Business Modeler IDE
Best Practices Guide

Siemens PLM Software

Version 2.18

Feb 2017

Change Summary

Version	Date	Comment
1.0	1/1/2008	• Initial version of Business Modeler IDE Template Guide
2.0	8/6/2008	• Changed name of document to Business Modeler IDE Best Practices Guide. • Added more information about upgrading from Engineering to Teamcenter unified architecture. • Added information about prefixes. • Added information about the Type Analysis Tool • Added information about Software Management Systems (SCM) in section 8.
2.1	10/1/2008	• Added detailed information about how to properly upgrade from Teamcenter unified architecture to another release of Teamcenter unified architecture in section 12. • Added detailed information about handling type name collisions and dataset merges in sections 10 and 12. • Added information about how to use the BMIDE Hot Deploy to update a production database in section 6.5 and section 8.3.
2.2	12/5/2008	• Added a “How to” section for details on how to accomplish specific tasks. • Added details on how to add relation properties in section10.1. • Added details on how to add runtime properties in section 10.2. • Added a subsection to the SCM chapter on how to coordinate continuous work on a test environment while performing operational data updates to a production database.
2.3	4/15/2009	• Added information on how to clean up any merged dataset definitions from the generated baseline.xml file. • Update section titled Upgrading a BMIDE Template to include more accurate information about how to upgrade a BMIDE template from a TcUA to TcUA release.
2.4	07/10/2009	• Added information to Orphaned Type Issues section about deleted types and classes. • Added information on how to re-order custom LOV values in section 10.3.
2.5	08/06/2009	• Added Prerequisite tasks before starting an upgrade that explains the tasks to perform before starting an upgrade in section 11.1.1.
2.6	08/24/2009	• Added Best Practices related to Form Storage Class Swapping in section 10.4
2.7	11/24/2009	• Added section 17.4 titled “How to remove COTS elements causing data model errors from custom BMIDE template section”.
2.8	10/22/2010	• Corrected the TEM option name from “Rebuild/Update the database” to “Update the database” in section 6.4 • Added details on hot deploy restrictions in the new section 6.6 • Corrected the release information and the sample prefix “x99_” in section 9 • Added details on applicable releases for section10.1, 10.2 and 10.3. • Corrected the download location of Type Analysis Tool in section 13.1 • Updated the postupgrade steps for Teamceter Engineering to Teamcenter Unified upgrade in section 13.3.2 • Corrected bmide_postupgradetotc utility syntax typo in section 13.3.2 • Added details on baseline file cleanup in the new section 13.7

		• Corrected the download location of Change Type Name Tool in section 13.1
2.9	11/01/2010	• Added 'Best practices for general Form creation' in new section 10.5 • Added a prerequisite task 'Correction of form storage class on custom form types' in 11.1.1. • Added 'Correction of form storage class on custom form types' in new section 12.1 • Added 'Codeful Extensions' in Appendix section
2.10	03/30/2011	• Updated the section 14.1 to mention the wizard location • Updated section 13.2 (Dataset type name collisions) to discuss about usage of prefix while adding custom dataset references to a COTS dataset • Corrected the typo in section 11.5(Baseline file cleanup) • Removed chapter called "Installation" since this is now covered in the standard Teamcenter documentation set. Information can be found in both the Teamcenter Installation manual and the BMIDE User Guide. • Cleaned up section "How to Create a BMIDE Template Project" to refer to the BMIDE User Guide for details • Cleaned up section "How to Use TEM to Add Your Custom Template" to refer to the BMIDE User Guide for details • Cleaned up section "How to Use TEM to Update Your Custom Template" to refer to the BMIDE User Guide for details • Cleaned up section "Upgrading from Teamcenter Engineering to Teamcenter" to refer to the Upgrade Guide for details • Cleaned up section "Upgrading from Teamcenter to a new release of Teamcenter" to refer to the Upgrade Guide for details
2.11	05/02/2011	• Added new section 6.7 "How to recover from a failed template deployment" • Added a new section 18 "How to rename a template"
2.12	05/17/2011	• Fixed missing links in the document.

2.13	7/12/2012	• Removed the following sections
		  ○ Enhancements to Managing and Deploying Extensions
		  ○ How to make frequent updates to a production environment using the BMIDE Hot Deploy

2.14	3/8/2012	• Added a new section "How to Use Custom Operation in a Condition Expression"
2.15	08/22/2012	• Added a new section "How to remove a template" • Added a new section "How to remove a dependency from your template" • Added a new section "How to delete business object instances from the Database" • Moved "Codeful Extensions" sub-section to a new section 23 "Codeful Extensions" • Added a new section 23.2 "Relation Navigation from Secondary Object to Primary Object" under Codeful Extensions
2.16	06/18/2013	• Updated the references to hot deploy, live update in section "Restrictions on Deploying Templates" • Added a new section 17.5 "How to add Smart Folder capability to Teamcenter without having to install the CPG Template" • Updated the section "Update files in TC_DATA/model" with a new step needed to rename a template

		• Updated the section "Update files in TC_DATA/model" with a new step needed to remove a template • Added a new section New Best Practices for Custom WorkspaceObject as Opposed to Custom Form • Added a new section "New Best Practices for Custom Item" • Revised the existing section "New best practice for parenting" • Revised the existing section "Best practices for subclassing Form Storage Classes"
2.17	05/25/2016	• Updated Section "Unregister the template from the database" to include additional command for updating installed_templates_info.txt • Updated Section "Before removing a template from Teamcenter" by replacing XXX with actual reference to correct Section.
2.18	02/22/2017	• Added a new section "Removing a template – from Teamcenter 11.2.3 and onwards" • Updated the existing sections "Before removing a template from Teamcenter", "Removing a template that you own" and "Removing a template owned by someone else".

TABLE OF CONTENTS

Document Audience

This paper is targeted at those people who need to use the Business Modeler IDE to extend the data model and business behavior of Teamcenter unified architecture. This paper focuses on the Business Analyst, Siemens PLM Software services, or third-party vendors who are responsible for implementing, managing, and distributing data model definitions. This paper will provide detailed topics of the new tools, new processes, and best practices for working with the Business Modeler IDE. This paper may be distributed to customers, 3rd party developers, and Siemens PLM Software Global Sales and Services.

1 Where can I download the latest version of this document?

This best practices guide will be continually updated as new information about the BMIDE is available. To get the latest version, please download from GTAC.

Steps for downloading the tool from GTAC:

- Go to http://support.ugs.com/
- Click on the “Documentation” link.
- Click on the “Teamcenter” link and login.
- Click on the tab with the Teamcenter release to which you plan to upgrade.
- Download the file title: “Business Modeler IDE Best Practices Guide”

2 Overview

Teamcenter introduces a new client called the Business Modeler IDE (BMIDE). The BMIDE is a tool for creating data model extensions and configuring the business behavior of Teamcenter. Teamcenter is easily configurable through both codeless and “codeful” extension points.

Codeless extension points are places within the Teamcenter code base that expose a configuration point through a graphical user interface. Wizards within the BMIDE guide the user through configuring the extension point to tailor Teamcenter to align within a business process. Some examples of codeless extensions are business objects, properties, naming pattern validation, number sequence generators, deep copy rules for revise, relation cardinality, and pick lists for properties.

Codeful extension points are places within the Teamcenter code base where code can be authored to provide advanced business behavior. Typically code is authored when behavior is required that is beyond the capabilities of the codeless extension points. Once the code is written, it can be connected through a codeless extension point to a hook point within the application that automatically executes the code when an action is performed in the user interface.

The Business Modeler IDE exposes these extension points to you through a graphical UI. As you configure these extension points, the BMIDE stores these definitions in XML files, thus abstracting the user from being required to have any knowledge or experience with the programming interface. The BMIDE is focused on providing codeless extension points as its primary goal since these are the easiest to configure and least expensive for a customer to maintain. A reduced cost of ownership is provided through codeless extension points because business behavior code is maintained by the software provider and the customer extends that behavior through a well-defined interface in the BMIDE. Because all extensions are performed consistently, upgrading from release to release is easier and inexpensive.

Teamcenter and the BMIDE are introducing new mechanisms and processes for creating, organizing, managing, and distributing extensions through a template. These new processes provide a consistent framework that can be utilized by customers, third-party product developers, Siemens PLM Software services, and Siemens PLM Software product development to build industry solutions or customer extensions to Teamcenter in a unified manner. All users create extensions to Teamcenter with the equal benefits and comply with the uniform rules.

The new template mechanism makes it easier to distribute the extensions and provides support for installation through the Teamcenter Environment Manager (TEM) wizard. Therefore, any extensions that you create can be presented as a selectable feature through TEM along with the other Teamcenter templates.

The BMIDE can also deploy the data model template directly from the BMIDE to any test server with a single button click. The BMIDE client supports both two-tier and four-tier configurations. Thus the user has the ability to use the BMIDE client in a two-tier configuration with a test environment for local testing and in a four-tier configuration for deploying to other test environments.

For those customers with large implementation projects that require two or more users to develop the data model and extensions to Teamcenter, the BMIDE can scale up easily to support a team of concurrent developers. The BMIDE easily integrates with any available Source Control Management (SCM) systems. Examples include ClearCase, Perforce, Visual Source Safe, Subversion, and CVS.

This guide will not give details on how to create extensions in the BMIDE or how to install or upgrade Teamcenter. This information is already covered in the standard product documentation. This document is intended to be a supplemental guide for those people who need to use the Business Modeler IDE. This guide provides guidance, tips, and best practices on how to successfully administer the BMIDE template. Since the BMIDE templates have an impact on the install and upgrade process, the Teamcenter documentation should be consulted for the primary source of information. This guide supplements those guides by providing the reader with the broader picture of the new BMIDE architecture and its role in the Teamcenter unified architecture platform.

For complete information about the impact of the BMIDE to Teamcenter installation and upgrade, and for information about use of the BMIDE, see the following documentation:

- Release Bulletin
- Installation Guide

- Business Modeler IDE User Guide

2.1 Who Should Read this Guide?

You should read this document if you plan to:

- Install a new Teamcenter installation and create extensions
- Use the Business Modeler IDE
- Create a new BMIDE template
- Understand the details of the files that make up a BMIDE template project
- Upgrade a Teamcenter Engineering installation to Teamcenter unified architecture
- Migrate your existing Teamcenter Engineering data model and behaviors to the BMIDE
- Understand how to coordinate the upgrade with multiple site who have the same, similar, or different customizations
- Set up environments for developing, testing, and deploying extensions
- Learn how to use your BMIDE with a Source Control Management System (SCM)
- Learn how to deploy your template
- Learn how to package your template into a feature to install through TEM
- Understand when to use Hot Deploy and TEM deploy
- Understand how to do concurrent development with the BMIDE
- Learn how to add commands to the template to create supporting data during install and upgrade
- Learn why and how to test your template for accuracy with respect to upgrades

3 Supported Extension Points

The following is a list of supported extension points within the Business Modeler IDE in Teamcenter. This list is a brief synopsis of the extension points available in Teamcenter that are configurable through the BMIDE. For more details about each extension point, see the Business Modeler IDE Guide.

Business objects

Business objects are the primary objects that Teamcenter users work with in to create product data. These business objects are modeled in the BMIDE. Business objects contain properties and behaviors and are hierarchical in nature. A new business object can be added anywhere in the hierarchy to inherit behavior from its parent business object.

- Item
- Form
- Dataset
- AppInterface
- GDELink
- Intermediate Data Capture
- StructureContext
- Runtime (View Only)

Properties

Properties are the attributes that describe the business object.

- Persistent Properties are stored in the database
- Compound Properties are properties from one object shown on another object
- Runtime Properties are properties that are computed at runtime
- Relation Properties are properties that show related objects

Schema

Schema is comprised of Classes and Attributes: the logical database representation of the business objects and properties. Attributes represent your meta-data and store the following data types.

- String
- Integer
- Date
- Float
- Character
- Logical
- Typed Reference
- Untyped Reference

Business Rules

Business rules are the business behavior extension points of Teamcenter. The following is a list of the behavioral extensions points in Teamcenter.

- Display Rules
- Naming Rules
- Extension Rules
- Deep Copy Rules
- GRM Rules
- ID Context Rules
- Property Rules
- Compound Properties
- Verification Rules

List of Values

List of values provide a pick list of options for a given property in the UI. Business analyst can set up predefined lists that match the property's data type: String, date, integer, etc. There are four types of LOVs.

- Standard
- Filter
- Hierarchal
- Interdependent

Options

Other extension points are available for configuring tools, change, unit of measure, and so on.

- Change
- Tool
- Status
- View Type
- PS Occurrence
- ID Context
- Unit of Measure
- Storage Media

Constants

A new constants framework is available by which you can configure extension points of the system and provide some of your own.

- Global Constants
- Business Object Constants
- Property Constants

Rules Engine

A rules engine framework is available for providing robust decision table logic to your customers for configuring system behavior.

- Application Extension Points
- Application Extension Rules
- Business Context

4 What are Templates?

A Business Modeler IDE template is a container that holds the extensions defined in the Business Modeler IDE. A template can contain any number of business objects, classes, LOVs, and business rules. For a complete list of definitions that can be stored in the BMIDE template, see section 3.

Since the template is a container of definitions it becomes a facility to group and apply functionality. Typically a template is used to organize a set of definitions for a specific industry, application, or functional area. It can also hold the custom extensions that a customer builds for tailoring Teamcenter to meet business objectives. Each of these templates can be optionally applied to Teamcenter like building blocks to build a robust combination of functionality. Each customer can select a combination of templates that provides the data model elements and behaviors needed to support their business practices and industry standards.

To support the building block capability, each template can declare dependencies on other templates. When a dependency is declared, TEM enforces that all dependent templates are installed before the template is installed. Additionally when a dependency is declared, the definitions in the dependent template are available for use, modification, or extension in the template that declared the dependency.

Figure 1 shows an example of how the base template “foundation” is extended by two optional templates called the Solid Edge Embedded Client template and the Wire Harness Configuration template. Because these two templates are dependent upon the foundation template, they can add attributes to the classes of the foundation template. Or they can create subclasses under the classes in the foundation template. Business rules from the foundation template can be altered in each of these templates per the guidelines of the BMIDE. In all cases, the BMIDE ensures that the user extends the other templates in a supported manner.

Likewise any new template can be created and declare dependencies on any number of templates. In the figure below, a customer has created a template called “Customer”. Because the customer wants to take advantage of the functionality offered by Foundation, Solid Edge Embedded Client, and the Wire Harness Configuration templates, the customer template declares dependencies on all three templates.

.

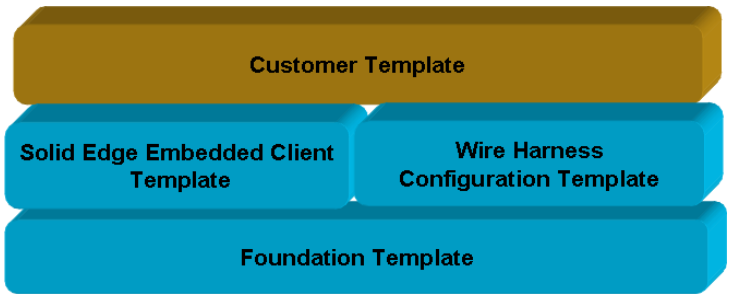


Figure 1: Sample template dependencies

Figure 2 shows some of the templates that are available inTeamcenter. These templates are all available with the Teamcenter unified architecture kit. Many of these templates were available as add-on solutions in Teamcenter Engineering and have been converted to the BMIDE template format. The base template is called the “foundation” template, and by default this template must always be installed into any Teamcenter database to give it the basic Teamcenter functionality. The Foundation template is installed anytime a Corporate Server is installed. The other Teamcenter templates are optional and can be installed as layers on top of the Foundation template. When a user creates a template in the BMIDE, they can choose any of these available Teamcenter templates to extend.

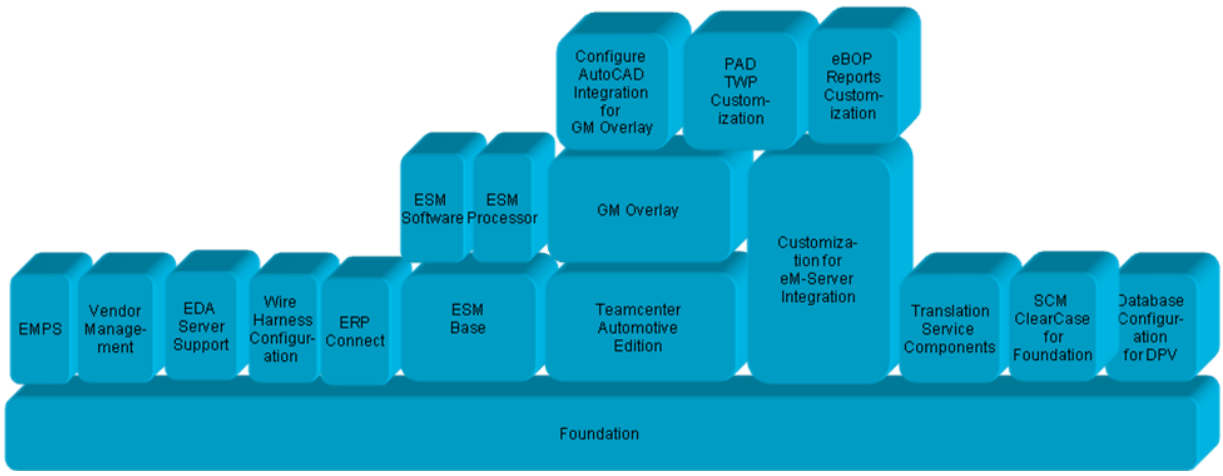


Figure 2: Teamcenter kit templates

Figure 3 shows some of the additional templates that are available with Teamcenter. The templates are release asynchronous to the Teamcenter release and are not supplied with the kit. However, they can typically be downloaded from GTAC or from the vendor who supplies the template.

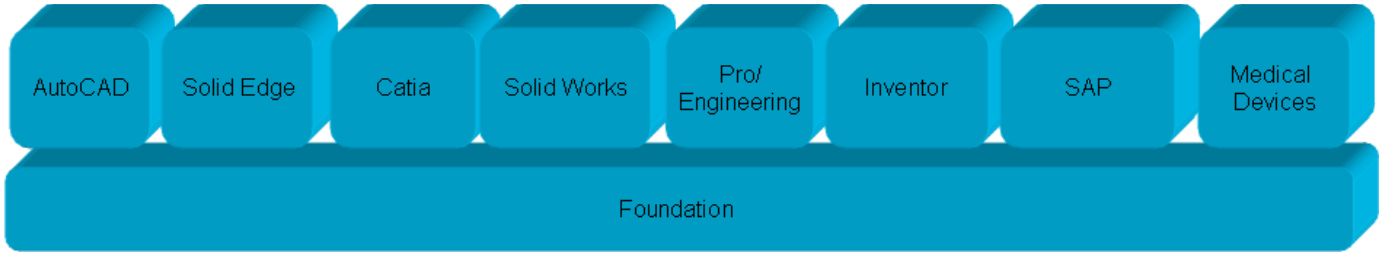


Figure 3 - Asynchronous templates

5 Using the Business Modeler IDE

This section is an introduction to using the BMIDE. This section will cover the BMIDE workbench, how to access help, how to create a BMIDE template project, the details about each file in the template project, source file organization, hot deployment, the Packaging wizard, and distributing templates. This section will not cover how to use the BMIDE to extend the Teamcenter data model and behaviors. For more information about how to use the BMIDE to create each of the extensions, see the Business Modeler IDE Guide.

5.1 Launching the BMIDE

The first time the BMIDE is launched, you will see the Welcome screen (Figure 4). The Welcome screen has three icons adjacent to the splash screen that you can click to take you to different areas of the BMIDE. The Overview icon takes you to the overview page that covers topics that are useful for getting started. The Tutorials icon launches a tutorials page that covers the most common tasks within the BMIDE and how to do them. Finally, you can click the Workbench icon to go to the BMIDE. To return to the Welcome screen at any time, select Help->Welcome.

Figure 4: Welcome screen

Click the Workbench icon on the Welcome screen; you see the Business Modeler IDE perspective (Figure 5). A perspective is a grouping of views. Initially all the views are empty and will remain empty until a template project is created. For more information about each of the views, see the Business Modeler IDE Guide or go to Help->Contents.

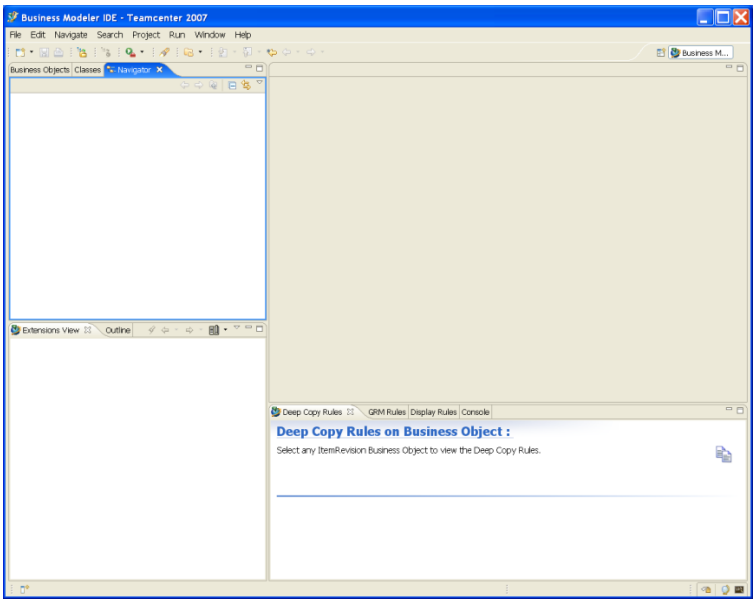


Figure 5: Workbench

5.2 Using the Help System

The BMIDE provides many varieties of help and is easily accessible within the BMIDE. The following sections address the help formats that are available and their usages.

5.2.1 Business Modeler IDE Guide

The *Business Modeler IDE Guide* is a complete reference on the BMIDE, templates, wizards, and deployment. It also contains instructions on how to create each type of extension available within the BMIDE. To launch this guide, select Help->Contents (see Figure 6).

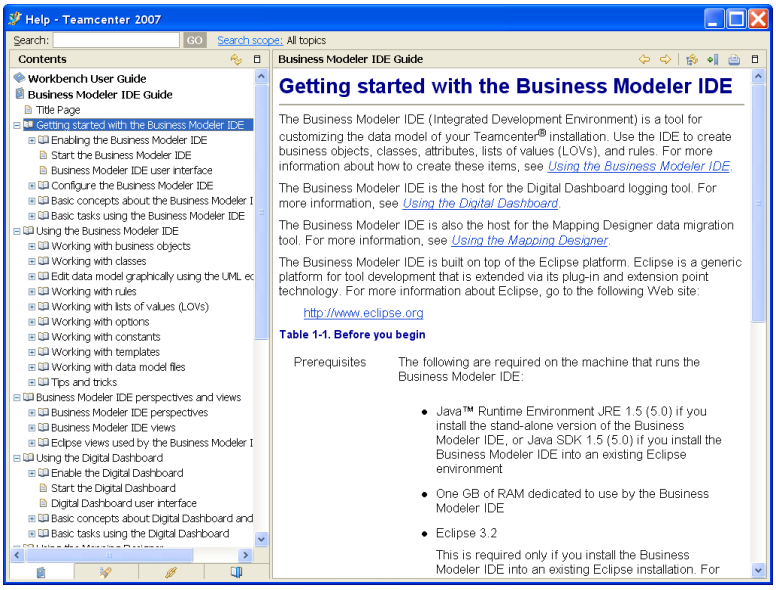


Figure 6: Business Modeler IDE Guide

5.2.2 BMIDE Overview

The BMIDE Overview is a list of topics from the Business Modeler IDE Guide related to getting started (see Figure 7). These topics are useful for getting a general overview of the BMIDE and its capabilities. To access this page select Help->Welcome and click Overview.

