files for the Save API. The xml file that the Pre-save API generates does not contain information about additional data. To save additional data, you must explicitly update the xml file.

  Example:

  The Save API requires an XML string as input. This string contains information about the models to be saved. This XML string can be based on the **BHMOperationsSchema.xsd** file as follows:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<BHMOperations xmlns="http://www.plmxml.org/Schemas/bhm">
 <operationData modelPath="D:\HSNModels">
  <modelInfo>
   <Model createdDate="Mon Sep 02 12:35:30 2013"
    tcType="Bhm0BehaviorModlRevision"
    instanceType="ModelReference" tcItemName="SingleModel"
    tcrevId="H"
    tcitemId="000205" tcuid="" modelIdentifier="SingleModel"
    modelName="SingleModel">
    <Attributes>
     <Key name="object_desc">
      <Value>My Desc</Value>
     </Key>
```

```
        <RevisionAttributes />
       </Attributes>
      </Model>
     </modelInfo>
     <options>
      <saveOperationOptions folderUID="g9UJCdVLopK8AC"
       checkIn="false" addAdditionalData="false" />
     </options>
    </operationData>
   </BHMOperations
```

Operations API

This API is an overloaded constructor that accepts a Teamcenter connection object. The Teamcenter connection object passed through this constructor is used for all Behavior Modeling operations. If you are using this API, you need not use the bhmLogin API.

```
public OperationsAPIs (Connection connection, String serverURL, String
toolType)
```

Inputs

connection

Specifies the Teamcenter connection object.

serverURL

Specifies the URL to the Teamcenter server.

toolType

Specifies the behavior modeling tool, for example, MATLAB.

bhmLogin API

You must start a Teamcenter session, using the bhmLogin API before using the Open and Save APIs.

```
BHMOperationOutput bhmLogin (String userName, String password,
 String group, String toolType)
```

Inputs

username

Specifies the Teamcenter user ID.

password

Specifies the password of the Teamcenter user.

group

(Optional) Specifies the group associated with the user.

toolType

Specifies the behavior modeling tool, for example, MATLAB.

Example

```
BHMOperationOutput bhmLogin (“dba”, “dba”, “”, “MATLAB”);
```

bhmOpenOperation API

Opens behavior models from Teamcenter.

```
List<BHMOperationOutput> bhmOpenOperation(List<String> dataOpenInputXML,
 String toolType)
```

Inputs

dataOpenInputXML

Specifies a list of input strings that are based on the **BHMOperationsSchema.xsd** file.

You can generate the xml string, using the JAXB utility available with the **BHMOperationData** and **BHMDesign** classes in the **com.teamcenter.behaviormodeling.common.xml** package.

toolType

Specifies the behavior modeling tool, for example, MATLAB.

In your Java code, you must specify the following:

- In the **BHMOperations** element, specify the values for **rootModelName** and **rootModelIdentifier**.
- In the **Model** element, specify the values for **modelName**, **modelIdentifier**, and **tcuid**.
- In the **openOperationOptions** element, specify the values of the following variables:
  - **checkout**

    To check out the root model, set the value to **true**.

  - **checkOutAll**

    To check out all submodels of the root model, set the value to **true**.

  - **addAdditionalData**

To download additional data, set the value to **true**.

Outputs

The output is a list of **BHMOperationOutput** objects.

The **BHMOperationOutput** class is available in the **com.teamcenter.behaviormodeling.common** package.

Each **BHMOperationOutput** object in the list corresponds to the input XML strings sent through the input parameter **dataOpenInputXML**.

BHMOperationOutput variables	Description
OperationStatus	Returns one of the following values: **OPERATION_FAILED** **OPERATION_WARNING** **OPERATION_SUCCESSFUL** **OPERATION_SUCCESSFUL_WITH_INFO**
message	Specifies the cause of the API failure.
inputXML	Specifies the input XML string based on **BHMOperationsSchema.xsd**.
outputXML	Specifies an XML string based on **BHMDesignSchema.xsd**.
rootModelName	Specifies the root model to open.
rootModelIdentifier	Specifies the identifier of the root model.
rootModelFullFilePath	Specifies the full path of the root model.

Example

Pass the following string to the **dataOpenInputXML** argument of the bhmOpenOperation API to load and check out a model.

To load multiple models, you must pass separate input XMLs to **dataOpenInputXML**.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<BHMOperations rootModelName="test" rootModelIdentifier="test"
xmlns="http://www.plmxml.org/Schemas/bhm">
  <modelInfo>
    <Model modelName="test" modelIdentifier="test"
tcuid="Q2TJlnST4BflXC" />
  </modelInfo>
  <options>
```

```
<openOperationOptions checkOut="true" checkOutAll="false"
 addAdditionalData="false" />
</options></BHMOperations>
```

bhmPreSaveOperation API

This API generates an XML string that can be used as the input XML string for the Save API.

```
List<BHMOperationOutput> bhmPreSaveOperation(List<String>
 lstRootModelFullFilePath, String toolType)
```

Inputs

lstRootModelFullFilePath

Specifies the full path of the root model.

toolType

Specifies the behavior modeling tool, for example, MATLAB.

Outputs

The output is a list of **BHMOperationOutput** objects.

The **BHMOperationOutput** class is available in the **com.teamcenter.behaviormodeling.common** package.

Each **BHMOperationOutput** object in the list corresponds to the input XML strings sent through the input parameter **lstRootModelFullFilePath**.

BHMOperationOutput variables	Description
OperationStatus	Returns one of the following values: **OPERATION_FAILED** **OPERATION_WARNING** **OPERATION_SUCCESSFUL** **OPERATION_SUCCESSFUL_WITH_INFO**
message	Specifies the cause of the API failure.
inputXML	Is not used.
outputXML	Specifies an XML string based on **BHMDesignSchema.xsd** that is used as input for the bhmSaveOperation API.
rootModelName	Specifies the root model.

BHMOperationOutput variables	Description
rootModelIdentifier	Specifies the identifier of the root model.
rootModelFullFilePath	Specifies the full path of the root model.

Example

To generate XML files for two root model files, you can define the paths as follows:

```
List<String> paths = new ArrayList<String>();
            paths.add("D:\\matlabModels\\myModel11.slx");
            paths.add("D:\\matlabModels\\myModel21.slx");
```

bhmSaveOperation API

Use the following API to save and check in models to Teamcenter.

```
List<BHMOperationOutput> bhmSaveOperation (List<String> dataSaveInputXML,
 String toolType)
```

Inputs

dataSaveInputXML

Specifies a list of input strings that are based on the **BHMOperationsSchema.xsd** file.