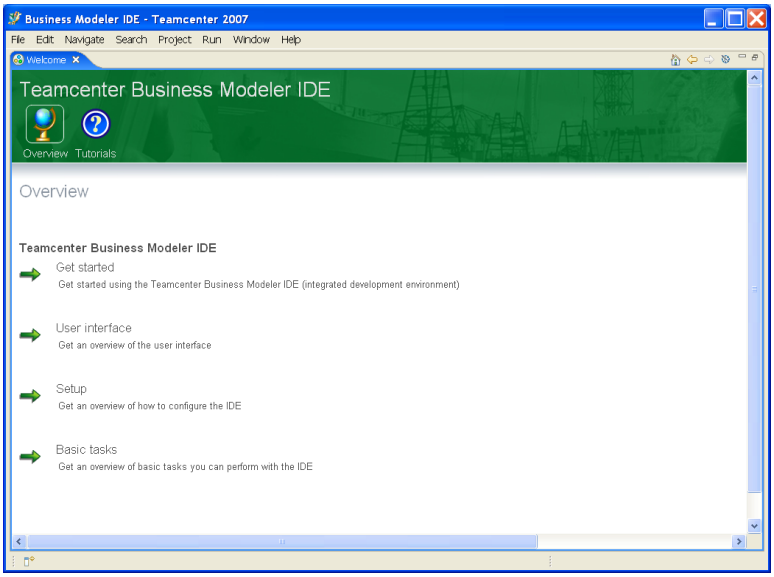


Figure 7: Overview

5.2.3 Tutorials

The BMIDE tutorials are lists of cheat sheets that guide you step by step on how to accomplish the most common tasks performed within the BMIDE. The tutorials can be used as a supplemental learning guide to the Business Modeler IDE Guide. To access the tutorials, select Help->Welcome and click Tutorials. Alternatively, you can access to the tutorials directly by selecting Window->Show View->Other->Cheat Sheets->Cheat Sheets. The following is a list of the available cheat sheets:

- Getting started
- Create a class
- Create a business object
- Use the UML Editor
- Create a list of values
- Create a naming rule
- Create a relationship rule
- Create a deep copy rule
- Create a display rule

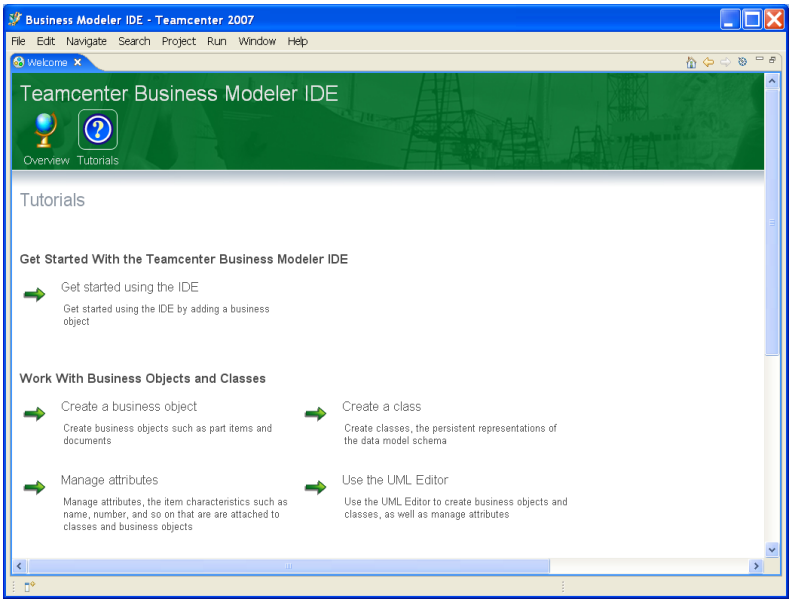


Figure 8: Tutorials

5.2.4 Dynamic Help

Dynamic help is accessible anywhere within the BMIDE (Figure 9). The dynamic help displays all help topics from the Business Modeler IDE Guide related to your current location within the BMIDE. Open any wizard, view, or editor and press the F1 key on your keyboard. The dynamic help system launches to the right of the BMIDE. Select any topic from the list to view more information.

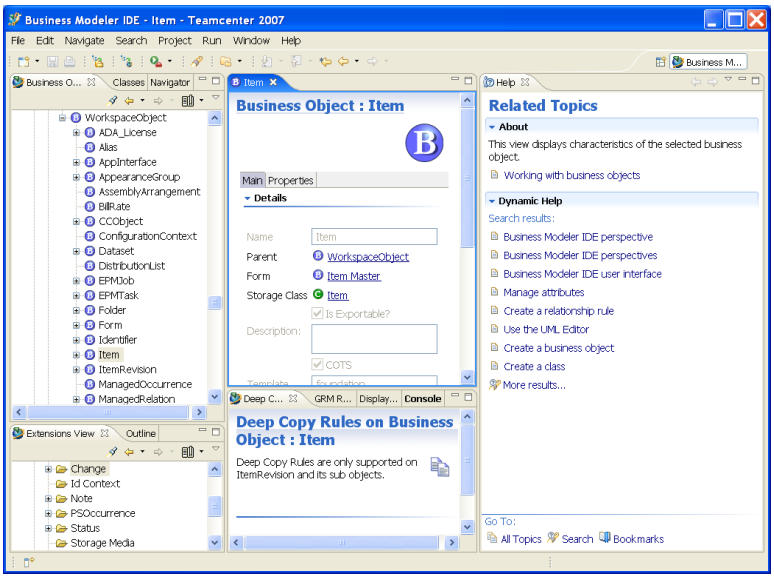


Figure 9: Dynamic Help

5.3 How to Create a BMIDE Template Project

For details on how to create a BMIDE Template project, see the BMIDE User Guide.

5.4 Loading the Data Model

After the template project is created, the BMIDE loads the data model definitions into your modeling environment. Each time the BMIDE is started the same steps are followed. The following are the details of how the model is loaded.

- For each template name listed in your <Project>/extensions/dependency.xml file, the BMIDE reads the XML template from the template folder path specified in the project creation wizard. As each definition is loaded, it is assigned COTS=true to indicate that the definition is from the dependent template list.
- For each custom source file listed in the <Project>/extensions/master.xml file, the definitions in the XML file are loaded into the BMIDE and appended to the existing COTS model. As each definition is loaded it is assigned COTS=false to indicate that the definition is from the custom template.
- If any parsing errors occur while loading the XML files, then the Console view shows the errors. The Console view can be found at the bottom right quadrant of the BMIDE (Figure 5). Errors can occur if files are manually edited and the proper XML format is not followed. Errors can also occur if the data model definitions are invalid and do not pass the model validation of the BMIDE. Typically errors will occur after manually editing a file. A manual edit circumvents the BMIDE validation and is dangerous if the model is not re-validated. Manual editing is permitted if you use an SCM to merge changes from two or more users. If there are any parsing errors, the Console view displays the errors showing the file name, line number, and an error description. If you receive errors, fix them and reload the project before you continue using the BMIDE. See section 5.10 for more details.

When the loading is complete, all views are populated with the full data model: COTS + Custom definitions. You can easily distinguish between the COTS and custom definitions by looking for the small green “c” icon next to any model definition icon: Class, Business Object, Attribute, and so on. The “c” indicates “Custom”. Additionally each extension is also tagged with the name of the template where it originated. To see the template name, open any object into an editor and look for the “Template” field or column in a table. Your modeling environment is now ready for use.

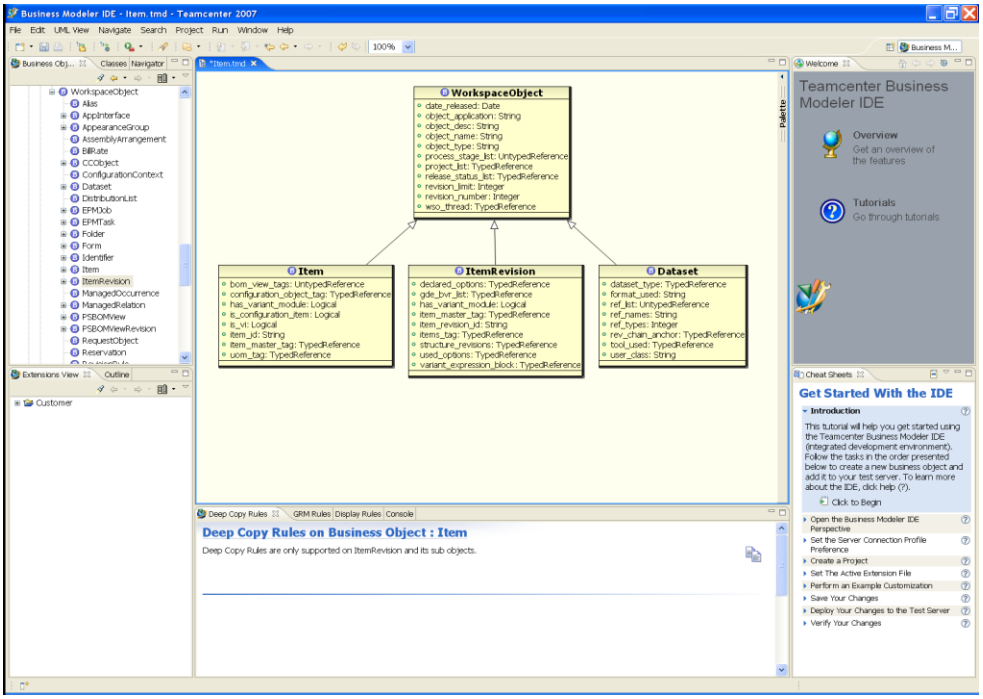


Figure 10: BMIDE with the data model loaded

5.5 File Organization of a Template Project

This section details the purpose of each folder and file in your template project. Figure 11 shows the template project structure that was created for the template as seen in the *Navigator View*. For a complete list of the files and their contents, see the *Business Modeler IDE Guide*.

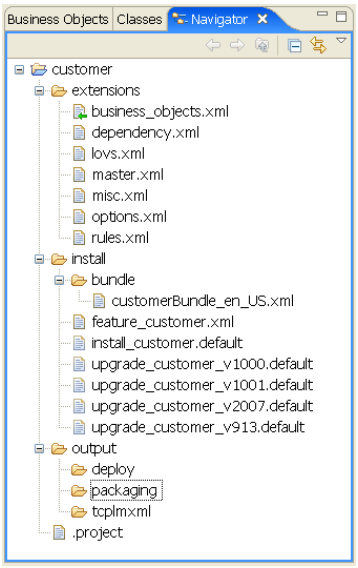


Figure 11: Sample template project structure

The BMIDE creates each new project in the workspace on your machine. The workspace is created automatically for you during the installation of the BMIDE client. If you have BMIDE running on Windows, the workspace is located at "C:\Documents and Settings\<*username*>\Teamcenter\BMIDE." If the BMIDE is running on UNIX, the workspace is located at "<*username*>/Teamcenter/BMIDE." Your template project folder is located in this workspace.

The BMIDE exposes the template project folder and its subdirectories and files in the Navigator view. However, the view does not directly show you the full path to the workspace. To see the absolute path of the files and folders on the machine, select any file or folder in the Navigator view, right-click, and choose Properties from the shortcut menu. The properties window displays the full path to the file using the *Location* field (Figure 12). Knowing the location of the workspace is important when you need to back up your template project, or when you need to delete or import a project into the BMIDE.

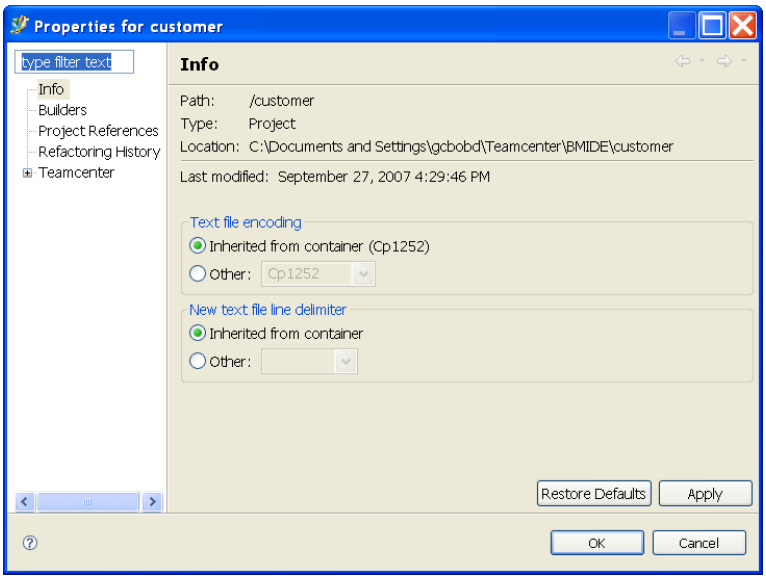


Figure 12: Properties window showing the absolute path

5.5.1 Navigator View

The *Navigator view* displays all the projects that are located in the workspace. Each project folder can be expanded to see its contents. Use this view to manage the files and folders of the project.

5.5.2 Template Project Directory

This is the top-level directory for the template project. In Figure 11, it is the “customer” directory. Each template project is created with a specific set of directories and files for managing your template and helping you to stay organized.

5.5.3 Extensions Directory

The “extensions” directory contains the XML source files for storing Teamcenter extensions, the dependency file, and the master file.

5.5.3.1 Source Files

Source files are XML files that contain the custom extension definitions to the data model and business rules. The definitions can be randomly saved into any source file. The BMIDE loads definitions in any order.

The user of the BMIDE can create any number of source files and folders. The names of the folders and files are arbitrary. Typically, you should never manually edit these source files because editing will circumvent any validation provided by the BMIDE. The BMIDE user interface presents a graphical tool for editing the data model and automatically writes the custom extensions to the source files for you. The only case where it is acceptable to manually edit a source file is when you need to merge two or more changes to a file when using an SCM with the BMIDE.

5.5.3.2 Dependency.xml

The *dependency.xml* file is an XML file contains the following information:

- The solution name
- The solution description
- The names of the dependent templates
- The custom prefixes designated for this project (reserved for use in a future release)
- The display name of the template
- The GUID of the template
- The optional flag that indicates whether the template is an optional template for Teamcenter. Only the Foundation template is required (flag=false); all other templates are optional (flag=true).

The solution display name, description, and list of dependencies can be edited through the BMIDE UI by selecting the project directory in the Navigator view, right mouse button, and choosing Project Properties -> Teamcenter -> BMIDE from the shortcut menu.

5.5.3.3 Master.xml

The *master.xml* file maintains the list of source files for the template. This file is automatically managed by the BMIDE. Additional source files and folders can be added through wizards. Whenever a file is added or deleted through the BMIDE, the master.xml file is updated to reflect this change. Note that this file controls the order in which source files are loaded. The order does not matter because the BMIDE can load in any order.

5.5.4 Install Directory

The *install* directory contains all the files that support the installation and upgrade of the template using TEM.

5.5.4.1 Bundle File

The <*template_name*>*_bundle_en_US.xml* file provides localizable strings for TEM to present your feature to a user. This file contains the description of your template.

5.5.4.2 Feature File

The *feature_*<*template_name*>*.xml* file is a file that wraps your template and makes it available to install through TEM. All templates must be wrapped with a feature file to be installable through TEM. The feature file declares the name of your template, dependencies on any other features, the names of the zip files that are included with the feature file, and any template matching tags. The feature file also contains the commands necessary to get your template installed or upgraded. For more information about editing this file to provide template_match tags, see section 11.2.

5.5.4.3 Installation Script

The *install_*<*template_name*>*.default* file is the script that is called by TEM when the template is installed for the first time into a database. This file is required for your template project. The installation file contains a set of ordered commands that installs your template definitions into a database. This file must be manually edited to keep it updated. For more information, see section 15.

5.5.4.4 Upgrade Scripts

The *upgrade_*<*template_name*>*_*<*version*>*.default* files are scripts that are called by TEM when your template is upgraded in a database. By default, four upgrade scripts are provided, one for each version of upgrade that is

supported by Teamcenter. The files contain tags that are used for organizing commands. If you do need to add commands, the appropriate files must be manually edited. For more information about what entries to edit in this file, see section 15.

5.5.5 Output Directory

The *output* directory is used by the BMIDE for files that are generated.

5.5.5.1 Deploy Directory

The *deploy* directory is used by the Deploy wizard. For each deployment, a new directory is created with a timestamp. Each directory contains the deployed template files and a returned log file from the server after all transactions are processed. You should check the log file after each hot deploy to verify that all new definitions were processed.

5.5.5.2 Packaging Directory

The *packaging* directory is used by the Packaging wizard to build the template package. The packaged feature can then be transported to the Teamcenter environment where it can be installed through TEM.

5.5.5.3 tcplmxml Directory

The *tcplmxm* directory is used by Global Multi-Site to generate the TcPLMXML file for the mapping designer.

5.5.6 Project File

The *.project* file tells the BMIDE what type of project it is. This file should never be edited or removed because it is required and managed by the BMIDE.

5.6 Source Files

Every template project is created with a default set of source files in the extensions directory. Each source file can store any type of definition regardless of the name of the file. The BMIDE does not force the user to store definitions in a particular file because of its name. Therefore, the file name is arbitrary, yet the name can provide a meaningful description of the kinds of elements stored in it.

Figure 13 shows the default set of source files. This default set is only a suggestion for organizing definitions. A user can store business object definitions in the business_objects.xml file, and LOVs in the lov.xml file. Alternatively, the user can store the LOVs and business objects in the same file.

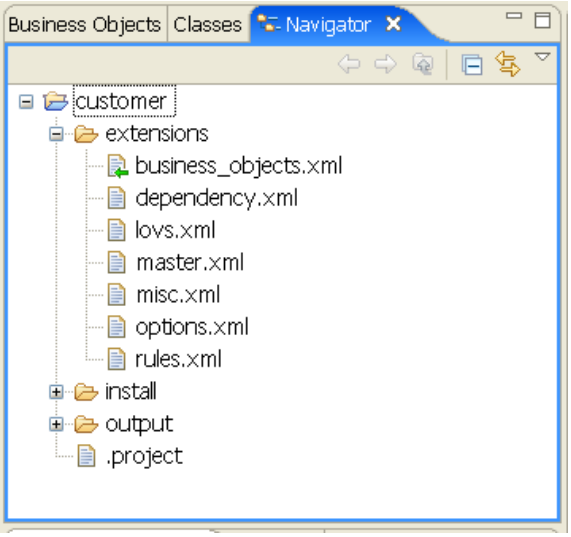


Figure 13: Default set of source files

Additional source files can be added by selecting the extensions folder in the project, right mouse button, and choosing Organize->Add new extension file. Additional folders can also be created by selecting the extensions folder, right mouse button, and choosing New->Other->General->Folder.

Figure 14 shows an alternative organization of source files where definitions are organized into functional folders first, then by files whose names indicate specific definition types. Figure 14 splits the functionality into two folders. The names “functionality1” and “functionality2” are used. However, as another example, these directories could have also been labeled as “bomline application” or “Program A”. The *document.xml* file may be used to store all schemas, business objects, and business rule definitions related to the document business object. The *tools.xml* file may be used to store all tools that were created to support the functionality.

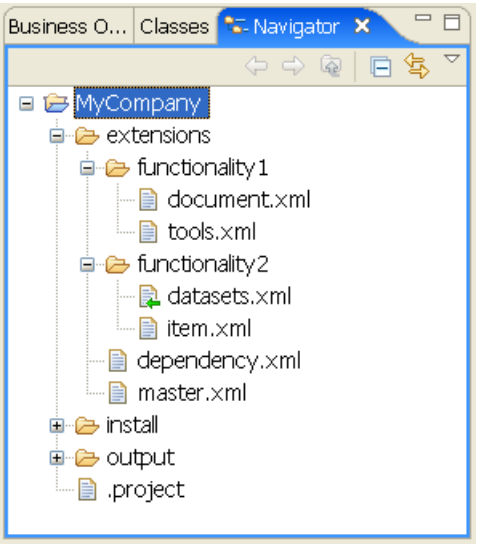


Figure 14: Alternative source file and folder organization

You can create any organization of files and folders that suits your needs. When using the wizards for creating new source files and folders, the BMIDE automatically updates the *master.xml* file with the new file information.

5.6.1 Best Practices for Source File Organization

Technically, all definitions created within the BMIDE can be stored in a single source file. This is useful if there is only one person using the BMIDE.

Ideally it is better to spread out your definitions into many source files especially when multiple users are working on the template. To do this, the BMIDE must be connected to a Source Control Management (SCM) system. This makes it easier for multiple users to work concurrently on the template's extensions. Each user should work in their own area of the source files, storing all definitions that they create into their own active source file. This ensures that when developers "check in" their source files to the SCM, there is little chance of having to merge two or more sets of definitions from different users.

A file merge is required when two or more users make simultaneous changes to the same file. A merging tool must be used by the last person who checks in the file to integrate their own changes with the other person's changes. Most SCM tools can automatically resolve a merge for you, but in some cases it is necessary to manually perform the merge. Merging is a necessary part of concurrent source code development, and can mostly be avoided with proper coordination and execution. The benefits of working concurrently on a template using an SCM far outweigh the tediousness of having to manually manage the files without an SCM.

5.7 Active Source File

The active source file is the XML file in your project template that receives all definitions created from wizards, views, and editors. Only one source file can be active at a time within the BMIDE client and remains active until another file is designated as the active source file. The active source file is marked with a green arrow on the active source file icon within the extensions folder in the Navigator view. As an example, see the green arrow icon on the *business_objects.xml* file in Figure 15.

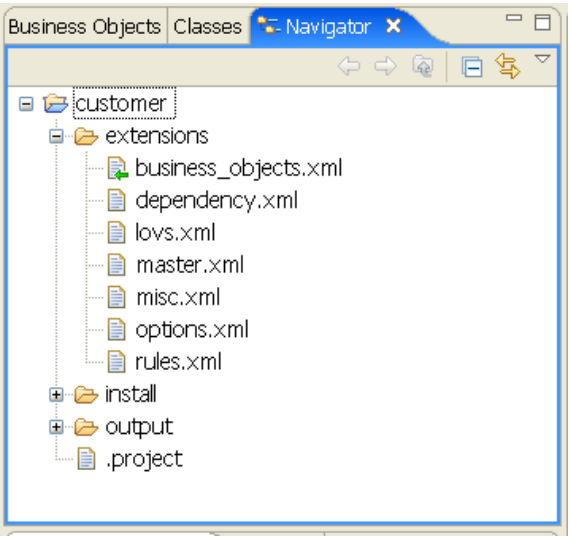


Figure 15: Navigator view showing the active source file

By default, an active source file is set for your project. The active source file can be changed to designate a new source file to receive the custom extensions. Figure 16 shows how to set the active source file. Go to the Navigator

view and select any resource, right-click, and choose Teamcenter-> Set active extension file. Use the Extension File Selection dialog box shown in Figure 17 to select the new active source file, and then click OK.

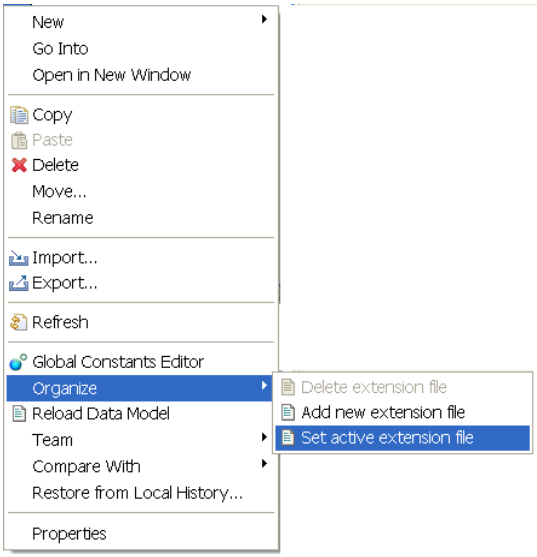


Figure 16: Setting the active source file

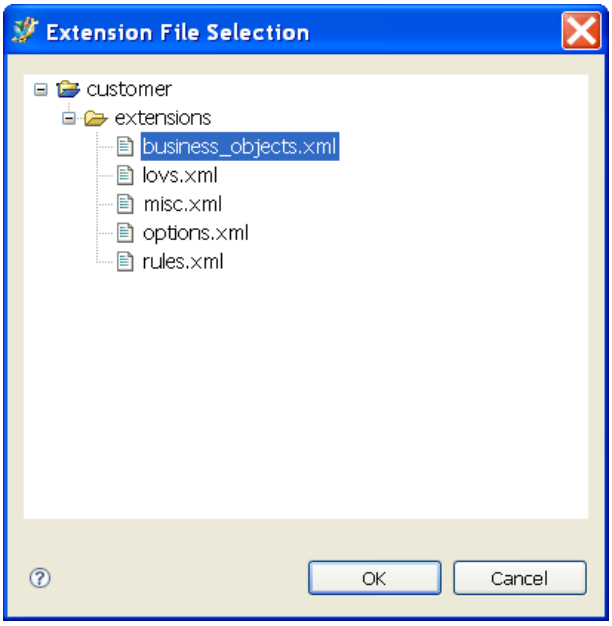


Figure 17: Extension File Selection dialog box

5.8 Saving the Template Project

While working in the BMIDE, you should save your project frequently. Saving the project writes all of your custom definitions into their designated source files. To save your project, select File -> Save Data Model. Select the project to save by its name and click OK. If you try to exit the BMIDE with unsaved changes in your project, the BMIDE prompts you to save.