Do not specify files that do not exist or are internal data such as the ModelFolder organization file. If you specify such files, you may get an error.

You can generate the xml string using the JAXB utility available with the **BHMOperationData** and **BHMDesign** classes in the **com.teamcenter.behaviormodeling.common.xml** package.

toolType

Specifies the behavior modeling tool, for example, MATLAB.

In your Java code, you must specify the following:

- In the **BHMOperations** element, specify the values for **rootModelFolder**, **rootModelName**, and **rootModelIdentifier**.
- In the **Model** element:
  - Specify the values for **modelName** and **modelIdentifier**.
  - Specify the value of **tcuid** for an existing model.

Do not specify a value for **tcuid** in new models.

- In the **Attributes** element, specify the value of the **item_id** for items and revisions. If you do not specify the IDs, Teamcenter automatically generates them.

  Do not specify values for **tcrevId** and **tcitemId** as they are for internal use.

```
<Attributes>
  <Key name="item_id">
    <Value>001001</Value>
  </Key>
  <RevisionAttributes>
    <Key name=" item_id ">
      <Value>P</Value>
    </Key>
  </RevisionAttributes>
</Attributes>
```

  You can also specify the description of the model in the **Attributes** element as follows:

```
<Attributes>
  <Key name="object_desc">
    <Value>This is S1 Desc</Value>
  </Key>
</Attributes>
```

- Specify the submodel information in the **ModelElement** element.

  Specify the ID and revision of the submodel in the **Attributes** element of **ModelElement**.

  If you do not specify the ID and revision for the submodel, Teamcenter automatically generates them.

- In the **saveOperationOptions** element, specify the values of the following variables:

  - **folderUID**

    Saves the model to the Teamcenter folder specified by the folder UID.

  - **checkIn**

    To check in the root model, set the value to **true**.

  - **addAdditionalData**

    To attach additional data, set the value to **true**, and specify the additional data details in the **AdditionalData** element.

Outputs

The output is a list of **BHMOperationOutput** objects.

The **BHMOperationOutput** class is available in the **com.teamcenter.behaviormodeling.common** package.

Each **BHMOperationOutput** object in the list corresponds to the input XML strings sent through the input parameter **dataSaveInputXML**.

BHMOperationOutput variables	Description
OperationStatus	Returns one of the following values: **OPERATION_FAILED** **OPERATION_WARNING** **OPERATION_SUCCESSFUL** **OPERATION_SUCCESSFUL_WITH_INFO**
message	Specifies the cause of the API failure.
inputXML	Specifies the input XML string based on **BHMOperationsSchema.xsd**.
outputXML	Specifies an XML string based on **BHMDesignSchema.xsd**.
rootModelName	Specifies the root model to be saved.
rootModelIdentifier	Specifies the identifier of the root model.
rootModelFullFilePath	Specifies the full path of the root model.

Example

To save a new model to Teamcenter, you can pass the following string through the **dataSaveInputXML** list of the bhmSaveOperation API.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<BHMOperations xmlns="http://www.plmxml.org/Schemas/bhm"
  rootModelFolder="D:\Models\MatlabModels" rootModelName="MyModel"
  rootModelIdentifier="MyModel">
  <modelInfo>
    <Model toolId="" teamcenterVersion="1.1"
tcType="Bhm0BehaviorModlRevision"
      tcItemName="MyModel" tcrevId="" tcitemId="" tcuid=""
      modelIdentifier="MyModel"
      modelName="MyModel">
      <Attributes>
        <Key name="object_name">
```

```
            <Value>MyModel</Value>
          </Key>
          <Key name="object_desc">
            <Value></Value>
          </Key>
          <Key name="item_id">
            <Value>000041</Value>
          </Key>
          <RevisionAttributes>
            <Key name="item_revision_id">
              <Value>A</Value>
            </Key>
          </RevisionAttributes>
        </Attributes>
        <AdditionalData>
          <File description="" relationType="Bhm0AdditionalData"
            reservation="true" path="D:\Models\MatlabModels\AdditionalData"
            fileext="txt" fileName="ReviewComments.txt"
namedReferenceType="Text"
            tcDatasetType="Text" tcuid="" action="add" />
        </AdditionalData>
        <Contents>
          <ModelElements>
            <ModelElement parent="S1" tcType="Bhm0BehaviorModlRevision"
              instanceType="MATLAB:ModelReference" instanceUid="" tcuid=""
              modelIdentifier="MyChild" modelName="MyChild" name="Model">
              <Attributes>
                <Key name="object_name">
                  <Value>MyChild</Value>
                </Key>
                <Key name="object_desc">
                  <Value>This is MyChild Desc</Value>
                </Key>
                <Key name="item_id">
                  <Value>001003</Value>
                </Key>
                <RevisionAttributes>
                  <Key name="item_revision_id">
                    <Value>Q</Value>
                  </Key>
                </RevisionAttributes>
              </Attributes>
            </ModelElement>
          </ModelElements>
        </Contents>
      </Model>
    </modelInfo>
    <options>
      <saveOperationOptions folderUID="" checkIn="true"
```

```
        addAdditionalData="true" />
    </options>
</BHMOperations>
```

bhmSaveAsOperation API

Use the following API to create a copy of an existing model and save it to Teamcenter.

If the model has reusable blocks, the SaveAs operation creates new copies of the reusable blocks.

```
List<BHMOperationOutput> bhmSaveAsOperation (List<String>
dataSaveAsInputXMLs,
 String toolType)
```

Inputs

dataSaveInputXML

Specifies a list of input strings that are based on the **BHMOperationsSchema.xsd** file.

You can generate the xml string using the JAXB utility available with the **BHMOperationData** and **BHMDesign** classes in the **com.teamcenter.behaviormodeling.common.xml** package.

> Note:
>
> You can also generate the input XML string, using the bhmPreSaveOperation API.
>
> Provide the details of the model to be copied, such as the staging directory, and generate the XML file and update the output of this API.

toolType

Specifies the behavior modeling tool, for example, MATLAB.

In your Java code, you must specify the following:

- In the **BHMOperations** element, specify the values for **rootModelFolder**, **rootModelName**, and **rootModelIdentifier**.
- In the **Model** element:
  - Specify the values for **modelName** and **modelIdentifier**.
  - Specify the value of **tcuid**.
- In the **Attributes** element, specify the value of **item_id**, **item_revision_id**, and **object_name**. If you do not specify these IDs, Teamcenter automatically generates them.