To view the contents of any source file, go to the Navigator view, find the source file under the extensions folder, and double-click the file. This opens the XML file into an editor. Do not manually edit these extensions files. Manually editing these files circumvents all the validation that the BMIDE provides to ensure that you are extending Teamcenter in a supported manner. The only time it is acceptable to manually edit a source file is when you are merging files. See section 5.6.1.

5.9 Moving Definitions to another Source File

Each custom extension that you create in the BMIDE is stored into a source file through the active source file setting. If you want to reorganize the definitions, any custom definition can be moved to another source file. To move a definition, first find the custom object in the Business Objects view, Classes view, or Extensions view. The icon of a custom object is marked with a green "c." Figure 18 shows how to move a definition. Select a custom object, right mouse button, and choose *Organize -> Move to extension file*. Use the Extension File Selection dialog box shown in Figure 19 to select the new active source file, and then click OK.

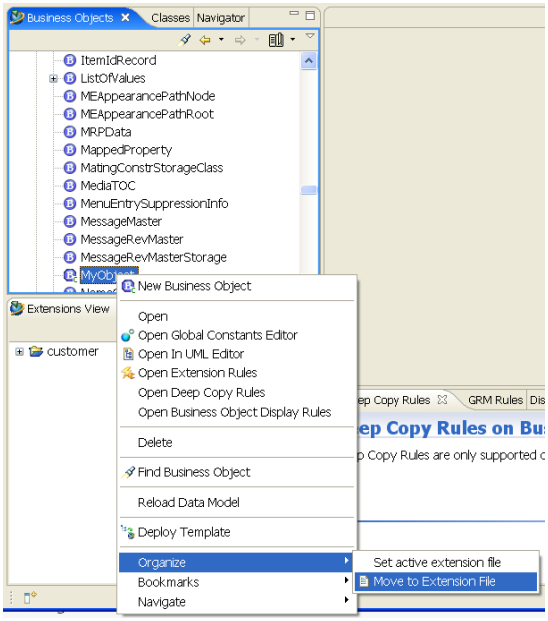


Figure 18: Moving an extension to another source file

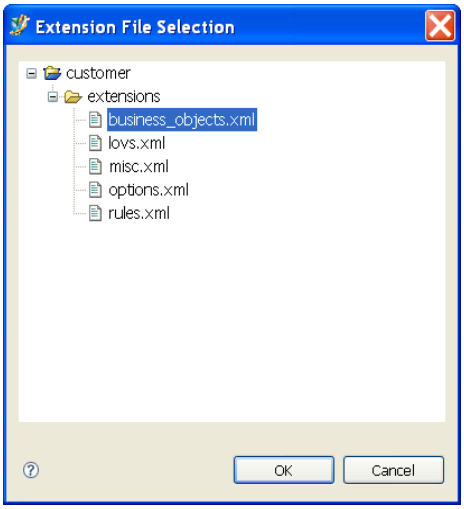


Figure 19: Extension File Selection dialog box

5.10 Reloading Data Model

In most cases, you should only use the BMIDE wizards and views to edit the data model. However, there are some circumstances where manually editing a file is acceptable. Typically, this would occur if a file merge is required when using an SCM.

After the manual file editing is completed, the XML files should be validated for correctness. To reload, right mouse button in any window and select *Reload Data Model*. This menu reloads the entire data model as detailed in section 5.4. During the load, all XML definitions are validated. Any errors that are found are displayed in the Console view. The Console view can be found at the bottom right quadrant of the BMIDE. If there are any parsing errors, the Console view gives you the file name, line number, and an error description. Before continuing to use the BMIDE, the errors should be fixed and the project reloaded. Continue fixing and reloading until all errors are removed.

5.11 Backing Up Your Custom Template Extensions

The template project within your BMIDE client contains your master copy of the extensions. These definitions represent company intellectual data much like the product data in the database and should be backed up regularly. If you use an SCM, ensure that the SCM repository is backed up frequently. If you do not use an SCM, then back up your BMIDE workspace frequently. If for any reason your BMIDE client machine malfunctions, you will be able to retrieve the previously backed up template project from the archives and continue working with minimal impact.

5.12 Connection Profiles

A connection profile stores information within the BMIDE so that the BMIDE can connect to a Teamcenter server. A connection is needed by the BMIDE to retrieve data from the server to assist you with configuring some of the extensions. A connection is also used for deploying the custom template directly from the BMIDE to a running server. The BMIDE is capable of storing multiple connection profiles, for deploying to more than one database. To configure a new connection profile, go to the Windows menu and select Preferences, then select *Teamcenter -> Server Connection Profiles*. Use this panel to create your connection profiles for your database server. All profile information is saved locally to your installation of the BMIDE client and is not shared with the template. For security reasons, the password is never saved with the profile and must always be entered when using the connection.

The following is a list of the BMIDE extension areas that require a connection to retrieve data from the server:

- Process templates are queried from the database to configure a Change object.
- Import and Export Transfer Modes are queried from the database to configure an *AppInterface* Business Object.
- Transfer Modes are queried from the database to configure an *IntermediateDataCapture* Business Object.
- Groups and Roles In Groups are queried from the database to configure Display Rules.
- Groups and Roles In Groups are queried from the database to configure a Business Context.
- Instance data is queried from the database to configure the values on some LOVs.

The BMIDE prompts you to log in using a connection profile the first time you access any of these areas. The session information is cached with the BMIDE for any subsequent requests for data. There is no need to log in again, unless you restart the BMIDE.

In addition to requesting data, the BMIDE also needs a connection to send your custom template definitions to a running test server. Hot deployment from the BMIDE works in both two-tier and four-tier mode. While you can have multiple connection profiles for deploying to different databases, only one connection can be used at time.

6 Deploying your custom template to a database

The BMIDE supports two methods for deploying your custom template definitions to a database: Hot Deploy and TEM Deploy.

Hot deployment (or “hot deploy”)

- Deploys directly from the BMIDE to a running test server for verification of custom extensions.
- Extensions created within the BMIDE can be deployed to a server and visualized with the RAC or thin client immediately.
- Hot deploy requires a user with DBA access.
- Supports two-tier and four-tier server configurations.
- Very useful for deploying extensions to a test database.
- Other logged-in users have to log out and back in to see the changes.

Package with BMIDE and deploy with TEM

- BMIDE wizard packages the template into a feature so that it can be installed to production databases through TEM.
- TEM requires a user with DBA access to install the template.
- Requires that the Teamcenter server be stopped, users logged out.
- Practical for production systems where downtime must be scheduled.
- Useful when the database update must be coordinated among many sites.
- Used for distributing your template to a partner or customer.

The next two sections address these two methods of deployment in further detail.

6.1 Hot Deploy

The BMIDE can “hot deploy” your custom template definitions directly from the BMIDE to any running test server using the Deploy wizard. FMS is required on the BMIDE client machine to transfer files to and from the server.

6.1.1 Using the Hot Deploy Wizard

For convenience, there are three places in the BMIDE where the Hot Deploy wizard can be initiated (see Figure 20).

1. There is a “Deploy Template” icon  on the main toolbar. The main toolbar is located under the top main menu bar in the BMIDE. Use this button for the quickest method for deploying your extensions.
2. Right mouse button on any object and select *Deploy Template*.
3. From the Wizards panel, select *New -> Other -> Business Modeler IDE -> Deployment*.

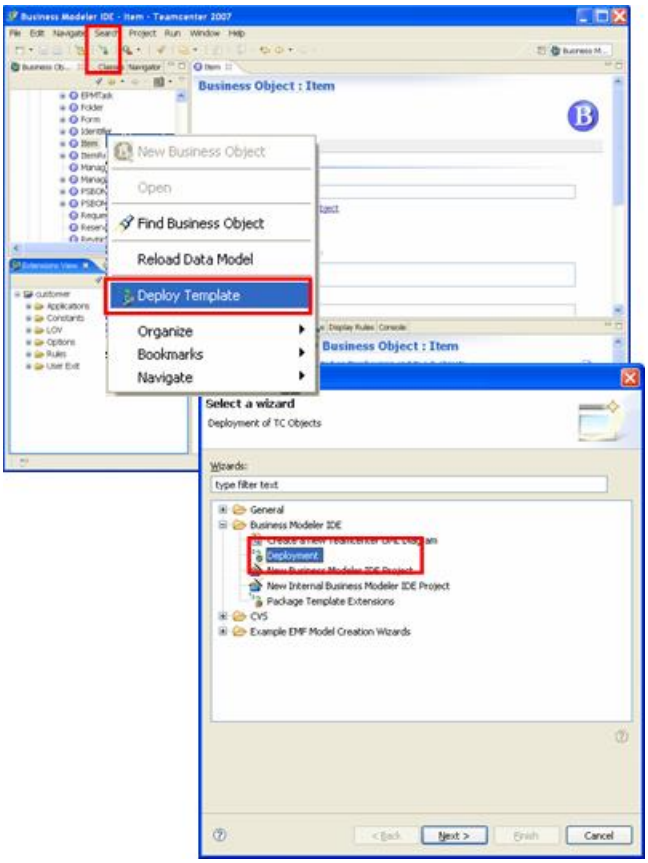


Figure 20: Three ways to hot deploy

The Deploy wizard prompts you to select the Project and the Server connection profile from your established list of profiles. See section 5.12. Selecting a profile from the profiles pull-down menu automatically fills out your profile information. If no connection profiles exist, then the wizard prompts you to fill out information and the wizard will automatically create a profile for you. Enter your password and click Finish. The Deploy wizard automatically requests that you save your template project before deploying. Click OK to save.

.

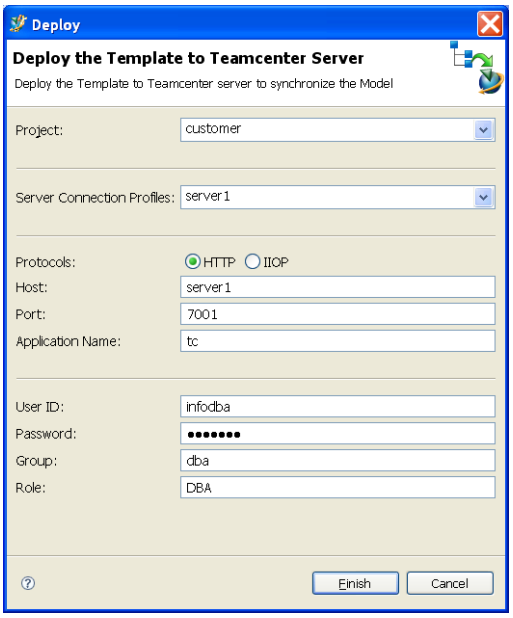


Figure 21: Deploy wizard

The Deploy wizard sends your template to the database server and synchronizes the data model in the database. A progress wizard will be displayed in the BMIDE client while you wait for the synchronization to complete.

For each hot deploy, the Deploy wizard does the following (see Figure 22):

- In the Navigator view, a new directory is created in your template project under the deploy directory using the name of your connection profile.
- A subdirectory is created under the profile directory using the current date and time of the deployment.
- The sources files listed in the extensions/master.xml file are consolidated into a single file and copied to <project>/output/deploy/<connection profile name>/<date>/<template_name>_template.xml file.
- The <project>/extensions/dependency.xml file is copied to the <project>/output/deploy/<connection profile name>/<date>/<template_name>_dependency.xml.
- Create a connection to the specified server using the login credentials in the connection profile and upload the <template_name>_template.xml and <template_name>_dependency.xml to the server using FMS. The two files are placed into the <TC_DATA>/model directory.
- The name of the deployed template is added to the <TC_DATA>/model/master.xml file if it is not already included.
- The system deletes the existing <TC_DATA>/model/model_backup.xml file.
- The <TC_DATA>/model/model.xml file is renamed to <TC_DATA>/model/model_backup.xml.
- A script is run that consolidates all template names listed in the <TC_DATA>/model/master.xml file and concatenates them into a single file called <TC_DATA>/model/model.xml. As a result the model.xml file contains the definitions from the COTS templates and the custom template.
- The <TC_DATA>/model/model.xml file is compared to the <TC_DATA>/model/model_backup.xml and the differences between the two models are written to a new file at <TC_DATA>/model/delta.xml. Because the model.xml file contains the COTS templates and custom template definitions and the model_backup.xml file contains the COTS template definitions, the delta.xml file contains all the definitions of the custom template.
- A server utility is executed, which processes the <TC_DATA>/model/delta.xml file and synchronizes the definitions into the database.

- As each element is processed, a log is kept of the processed and unprocessed elements.
- The log file is sent back to the BMIDE and placed in the *<project>/output/deploy/<connection profile name>/<date>* directory.

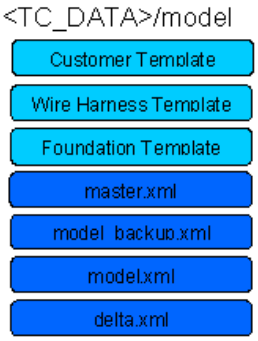


Figure 22: TC_DATA/model directory after the custom template deployment

6.1.2 Deploy Log File

After the hot deploy is complete, a log file is placed into your *<project>/output/deploy/<connection profile name>/<date>* directory. Always check this file to ensure that all elements were deployed properly. Figure 23 below shows a sample of the contents in this file.

```
-----------------------------------------
 Update Option   : all
 Log File Name   : C:\Temp\gcbobd2TierTransientVolume\foundation_model_21-Jan-2008_16-39-44.log
 Model File Name : D:\Programs\UGS\TEAMCE~1\Tc2007\tcdata\model\delta.xml
 Run Mode        : upgrade
 Process Option  : all
-----------------------------------------


Operation| Element Type   |Element Name                    | ErrorNo|    Error
---------|----------------|--------------------------------|--------|--------------------------
Processed Schema Changes:
Add      |TcClass         |BFormStorageClass               |       0|Success
---------|----------------|--------------------------------|--------|--------------------------

Unprocessed Schema Changes:
---------|----------------|--------------------------------|--------|--------------------------

Processed Non-Schema Changes:
Add      |TcStandardType  |BFormStorageClass               |       0|Success
Change   |TcForm          |AForm                           |       0|Success
---------|----------------|--------------------------------|--------|--------------------------

Unprocessed Non-Schema Changes:
---------|----------------|--------------------------------|--------|--------------------------

Memory=10181601, SQL=5123, Time 1.109000s cpu, 3.093000s real at-Statistics
```

Figure 23 - Sample deploy log

The file is divided into four sections and each section may contain add, change, or delete tags.

Processed Schema Changes:

A list of classes and attributes that were successfully added, deleted, or changed.

Unprocessed Schema Changes

A list of classes and attributes that were not successfully added, deleted, or changed.

Processed Non-Schema Changes

A list of all other BMIDE definitions that were successfully added, deleted, or changed.

Unprocessed Non-Schema Changes

A list of all other BMIDE definitions that were not successfully added, deleted, or changed.

The log file is provided for your benefit. You should always scan this file and look for any “Unprocessed Schema Changes” or “Unprocessed Non-Schema Changes”. Unprocessed elements can occur when you try to perform an action in the BMIDE that cannot be successfully reconciled in the database.

For example, assume you defined a deployed a custom class. You also used the RAC client to create instances of the custom class in the database. You decide that you do not need the class anymore, so you delete the class definition in the BMIDE and deploy to the test database. In this case, the log file will show the class name in the “Unprocessed Schema Changes” area with an error message. The error message gives you an idea why the element could not be processed. In this case it may say that the class or type is referenced or that the class/type has instances. See Figure 24.

```
-------------------------------------------
 Update Option   : all
 Log File Name   : C:\Temp\gcbobd2TierTransientVolume\foundation_model_12-Mar-2008_22-01-23.log
 Model File Name : D:\Programs\UGS\TEAMCE~1\Tc2007\tcdata\model\delta.xml
 Run Mode        : upgrade
 Process Option  : all
-------------------------------------------
ERROR: Exception thrown while processing "AFormStorage" of type "TcClass"
ERROR: Continuing with other elements


Operation| Element Type   |Element Name                    | ErrorNo|    Error
---------|----------------|--------------------------------|--------|-------------------------------------------------
Processed Schema Changes:
---------|----------------|--------------------------------|--------|-------------------------------------------------

Unprocessed Schema Changes:
Delete   |TcClass         |AFormStorage                    |  226018|Internal error in BMIDE module
Delete   |TcClass         |BFormStorageClass               |  226036|Class is referenced by a type
---------|----------------|--------------------------------|--------|-------------------------------------------------

Processed Non-Schema Changes:
Delete   |TcStandardType  |AFormStorage                    |       0|Success
---------|----------------|--------------------------------|--------|-------------------------------------------------

Unprocessed Non-Schema Changes:
Delete   |TcForm          |AForm                           |   28023|Form type instance referenced : Cannot Delete
Delete   |TcStandardType  |BFormStorageClass               |      -1|BFormStorageClass ( TcStandardType ) is referenc
         |                |                                |        |ed
---------|----------------|--------------------------------|--------|-------------------------------------------------

Memory=8615933, SQL=652, Time 15.953000s cpu, 83.000000s real at-Statistics
```

Figure 24 - Sample deploy log with unprocessed elements

In this case to resolve the deploy error you have to paths.

1) Add the class back to your template if you want to keep the instances and then re-deploy return the database XML back to normal.
2) If you want to remove the class, use the RAC client to query and delete all instances of the class. Then use the BMIDE to redeploy the template again. The subsequent deployment process the class deletion and show the deletion in the deployment log in the “Processed Schema Changes” list.

Continue working with all unprocessed elements until there are no longer any definitions in the “Unprocessed” areas.

6.1.3 Blocking Hot Deploy

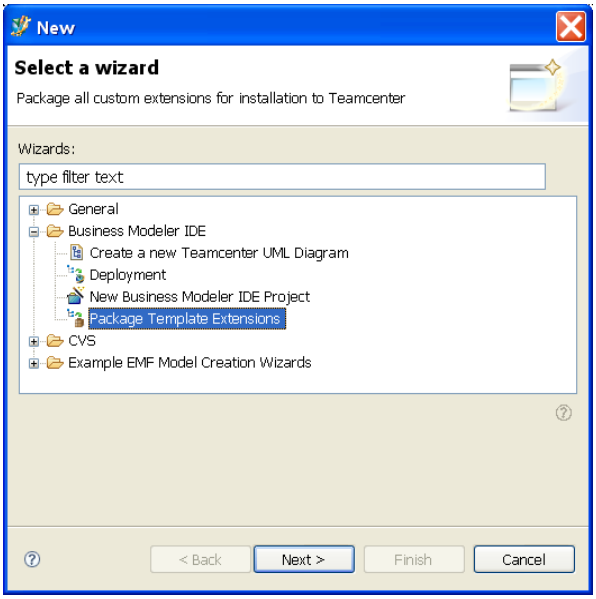
Although a hot deploy is very useful for deploying your custom template definitions to a test database, there are cases where a hot deploy is undesirable. For example, you may want to guard a production system against a hot deploy. Such a change to the system data model or behavior could potentially confuse users who are working on product data midstream. Instead, schedule a system downtime and notify users of an update to the system.

To block hot deploy in any database, use the rich client to set the BMIDE_ALLOW_DEPLOYMENT_FROM_CLIENT preference to FALSE. This prevents any BMIDE client from deploying changes directly to the database. By default the preference is always set to TRUE. When this preference is set to FALSE, a system administrator is force to use TEM to install the template, as covered in section 6.2.

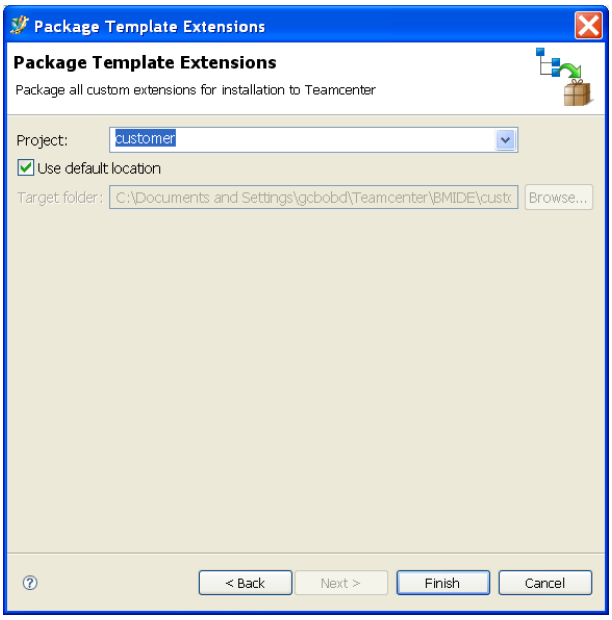
6.2 Package Wizard and TEM Installation

Another option for deploying your custom template is to package it into a feature. The package wizard can build your template into a feature that is installable through TEM.

To launch the Package Template Extensions wizard, go to File->New->Other. In the “Business Modeler IDE” folder, select the *Package Template Extensions* wizard.



On the Package Template Extensions page, select the project from the combo box field and then click Finish.



After you click Finish, the packaging wizard does the following:

- Prompts the user to save the project if there are any unsaved changes.
- Copies the feature_<template_name>.xml and <template_name>Bundle_<language code>_<country_code>.xml files to the <Project>/output/packaging directory.
- Creates a <Project>/output/packaging/<template_name>_template.zip with the following contents:
    - Consolidates all template source file definitions into a single <template_name>_template.xml file and adds it to the zip file.
    - The <Project>/extensions/dependency.xml is copied into the zip file.
    - If a <Project>/<template_name>_tcbaseline.xml file exists, it is copied to the zip file.
- Creates a <Project>/output/packaging/<template_name>_install.zip with the following contents:
    - Copies the <Project>/install/install_<template_name>.default file into the zip file.
    - Copies all the <Project>/install/upgrade_<template_name>_<version>.default files into the zip file.
    - Copies any other data files in the <Project>/install directory to the zip file.

When the packaging wizard is complete, the following four files are located in the <Project>/Output/packaging directory (see Figure 25):

feature_<template_name>.xml

<template_name>Bundle_<language_code>_<country_code>.xml

<template_name>_template.zip

<template_name>_install.zip

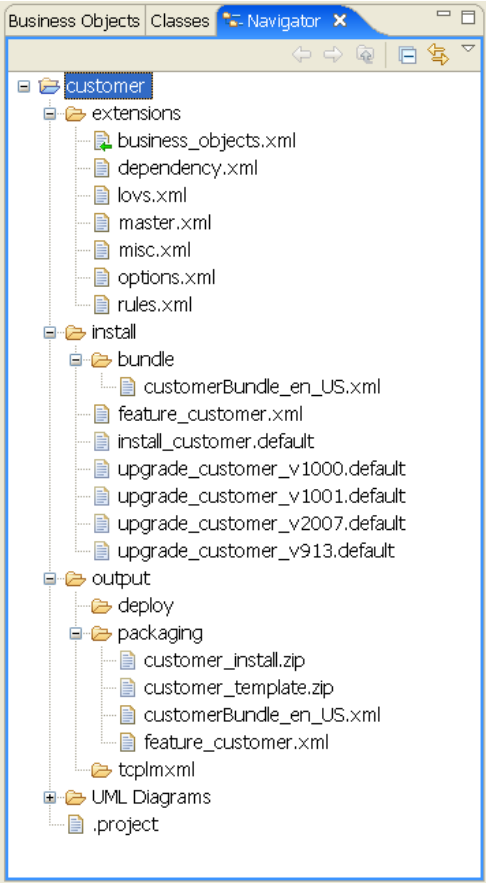


Figure 25: Navigator view showing the results of the Package Template Extensions wizard

6.2.1 Distributing the Template Feature

Now that the template feature files are created, you are ready to distribute your custom template to any site, partner, or customer. Because TEM needs these four files locally to install your template on the server, you need to copy these files to the server or burn them to a portable drive or CD to transport them to the machine.