```
<Attributes>
  <Key name="item_id">
    <Value>000300</Value>
  </Key>
  <RevisionAttributes>
    <Key name="object_name">
      <Value>M11SaveAs</Value>
    </Key>
    <Key name="item_revision_id">
      <Value>D</Value>
    </Key>
  </RevisionAttributes>
</Attributes>
```

- In the **saveOperationOptions** element, specify values for the following variables:
  - **folderUID**

    Saves the model to the Teamcenter folder specified by the folder UID.
  - **checkIn**

    Set the value to **false** as you cannot check in a new model.
  - **addAdditionalData**

    To copy additional data from the original model to the new model, set the value to **true**.

Outputs

The output is a list of **BHMOperationOutput** objects.

The **BHMOperationOutput** class is available in the **com.teamcenter.behaviormodeling.common** package.

Each **BHMOperationOutput** object in the list corresponds to the input XML strings sent through the input parameter **dataSaveInputXML**.

BHMOperationOutput variables	Description
OperationStatus	Returns one of the following values: **OPERATION_FAILED** **OPERATION_WARNING**

BHMOperationOutput variables	Description
	OPERATION_SUCCESSFUL **OPERATION_SUCCESSFUL_WITH_INFO**
message	Specifies the cause of the API failure.
inputXML	Specifies the input XML string based on **BHMOperationsSchema.xsd**.
outputXML	Specifies an XML string based on **BHMDesignSchema.xsd**.
rootModelName	Specifies the root model to be saved.
rootModelIdentifier	Specifies the identifier of the root model.
rootModelFullFilePath	Specifies the full path of the root model.

Example

To create a copy of a model and save it to Teamcenter, you can pass the following string through the **dataSaveInputXML** list of the bhmSaveOperation API.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<BHMOperations rootModelFolder="D:\staging_11\M11_000260_A\M112"
rootModelName="M11"
 rootModelIdentifier="M11"xmlns="http://www.plmxml.org/Schemas/bhm">
  <modelInfo applicationName="MATLAB">
    <Model modelName="M11" modelIdentifier="M11" tcuid="g_Zhgh35IvwBJA"
    tcitemId="000295" tcrevId="A" tcItemName="M11"
instanceType="ModelReference"
    tcType="Bhm0BehaviorModl" lastSavedTimeStamp="0"
    createdDate="Tue May 12 12:24:22 2015" isRoot="true"
    containerModel="true" objectType="Model">
      <Attributes>
        <Key name="item_id">
            <Value>000300</Value>
        </Key>
        <RevisionAttributes>
            <Key name="object_name">
                <Value>M11SaveAs</Value>
            </Key>
            <Key name="item_revision_id">
                <Value>D</Value>
            </Key>
        </RevisionAttributes>
      </Attributes>
      <Contents>
        <ModelElements/>
```

```
      </Contents>
    </Model>
  </modelInfo>
  <options>
    <saveOperationOptions folderUID="AWSdOLciIvwBJA" checkIn="false"
    addAdditionalData="true"/>
  </options>
</BHMOperations>
```

bhmOpenBlockLibrary API

Opens behavior library models from Teamcenter.

```
List<BHMOperationOutput> bhmOpenBlockLibrary(List<String>
dataOpenInputXML,
 String toolType)
```

Inputs

dataOpenInputXML

Specifies a list of input strings that are based on the **BHMOperationsSchema.xsd** file.

You can generate the xml string, using the JAXB utility available with the **BHMOperationData** and **BHMDesign** classes in the **com.teamcenter.behaviormodeling.common.xml** package.

toolType

Specifies the behavior modeling tool, for example, MATLAB.

In your Java code, you must specify the following:

- In the **BHMOperations** element, specify the values for **rootModelName** and **rootModelIdentifier**.
- In the **Model** element, specify the values for **modelName**, **modelIdentifier**, and **tcuid**.
- In the **openOperationOptions** element, specify the values of the following variables:
  - **checkout**

    To check out the root model, set the value to **true**.
  - **checkOutAll**

    To check out all submodels of the root model, set the value to **true**.
  - **addAdditionalData**

To download additional data, set the value to **true**.

Outputs

The output is a list of **BHMOperationOutput** objects.

The **BHMOperationOutput** class is available in the **com.teamcenter.behaviormodeling.common** package.

Each **BHMOperationOutput** object in the list corresponds to the input XML strings sent through the input parameter **dataOpenInputXML**.

BHMOperationOutput variables	Description
OperationStatus	Returns one of the following values: **OPERATION_FAILED** **OPERATION_WARNING** **OPERATION_SUCCESSFUL** **OPERATION_SUCCESSFUL_WITH_INFO**
message	Specifies the cause of the API failure.
inputXML	Specifies the input XML string based on **BHMOperationsSchema.xsd**.
outputXML	Specifies an XML string based on **BHMDesignSchema.xsd**.
rootModelName	Specifies the root model to open.
rootModelIdentifier	Specifies the identifier of the root model.
rootModelFullFilePath	Specifies the full path of the root model.

Example

Pass the following string to the **dataOpenInputXML** argument of the bhmOpenOperation API to load and check out a library model.

To load multiple library models, you must pass separate input XMLs to **dataOpenInputXML**.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<BHMOperations rootModelName="test" rootModelIdentifier="test"
xmlns="http://www.plmxml.org/Schemas/bhm">
  <modelInfo>
    <Model modelName="test" modelIdentifier="test"
tcuid="Q2TJlnST4BflXC" />
  </modelInfo>
  <options>
```

```
<openOperationOptions checkOut="true" checkOutAll="false"
 addAdditionalData="false" />
</options></BHMOperations>
```

bhmSaveBlockLibrary API

Use the following API to save and check in library models to Teamcenter.

```
List<BHMOperationOutput> bhmSaveBlockLibrary (List<String>
dataSaveInputXML,
 String toolType)
```

Inputs

dataSaveInputXML

Specifies a list of input strings that are based on the **BHMOperationsSchema.xsd** file.

You can generate the xml string using the JAXB utility available with the **BHMOperationData** and **BHMDesign** classes in the **com.teamcenter.behaviormodeling.common.xml** package.

toolType

Specifies the behavior modeling tool, for example, MATLAB.

In your Java code, you must specify the following:

- In the **BHMOperations** element, specify the values for **rootModelFolder**, **rootModelName**, and **rootModelIdentifier**.
- In the **Model** element:
  - Specify the values for **modelName** and **modelIdentifier**.
  - Specify the value of **tcuid** for an existing model.

    Do not specify the value of **tcuid** for new models.
- In the **Attributes** element, specify the value of **item_id** for items and revisions. If you do not specify values for these, Teamcenter automatically generates them.

  If you do not specify values for **tcrevId** and **tcitemId** as they are for internal use.