6.2.2 Distributing Custom Utilities

If you wrote any custom utilities that are executed within the installation or upgrade scripts, you must manually place the appropriate platform binaries in the TC_ROOT/bin directory before using TEM to install or upgrade the template feature.

6.3 How to Use TEM to Add Your Custom Template

For details on how to use TEM to add your customer template to a Teamcenter, see the BMIDE User Guide.

6.4 How to Use TEM to Update Your Custom Template

For details on how to use TEM to update your customer template to a Teamcenter, see the BMIDE User Guide.

6.5 How to make live updates to a production environment

For details on how to make live updates to a production environment, see the BMIDE User Guide.

6.6 Restrictions on Deploying Templates

This section describes the scenarios when TEM or Hot Deploy should be used to deploy your template to the database and the restrictions that apply to Hot Deploy of your template.

6.6.1 When to use TEM Deploy

6.6.1.1 Production Environment

Always use TEM to install your template to a Production environment. Updates to your template must also be done using TEM except while deploying operational data as stated in section "*How to make live updates to a production environment*" of this guide.

There is a scheduled downtime during TEM install or update of a template, which ensures that all users are off the system and you get exclusive access to the Production environment. With exclusive access, all the schema edits to your template can be deployed successfully.

During install and update of your template, TEM executes the install scripts for your template. This ensures that all non-BMIDE managed data model elements like Transfer Modes, Process Templates, Organization information etc are deployed to the database. TEM copies your template specific libraries to various directories in the TC_ROOT so that they are available when you deploy your new data model definitions.

6.6.1.2 Test Environment

In a Test environment, you will be required to use TEM if your template contains data model changes that are stated in the section "*Restrictions on using Hot Deploy to a Test system*" of this guide.

NOTE: Very often a new template would contain schema changes, dependency on data in install scripts and template specific libraries. Hence it is recommended to use TEM instead of Hot Deploy to install your new template to a Test system.

6.6.2 When to use Live Update

6.6.2.1 Production Environment

Live update to a Production environment must follow the guidelines outlined in section "*How to make live updates to a production environment*" of this guide.

6.6.2.2 Test Environment

The BMIDE manages around 100 elements like Classes, Attributes, Business Objects, Properties, Status, Unit of Measure, LOVS, Rules, etc. Live Deploy can be used to deploy a template to Test environment for most use cases.

First, it is important to understand the difference between schema and non-schema. The term "schema" refers to Classes and Attributes managed by the BMIDE template. The term "non-schema" refers to all elements managed by the BMIDE template except for Classes and Attributes. Examples of non-schema are Business Objects, Properties, Status, Unit of Measure, LOV, Rules etc.

Schema edits refers to add/modify/delete of custom attributes, and add/modify/delete of custom classes. You can Live Deploy most schema edits except for the restrictions mentioned in section "*Restrictions on Live Deploy of schema edits*" of this guide. Use TEM to deploy your template when you have such restricted schema edits in your template.

You can use Live Deploy for deploying non-schema elements except if they have dependency on some data in the install script and template specific libraries. For example, if you created a new Application Interface business object that uses a Transfer Mode which is not already in your database, then you cannot Live Deploy your template. You must ensure that the Transfer Mode exists in the database by executing the install scripts. In such cases use TEM to deploy your template.

6.6.3 Restrictions on using Live Deploy (Hot Deploy) to a Test system

This section describes the restrictions that apply in using Live Deploy to a Test system.

6.6.3.1 General Restrictions

1) Cannot use Live Deploy if template deploys require execution of install scripts to add the dependent definitions like Transfer Modes, Process Templates, Organization information etc to the database.
2) Cannot use Live Deploy if template deploys require template specific libraries to be available in the TC_ROOT.

6.6.3.2 Restrictions on Live Deploy of schema edits

1) You cannot use Live Deploy to perform schema edits to classes whose instances are loaded during SOA connection(in general Teamcenter login). Use TEM to deploy these changes.
    a. Following are the list of classes that cannot be edited during Live Deploy. But you can perform schema edits on its sub-classes. The below list is for Teamcenter 8.3.0 and it may vary in future releases.

AM_ACE	AM_ACL	AM_named_tag
AM_privileges	AM_tree	BusinessRule
Condition	ConditionParameter	Constant
ConstantAttach	Dataset	DatasetType
EffectivityMode	EncryptionKey	EventTypeMapping
Fnd0BMIDEResource	Fnd0DeployVersion	Folder
GlobalConstant	GlobalConstantAttach	Group
GroupMember	GroupSecurityNamedTag	ImanCompoundPropDef
ImanEventType	ImanFile	ImanGRM
ImanType	ImanVolume	ListOfValues
ListOfValuesString	ListOfValuesTag	NameField
NameRule	Person	POM_application_object
POM_data_manager	POM_group	POM_imc
POM_member	POM_object	pom_session
POM_system_class	POM_user	ProjectPreference
PropertyConstant	PropertyConstantAttach	RevisionAnchor
RevisionNameRule	RevisionNameRuleAttach	Role
RoleInSchedule	TaskInbox	TC_Preferences
TypeConstant	TypeConstantAttach	User
User_Inbox	WorkspaceObject	

b. If you have any custom classes whose instances are loaded during SOA connection or a Teamcenter login, you cannot perform schema edits to these custom classes and its parent classes using Live Deploy. But you can edit its sub-classes.

2) You cannot Live Deploy a new attribute to an existing class if -

   a. The new attribute is of reference type. For example, you cannot add new attributes of type TypedReference / UntypedReference / ExternalReference.

   b. The new attribute does not allow nulls. For example, you cannot add a new attribute with “Is Nulls Allowed” flag set to false.

   c. The new attribute is added to a non-leaf class. A non-leaf class is a class that has children. A leaf class is a class that has no children. For example, you cannot add a new attribute to “Item” class because it is a non-leaf class.

   NOTE: If you are the only person logged into the database (exclusive access) during Live Deploy, then the above three restrictions in 2a, 2b, 2c for adding new attributes do not apply.

6.7 How to recover from a failed template deployment

This section describes the steps to recover your database when deploy of a template fails. Deploy of a template could have been done using either Hot Deploy from BMIDE client or TEM.

Based on the feedback we have received from our customers using the BMIDE, customers have been restoring the database from a dump after a deploy fails. While we feel that this strategy will work, we would like to suggest an alternative for troubleshooting the failed deploy so that you can identify the issue, fix it and re-deploy the template. This method will be much easier to maintain than restoring the database every time there is a database failure.

Besides the possible reasons provided below, there could be several other reasons for the deploy failure. For example, the Teamcenter server connection could have failed, there might be FMS issues, license issues etc. For the full list of these other possibilities, be sure to read the section titled “*Troubleshooting the Business Modeler IDE → Deployment errors*“in the Business Modeler IDE User Guide.

6.7.1 How do I recognize a failed template deploy?

Before we can describe the recovery process, we have to make sure that you are at the correct point in your development process to use this recovery method. The steps below describe the high level process steps that you must have been followed to reach this failed state.

1) You have installed a BMIDE client and created a custom BMIDE template to contain your extensions to Teamcenter.

2) You may have been working on adding extensions to the template through the BMIDE. You may have already deployed these previous extensions to a Teamcenter server successfully without any issue.

3) You have now added some more extensions, changed or deleted data model elements and deployed your template.

4) The deployment of this version of the template failed.

   a. If you performed Hot Deploy, BMIDE displays an error dialog indicating a deployment failure.

   b. If you performed TEM Deploy, then TEM displays failure message in the TEM dialog.

6.7.2 Understanding the template deployment process

This section provides an overview of the various steps that occur during the template deployment process.

6.7.2.1 Hot Deploy process

The below table shows the steps that occur in the BMIDE Client and Teamcenter Server during Hot deploy.

BMIDE Client	**Server**
1) Package & Upload template files to FMS volume	2) Download files to TC_DATA/model from datasets *****
	3) Download template package from FMS volume
	4) Generate delta.xml
	5) Update the database with delta.xml using business_model_updater
	6) Generate CLIPS rules, PLMXML schema file using the extracted file
	7) Upload files from TC_DATA/model to datasets *****
	8) Upload log files back to FMS volume
9) Download log files from FMS volume	
	*Only from Teamcenter 8.3 onwards

Let's understand these steps in detail.

1) Package & Upload template files to FMS volume
    a. In this step the BMIDE client will generate the new version of the template package and upload this package to FMS volume using a SOA.
2) Download files to TC_DATA/model from dataset *(Applicable from Teamcenter 8.3 onwards)*
    a. In this step, we download files from the dataset instead of relying on files residing in TC_DATA/model folder. For details see section titled "Storage of TC_DATA/model files in dataset"
3) Download template package from FMS volume
    a. In this step, we download the latest version of the template that the BMIDE client uploaded to FMS volume.
4) Generate delta.xml
    a. In this step, the delta.xml is generated using the files in TC_DATA/model
        i. Copy model.xml to model_backup.xml
        ii. Copy model_lang.xml to model_backup_lang.xml
        iii. Consolidate template listed in master.xml to model.xml and model_lang.xml
        iv. Load model_backup.xml and model_backup_lang.xml as old model
        v. Load model.xml and model_lang.xml as new model
        vi. Compare and generate delta.xml

b. Teamcenter 8.3 onwards, if any errors occur during this step, we restore all files in TC_DATA/model from datasets (except for model.xml and model_backup.xml)

5) Update the database with delta.xml using business_model_updater
    a. This is the step where the database is updated with the new data model. The utility business_model_updater updates the database with the contents delta.xml.
    b. If the update of the database fails, the contents of database will be extracted into model.xml and model_lang.xml
6) Generate CLIPS rules, PLMXML schema file using the extracted file
    a. The CLIPS rule file is re-generated if any Condition definitions were updated
    b. The PLMXML XSD is also re-generated
7) Upload files from TC_DATA/model to datasets *(Applicable from Teamcenter 8.3 onwards)*
    a. Finally the contents of TC_DATA/model folder are uploaded back to datasets to ensure it is in sync with the database
8) Upload log files back to FMS volume
    a. All deploy log files are uploaded to the FMS volume
9) Download log files from FMS volume
    a. BMIDE client downloads log files to the client after deployment completes.

6.7.2.2 TEM Deploy process

The below table shows the steps that occur in the TEM and BMIDE Utilities during Hot deploy.

TEM	**BMIDE Utilities**
1) Copy the template files, install scripts to TC_ROOT/install/<template> 2) Copy libraries to various folders in TC_ROOT	3) Download files to TC_DATA/model from datasets ***** 4) Copy the latest template files from TC_ROOT/install/<template> to TC_DATA/model 5) Generate delta.xml 6) Upload files from TC_DATA/model to datasets ***** 7) Run install scripts for all selected templates 8) Update the database with delta.xml using business_model_updater 9) Generate CLIPS rules, PLMXML schema file using the extracted file *****Only from Teamcenter 8.3 onwards

Let's understand these steps in detail.

1) Copy the template files, install scripts to TC_ROOT/install/<template>
    a. In this step the TEM will unzip files in your template package to the folder TC_ROOT/install/<templatename>
2) Copy libraries to various folders in TC_ROOT
    a. TEM will copy the libraries or executables in your template package to respective folders in TC_ROOT
3) Download files to TC_DATA/model from datasets *(Applicable from Teamcenter 8.3 onwards)*
    a. BMIDE utilities will download the current TC_DATA/model files from dataset
4) Copy the latest template files from TC_ROOT/install/<template> to TC_DATA/model
5) Generate delta.xml
    a. In this step, the delta.xml is generated using the files in TC_DATA/model
        i. Copy model.xml to model_backup.xml
        ii. Copy model_lang.xml to model_backup_lang.xml
        iii. Consolidate template listed in master.xml to model.xml and model_lang.xml
        iv. Load model_backup.xml and model_backup_lang.xml as old model
        v. Load model.xml and model_lang.xml as new model

vi. Compare and generate delta.xml

b. Teamcenter 8.3 onwards

i. If any errors occur during this step, all files in TC_DATA/model are restored from datasets (except for model.xml and model_backup.xml).

6) Upload files from TC_DATA/model to datasets *(Applicable from Teamcenter 8.3 onwards)*

a. If delta.xml file generation is successful, files in TC_DATA/model folder are uploaded to the database.

7) Run install scripts for all selected templates

a. If template install, run the install_<template>.script for the template being installed.

b. If template update, run the install_<template>.script for all templates being updated. While running the install script, TEM will skip the call to business_model_updater in the file install_<template>.default

8) Update the database with delta.xml using business_model_updater

a. The utility business_model_updater updates the database with the contents delta.xml

9) Generate CLIPS rules, PLMXML schema file using the extracted file

a. Invoke utility to generate the CLIPS rule file. This utility does the following..

i. Extract data model from database.

ii. Load the extracted data model. If there are loader errors, TEM deploy will report failure.

iii. Generate CLIPS rules using the loaded model.

b. Invoke utility to generate the PLMXML XSD file. This utility does the following..

i. Extract data model from database.

ii. Load the extracted data model. If there are loader errors, TEM deploy will report failure.

iii. Generate required schema file.

6.7.2.3 Storage of TC_DATA/model files in dataset

Starting at Teamcenter 8.3.0, files in TC_DATA/model folder are uploaded and stored in datasets within the database. The files in the dataset are always in sync with your current contents of the database. Thus we now have a reliable copy of the TC_DATA/model in the database to work with when we perform template deployment. This storage of TC_DATA/model files was also done to ensure that Admins don't have to update all TC_DATA/model folders every time they deploy a template using one of the multiple TC_DATAs setup for your database.

The following files from TC_DATA/model are stored in dataset

- TC_DATA/model/*_template.xml
- TC_DATA/model/*_dependency.xml
- TC_DATA/model/lang/*_template_<locale>.xml
- TC_DATA/model/master.xml
- TC_DATA/model/model.xml
- TC_DATA/model/model_lang.xml
- TC_DATA/model/model_backup.xml
- TC_DATA/model/model_backup_lang.xml
- TC_DATA/model/delta.xml

- TC_DATA/model/delta_lang.xml

Caution: There are few scenarios where you would be required to run some utilities manually to ensure that the TC_DATA/model files in datasets is in sync with the contents of the database. Following are the scenarios…

1) When deploy failures occur and you update any of the files listed above, either manually by fixing some elements in the XML file or by copying template files from other locations, then run the below command to update the datasets.

   ```
   manage_model_files –u=<user name> -p=<password> -g=dba –option=upload
   ```

2) When deploy failures occur, if you manually update your database to fix deployment failure issues, you must ensure that you are also updating the files in these datasets. For example, if you run the business_model_updater to fix an issue in your database, then you must run the below commands to keep the dataset files and database in sync.

   a. Extract the latest data model elements from your database

      ```
      business_model_extractor –u=<user name> -p=<password> -g=dba –
      outfile=TC_DATA/model/model.xml
      ```

   b. Upload files to dataset

      ```
      manage_model_files –u=<user name> -p=<password> -g=dba –option=upload
      ```

6.7.3 Best Practices for backing up your template

When we talk about recovery, we also need to talk about best practices for managing the recovery. This begins with ensuring that you perform the following steps each time you deploy your template. This is to ensure that you have a backup of the BMIDE template data for use during the recovery of your database.

1) Keep a copy of each version of the template package that you successfully deployed to the database.

   a. After each successful deploy (using TEM or Hot Deploy), keep a copy of the template package that you successfully deployed to the database. In the case where the deployment errors cannot be fixed, this backup will be used to return to the previous working state of the database.

2) Keep a backup your TC_DATA/model directory.

   a. After each successful deploy (using TEM or Hot Deploy), you must keep a backup of your TC_DATA/model folder. It will be used if there is a need to restore your database from a dump.

3) For Production sites you should backup the database before you deploy.

6.7.4 What steps should you take when deploy fails?

The following steps must be applied when you encounter deploy failures.

1) Review deployment logs
2) Identify the issue and resolution
3) Apply the recovery process

To help explain these steps we will assume the following setup exists.

- There exists a template named “customer” and it is dependent only on “foundation” template.
- Database has the following templates installed
  a. foundation
  b. common (dependent on foundation)
  c. customer

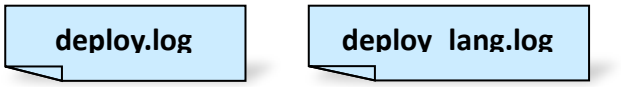
6.7.4.1 Review deployment logs

The first step is to find the correct log file to start investigating.

Hot Deploy:

If the deploy failure occurs during Hot deploy, then you must start by reviewing the deploy log file.

- The log files are located at *<project>/output/deploy/<Server Profile>/<timestamp>.*
    i. For example at, *<project>/output/deploy/tcdata1/2010_11_09_17-39-43*
- For every Hot deploy, a new <timestamp> folder will be created under <project>/output/deploy/<Server Profile>
- The name of the log files are "*deploy.log"* and *"deploy_lang.log".* The "deplo_lang.xml" exists only from Teamcenter 8.3 onwards. From here on, we will refer to these two files as *deploy log files*.

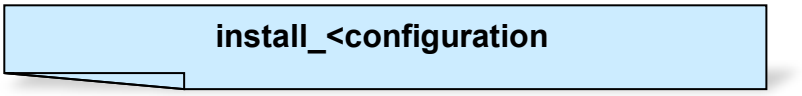


- The deploy log files provide details of all steps executed and the steps that failed during template deployment.
- In addition to the deploy files, this <timestamp> folder contains a copy of the template package that was deployed to your database.
- Prior to Teamcenter 8.3
    i. Sometimes the BMIDE Client does not pop-up a dialog indicating the deployment failure.
    ii. So you will have to take a look at the TAO window and the deploy.log file
    iii. The deploy.log file is a copy of the log file generated by the utility business_model_updater. The deploy.log file is created only if the business_model_updater utility ran during the deploy process. If deployment failure before the execution of business_model_updater, say during generation of delta.xml, then deploy.log files is not created.
    iv. A link to the deploy.log file is added in the Console View if deploy.log is generated.
- Teamcenter 8.3 onwards
    i. BMIDE Client always displays a pop-up a dialog indicating deployment failed.
    ii. The deploy.log file is always generated and has detailed steps of execution.
    iii. The deploy_lang.log is also generated and this contains the results of adding localizations to your database
    iv. Links to deploy.log and deloy_lang.log files are added in the Console View.

TEM Deploy:

If the deploy failure occurs during TEM deploy, then you must start by reviewing the install log file.

- The install log file is located at *TC_ROOT/install.*
- The name of the log file is "*install_<configuration ID>_yyMMddHHmm.log*".
    i. *<configuration ID>:* Indicates the name of the configuration from which you deploy the template.
    ii. *yyMMddHHmm:* Indicates the data and time of deploy.
    iii. For example, install_MYDB_1104281043.log.



- From here on, we will refer to "*install_<configuration ID>_yyMMddHHmm.log*" as *install log file*.
- The install log file has a record of the steps executed during deploy and the errors encountered during deploy.
- If there are more than one install log in TC_ROOT/install, ensure that you pick the file that matches with the date and time of your failed deploy.

6.7.4.2 Identify the issue and resolution

Once you have found the correct log file to start investigations, your next step is to identify the issue and fix it.

We will broadly classify the deployment failure issues into the following categories and discuss how to identify and fix each of these error categories -

1) Loader / Model errors
2) Updater errors
3) Extractor errors

The below table shows the types of error that can occur in in the Hot Deploy steps.

BMIDE Client	Server
1) Package & Upload template files to FMS volume	2) Download files to TC_DATA/model from datasets *
	3) Download template package from FMS volume
	4) Generate delta.xml **[Loader errors]**
	5) Update the database with delta.xml using business_model_updater **[Updater errors]**
	6) Generate CLIPS rules, PLMXML schema file using the extracted file **[Loader errors & Extractor errors]**
	7) Upload files from TC_DATA/model to datasets*
	8) Upload log files back to FMS volume
9) Download log files from FMS volume	

6.7.4.2.1Loader / Model errors

1) *What are loader errors?*

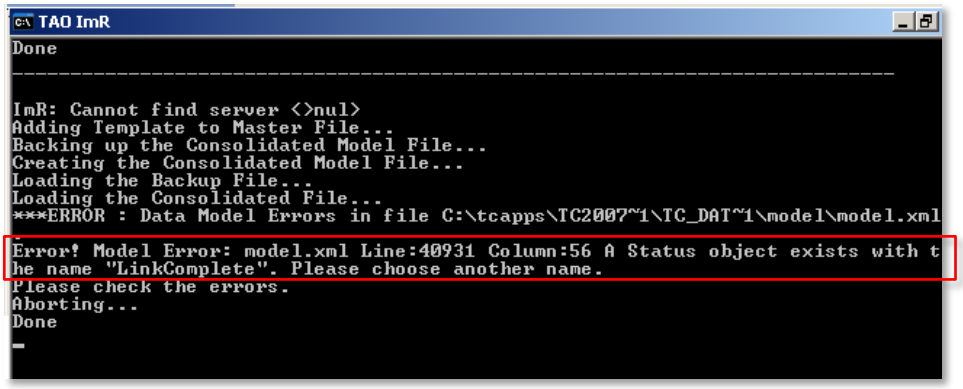
- Errors reported during loading of the data model are termed as Loader errors. The loader errors indicate that the data model does not satisfy the BMIDE validations or does not follow the template XSD.
- During deployment, these are usually seen in generation of delta.xml and in CLIPS/PLMXML schema file generation.
- To generate the delta.xml, the BMIDE utility has to load the latest template XML files (new model) and the data model extracted from the database (old model). If loader errors are reported at this stage it indicates that something in your new version of the template is wrong. Hence you will have to inspect your template to detect and resolve the issue.
- During the generation of CLIPS rules or generation of PLMXML XSD the BMIDE utilities extract the data model from the database and load up the data model. If loader errors occur at this stage, it indicates that after your template was updated in the database something went wrong so you will have to inspect the database.
- Loader errors are also referred as “Model error” or “Parser error”.

2) *Where to start debugging when deploy fails?*

- Hot Deploy

  Prior to Teamcenter 8.3

  - You have to review the TAO window.
  - The deploy log files will not exist because model loading step failed and the database is not yet updated.



  Teamcenter 8.3 onwards

  - Review the deploy.log file at *<project>/output/deploy/<Server Profile>/<timestamp>*

deploy.log

```
------------------------------------------------------------
LogFileName    : Deploy.log
TimeStamp      : 17-Feb-2011_21-13-09
Options        : Full Update
Deploy Source  : BMIDE
------------------------------------------------------------

------------------------------------------------------------
Deploy Summary
------------------------------------------------------------
```

```
Deploy Step 1: Success
Deploy Step 2: Success
Deploy Step 3: Success
Deploy Step 4: Success
Deploy Step 5: Failure
Deploy Step 6: Success
```

deploy.log

```
------------------------------------------------------------
Deploy Step 5:
------------------------------------------------------------
Description: Loads the new model (deployed from BMIDE) and the old
model (previous) to compute the elements that were added, deleted,
or changed in the BMIDE. Verifies that each of these elements are
permitted to be updated by verifying against the Operational Data
List from the database. Writes the differences to the delta.xml
file differences and writes the added, deleted, changed elements
to a delta.xml file.
Loading the Backup File...
Loading the Consolidated File...
Error in Loading backup Model
Result: Failure

------------------------------------------------------------
*** Process Template Files -  Date : Mon Feb 21 23:07:00 GMT 2011
***
------------------------------------------------------------
Adding Template to Master File...
Backing up the Consolidated Model File...
Backing up the Consolidated Model Localization File...
Finding Applicable Locales...
Creating the Consolidated Model File...
Creating the Localization Consolidated Model File...
Loading the Backup File...
Loading the Consolidated File...
***ERROR : Data Model Errors in file
C:\apps\tc\tc830\TD\model\model.xml.
Error! Model Error: C:\apps\tc\tc830\TD\model\model.xml Line:94689
Column:56 Failed to add TcStatus:LinkComplete. A Status object
exists with the name "LinkComplete". Please choose another name.
Aborting...
```

- TEM Deploy
  - Review the install log file at TC_ROOT/logs

install_MYDB_1104281043.log

```
....
....
Output from command:  "C:\Teamcenter\TR\bin\bmide_processtemplates.bat" -
u=infodba -p=***** -g=dba -mode=server -dir=C:\TEAMCE~1\TD\model -
templates=custom -deployvalidation=yes -update=all
Adding Template to Master File...
Backing up the Consolidated Model File...
Backing up the Consolidated Model Localization File...
Finding Applicable Locales...
Creating the Consolidated Model File...
Creating the Localization Consolidated Model File...
```

```
Loading the Backup File...
Loading Localization File
C:\TEAMCE~1\TD\model\lang\model_backup_lang.xml...
Loading the Consolidated File...
Error! Parsing Error: C:\Teamcenter\TD\model\model.xml Line:14574
Column:103 A status with the name "LinkComplete" already exists. Please
choose another name.
Please check the errors.
Aborting...
```

3) Anatomy of the loader error message

Every loader/model error is of the format –

Error Format:

```
Error! Parsing Error:<filename>Line:<line number>Column:<column number>:
<message>
```

Error sample:

```
Error! Parsing Error: C:\Teamcenter\TD\model\model.xml Line:14574 Column:103 A
status with the name "LinkComplete" already exists. Please choose another name.
```

- *"Error! Parsing Error":* This is a static string and indicates this is a model loading validation error.
- <*filename*>: Indicates the name of the file in which the error occurred.
    - In the above example the name of the file is "model.xml" and it is located at "C:\Teamcenter\TD\model".
- <*line number*>: Indicates the line number in the file where the error occurred.
    - In the above example, the line number is indicated as "Line: 14574".
- <*column number*>: Indicates the column number in the file where the error occurred.
    - In the above example, the column number is indicated as "Column: 103".
- <*message*>: A short description of the error.
    - In the above example, the error message is "A status with the name "LinkComplete" already exists. Please choose another name."

4) Analyze and resolve the loader error

- Go to the file specified in the loader error. During template deploy this file is usually model.xml in TC_DATA/model folder.
- Review the element indicated at line <line number> and column <column number>.
- Read the <message> indicated in the loader error and see if the loader error can be fixed.
- During deployment, loader errors are usually reported on model.xml and model_lang.xml or model_backup.xml and model_backup_lang.xml. These files are only generated files. Hence any fixes you make to resolve the loading problem should not be made to these files. Instead you should fix your template XML file.
- Let's analyze and resolve the sample loader error…

```
Error! Parsing Error: C:\Teamcenter\TD\model\model.xml Line:14574
Column:103 A status with the name "LinkComplete" already exists. Please
choose another name.
```

- The error states there are duplicate "LinkComplete" statuses in model.xml. If you look in the model.xml, you would find two TcStatus definition with the same name.

```
<TcStatus
statusName="LinkComplete"
description="Status during
```

- However model.xml file is a generated file (consolidation of individual templates)
- To find the templates that define "LinkComplete", you have to search in all the "***_template.xml**" or "***_template_<locale>.xml**" files located in TC_DATA/model folder
- A search in TC_DATA/model revealed that the TcStatus "LinkComplete" was found in two templates
  1. commom_template.xml
  2. customer_template.xml
- This means the status "LinkComplete" was already defined in the "common" template and deployed to the database. Hence you have to remove the "LinkComplete" definition from your "customer" template.
- Resolution
  1. Go back to BMIDE
  2. Open "customer" template project
  3. Remove the duplicate status "LinkComplete" since it is already defined in "common" template
  4. Re-deploy "customer" template

5) Known loader errors and resolutions

Error Message	`Error! Parsing Error: C:\Teamcenter\TD\model\model.xml Line:14574 Column:103 A status with the name "LinkComplete" already exists. Please choose another name.`
Cause	This error indicates that the model element Status named "LinkComplete" already exists. This means that another template in TC_DATA/model already has this Status defined. The reason BMIDE UI did not detect this when you created "LinkComplete" in your own template project could be because your template was not dependent on the template that already defined this Status. Since the BMIDE only loads your dependent templates, it could not detect the collision before template deploymnet.
Resolution	The fix is to remove the Status called "LinkComplete" from your own template and redeploy the template.

6.7.4.2.2 Updater errors

1) What are updater errors?

- Error that are reported when your database is being updated with the new data model is termed as “Updater error”
- These errors occur only during the execution of the business_model_updater to update your database with the contents in delta.xml. This is the only utility that can update the database with the new data model.
- When you review deploy or install log files, it will indicate that the execution of the above utility failed. It will also point you to additional log files associated with business_model_updater utility.
- Updater errors are reported in log files that has the following naming pattern -
    - business_model_updater_<timestamp>.log
    - business_model_updater*.syslog
    - For example
        - business_model_updater_2010_11_12_07-59-42.log
        - business_model_updater980a134.syslog
- Most often just reviewing the “*.log” file is sufficient to decipher the issue. If you need more information, it is a good idea to review the updater “*.syslog”.
- There are some common updater errors and we can categorize these as below. The details of these error categories and resolutions are discussed in the section “*Known updater errors and resolutions*”.
    - Schema issues
    - Data model deletion issues
    - Missing non-BMIDE managed element issues

2) Where to start debugging when deploy fails?

- Hot Deploy

  Prior to Teamcenter 8.3

    - Review the deploy.log file at *<project>/output/deploy/<Server Profile>/<timestamp>.*
    - The deploy.log is a copy of the business_model_updater_<timestamp>.log
    - The syslog will be located in the $TEMP folder associated with your TcServer or at TC_TMP_DIR

deploy.log

```
=====================================================
 Log File Name   : ..\process\business_model_updater.log
 Model File Name : C:\apps\tc\tc830\TD\model\delta.xml
 Update Option   : all
 Run Mode        : upgrade
 Process Option  : all
=====================================================


=================================================================
Processed Schema Changes:
=================================================================
Action | Element Type    | Element Name      | Error # | Error
-------|-----------------|-------------------|---------|-----------------


=================================================================
```

```
Unprocessed Schema Changes:
=============================================================================
Action | Element Type    | Element Name      | Error # | Error
-------|-----------------|-------------------|---------|------------------

=============================================================================
Processed Non-Schema Changes:
=============================================================================
Action | Element Type    | Element Name      | Error # | Error
-------|-----------------|-------------------|---------|------------------
Add    | TcLOV           | Colors            |       0 | Success

=============================================================================
Unprocessed Non-Schema Changes:
=============================================================================
Action | Element Type    | Element Name      | Error # | Error
-------|-----------------|-------------------|---------|------------------
Delete | TcDataset       | Rx7JPEG           |      -1 | Cannot delete
       |                 |                   |         | Rx7JPEG (TcDataset)
       |                 |                   |         | because it is
       |                 |                   |         | referenced.
```

Teamcenter 8.3 onwards

- Review the deploy.log file at *<project>/output/deploy/<Server Profile>/<timestamp>*
- The content of the updater log file is embedded into the deploy.log itself.
- The syslog will be located in the $TEMP folder associated with your TcServer or at TC_TMP_DIR

deploy.log

```
-------------------------------------------------------------
LogFileName    : Deploy.log
TimeStamp      : 18-Feb-2011_13-18-13
Options        : Full Update
Deploy Source  : BMIDE
-------------------------------------------------------------

-------------------------------------------------------------
Deploy Summary
-------------------------------------------------------------
Deploy Step 1: Success
Deploy Step 2: Success
Deploy Step 3: Success
Deploy Step 4: Success
Deploy Step 5: Success
Deploy Step 6: Success
Deploy Step 7: Failure
```

deploy.log

```
-------------------------------------------------------------
Deploy Step 7:
-------------------------------------------------------------
Description: Process the delta.xml file committing each data model
element to the database.
Executing: business_model_updater.
Result: Failure
```

```
=====================================================
Command output for the bmide_model_updater
=====================================================
 Log File Name   : ..\process\business_model_updater.log
 Model File Name : C:\apps\tc\tc830\TD\model\delta.xml
 Update Option   : all
 Run Mode        : upgrade
 Process Option  : all
=====================================================

=====================================================================
Processed Schema Changes:
=====================================================================
Action | Element Type   | Element Name     | Error # | Error
-------|----------------|------------------|---------|-----------------

=====================================================================
Unprocessed Schema Changes:
=====================================================================
Action | Element Type   | Element Name     | Error # | Error
-------|----------------|------------------|---------|-----------------

=====================================================================
Processed Non-Schema Changes:
=====================================================================
Action | Element Type   | Element Name     | Error # | Error
-------|----------------|------------------|---------|-----------------
Add    | TcLOV          | Colors           |       0 | Success

=====================================================================
Unprocessed Non-Schema Changes:
=====================================================================
Action | Element Type   | Element Name     | Error # | Error
-------|----------------|------------------|---------|-----------------
Delete | TcDataset      | Rx7JPEG          |      -1 | Cannot delete
       |                |                  |         | Rx7JPEG (TcDataset)
       |                |                  |         | because it is
       |                |                  |         | referenced.
```

- TEM Deploy
  - Start by reviewing the install log file located at TC_ROOT/install
  - The install log file will tell you the specific updater log file to investigate
  - In the example install log file below, it referring to the updater log file named "**business_model_updater_26-May-2009_16-17-34.log**". This is the file you must review to understand why the updater failed.

install_MYDB_0905261043.log

```
....
....
Executing 'install -regen_schema_file infodba ***** dba
>>"D:\SPLM\Teamcenter\TC_AMER_DEVL\Tc8.3\logs\tc_install.log"'...
Output from command:  install -regen_schema_file infodba ***** dba
>>"D:\SPLM\Teamcenter\TC_AMER_DEVL\Tc8.3\logs\tc_install.log"
command_exit=0
Exit Status  0, elapsed time 0:00:03

Executing 'business_model_updater -u=infodba -p=***** -g=dba -
file=C:\TEAMCE~1\TD/model/delta.xml -update=all -mode=upgrade
>>"C:\TEAMCE~1\TR\logs\tc_install.log"'...
```

```
Output from command:  business_model_updater -u=infodba -p=***** -g=dba -
file=C:\TEAMCE~1\TD/model/delta.xml -update=all -mode=upgrade
>>"C:\TEAMCE~1\TR\logs\tc_install.log"
Unprocessed schema and/or non-schema updater transactions exist.
Please see updater log at:
C:\Teamcenter\TR\logs\business_model_updater_26-May-2009_16-17-34.log for
status of processed element/s.
```

3) Anatomy of the updater error message

Let us now understand the anatomy of the log file *business_model_updater_<timestamp>.log*

business_model_updater_26-May-2009_17_17_34.log

```
-----------------------------------------------------
 Log File Name   : ..\process\business_model_updater.log
 Model File Name : C:\apps\tc\tc830\TD\model\delta.xml
 Update Option   : all
 Run Mode        : upgrade
 Process Option  : all
-----------------------------------------------------

=====================================================================
Processed Schema Changes:
=====================================================================
Action | Element Type   | Element Name     | Error # | Error
-------|----------------|------------------|---------|-----------------

=====================================================================
Unprocessed Schema Changes:
=====================================================================
Action | Element Type   | Element Name     | Error # | Error
-------|----------------|------------------|---------|-----------------

=====================================================================
Processed Non-Schema Changes:
=====================================================================
Action | Element Type   | Element Name     | Error # | Error
-------|----------------|------------------|---------|-----------------
Add    | TcLOV          | Colors           |       0 | Success

=====================================================================
Unprocessed Non-Schema Changes:
=====================================================================
Action | Element Type   | Element Name     | Error # | Error
-------|----------------|------------------|---------|-----------------
Delete | TcDataset      | Rx7JPEG          |      -1 | Cannot delete
       |                |                  |         | Rx7JPEG(TcDataset)
       |                |                  |         | because it is
       |                |                  |         | referenced.
```

- **Header**
  - The updater log file header indicates the XML file and the options used during database update

- **4 sections**
    - This updater log file has 4 sections as shown in the table below.
    - Whenever you review the updater log, you must first check if any errors are reported in the sections “Unprocessed Schema Changes” OR “Unprocessed Non-Schema Changes”. If there are errors then you need to worry about your deployment failure. Otherwise it means updater was successful.

Section Name	Purpose
Processed Schema Changes	This section lists schema elements (classes and attributes) that were successfully deployed to the database.
Unprocessed Schema Changes	This section lists the schema elements (classes and attributes) that failed to get deployed to the database. If the updater encounters at least one error in the “Unprocessed Schema Changes” section, then the updater will not process any of the non-schema elements in the delta XML file (like LOV, Business Objects, Rules, etc)
Processed Non-Schema Changes	This section lists non-schema elements (Business Objects, LOV, Rules) that were successfully deployed to the database
Unprocessed Non-Schema Changes	This section lists non-schema elements (Business Objects, LOV, Rules) that failed to get deployed

- **Table in each section**
    - Each of the section contains a table listing the success/failure status of elements updated in the database. Let’s understand the various columns of this table.

Column Name in the log file	Purpose
Action	Indicates if the element is being added, changed or deleted. The only values seen in this column are: Add → Indicates element has to be added Change → Indicates element has to be changed Delete → Indicates element has to be deleted
Element Type	Indicates the category of the element being added. This value usually matches with the XML tag name in delta.xml. For e.g., if a new LOV is being added to the database then this column has the value “TcLOV”.
Element Name	Indicates the name (unique identifier) of the element being processed. For e.g. if we are processing an element of type TcTool, then this column displays the name of the tool. For e.g. “AcroRd32”

	In the above sample, name of LOV is “Color” and name of Dataset is “Rx7JPEG”
Error #	Indicates the error number that can be later used to obtain additional information In the above sample, the error number is “-1”
Error	Indicates the error message. This is a short description of the issue that caused this element processing to fail. In the above sample, the error message indicates that dataset could not be added because it is referenced

4) Analyze and resolve the updater error

- For TEM deploy, review the business_model_updater_<timestamp>.log. For Hot deploy, review contents of deploy.log.
- Updater errors will be reported in the sections “Unprocessed Schema Changes” and “Unprocessed Non-Schema Changes”. Review each of the error recorded in these sections and find a resolution. Sometimes there might be multiple errors due to a single issue. So you must carefully read the errors messages and find the root cause.
- Let’s analyze and resolve the sample updater error…

```
....

================================================================================
Processed Non-Schema Changes:
================================================================================
Action | Element Type    | Element Name         | Error # | Error
-------|-----------------|----------------------|---------|----------------------
Delete | OperationInput  | Rx7JPEGCreI          |       0 | Success
       | Type            |                      |         |

================================================================================
Unprocessed Non-Schema Changes:
================================================================================
Action | Element Type    | Element Name         | Error # | Error
-------|-----------------|----------------------|---------|----------------------
Delete | TcTypeConstant  | CreateInput:Rx7JPEG  |      -1 | The Create Input object
       | Attach          |                      |         | for type already exists
       |                 |                      |         | in the database
Delete | TcDataset       | Rx7JPEG              |      -1 | Cannot delete Rx7JPEG
       |                 |                      |         | (TcDataset) because
       |                 |                      |         | it is referenced.
```

  - There are two errors reported in the “Unprocessed Non-Schema Changes” section. Everything is related to deletion of “Rx7JPEG” dataset.
  - The root cause is that the dataset we want to delete is referenced. It means that the database contains instances of this Dataset type.
  - If we are deleting any Business Object then we must ensure that there are no instances of this Business Object in the database. In our sample use case, users had created instances of the Dataset type “Rx7JPEG”.

- To resolve this error, we have to delete all dataset instances and then redeploy. The steps to resolve this issue is discussed in detail in the section "*Known updater errors and resolution*".

5) Known updater errors and resolutions

Schema issues

These are usually seen during Hot Deploy. There are certain restrictions that apply while deploying schema changes using Hot Deploy (see section 8.6). In such cases, you can resolve the issue by re-deploying your template (without any changes) using TEM.

Error Message	``` ========== Unprocessed Schema Changes: ========== Action |Element Type |Element Name |Error# | Error -------|----------|----------|---------|---------- Add |TcAttributeAttach |Folder:rx7Attr1 | 515062 | The given | | | class is | | | referenced. Delete | TcClass | Rx7MyDS | 226055 | Not processed | | | | because some | | | | some schema | | | | updates | failed. ```
Cause	The log indicates that there are 2 errors in the "Unprocessed Schema Changes" section – The addition of an attribute "rx7Attr1" to the Class "Folder". The deletion of a Class called "Rx7DS" has failed. We need to determine why they were unprocessed. The root cause is that we are adding a new attribute "rx7Attr1" to Folder class. The "Folder" class is a bootstrap class that cannot be updated during Hot deploy. For more details see section titled "Restrictions on Hot Deploy". Due to this restriction, the attribute could not be added during a Hot Deploy. Instead it has to be deployed using TEM because schema changes to bootstrap classes require exclusive access to the database.
Resolution	The resolution is to package the template and deploy it using TEM.

Data model deletion issues

The most common reason for failures to occur during deletion of data model elements is because they are referenced by another data model element or instances are created in the database.

Error Message	``` ========== Unprocessed Non-Schema Changes: ========== Action |Element Type |Element Name |Error# | Error -------|----------|----------|---------|---------- Delete |TcTool | AcroRd32 | 226049 | AcroRd32 (TcTool) | | | | is referenced ```
Cause	The log indicates that there are errors in the "Unprocessed Non-Schema Changes" section – The tool named "AcroRd32" could not be deleted from the database. We need to determine why they were unprocessed. The root cause of this issue is that tool "AcroRd32" is referenced in other elements. The elements where tool is references are in the Dataset and

	Application Interface data model definitions. Due to this the tool cannot be deleted. You must ensure that all references to this tool are removed in your Dataset and Application Interface elements before deploying.
Resolution	To resolve this issue you have two options – 1) Go back to BMIDE, re-add the tool and deploy the template OR 2) Go back to BMIDE and remove references to this tool from your Dataset and Application Interface definitions. Then deploy your template.

Error Message	``` == Unprocessed Non-Schema Changes: == Action |Element Type |Element Name |Error# | Error -------|--------------|-------------|---------|---------------------- Delete |TcTool | AcroRd32 | 226049 | AcroRd32 (TcTool) | | | | is referenced Delete | TcTypeConstant| CreateInput | -1 | The Create Input | Attach | :Rx7JPEG | | for type already | | | | exists in the | | | | database Delete | TcDataset | Rx7JPEG | -1 | Cannot delete Rx7JPEG | | | | (TcDataset) because | | | | it is referenced. ```
Cause	The log indicates that there are errors in the “Unprocessed Non-Schema Changes” section – 1) The tool named “AcroRd32” could not be deleted from the database. We need to determine why they were unprocessed. 2) The Type Constant Attach for Constant “CreateInput” and Business Object “Rx7JPEG” could not be deleted 3) The dataset named “Rx7JPEG” could not be deleted. The root cause of this issue is that there are instances in the database for the dataset “Rx7JPEG”. Hence the dataset definition cannot be removed from the database. To resolve this issue you must deleted all instances of this dataset from the database before deploying again. The tool “AcroRd32” could not be deleted because it is still referenced by dataset “Rx7JPEG”. The tool “AcroRd32” can be deleted only if the dataset “Rx7JPEG” is deleted from the database. The Type Constant Attach could not be deleted because an updater issue. This is being fixed.
Resolution	To resolve this issue do the following.. 1. Shutdown BMIDE 2. Launch RAC, find and delete all Dataset instances of the type “Rx7JPEG” 3. Shutdown RAC. If using 2-tier close the TAO window 4. Prepare cleanup XML to fix database

	5. Copy delta.xml to cleanup_delta.xml
	6. Open cleanup_delta.xml
	7. In the <Delete> tag, move the XML tags <TcTypeConstantAttach> to the beginning. These should be the very first tags under <Delete>
	8. Save and close cleanup_delta.xml
	9. Run updater to fix the database
	a. Open a Teamcenter command shell and run the command.
	`business_model_updater -u=<user name> -p=<password> -g=<dba> -mode=upgrade -update=all -process=delete -file=$TC_DATA/model/cleanup_delta.xml`
	b. Ensure that the updater log file has not failures
	10. Sync datasets and DB
	a. Run extractor to extract the DB contents into TC_DATA/model/model.xml
	b. Run manage_model_files to update the database
	`manage_model_files -u=<user name> -p=<password> -g=<dba> -option=upload`
	11. Finally, launch BMIDE and perform Live Update to ensure the database is in sync with your project.

Missing non-BMIDE managed element issues

There are certain data model elements in the BMIDE template that can refer to objects in the database which are not managed by the BMIDE. For example a Classic Change element in the BMIDE has references to Process Template names. An Application Interface business object has references to Import Transfer Mode. An IRDC in the BMIDE has references to Create Template. The objects Process Template, Import Transfer Mode, Create Template are not managed by the BMIDE. These are termed as non-BMIDE managed elements. Such objects must be pushed into the database before the template gets processed by the updater. The expectation is that the install scripts (install_<template name>.default) have utilities that create these non-BMIDE managed objects.

Sometimes when you deploy your template, these non-BMIDE managed objects might not be present in the database. This could be because the utilities that push them to the database did not get executed or utilities were never updated to push in these objects.

During Hot deploy the install scripts that pushes in these elements to the database are not executed. Hence it is common to see errors in the updater indicating that the dependent non-BMIDE managed object does not exist in the database. In such cases, you can deploy using TEM, since TEM executes the install script to add the non-BMIDE managed objects to the database.

Error Message	```
	==========
	Unprocessed **Non-Schema** Changes:
	==========
	Action |Element Type |Element Name |Error# | Error
	-------|-------------|-------------|--------|-----------------------
	Add |TcChange | Rx7CN | -1 |Process template name
	       | | | |"Rx7ChngNotice" does not
	       | | | |exist
	       | | | |
	Add |TcAppInterface| Rx7AppI | 226331|rx7incr import
	``

	| | | |(importTransferMode) | | | |required by Rx7AppI | | | |(TcAppInterface)does | | | |not exist
Cause	The log indicates that there are errors in the “Unprocessed Non-Schema Changes” section – A new Classic Change named “Rx7CN” could not be added to the database. A new Application Interface named “Rx7AppI” could not be added to the database. We need to determine why they were unprocessed. The Classic Change “Rx7CN” refers to the Process Template “Rx7ChngNotice” which is a required for adding Classic Change. Process Template is a non-BMIDE managed element. By the time the updater processes your Classic Change elements, it expects all such elements to reside in the database. Since the Process Template was not in the database, the updater did not add the Classic Change to the database. The Application Interface “Rx7AppI” could not be added for a similar reason. It requires the transfer mode name “rx7incr_import” and this transfer mode is not in the database. The import and export transfer modes are non-BMIDE managed elements and these should have been added to the database before the updater started processing the data model elements.
Resolution	The resolution is to include commands in your template’s install script – install_<template>.default so that the new the Process Template “Rx7ChngNotice” and the Import Transfer mode “rx7incr_import”. Next, package your template and perform TEM deploy. You cannot perform Hot Deploy because the install scripts are not executed during Hot Deploy.

6.7.4.2.3 Extractor errors

1) What are extractor errors?

- Errors reported when extracting the data model from the database is termed as “Extractor error”.
- These errors occur only during the execution of the utility business_model_extractor since this is the only utility that can extract the data model from the database.
- When you review deploy or install log file, it will indicate that the execution of the above utility failed. It will also point you to additional log files associated with business_model_extractor utility.
- The extractor errors are reported in log files with naming pattern.. -
    - business_model_extractor_<timestamp>.log
    - business_model_extractor*.syslog
    - For example
        - business_model_extractor_2010_11_12_06-54-09.log
        - business_model_extractor284052a.syslog
- You should always start by reviewing the extractor “*.log” file. If you need more information, it is a good idea to review the extractor “*.syslog” and also re-run the failed extractor command by setting extra journaling or logging variables.

2) Where to start debugging when deploy fails?