```
<Attributes>
  <Key name="item_id">
    <Value>001001</Value>
  </Key>
  <RevisionAttributes>
    <Key name=" item_id ">
```

```
        <Value>P</Value>
      </Key>
    </RevisionAttributes>
  </Attributes>
```

You can also specify a description for the model in the **Attributes** element as follows:

```
<Attributes>
  <Key name="object_desc">
    <Value>This is S1 Desc</Value>
  </Key>
</Attributes>
```

- Specify the submodel information in the **ModelElement** element.

  Specify the ID and revision of the submodel in the **Attributes** element of **ModelElement**.

  You need not specify the ID and revision for the submodel, Teamcenter automatically generates them.

- In the **saveOperationOptions** element, specify the values of the following variables:

  - **folderUID**

    Saves the model to the Teamcenter folder specified by the folder UID.

  - **checkIn**

    Checks in the root model when the value is set to **true**.

  - **addAdditionalData**

    Attaches additional data when the value is set to **true**, and specify the additional data details in the **AdditionalData** element.

Outputs

The output is a list of **BHMOperationOutput** objects.

The **BHMOperationOutput** class is available in the **com.teamcenter.behaviormodeling.common** package.

Each **BHMOperationOutput** object in the list corresponds to the input XML strings sent through the input parameter **dataSaveInputXML**.

BHMOperationOutput variables	Description
OperationStatus	Returns one of the following values: **OPERATION_FAILED** **OPERATION_WARNING** **OPERATION_SUCCESSFUL** **OPERATION_SUCCESSFUL_WITH_INFO**
message	Specifies the cause of the API failure.
inputXML	Specifies the input XML string based on **BHMOperationsSchema.xsd**.
outputXML	Specifies an XML string based on **BHMDesignSchema.xsd**.
rootModelName	Specifies the root model to be saved.
rootModelIdentifier	Specifies the identifier of the root model.
rootModelFullFilePath	Specifies the complete path of the root model.

Example

To save a new library model to Teamcenter, you can pass the following string through the **dataSaveInputXML** list of the bhmSaveBlockLibrary API.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<BHMOperations xmlns="http://www.plmxml.org/Schemas/bhm"
  rootModelFolder="D:\Models\MatlabModels" rootModelName="MyModel"
  rootModelIdentifier="MyModel">
  <modelInfo>
    <Model toolId="" teamcenterVersion="1.1"
tcType="Bhm0BehaviorModlRevision"
      tcItemName="MyModel" tcrevId="" tcitemId="" tcuid=""
      modelIdentifier="MyModel"
      modelName="MyModel">
      <Attributes>
        <Key name="object_name">
          <Value>MyModel</Value>
        </Key>
        <Key name="object_desc">
          <Value></Value>
        </Key>
        <Key name="item_id">
          <Value>000041</Value>
        </Key>
        <RevisionAttributes>
          <Key name="item_revision_id">
            <Value>A</Value>
```

```
          </Key>
        </RevisionAttributes>
      </Attributes>
      <AdditionalData>
        <File description="" relationType="Bhm0AdditionalData"
          reservation="true" path="D:\Models\MatlabModels\AdditionalData"
          fileext="txt" fileName="ReviewComments.txt"
namedReferenceType="Text"
          tcDatasetType="Text" tcuid="" action="add" />
      </AdditionalData>
      <Contents>
        <ModelElements>
          <ModelElement parent="S1" tcType="Bhm0BehaviorModlRevision"
            instanceType="MATLAB:ModelReference" instanceUid="" tcuid=""
            modelIdentifier="MyChild" modelName="MyChild" name="Model">
            <Attributes>
              <Key name="object_name">
                <Value>MyChild</Value>
              </Key>
              <Key name="object_desc">
                <Value>This is MyChild Desc</Value>
              </Key>
              <Key name="item_id">
                <Value>001003</Value>
              </Key>
              <RevisionAttributes>
                <Key name="item_revision_id">
                  <Value>Q</Value>
                </Key>
              </RevisionAttributes>
            </Attributes>
          </ModelElement>
        </ModelElements>
      </Contents>
    </Model>
  </modelInfo>
  <options>
    <saveOperationOptions folderUID="" checkIn="true"
      addAdditionalData="true" />
  </options>
</BHMOperations>
```

bhmExportOperation API

Downloads behavior models, related datasets, additional data, and referenced models from Teamcenter to the specified folder. This API also restores the original folder structure and unzips the additional data that was zipped during the save operation.

```
List<BHMOperationOutput> bhmExportOperation(List<String>
dataExportInputXMLs)
```

Input

dataExportInputXMLs

Specifies a list of xml input strings that are based on the **BHMOperationsSchema.xsd** file.

You can generate the xml string, using the JAXB utility available with the **BHMOperationData** and **BHMDesign** classes in the **com.teamcenter.behaviormodeling.common.xml** package.

In your Java code, you must specify the following:

- In the **BHMOperations** element, specify the path of the folder where the model is to be exported in the **rootModelFolder** variable.
- In the **modelInfo** element, specify values for the following variables: **applicationName** and **tcuid**.
  - **applicationName**

    Specifies the name of the behavior modeling application, for example, **MATLAB**.
  - **tcuid**

    Specifies the Teamcenter ID of the model.
- In the **exportOperationOptions** element, specify the values of the following variable:
  - **getAdditionalData**

    To download additional data, set the value to **true**.

Output

The output is a list of **BHMOperationOutput** objects.

The **BHMOperationOutput** class is available in the **com.teamcenter.behaviormodeling.common** package.

Each **BHMOperationOutput** object in the list corresponds to the input XML strings sent through the input parameter **dataOpenInputXML**.

BHMOperationOutput variables	Description
OperationStatus	Returns one of the following values:

BHMOperationOutput variables	Description
	OPERATION_FAILED **OPERATION_WARNING** **OPERATION_SUCCESSFUL** **OPERATION_SUCCESSFUL_WITH_INFO**
message	Specifies the cause of the API failure.
inputXML	Specifies the input XML string based on **BHMOperationsSchema.xsd**.
outputXML	Specifies an XML string based on **BHMDesignSchema.xsd**.
rootModelName	Specifies the root model to export.
rootModelIdentifier	Specifies the identifier of the root model to be exported.

Example

Pass the following string to the **dataExportInputXML** argument of the bhmExportOperation API to export a model.

To load multiple models, you must pass separate input XMLs to **dataOpenInputXML**.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<BHMOperations xmlns="http://www.plmxml.org/Schemas/bhm
rootModelFolder="test">
  <modelInfo applicationName="MATLAB">
    <Model tcuid="Q2TJlnST4BflXC" />
  </modelInfo>
  <options>
    <openOperationOptions checkOut="true" checkOutAll="false"
     addAdditionalData="false" />
    </options></BHMOperations>
```

bhm0GetModelOrgData API

This is a server API that returns the model organization data that is saved in the model object in Teamcenter. The model organization data consists of a model folder name, relative paths of all the submodel files, and model-configured data. This API is available for the Bhm0BehaviorModelRevision and Bhm0CompRepositoryRevision objects and all inherited business objects.

The model organization information is returned for the model object on which the API is invoked. To obtain the organization information for all the submodels, the API should be invoked recursively on all the submodels.

```
int bhm0GetModelOrgData ( behaviormodeling::BhmModelOrgDataS
*modelOrgData )
```

Inputs

BhmModelOrgDataS

Specifies a reference to an empty **BhmModelOrgDataS** structure object. The API returns model organization data in this input structure object.

Outputs

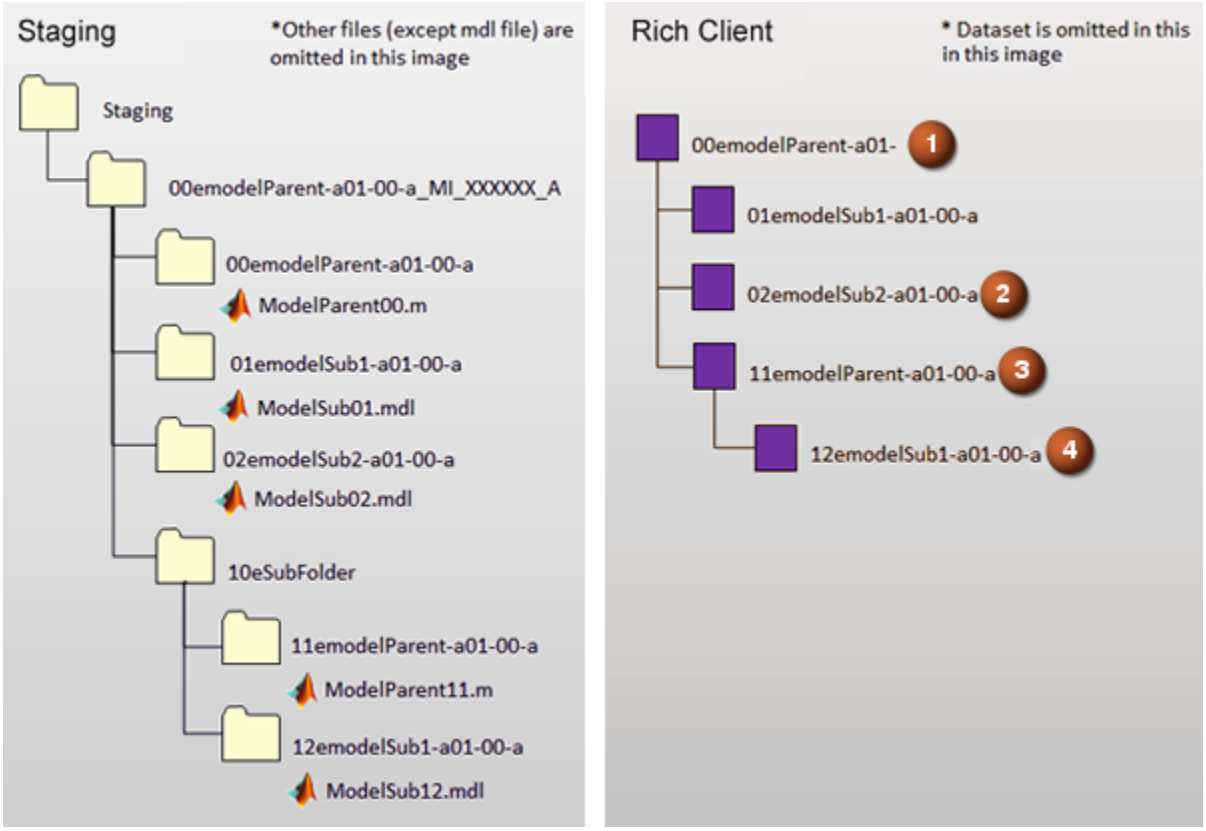
The API outputs the model organization data in the **BhmModelOrgDataS** structure object.

The **BhmModelOrgDataS** structure object returns the following:

Output object	Description
modelName	Specifies the name of the model.
modelFolderPath	Specifies the model path relative to the model folder.
configFilesFolderPath	Specifies the paths of the configuration data files relative to the model, including the name of the file.
mapSubModelFilesFolderPath	Specifies the submodel name and its path relative to the parent model excluding the submodel folder name.

Example

Assume that your model organization appears as follows in the staging directory and the rich client.



On using the API, you obtain the following output:

Output object	Object name
modelName	*ModelParent00*
modelFolderPath	*00emodelParent-a01-00-a*
configFilesFolderPath	*./ const / SomeDataModel.mat* *./ simdata /SimDataSpec.doc* *./pic/SimResult.jpg* *./extern/ ExternDataModel.m*
mapSubModelFilesFolderPath	*ModelSub01 - ../* *ModelSub02 - ../* *ModelParent11 - ../10eSubFolder*
modelName	*ModelSub02*
modelFolderPath	*02emodelSub2-a01-00-a*

Output object	Object name
configFilesFolderPath	*./ const / SomeDataModel.mat*
	./ simdata /SimDataSpec.doc
	./pic/SimResult.jpg
	./extern/ ExternDataModel.m
mapSubModelFilesFolderPath	
modelName	*ModelParent11*
modelFolderPath	*11emodelParent-a01-00-a*
configFilesFolderPath	*./ const / SomeDataModel.mat*
	./ simdata /SimDataSpec.doc
	./pic/SimResult.jpg
	./extern/ExternDataModel.m
mapSubModelFilesFolderPath	*ModelSub12 - ../*
modelName	*ModelSub12*
modelFolderPath	*12emodelSub1-a01-00-a*
configFilesFolderPath	*./ const / SomeDataModel.mat*
	./ simdata /SimDataSpec.doc
	./pic/SimResult.jpg
	./extern/ExternDataModel.m
mapSubModelFilesFolderPath	

bhmReviseOperation API

Use the following API to revise models that are managed in Teamcenter.