- Hot Deploy
  - If the extractor has failed then the TAO window and the deploy log file will indicate the extractor failure and specify the log file path.
  - The extractor log and syslog will be located in the $TEMP folder associated with your TcServer or at TC_ROOT/logs
  - The extractor log files will be located in the platform temp folder or TC_ROOT/logs on the server machine.
  - *Sample error when extractor has failed*

```
....

Exrtacting ...
 *** Errors occurred during extraction of TcTool ***
business_model_extractor failed, see the extractor log and syslog for
details.
Please refer to
[C:\Teamcenter\TR\logs\business_model_extractor_2010_04_01_19-31-47/log]
for log information.
```

- TEM Deploy
  - Start by reviewing the install log file located at TC_ROOT/install
  - The install log file will tell you the specific extractor log file to investigate
  - In the example install log file below, it is referring to the extractor log file named "**business_model_extractor_2010_04_01_19-31-47.log**". This is the file you must review to understand why the extractor failed.

```
...
Executing 'install -regen_schema_file infodba ***** dba
>>"D:\SPLM\Teamcenter\TC_AMER_DEVL\Tc8.3\logs\tc_install.log"'...
Output from command:  install -regen_schema_file infodba ***** dba
>>"D:\SPLM\Teamcenter\TC_AMER_DEVL\Tc8.3\logs\tc_install.log"
command_exit=0
Exit Status  0, elapsed time 0:00:03

Executing 'business_model_extractor -u=infodba -p=***** -g=dba -
outfile=C:\TEAMCE~1\TD\model\model_backup.xml
>>"C:\TEAMCE~1\TR\logs\tc_install.log"'...
Output from command:  business_model_extractor -u=infodba -p=*****
-g=dba -outfile=C:\TEAMCE~1\TD\model\model_backup.xml
>>"C:\TEAMCE~1\TR\logs\tc_install.log"
Extracting....

**** Errors occurred during extraction of TcTool ****
business_model_extractor failed, see the extractor log and syslog
for details
Please refer
[C:\TEAMCE~1\TR\logs\business_model_extractor_2010_04_01_19-31-
47.log] for log information
```

3) Anatomy of the extractor error

Here is a sample error from an extractor log file..

```
...
...
*****************************************************************
Start Extracting PSOccurrenceType.....
End Extracting PSOccurrenceType.....
*****************************************************************
Start Extracting StorageMedia.....
End Extracting StorageMedia.....
*****************************************************************
Start Extracting Tool.....
    Failed to get Tool object from the tag
    **** Some Errors occurred during extraction of Tool. Please
    look into syslog ****
End Extracting Tool.....
*****************************************************************
Start Extracting Change.....
End Extracting Change.....

*****************************************************************
```

Error Format

```
*********************************************************
Start Extracting <Element Type> .....
<Error messages>
End Extracting <Element Type> .....
*********************************************************
```

- <*Element Type*>: Indicates the category of the element being extracted. In the sample above, we are extracting all tools. Hence the element type "Tool" is seen.
- <*Error messages*>: This section displays the errors encountered while extracting a specific type of data model element. In the sample above, we see "Failed to get Tool object from the tag".

4) *Analyze and resolve the updater error*

- Open the install and deploy log file to check if the extractor has failed.
- If they indicate that the extractor has failed, then review the extractor log *business_model_updater_<timestamp>.log.*
- Check the log file to find the element whose extractor failed.
- Review the <Element Type> section where the failure occurred.
- If you are unable to decipher the issue, open the *business_model_extractor*.syslog* and analyze the reasons for the failure of the <Element Type> extraction.
- Normally extractor errors never occur. But if they do occur, it is a serious error. There could be several reasons for the extractor failure. For instance, incorrect processing code in the customer libraries or OOTB libraries, corrupted data in the database (due to a previous update), bug in the extractor itself.
- Extractor error most often require fixes to your database. If you are unable to decipher the extractor error and fix your database then please contact GTAC for assistance.

5) Known extractor errors and resolutions

Error Message	** Start Extracting Tool..... Failed to get Tool object from the tag **** Some Errors occurred during extraction of Tool. Please look into syslog **** End Extracting Tool..... **
Cause	The root cause for this issue was that the database was had corrupted Tool definition and the extractor could not extractor the incorrect Tool.
Resolution	SQL commands were executed to fix the Tool corruption in database. These SQL commands were very specific to this database that had the corruption. In most cases where you see such data model corruption issues please contact GTAC.

6.7.4.3 Apply the recovery process

There are 3 options to recover from a failed deployment…

1) Analyze and fix the deploy issue
2) Restore database using from your backup template package
3) Restore database from a database dump

The following sections describe each of the options in detail.

6.7.4.3.1 Analyze and fix the deploy issue

This option must always be your first choice – To find the reason for deploy failure, fix the issue and continue working.

1) Study the various errors we discussed earliest, apply this to your current deploy failure and find the resolution.
    a. If the resolution is to fix your template, then launch BMIDE and fix your template
    b. If the fix is not in your template but in the database, then execute the required utilities to fix your database. Ensure that after you fix your database, you have updated the TC_DATA/model files uploaded to datasets.
2) After the fix ensure that you redeploy your template in BMIDE to the database. This will ensure that your template is in sync with the database.
3) If the second deploy of the updated template succeeds without any errors, then you are done. You are good to start using your database.
4) If the second deploy of the updated template fails, repeat the steps to analyze and fix the issue.

6.7.4.3.2 Restore database using from your backup template package

This option can be your choice if you are unable to analyze and resolve the deploy failure..

1) As a best practice for deploy, we recommended that you always keep a backup of your template package that was successfully deployed to your database.
2) Fetch this template package from your backup and deploy it to the database.
3) You must use TEM to deploy this backup template package. You cannot use Hot Deploy in this case since your BMIDE template project has changed since the backed-up template package was created.
4) If the re-deploy of the backed-up template package succeeds, then you are done.
5) If re-deploy fails, you can follow either Option #1 or Option#3 to resolve the issue.

6.7.4.3.3 Restore database from a database dump

This option can be your choice if you are unable to apply the Option #! And Option #2 to recover your database.

1) Restore your database from a backup dump
2) Restore your TC_DATA/model folder that was backed-up before starting deploy
3) Run the below command to upload the files in TC_DATA/model to datasets. This step is required only if you are working with Teamcenter 8.3 or later releases.

   manage_model_files -=<user> -p=<password> -g=dba –option=upload

NOTE: During the recovery process, if you are planning to use TEM to deploy your template package, here is a caution.

a. When you launch TEM from the maintenance mode, it can display that the previous deploy had failed and hence it will attempt to recover from the failed step. TEM can end up in this scenario if your previous attempt to deploy the template had failed.
b. If you find TEM going into recovery mode and does not provide you an option to select your updated template package, then perform the below steps…
    a. Shutdown TEM.
    a. Unzip your updated template package to a temporary location.
    b. Copy the following files to TC_ROOT/install/<template-name> folder on the server machine:
        i. <template-name>_template.xml
        ii. <template-name>_dependency.xml
        iii. <template-name>_tcbaseline.xml (if the file exists)
    c. Restart TEM and continue.
    d. TEM will now pick up your latest templates from TC_ROOT/install/<template-name>.

7 Development Environments

Now that you know how to use the BMIDE for creating extensions, packaging, and deploying, you may have some questions about best practices for developing, testing, and rolling them out to production. Should you have a test environment? Is an integration environment required? These are some questions that you may be asking. This section gives suggestions about how to set up an environment that suits your needs. Factors that may affect your choice will be the number of users, developers, sites, and hardware availability.

7.1 Single Production Database

The first scenario is the simplest. It consists of a single production database and the BMIDE client. In this environment a single user will use the BMIDE client to create a template for extending the data model. The production database may support one to a small number of users. If no other users are logged on, the BMIDE user can use Hot Deploy to deploy the custom template. If other users are logged on, the users should be notified of a system downtime. When the system is taken down, use the package wizard in the BMIDE to build the template feature, and then use TEM to install/update the feature into the production environment.

This scenario is ideal for a small community of users.

Caution: One disadvantage of this scenario is that the extensions cannot be tested in a safe environment before going live with the production environment.

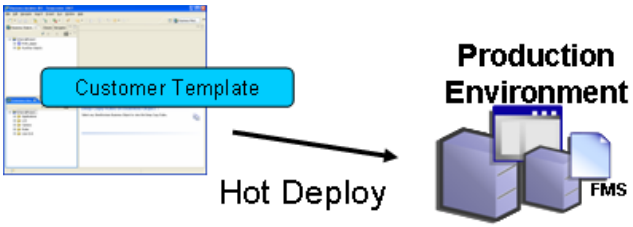


Figure 26: Single production database

7.2 Test and Production Database

This scenario is very similar to the first scenario except that it adds a test environment for the BMIDE user. In this scenario, the BMIDE user creates a template for extending the data model. The BMIDE user hot deploys the template to a test database, where the extensions can be tested. After the extensions have been tested to a satisfactory level, the user community is notified of a system downtime. When the system is taken down, use the package wizard in the BMIDE to build the template feature, and then use TEM to install or update the feature, as the case may be, into the production environment.

This scenario offers a safer process for extending, testing, and rolling out to production than the first scenario. Extensions can be created, deployed, and tested in the safe test environment and the process is reiterated until requirements have been met. This process protects the production environment from any undesirable changes or stability issues, thus keeping the user community up and running while reiterating the development and test process.

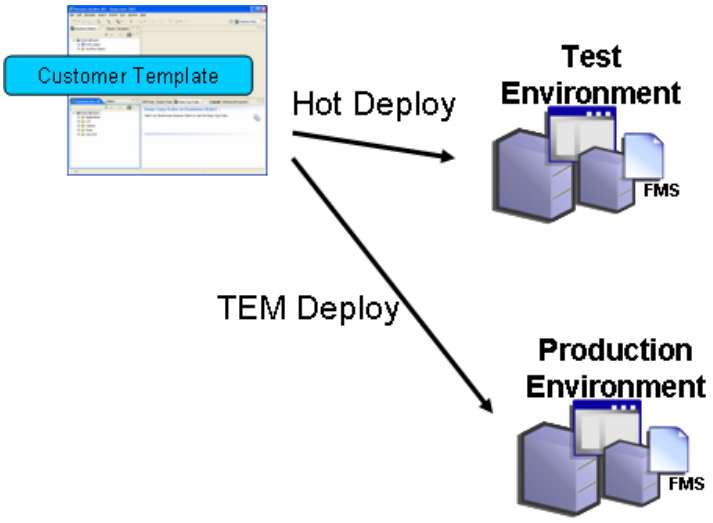


Figure 27: Test and production database

7.3 Test, User Testing, and Production Database

This environment is very similar to the previous scenario except that it adds a third environment specifically for user testing or user training.

In this scenario, the BMIDE user creates a template for extending the data model. The BMIDE user hot deploys the template to a test database, where the extensions can be tested. After the extensions have been tested to a satisfactory level, the company is ready to roll out these changes.

Before rolling the template out, the customer will want to set up a safe environment for training or for user acceptance testing. Use the package wizard to build the template feature, and then use TEM in the user testing environment to install/update the template. If user acceptance testing fails, fix and test the extensions in the test environment then roll out to the user testing environment again. If user acceptance testing passes, then notify the user community of a system down time. Then take the template feature created for the user testing environment and use TEM in the production environment to install or update it as the case may be.

This scenario has the same benefits of a reiterative process as the previous scenario. It also adds the ability to let the user community test or train in a safe environment without impact to the real data in the production system. The user testing environment can be a useful place to work out any issues between requirements and design. The training environment allows the user community to get comfortable with any new processes, objects, or behaviors before going live with the production environment.

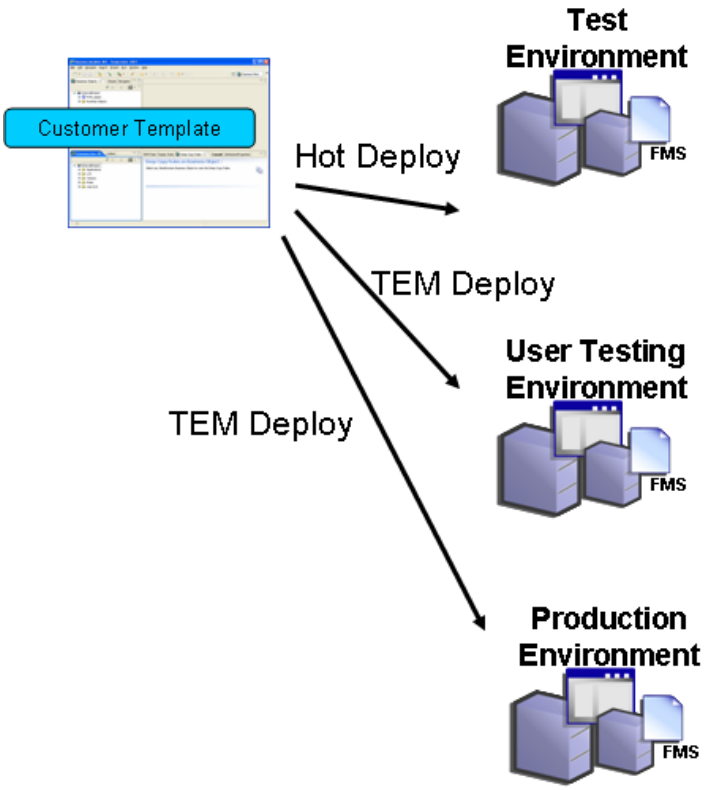


Figure 28: Test, user testing, and production database

7.4 Multiple Developer Environments

This scenario is similar to the previous two scenarios with the added requirement that there are two or more users of the BMIDE working on the template, rather than only one. In this scenario many users are contributing extensions to the same template.

For two or more developers to work on a template project concurrently, a Source Control Management (SCM) system must be connected to the BMIDE client. Examples include: CVS, Subversion, ClearCase, Perforce, and Visual Source Safe. By default the BMIDE is equipped to connect to a CVS repository. All other integrations must be added separately to the BMIDE. Factors that influence your decision to use an SCM may be budget and functionality. CVS and Subversion are free and have basic tools for managing source code. Others, like ClearCase have license fees and more robust functionality. To use an SCM with the BMIDE, the SCM must have a plug-in that integrates Eclipse with the SCM repository.

In this scenario, one BMIDE user should create the template project and add the project directory and all of its subfolders and files into the SCM. This is usually performed by doing a check-in of all files and directories. Next all other BMIDE users should synchronize their SCM repository to get the project files and directories downloaded to their machine, and then import the project into the BMIDE using the Import wizard. Care should be taken to set up the source files so that each developer is extending the system and saving the extensions into their own files. Working with your own set of files will reduce the amount of file merges. See section 5.6 for more information on how to organize source files for concurrent development of a template. See the Business Modeler IDE User Guide for information on how to import a project. Finally for more information about SCM's see section 8.

Each developer should have their own test environment for unit testing. Following the process established in the previous scenarios, each developer should create extensions, hot deploy them to their own test environment, and then test the changes. Repeat the process until each developer achieves the acceptance level of individual testing. After completing the developer testing, each developer should check in (submit) their changes to the main SCM repository.

At this point, all extensions have been unit tested individually by each developer independent of the other extensions. Now all extensions need to be tested together in an integration environment. A project administrator or one of the developers should synchronize the BMIDE with the SCM to download the latest source files from the main repository. This will gather up all of the latest XML definitions from each developer to this BIMDE client. Note that after each synchronization with the repository, you should use the “Reload Data Model” feature in the BMIDE to reload the data model and validate it for correctness (see section 5.10). Use the package wizard to build the template feature, and then use TEM to install the template package into the integration environment.

The integration environment is the environment where all of the individually developed extensions are tested together in one environment to ensure there are no integration issues. Use TEM to install the custom template into this environment and test all functionality.

After all extensions have been tested in the integration environment, the template feature can be installed into the user testing environment and production environments using TEM.

Connecting the BMIDE to an SCM provides an extra benefit of backing up the project files in the SCM. Care should be taken to back up the SCM on a regular basis.

This scenario has the same benefits of the previous two scenarios with added scalability to add more BMIDE developers and increase productivity through collaboration. Each user should understand the process and follow it so that extensions are successfully created, tested, and pushed to each environment with confidence in the quality.

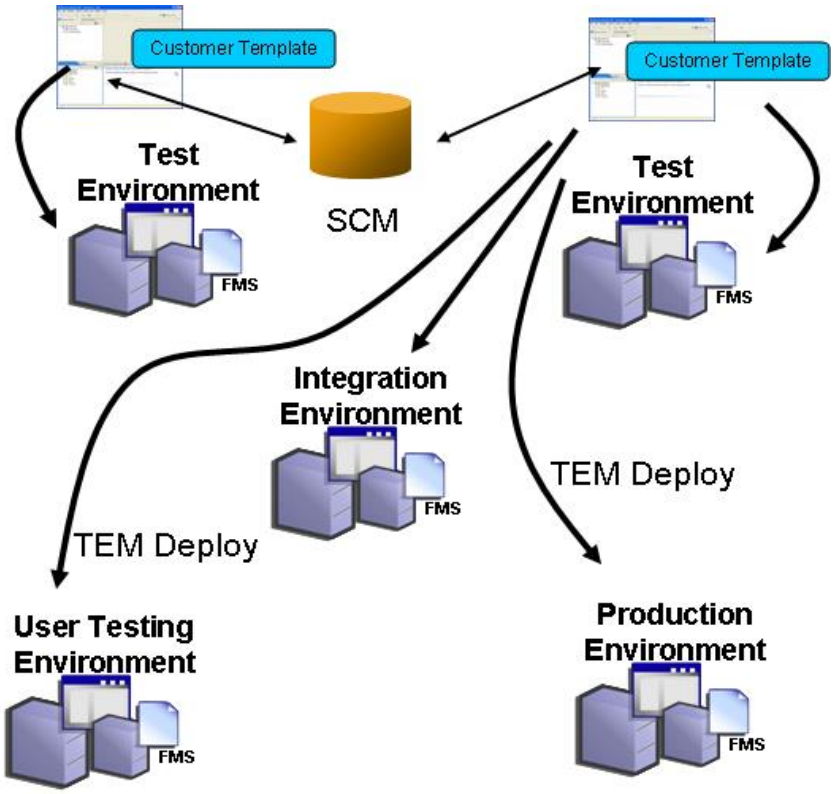


Figure 29: Multiple developer environments

8 Source Control Management System (SCM)

This section contains additional information about questions commonly asked about using an SCM with the BMIDE.

8.1 How to select an SCM

Many customers ask for guidance from Teamcenter product development on which SCM to use and how to use them. Typically this is not something that product development provides guidance on because there are many vendors who supply SCM systems that are compatible with the BMIDE (Eclipse) and each has their own documentation on how to use it. However, we would like to give you some thoughts on how to find information and some best practices to get you started.

To find more information about SCM, perform an internet search for “SCM” or “Source Control System”. There are many white papers and websites that provide in-depth information about the advantages and how to use them. Additionally, review section 7.4 in this document which explains how to set up a multi-developer environment integrated through an SCM system.

The BMIDE can be integrated with any SCM that has a plug-in compatible with Eclipse. To find out, perform an internet search that includes the name of your product plus the word “eclipse” or “plug-in”. Or contact the SCM vendor to see if they have a plug-in for integrating eclipse with their SCM repository. While there are many SCMs available that integrate with eclipse, here are a few that we have tried and work well: CVS, Subversion, Perforce, and ClearCase.

CVS and Subversion are free shareware products. The BMIDE by default already has the CVS plug-in built into it, this will save you time from having to download it. If you have a CVS repository setup, you can connect to it from the BMIDE in the matter of a few minutes.

Subversion is the next generation of CVS. Both Subversion and CVS both offer adequate tools for checking in and out source files and synchronizing with a central repository.

Perforce and ClearCase are commercial products. These commercial products offer a richer set of tools and functionality for managing source files.

Product development has utilized all four of these SCM products with eclipse and was able to work well with multi-developer environment. Your decision to use one of these products may be based on factors such as resources, IT administration, and purchase costs for licenses. If unsure how to proceed, you may want to start with CVS or Subversion since these are free. After gaining some experience on using these tools you could look into a commercial system.

8.2 Getting started

To get started, install the SCM server onto a computer which is available to the network. The SCM server will be the central repository for the template source files. Next, one developer should go first and install the BMIDE client on his local machine. Then install the SCM plug-in into the BMIDE client which integrates the BMIDE (Eclipse) with the SCM. Next create a new BMIDE template project using the BMIDE (see Figure 30). This template will be the template that all other users or developers will share and add definitions too. Note that you do not want to have a separate template (different template names) for each developer, because if you do all definitions will remain in their own template and cannot be merged together into one template.

After this user creates the template project, the user should create many source files. See section 5.6 for more information. The best way for multiple developers to add definitions to one template is to have each person work in a separate file. This will cut down on the number of file merges that need to occur. One good way to organize and name new files is by functionality. All definitions related to the functionality can be added to the file. If each developer is given an area of functionality to work on, then they can add all of these definitions to their file and not step on top of other developers adding definitions to another file. Therefore this first user should work with the project manager to discuss the kinds of new functionality that is needed and establish the names for these new files. If you find that you have too many new source files to keep everyone organized, the files can also be organized into source folders. Use the wizards listed in the user documentation for adding new folders and files.

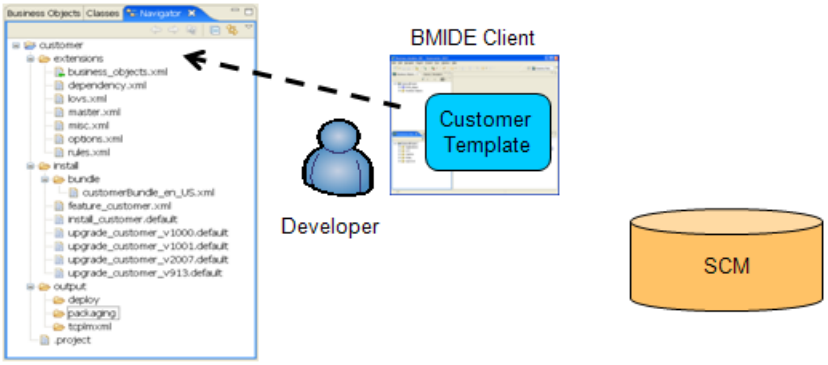


Figure 30

Once all files and folders are arranged, this first user should synchronize in the entire template project into the SCM. This will push all source files from the local BMIDE client to the SCM repository as shown in Figure 31.

For systems like ClearCase and Perforce, you would add all source files to the repository. For systems like CVS and Subversion, go to the Navigator View, select the top level template project folder, right mouse button click and select Team->Share Project. Follow the SCM user documentation on how to fill out the fields in this wizard. After completing this step, the template project will be connected to the SCM. Next “synchronize” all files and folders to the repository. This will push the files and folders to the central repository where others can start downloading them. After the SCM is synchronized, the SCM will now track the change history of each file as developers change them.

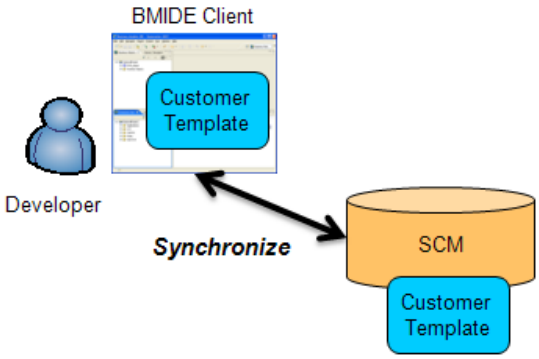


Figure 31

The next step for each developer who wants to contribute to this template is to install their own BMIDE client on their local machine. Next install the SCM plug-in into the BMIDE to integrate the BMIDE with the SCM. Then download the BMIDE template project from the SCM to their BMIDE client (see Figure 32). For ClearCase and Perforce, you may have to launch a separate administrator tool to synchronize or download the latest BMIDE template files from the repository. This will place the template project directory and all of the files in a directory on your machine.

The next step is to import the downloaded template project into the BMIDE client. Note that each developer should not create a new template project using the same name as the first developer, nor should they create one with a separate name. The correct procedure is to import the project. Use the File->Import->Import a Business Modeler IDE Template wizard to import the template folder from local machine into your BMIDE client. For more information about the import wizard see the Business Modeler IDE User Guide. For CVS or Subversion users, you can use the CVS Repositories

View to locate and download the template folder to your BMIDE. After downloading, use the “Reload Data Model” menu to load the template project into the BMIDE.