```
List<BHMOperationOutput> bhmReviseOperation (List<String>
dataReviseInputXML,
 String toolType)
```

Inputs

dataReviseInputXML

Specifies a list of input strings that are based on the **BHMOperationsSchema.xsd** file.

You can generate the xml string, using the JAXB utility available with the **BHMOperationData** and **BHMDesign** classes in the **com.teamcenter.behaviormodeling.common.xml** package.

toolType

Specifies the behavior modeling tool, for example, MATLAB.

In your Java code, you must specify the following:

- In the **BHMOperations** element, specify the values for **rootModelName** and **rootModelIdentifier**.
- In the **Model** element, specify the values for **modelName**, **modelIdentifier**, and **tcuid**.
- In the **reviseOperationOptions** element, specify the values of the following variables:
  - **revisionID**

    Specifies the revision ID to be set when revising the behavior model. If this value is empty, the revision ID is automatically assigned. This value must contain alphanumeric characters and must not contain the _ (underscore) character or the . (period) character.

  - **carryForwardAdditionalData**

    Specifies whether additional data must be revised and carried forward to the revised model. The value of this variable is a boolean parameter. If the value if **true**, additional data is carried forward. If not, the revised model will not have any additional data.

Output

The output is a list of **BHMOperationOutput** objects.

The **BHMOperationOutput** class is available in the **com.teamcenter.behaviormodeling.common** package.

Each **BHMOperationOutput** object in the list corresponds to the input XML strings sent through the input parameter **dataReviseInputXML**.

BHMOperationOutput variables	Description
OperationStatus	Returns one of the following values: **OPERATION_FAILED** **OPERATION_WARNING**

BHMOperationOutput variables	Description
	OPERATION_SUCCESSFUL **OPERATION_SUCCESSFUL_WITH_INFO**
message	Specifies the cause of the API failure.
inputXML	Specifies the input XML string based on **BHMOperationsSchema.xsd**.
outputXML	Specifies an XML string based on **BHMDesignSchema.xsd**.
rootModelName	Specifies the root model to be saved.
rootModelIdentifier	Specifies the identifier of the root model.
rootModelFullFilePath	Specifies the full path of the root model.

Example

To revise a model, you can pass the following string through the **dataReviseInputXML** list of the **bhmReviseOperation** API.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<BHMOperations rootModelName="test" rootModelIdentifier="test"
   xmlns="http://www.plmxml.org/Schemas/bhm">
  <modelInfo>
    <Model modelName="test" modelIdentifier="test"
     tcuid="Q2TJlnST4BflXC" />
  </modelInfo>
  <options>
    <reviseOperationOptions revisionID="D"
    carryForwardAdditionalData="false" />
  </options>
</BHMOperations>
```

bhmCancelCheckOutOperation API

Use the following API to cancel the checkout of a model saved in Teamcenter.

```
List<BHMOperationOutput> bhmCancelCheckOutOperation
(List<String> dataCancelCheckOutInputXML,  String toolType)
```

Inputs

dataCancelCheckOutInputXML

Specifies a list of input strings with information about the models whose checkout must be canceled. These strings are based on the **BHMOperationsSchema.xsd** file.

You can generate the xml string, using the JAXB utility available with the **BHMOperationData** and **BHMDesign** classes in the **com.teamcenter.behaviormodeling.common.xml** package.

toolType

Specifies the behavior modeling tool, for example, MATLAB.

In your Java code, you must specify the following:

- In the **BHMOperations** element, specify the values for **rootModelFolder** and **rootModelIdentifier**.
- In the **Model** element:
  - Specify the values for **modelName** and **modelIdentifier**.
  - Specify the value of **tcuid** for an existing model.
- In the **ReservationOperationOptions** element, specify the value for **BackUpExistingModel**. This value backs up the model before the checkout is canceled.

Output

The output is a list of **BHMOperationOutput** objects.

The **BHMOperationOutput** class is available in the **com.teamcenter.behaviormodeling.common** package.

Each **BHMOperationOutput** object in the list corresponds to the input XML strings sent through the input parameter **dataCancelCheckOutInputXML**.

BHMOperationOutput variables	Description
OperationStatus	Returns one of the following values: **OPERATION_FAILED** **OPERATION_WARNING** **OPERATION_SUCCESSFUL** **OPERATION_SUCCESSFUL_WITH_INFO**
message	Specifies the cause of the API failure.
inputXML	Specifies the input XML string based on **BHMOperationsSchema.xsd**.
rootModelName	Specifies the root model to be saved.
rootModelIdentifier	Specifies the identifier of the root model.

BHMOperationOutput variables	Description
rootModelFullFilePath	Specifies the full path of the root model.
backUpDirectory	Specifies the directory to back up the model before canceling the checkout.

Example

To cancel a checkout, you can pass the following string through the **dataCancelCheckOutInputXML** list of the **bhmCancelCheckOutOperation** API.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<BHMOperations rootModelFolder="D:\staging_11\M11_000260_A\M112"
rootModelIdentifier="M11"
xmlns="http://www.plmxml.org/Schemas/bhm">
  <modelInfo applicationName="MATLAB">
    <Model modelName="M11" modelIdentifier="M11" tcuid="g_Zhgh35IvwBJA">
      <Attributes/>
      <Contents>
        <ModelElements/>
      </Contents>
    </Model>
  </modelInfo>
  <options>
    <reservationOperationOptions backUpExistingModel="true/>
  </options>
</BHMOperations>
```

bhmLogout API

Use this API to end the Teamcenter session from the behavior modeling integration.

```
BHMOperationOutput bhmLogout()
```

getStateOfCurrentModel API

Provides information about the model that is cached in the derby database.

When modeling tool artifacts are saved in Teamcenter, model dependencies and some of the metadata, such as attributes, are saved as properties of Teamcenter objects. When the model is downloaded to a staging directory, the modeling tool refers to only the files associated with the model objects. When these models are saved back to Teamcenter, the information about which Teamcenter objects these model files are associated with is cached in a derby database on the client machine. The derby database also keeps track of the checkout status of the model object.

This API provides the model information (TcUID) and the checkout status and is available in the **com.teamcenter.behaviormodeling.commonclient.operations.api** package.

```
BHMObjectStatusOutput getStateOfCurrentModel( String modelIdentifier,
String modelFileFullPath, String strToolType)
```

Inputs

modelIdentifier

Specifies the model identifier.

modelFileFullPath

Specifies the full path of the model

strToolType

Specifies the name of the modeling tool, for example, MATLAB.

Outputs

The output is a list of **BHMObjectStatusOutput** objects.

The **BHMObjectStatusOutput** class is available in the **com.teamcenter.behaviormodeling.common** package.

The output object has the following information:

BHMOperationOutput variables	Description
String modelIdentifier	Specifies the model identifier.
String modelDirectory	Specifies the full path of the model.
String tcUID	Species the UID of the model.
String checkoutStatus	Specifies the checkout status. It can be one of the following: **MODEL_NOT_IN_TEAMCENTER** **NOT_CHECKEDOUT_BY_ANYONE** **CO_BY_DIFFERENT_USER** **CO_BY_SAME_USER_IN_TEAMCENTER_ONLY** **CO_BY_SAME_USER_IN_TEAMCENTER_AND_BHM_IN_DIFFERENT_WORKING_DIR** **CO_BY_SAME_USER_IN_TEAMCENTER_AND_BHM_IN_SAME_WORKING_DIR**
TCMEOperationOutput.-OperationStatus operationStatus	This is a string message. The message status is either **SUCCESS** or **FAILURE**. In the case of **SUCCESS**, the string

BHMOperationOutput variables	Description
	is empty, but in the case of **FAILURE**, the string contains the error message.

Example

```
BHMObjectStatusOutput getStateOfCurrentModel
( “BHM_Save002”,
ModellingGatewayDataStore.getCachedObjectsModel(rootModelName).
getCachedModel().getPrimaryData().getFile().get(0).getFilePath(),"MATLAB"
)
```

ModelManagementFCCAPIs API

Provides the status of the File Client Cache (FCC).

The FCC deals with uploading, downloading, and caching files saved in Teamcenter. For any integration operation to be successful it is important to understand the status of FCC.

This API allows you to initialize the FCC, stop the FCC, and get its status.

For this API to work ensure the following:

- The following entry is added to the PATH environment variable: `%FMS_HOME%\bin`
- The **FMS_HOME** environment variable is configured.

you must add the location of FMS HOME to the path as follows: `%FMS_HOME%\bin`

```
TCMEFCCOutput out = ModelManagementFCCAPIs.tcmeInitializeFCC();

TCMEFCCOutput out = ModelManagementFCCAPIs.tcmeStopFCC();

TCMEFCCOutput out = ModelManagementFCCAPIs.getFCCStatus();
```

Inputs

None.

Outputs

The output is a **TCMEFCCOutput** object. It contains the following information:

BHMOperationOutput variables	Description
Enum out.getOperationStatus	Specifies if the operation has failed or succeeded. Returns the following: • **OPERATION_FAILED** • **OPERATION_SUCCESSFUL**
List out.getFccInfo	Provides additional information about the FCC status. For the **tcmeInitializeFCC** method: • **FCC Started** This indicates that the FCC is successfully started through the API. • **Assigned FSC FCC URL is currently active** For the **tcmeStopFCC** method: • **FCC Stopped** This indicates that the FCC is successfully stopped through API. • **FCC Clients are connected to the FCC segment cache. Stop all client applications or use the '-kill' option.** If FCC is not stopped. For the **getFCCStatus** method: • **FCC Started** This indicates that the assigned FSC FCC URL is currently active. • **FCC Offline** This indicates that the FCC is offline.
Boolean out.isFccOnline	Returns true if the FCC is online and false if it is offline.

Rich client APIs

BHMOperations API

```
BHMOperations (String strToolType, TCComponent modelrevision)
```

This API is an abstract base class for all the behavior modeling operations supported through the rich client. The constructor considers the model tool name and the model revision object for which the behavior modeling operations are to be performed.

Inputs

strToolType

Specifies the name of the modeling tool, for example, MATLAB.

modelrevision

Specifies a reference to a model revision or a BOMLine of the model revision.

BHMModelOrgOperations::bhmSetFolderName API

```
void setFolderName( String strFolderName ) throws TCException,
 BHMCommonException, JAXBException
```

Using this API, a custom environment can update the folder organization metadata for new files that are associated to a model and that are expected to be downloaded in the configured folders.

If the input folder name and the relation with which the dataset is associated to the model revision object do not match, an error is displayed. The API supports the model revision object or the BOMLine of a model revision object.

Inputs

strFolderName

Specifies the name of the folder.

BHMModelOrgOperations::bhmSetFileRelativePath API

```
void setFileRelativePath( TCComponent dataset, String configFolderName,
String relativePath )
 throws BHMCommonException, TCException, JAXBException
```

Using this API, a custom environment can update the file organization meta-data for new files that are associated to a model and that are expected to be downloaded in the configured folders.

If the input folder name and the relation with which the dataset is associated to the model revision object do not match, an error is displayed. The API supports the model revision object or the BOMLine of a model revision object.

Inputs

dataset

Specifies the dataset corresponding to the file.

configFolderName

Specifies the name of the configured folder. This folder name must match the folder name from the integration definition file.

relativePath

Specifies the relative location of the file with the relation to the model folder.

Analysis Request APIs

getAttributeData API

This API extracts the input and output attribute information for the specified analysis request or study object.

Inputs

InputRevisions

Specifies a vector of the analysis request revision UIDs, study revision UIDs, or a mix of both.

Output

operationStatus

Specifies the following:

- Status of the operation
- Response data corresponding to the input analysis request or study revision

setAnalysisRequestData API

This API updates the attribute values based on the simulation results in Teamcenter. It also uploads result data in the form of result files. These files are then associated with the analysis request or the study revision.

Inputs

InputAttributeData

Specifies a vector of attribute values. These values are generated after the simulation. The attribute values are grouped per analysis request or study revision. You can also specify SMC files to be associated with the analysis request of study objects.

Outputs

operationStatus

Specifies the following:

- Status of the operation
- Response data corresponding to the input analysis request or study revision

getAnalysisRequestData API

This API is a wrapper around the Teamcenter SOA that extracts the analysis request data. The API returns the entire analysis request data, including the input-output objects, input-output attributes, all study objects and their data, end objects that are associated through trace links to the system blocks that are added to the analysis request, analysis request system blocks hierarchy up to the root node, and the associated data such as requirements and test cases. The API also provides an option to filter the unwanted data.

Inputs

InputRevisions

Specifies a vector of analysis request revision UIDs, study revision UIDs, or a mix of both.

ARDataFilter

Specifies an argument to filter unwanted data.

Outputs

operationStatus

Specifies the following:

- Status of the operation
- Response data corresponding to the input analysis request or study revision

Customizing behavior modeling integration operations by using extensions

Teamcenter provides extension points for the following behavior modeling tool integration operations:

- Save
- Save-as

- Open
- Insert

Extension points are available for the following concurrent modeling operations:

- Import
- Export
- Add to branch
- Update

You can implement these extensions as java programs, tool-specific scripts, or batch files.

Use the following extension points to extend behavior modeling integration:

Extension name	Description
PRE_UI_ACTION	Performs the configured actions before the behavior modeling tool launches Teamcenter dialog boxes. This extension supports only Save and Save-as operations. You can implement this extension using Java only.
PRE_UI_INSERT	Performs the configured actions before the behavior modeling tool launches the Teamcenter insert dialog box. You can implement this extension using Java only.
PRE_CONDITION	Checks the specified conditions before performing the behavior modeling operations. If the **PRE_CONDITION** extension fails, the operation is not performed.
PRE_ACTION	Performs the specified actions before performing the behavior modeling operations.
POST_ACTION	Performs the specified actions after performing the behavior modeling operations.

The **PRE_UI_ACTION** and **PRE_UI_INSERT** extensions take the input as an XML string that is based on the **BHMOperationsSchema.xsd** file. The **PRE_CONDITION**, **PRE_ACTION**, and **POST_ACTION** extensions take the input as an XML string based on the **BHMDesignSchema.xsd** file.

Configuring extensions

Update the *TOOL-NAME*_**BHMIntegrationDefinition.xml** file with information about the extensions. If you are configuring the concurrent modeling extensions, update the *PROJECT_IntegrationDefinition.xml* file.

For example, for integration with MATLAB, you can update the Matlab_BHMIntegrationDefinition.xml file as follows:

```
<Extensions>
 <Extension operationName="SAVE" extensionPoint="PRE_UI_ACTION" >
      <Impl type="JAVA" location=""
name="com.teamcenter.matlabcustom.SaveCustom" />
    </Extension>
    <Extension operationName="SAVE" extensionPoint="PRE_CONDITION" >
      <Impl type="SCRIPT" location="" name="SAVE_PRE_CONDITION" />
    </Extension>
    <Extension operationName="SAVE" extensionPoint="PRE_ACTION" >
      <Impl type="BATCH" location="D:\temp" name="RunMe.bat" />
    </Extension>
    <Extension operationName="SAVE" extensionPoint="POST_ACTION" >
      <Impl type="JAVA" location=""
name="com.teamcenter.matlabcustom.SaveCustom" />
    </Extension>
</Extensions>
```

- operationName

  Specifies the name of the operations that the extension point supports.

  The extension points support Save, Save as, Open, and Insert operations.

  The PRE_UI_ACTION extension supports only the Save and Save as operations.

- extensionPoint

  Specifies the name of the extension points such as PRE_UI_ACTION, PRE_CONDITION, PRE_ACTION or POST_ACTION.

- implType

  Specifies how the extensions are implemented.

  implType can have the following values:

  - JAVA

Specifies that the extension is implemented in Java.

The Java implementation must either implement the **com.teamcenter.behaviormodeling.common.bhminterface.IBHMOperations Extendable** interface or extend the **com.teamcenter.behaviormodeling.common.DefaultBHMOperations Extendable** class.

The PRE_UI_ACTION extension point supports only Java as the implementation type.

- BATCH

  Specifies that the extension is implemented as a batch file.

- SCRIPT

  Specifies that the extension is implemented in the behavior modeling tool-specific script file.

- location

  Specifies the location of the batch file. You must set a value for this variable only when the **implType** is **BATCH**.

- name

  - Specifies the class name when the **implType** is Java.

    The class must be in a JAR file named **customjars** that is located in the **bhm** folder of the Teamcenter installation.

  - Specifies the name of the batch file when the implType is BATCH.

  - Specifies the command ID of the script file when the implType is SCRIPT.

    The script file name corresponding to the command ID is defined in the **MATLAB_BHMConnectorCommands.xml** file.

Configure the PRE_UI_INSERT extension

The PRE_UI_INSERT extension allows you to restrict the search options on the Teamcenter Insert dialog box.

You can implement this extension using JAVA only.

The method for implementing this extension is as follows:

- Implement a custom JAVA class and override the **public BHMExtensionOutput executePreUIAction(String input, Connection connection)** method in this custom class.
- The input string of the **BHMExtensionOutput executePreUIAction** method is an XML file based on the **BHMOperationSchema.xsd** schema.
- Use the **JAXB** utility to generate the XML input strings. This utility is available in the **BHMOperationData** and the **BHMDesign** classes located in the **com.teamcenter.behaviormodeling.common.xml** package.
- The JAXB utility also does the following:
  - Converts the input XML string into a JAVA object.
  - Sets the UID in the JAVA code.
  - Converts the JAVA object back into XML.
  - Sets the XML file as the output in the XML field of the **BHMOperationOutput** object.
- In the input XML file, you can provide two types of UIDs. These UIDs will be used to populate the **Insert** dialog box.
  - The list of UIDs for the insert search operation. In the **Insert** dialog box, you can search using Teamcenter folders or Classification, or you can use Teamcenter search. You can provide UIDs only for one out of the three search options.
  - The list of UIDs for the behavior models that you want to appear in the **Model selected for Insert** list of the **Insert** dialog box.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<BHMOperations xmlns="http://www.plmxml.org/Schemas/bhm">
  <options>
    <insertOperationOptions stopOperationOnError="true">
    <!-- List of Teamcenter UIDs for browsing on Insert dialog.
            Enable one of the following: -->
      <BrowseList>
        <TCFolderUIDs>
        <!-- 1. Specifies the Teamcenter folder UIDs -->
          <UID>AkTRBcBQI2ELbD</UID>
          <UID>gQYRBcGLI2ELbD</UID>
        </TCFolderUIDs>
        <!-- 2. Specifies the Classification UIDs -->
        <TCClassUIDs/>
        <!-- 3. Specifies the UIDs available in Teamcenter search
dialog-->
```

```
          <TCObjectUIDs/>
        </BrowseList>
        <!-- Specifies the UIDs of objects that will be displayed
                in the Models selected for insert list
        -->
        <SelectList>
          <UID>A6URBk7FI2ELbD</UID>
        </SelectList>
      </insertOperationOptions>
    </options>
  </BHMOperations>
```

In the example input XML file:

- UIDs are specified for objects to be displayed in for the **Teamcenter Folder** tab. The **Classification Tree** and **Teamcenter Search** tabs are disabled.

  Only one tab can be enabled at a time.

- UIDs of objects that must always appear in the **Models selected for insert** list are added in the **SelectList** element.