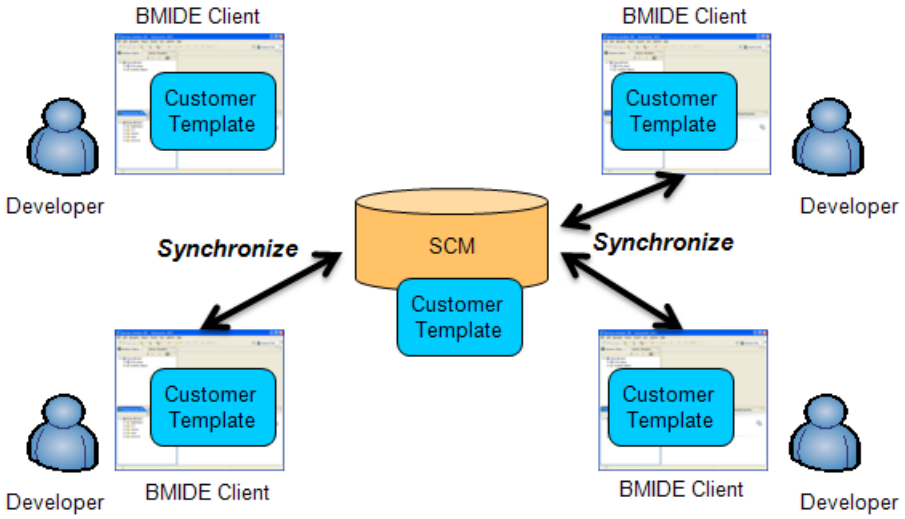


Figure 32

Once each user has the template project in the BMIDE, they should set the active source file to the file that is assigned to them. Then they can start to use the BMIDE to add definitions. After each person completes their work, they should test first by deploying to a local test environment (see Figure 33). Note that the test environment should not be shared with other developers.

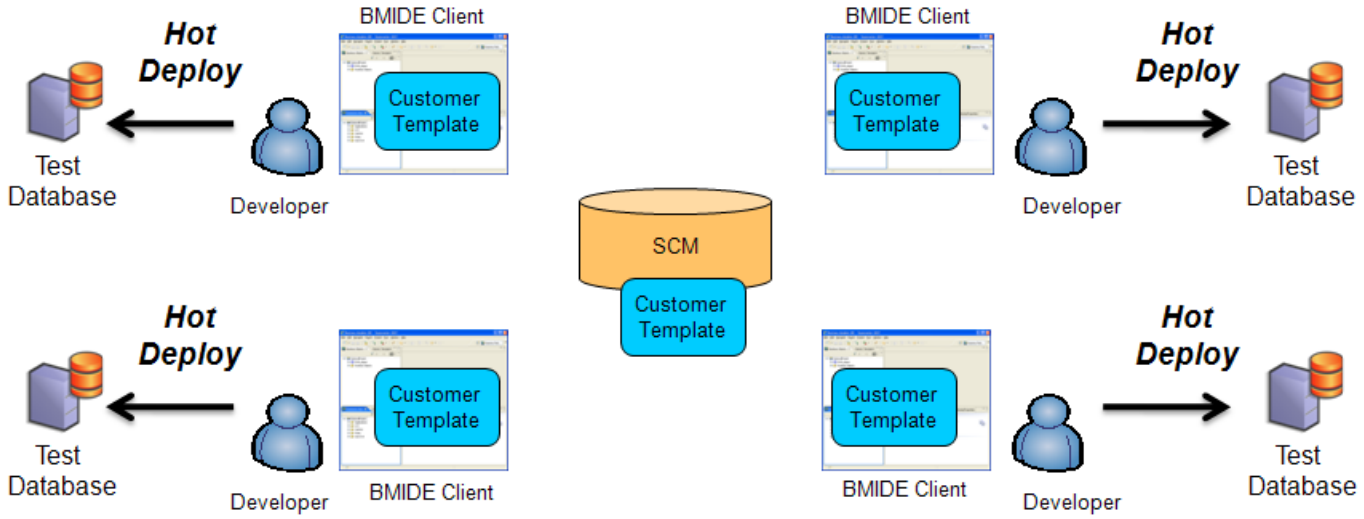


Figure 33

After testing in the individual environment is complete, each user should synchronize their BMIDE template files to the SCM (see Figure 34). This step will push all changes created by each individual developer’s BMIDE to the SCM repository. Additionally synchronization will pull the latest versions of the files from the SCM to the local developer machine.

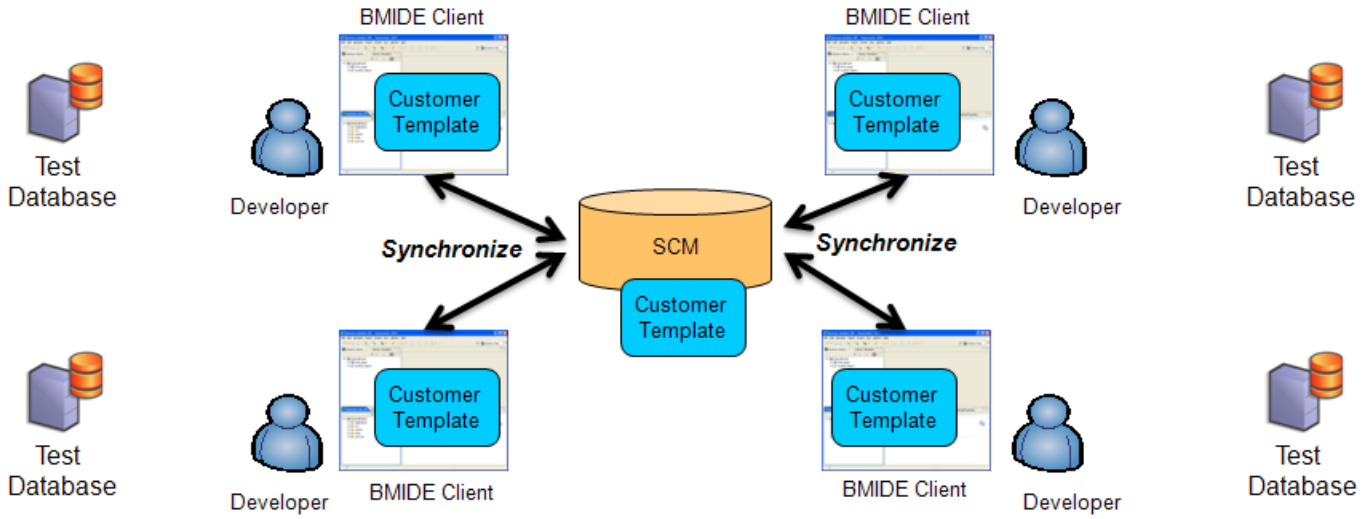


Figure 34

After synchronizing each developer's environment will have all the latest extensions in their BMIDE. Each developer can continue with additional rounds of creating new extensions, deploying, testing and then synchronizing with the SCM again (see Figure 35).

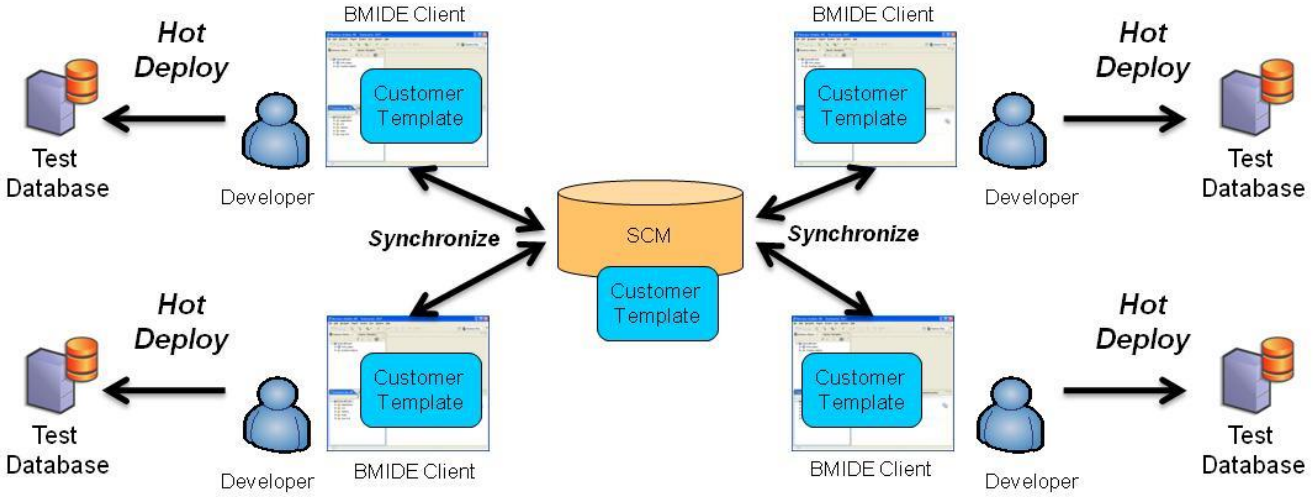


Figure 35

Up to this point, each developer has been testing his own extensions in his own test environment. Once all development activities are complete, all extensions should be tested together. At this point typically a project leader or administrator should synchronize the latest code to his own BMIDE client (see Figure 36). Use the Package Wizard to build a deployment package and then use TEM to deploy this package into an integration testing database.

If integration testing fails, then fix the issues in an individual test environment and synchronize to the repository. The administrator can synchronize again, to bring in the latest changes, then rebuild the package and deploy again to the integration environment. If testing in the Integration Environment passes, the tested template package can be deployed to a QA testing environment for user testing.

Once user testing passes the final template package can be deployed to a production database.

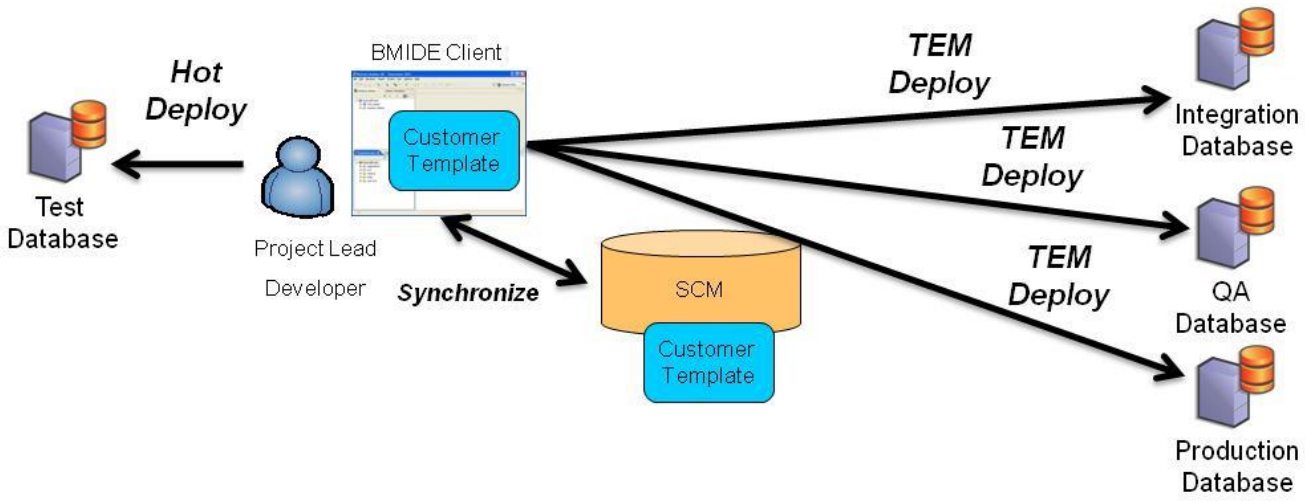


Figure 36

8.3 Coordinating template development with frequent operational data updates

This section will cover the use case where you have an administrator who needs to make frequent updates to the production database using the BMIDE while developers continue to add extensions to the template for the next system downtime. Before reading this section, please read section 6.5 for more information about using the BMIDE for an operation data update to a production system.

To get started, the process is similar to the process detailed in section 8.2. Figure 37 below shows that first a developer should install a BMIDE client, add a plug-in to the BMIDE to support a connection to the SCM repository, create a new BMIDE template project, then synchronize the entire contents of the template project into the SCM.

After the project is synchronized with the SCM, the administrator can install a BMIDE client and SCM plug-in. Synchronize the BMIDE with the SCM to pull down the template project. Then import the template project into the BMIDE client.

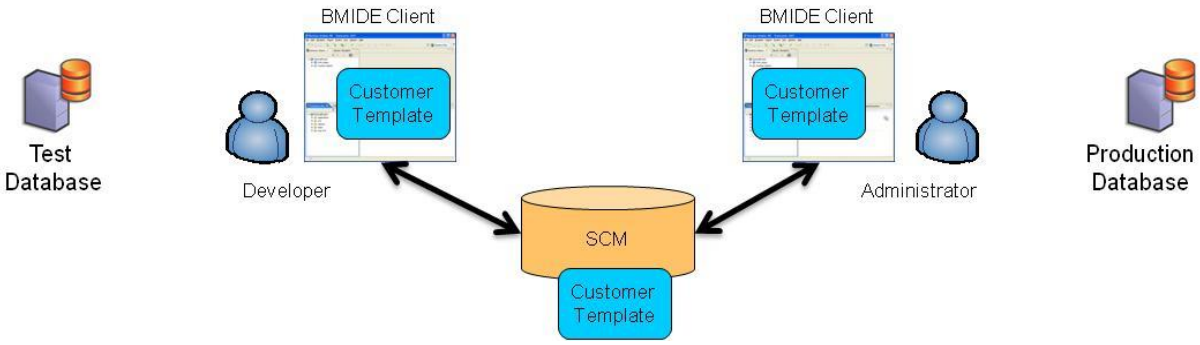


Figure 37

Next Figure 38 shows that the developer can create extensions using the BMIDE. His extensions are stored in the XML files that are included with the template project. The developer can add both schema (Classes, Attributes, and Business Objects) and operational data (LOVs, Display Rules, etc). The developer should hot deploy his template to the test database to test the extensions.

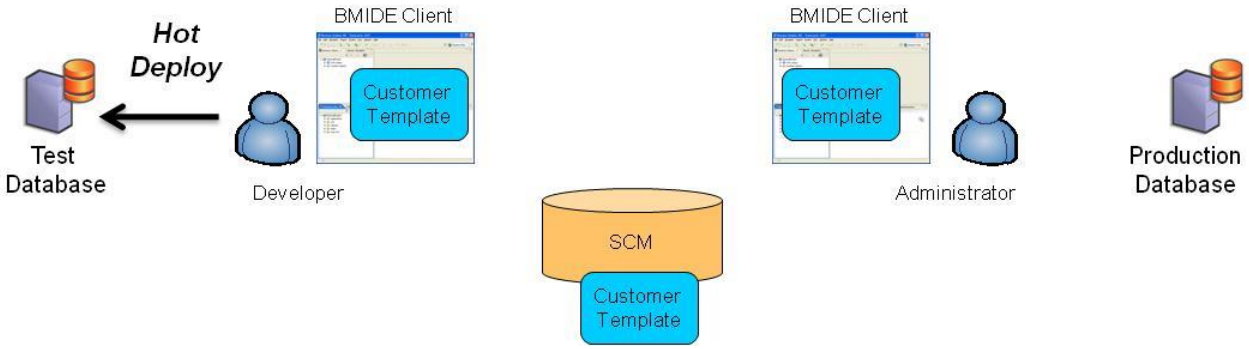


Figure 38

When the developer is finished creating and testing his template extensions, Figure 39 shows that the developer should synchronize his BMIDE with the SCM. This will push all of his changes to the files to the SCM. Then he should notify the administrator that the template is ready for deploying to the production database.

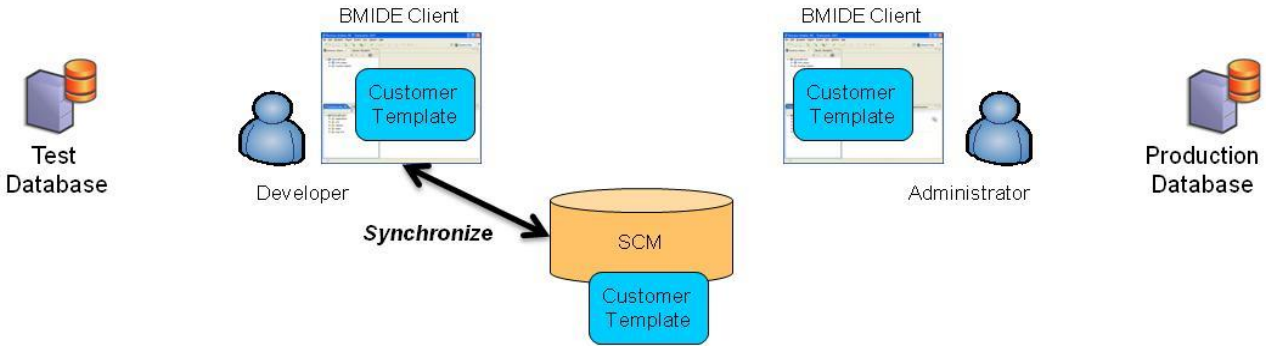


Figure 39

After getting the notification from the developer, the administrator should synchronize his BMIDE with the SCM. This will pull all the file changes from the SCM to the BMIDE as shown in Figure 40. The administrator should Reload the Data Model and then review the latest configurations in the BMIDE client.

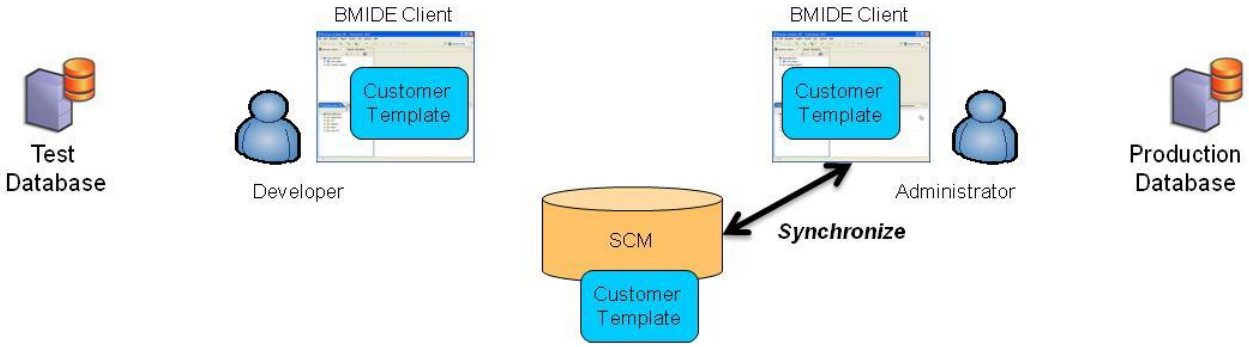


Figure 40

Since the administrator's template now includes schema changes that the developer contributed to the template, the administrator must take down the production server in order for the schema changes to be processed. Use the Package Wizard to build a deployment package and then use TEM to deploy this package into the production database. Restart the server and notify the users to log back on.

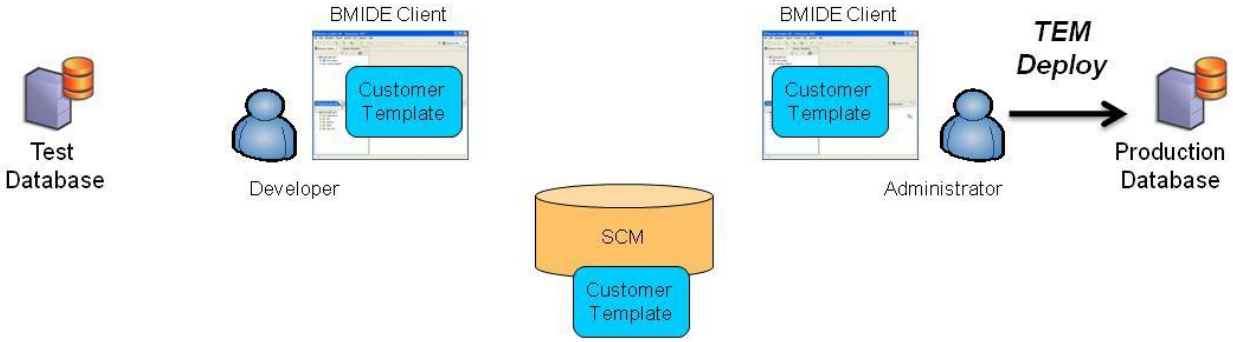


Figure 41

Follow the convention outlined in section 6.5 about operational data updates, Figure 42 shows that the administrator can continue to make regular updates to the production server using the BMIDE as long as the template does not include any schema changes (Classes, Attributes, Business Objects). Because the production server must remain up and running, the administrator uses the BMIDE with Hot Deploy to deploy the template to the production database. However, any users currently logged into the system during the deploy will not see the changes until they log out and log back in again.

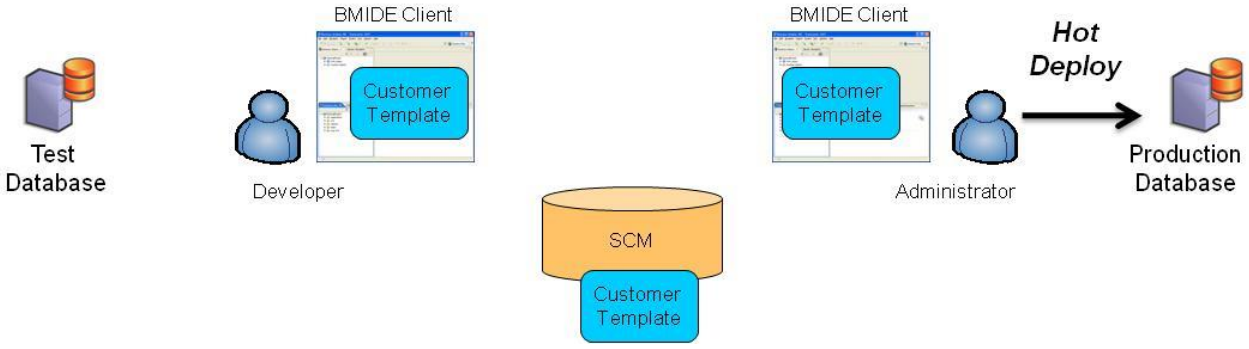


Figure 42

While the administrator is making frequent updates to the production database, the development team may wish to continue adding extensions to the template to prepare it for the next scheduled system downtime. Figure 43 shows how the developer can continue to add new schema and operational data to support the update to the database. The developer should use Hot Deploy to test the extensions in his test environment. After the developer finishes testing, he should synchronize his BMIDE with the SCM. This will allow any other developers who may also be working on the template, to stay current and pickup the latest changes in the SCM.

Note that during this timeframe, the administrator should not synchronize with the SCM. Because if the administrator did synchronize, he would pick up schema changes and inadvertently Hot Deploy them to the production server. As outlined in section 6.5, schema changes should not be deployed to a production server.

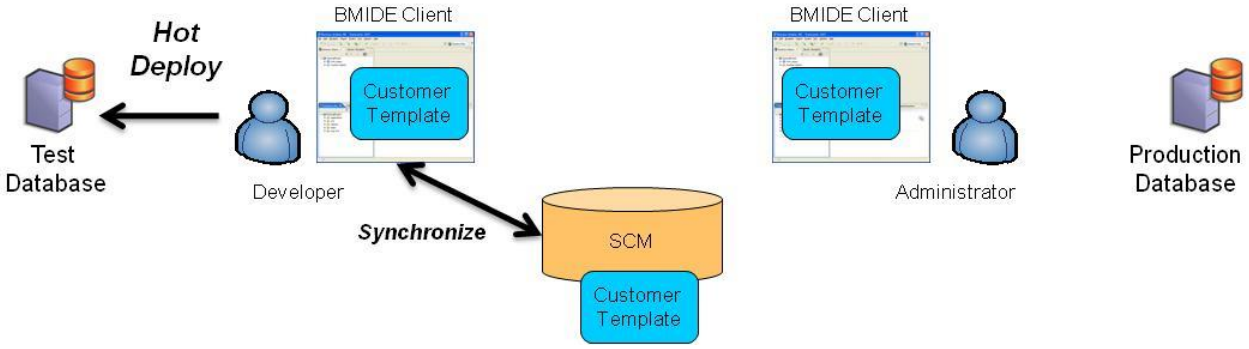


Figure 43

Before the next system down time, the administrator should push all of his operation data updates back to the SCM so that they can be integrated with the latest template extensions added by the development team. Figure 44 shows the administrator synchronizing his BMIDE with the SCM. This pushes his latest template files into the SCM.

Note that synchronization will synchronize file changes in both directions. That is, the synchronization will push the latest changes from the administrator's BMIDE to the SCM and pull the latest changes from the SCM to the administrator's BMIDE. Since this means that the administrator could potentially pick up schema changes from the SCM, the administrator should not perform a BMIDE Hot Deploy during this time. He should wait until all of his extensions are combined with the developers' extensions and tested, then take the system down to do a TEM deploy.

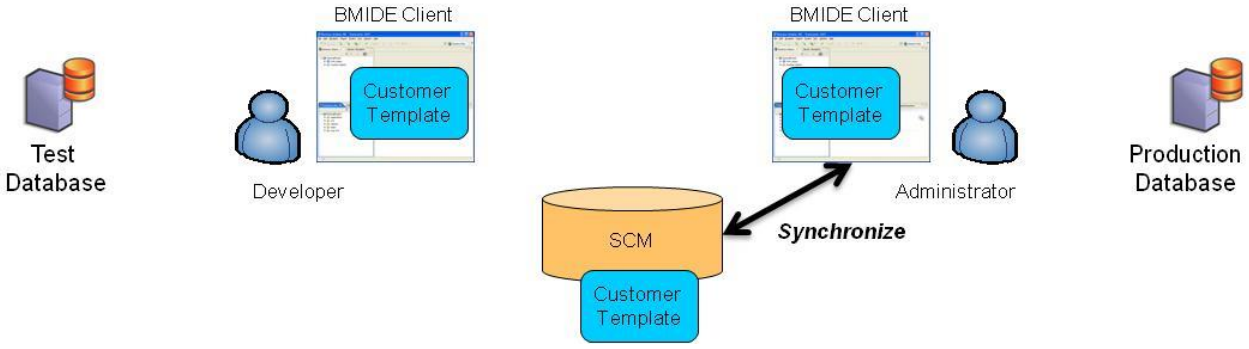


Figure 44

To pick up all of the operation data extensions added by the administrator, the developer should synchronize his BMIDE with the SCM as shown in Figure 45. The developer should reload the data model into the BMIDE and view the latest extensions added by the administrator to see how they blend with the new extensions added by the development team. Use Hot Deploy to deploy the combined extensions to the test environment and test.

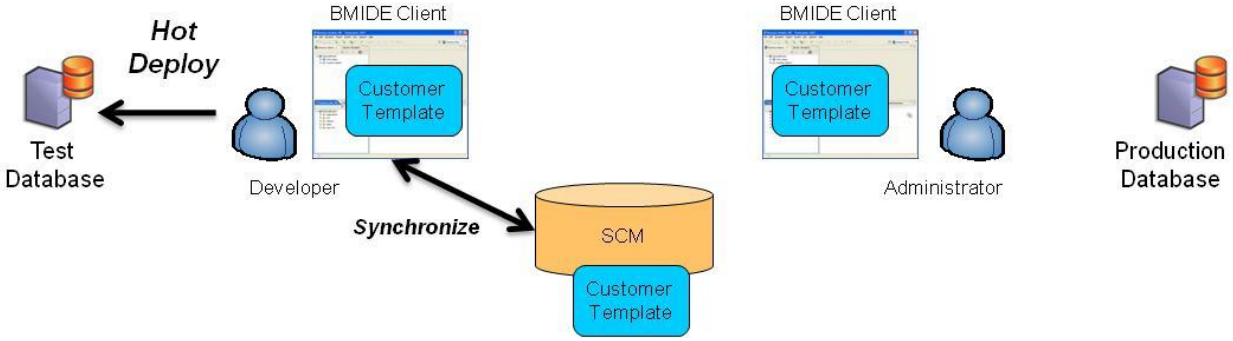


Figure 45

After the development team completes the integration testing, the developer should synchronize one more time with the SCM. This will push all the integrated files back to the SCM as shown in Figure 46.

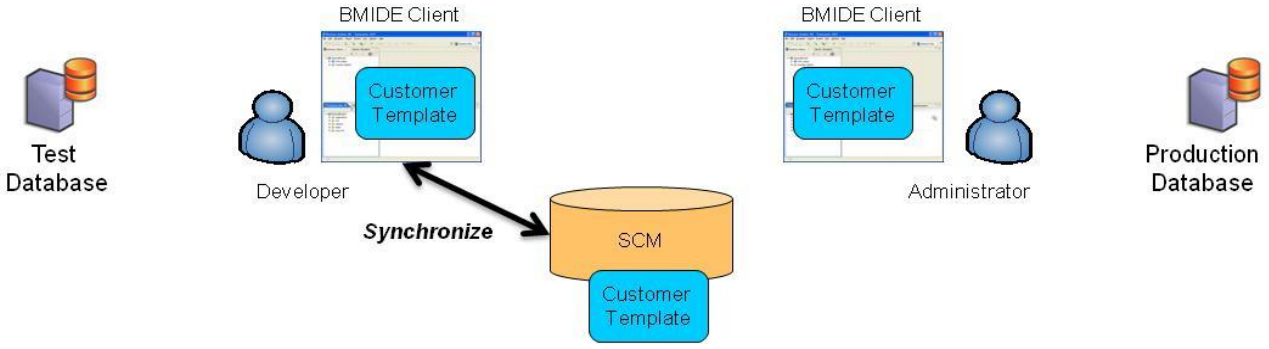


Figure 46

Finally the administrator or any developer should synchronize his BMIDE with the SCM. This will pull all the latest files that were tested from the SCM to the local BMIDE. Reload the data model and review the configurations. Because the template includes schema changes, the administrator must take down the production server. Use the Package Wizard to build a deployment package and then use TEM to deploy this package into the production database. Restart the server and notify the users to log back on.

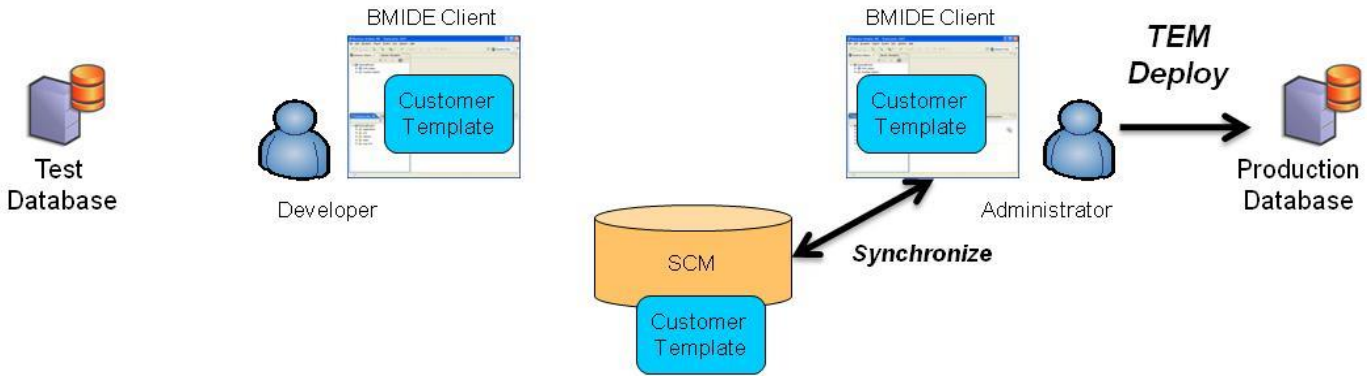


Figure 47

9 Prefix Policy

With the constant introduction of many new Classes, Business Objects, LOVs, rules, and other data model elements, the risk of a customer defining a new object that has the same name as a Teamcenter object is increasing. Teamcenter will only allow one type with a given name. So if both the customer and Teamcenter have defined a type by the same name, the customer will have to rename their type.

There is only one way to ultimately ensure that no one will ever have a collision. The answer is prefixes. In a future release of Teamcenter unified architecture, product development is going to start enforcing the usage of a prefix for every data element defined by a customer. This will be enforced through the Business Modeler IDE. There are two aspects to this. First a prefix policy needs to be established. Second, prefixes are used on type names, and then there has to be a method for displaying the type in the client UI without the prefix. For example, if you name your type as "B9_Part"; you would want the UI to display it as "Part" so that is user friendly.

Note that the prefix policy listed here will be enforce by the BMIDE client in Teamcenter 8.3.

In the meantime, you can start using the prefix policy if you feel that you want to avoid potential collisions. We have found that many customers are already prefixing their definitions in the BMIDE. We are not forcing you to prefix at this time, however, it is available if you want to utilize it.

When choosing a prefix, here is the policy that you must follow:

- A prefix must be used on all class names, type names, rule names, LOV names, etc within a single BMIDE template. A prefix should not be shared across two or more templates.
- The prefix must be placed at the beginning of the model element definition name. Example: "B9Part" is correct. "PartB9" is not correct.
- A prefix must contain 2 to 4 characters.
- Characters can be a digit, letter, or underscore.
- The prefix must have a digit in character position 2, 3, or 4. Digits may not be in the first character position.
- The first character of a prefix must always be a capital letter.
- The use of the "0" or "1" digit in a prefix is reserved for Siemens PLM Software Templates
- The use of the "2" or "3" digit in a prefix is reserved for 3rd party template development. If you are a 3rd party please contact GTAC to obtain an assigned prefix.
- Customers are to use digits "4" through "9" in the prefix.

It is not necessary for a customer to register their prefix with product development. If a customer prefix is the same as another customer prefix, it is not an issue as long as these two customers never share data models (BMIDE templates).

Here are some examples of allowable prefixes for a customer

- A3
- Ab5
- A5c
- C4
- B9
- F8_

- X99_
- Zwt5

Using the same prefixes above, here is how the prefixes would be used for a new type name that includes the word “Part”.

- A3Part
- Ab5Part
- A5cPart
- C4Part
- B9Part
- F8_Part
- X99_Part
- Zwt5Part

Additionally here is how the prefixes would be used for a new type name that includes the word “Document”.

- A3Document
- Ab5Document
- A5cDocument
- C4Document
- B9Document
- F8_Document
- X99_Document
- ZwtDocument

Note that it is not necessary that you rename any existing data model definitions to use the prefix. All existing data model definitions will be grandfathered in and will not require a prefix. The only exception to this rule is when one of your existing definitions collides with a Teamcenter definition. In that case, you will need to rename the definition. The new name should include the prefix to ensure that there will be no future collisions.

If you have already been using a prefix with your data model definitions, you may continue to use it as long as it follows the prefix policy defined in this document. If it does not align, you must alter the prefix to align with this policy.

10 Best Practices

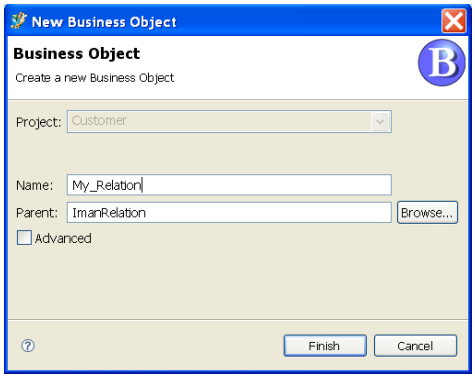
This section will cover best practices within the BMIDE.

10.1 How to add relation properties

Applicable Releases: Tc2007.x

A relation property is a property on a business object that shows the related objects via a specific relation type. In Tc2007.1 relation properties must be added manually to both the database and to the BMIDE. After following these steps you will be able to create and relate objects via your relation in the rich client. Additionally you will be able to create business rules such as compound property rules in the BMIDE which utilize your custom relation.

1. In the BMIDE, create a new relation business object by creating a new Business Object under the "ImanRelation" business object.
    a. Go to the Navigator View and select the ImanRelation business object.
    b. RMB-> New Business Object
    c. In the name field type in the name of your new relation and click Finish.



2. Add the relation as a property in the BMIDE template.
    a. Currently in Tc2007.1.x, there is no wizard for adding relation properties to a business object. Therefore this step requires you to manually edit one of the XML files that belong to the template.
    b. Go to the Navigator view, expand to the source files below the <Project>/extensions directory. Open the XML source file and add the following XML code between the two <Add></Add> tags as shown below. For the typeName value, enter the type name where the property should be added to. In this case we want to see the relation on the ItemRevision. For the propName field, enter the new relation business object name created in the previous step. In this case we entered My_Relation.

```
<Add>
   <TcPropertyAttach typeName="ItemRevision">
      <TcProperty arrayLength="-1" isArray="true" propMaxStringLength="0"
            propName="My_Relation" propTypeName="Relation"
            propValueType="PROP_untyped_relation"/>
   </TcPropertyAttach>
```

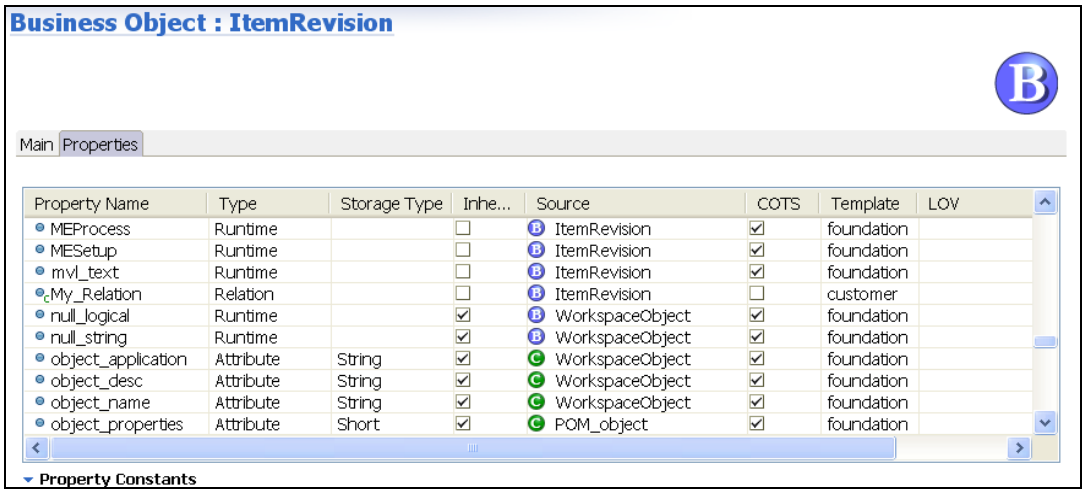
```
</Add>
```

c. Save the file.

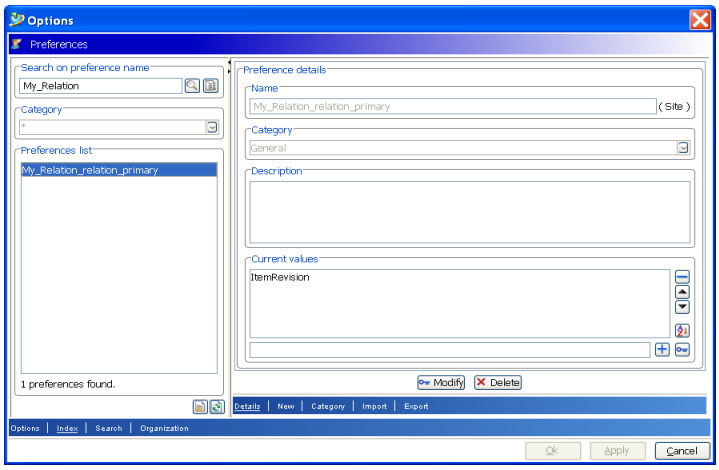
d. Select the file, RMB->Reload the Data Model.

e. Go to the Business Objects View and find the business object where you added the property to.

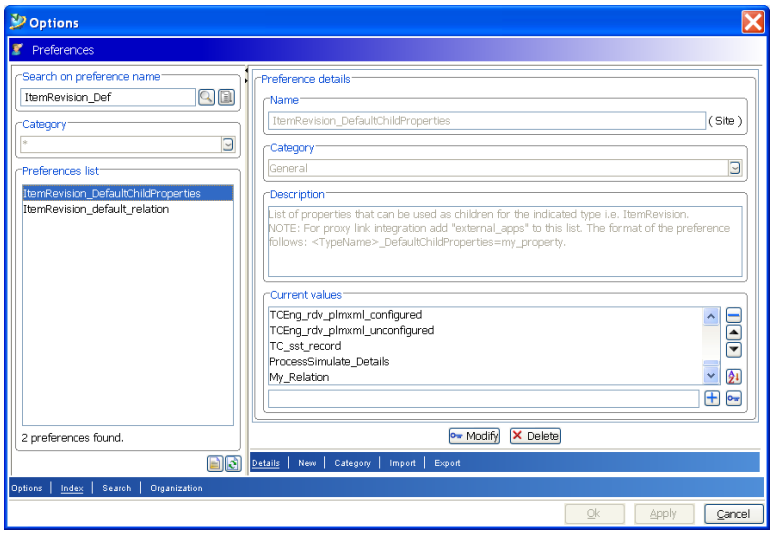
f. Open the business object and find the property in the Properties tab.



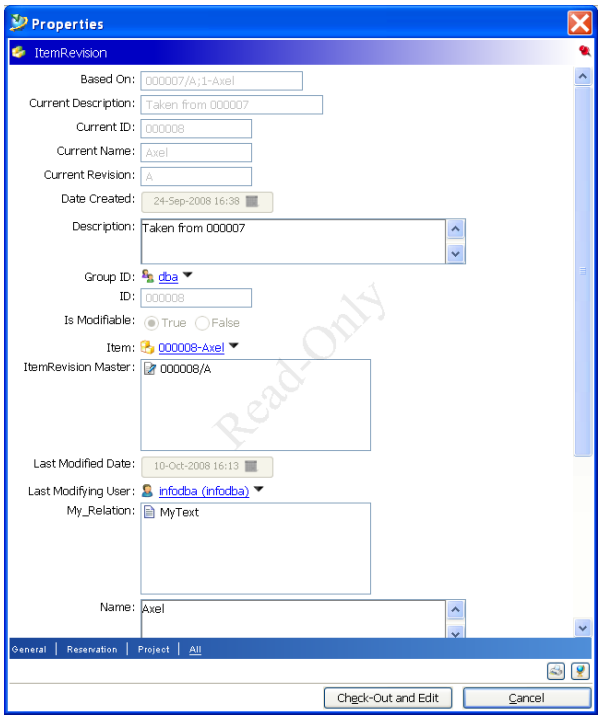
3. Now that the relation and relation property have been added to data model in your BMIDE, you can configure other rules such as Compound Property rules to utilize this new relation and property.
4. Deploy your template to the server to add the relation to the database.
5. To allow users to relate objects via this new relation in the rich client, you will need to add or edit two preferences.
    a. For more information about these two preferences and how to use them, see the *Preferences and Environment Variables Reference*.
    b. In Rich Client, add the following preferences to expose your custom relation as a property on a set of types. Format: <relation_type>_relation_primary
        i. In our example, we want to expose the My_Relation relation business object as a property on ItemRevision. This would by creating a new preference called "My_Relation_relation_primary". In the Current values field enter "ItemRevision".



c. To allow users to create a My_Relation relation using ItemRevision, find the "ItemRevision_DefaultChildProperties" preference and add your relation name "My_Relation" to the Current values field.



6. Log out of the Rich Client and back in.
   a. Create your custom relation "My_Relation" between two objects.
   b. Open up the primary object into the Properties panel. Click "All" at the bottom to view your relation property in the panel.



10.2 How to add runtime properties

Applicable Releases: Tc2007.x

A runtime property is a property on a business object that shows displays the value that is retrieved via custom code. Whereas a persistent property gets its value directly from the object in the database, a runtime property gets its value from another object or database via custom code.

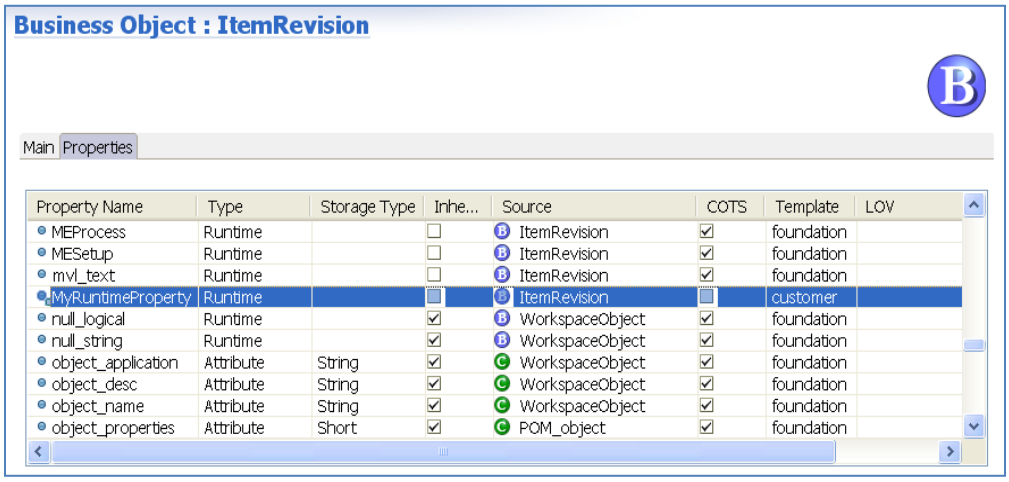
In Tc2007.1 runtime properties must be registered on the server via code. The code must register the name of the runtime property and the names of the methods that get and set the values for the given runtime property. For more information about how to author and register this code, see the section titled “Types, properties, and methods” in the *Integration Toolkit Programmer’s Guide*.

After your you have registered the runtime property in the server, you may want to configure business rules in the BMIDE that utilize the new runtime property. In Tc2007.1 runtime properties must be manually added to the BMIDE via editing XML source files.

1. Add the runtime property to the BMIDE template.
    a. Currently in Tc2007.1.x, there is no wizard for adding relation properties to a business object. Therefore this step requires you to manually edit one of the XML files that belong to the template.
    b. Go to the Navigator view, expand to the source files below the <Project>/extensions directory. Open the XML source file and add the following XML code between the two <Add></Add> tags as shown below. For the typeName value, enter the type name where the property should be added to. In this case we want to see the relation on the ItemRevision. For the propName field, enter the new relation business object name created in the previous step. In this case we entered My_Relation.

```
<Add>
   <TcPropertyAttach typeName="ItemRevision">
      <TcProperty arrayLength="-1" isArray="true" propMaxStringLength="0"
            propName="MyRuntimeProperty" propTypeName="Runtime"
            propValueType="PROP_untyped_reference"/>
      </TcPropertyAttach>
</Add>
```

2. Save the file.
3. Select the file, RMB->Reload the Data Model.
4. Go to the Business Objects View and find the business object where you added the property to.
5. Open the business object and find the property in the Properties tab.



6. Now that the relation and runtime property have been added to data model in your BMIDE, you can configure other rules to utilize this new runtime property.
7. Deploy your template to the server to add your data model changes to the database.

10.3 How to reorder the values on a custom LOV

Applicable Releases: Prior to Teamcenter 8.3

Note: In Teamcenter 8.3 reorder is supported in the BMIDE client.

The following is a description of a known issue in the BMIDE. A customer can define their own LOV and add some values to it. After the customer deploys the LOV to the database, the customer wants to re-order some of the values in the LOV. The BMIDE will allow the reorder, but any subsequent deploys to the database, do not reflect the reorder.

The following steps describes a work around to this issue.

1. Launch BMIDE and open the target LOV in LOV editor
2. We want to ensure that the LOV you want to change, is in its own source file. This will make it easier to manipulate in later steps. Create a new source file called “my_lov.xml” and set it to the active source file.

3. Change the orders of LOV values by using Move Up or Move Down button
4. Save the order change to template extension file: File -> Save Data Model. Note the extension file that the LOV is written to. Example my_lov.xml.
5. Copy the extension file from the BMIDE workspace to a new location on your machine. Example: copy <workspace>/<template project>/extensions/my_lov.xml to C:/my_lov.xml
6. Rename the file to a new name to reflect that we not changing the original source file. Example: rename my_lov.xml to my_lov_reorder.xml.
7. Open the C:/my_lov_reorder.xml and replace the <Add> </Add> XML tags with <Change> </Change> XML tags.

   Example of LOV definition before conversion

```
<Add>
   <TcLOV name="MyLOV" …>
      <TcLOVValue value="A" …/>
      <TcLOVValue value="B" …/>
      <TcLOVValue value="C" …/>
   </TcLOV>
</Add>
```

   Example of LOV definition after conversion

```
<Change>
   <TcLOV name="MyLOV" …>
      <TcLOVValue value="A" …/>
      <TcLOVValue value="B" …/>
      <TcLOVValue value="C" …/>
   </TcLOV>
</Change>
```

8. Run business_model_updater with the file to update the database. See the Utilities guide for more information about this command.

   Example:

```
business_model_updater –u=infodba –p=<password> -g=dba –file=my_lov_reorder.xml
–mode=upgrade –update=all –log=log.txt
```

9. If you want to verify the order without launching a client, you can use the business_model_extractor to extract the full data model. Run the extractor and then search for your LOV in the file to review the order of the LOV values. See the Utilities guide for more information about this command.

Example:

```
business_model_extractor –u=infodba –p=<password> -g=dba –all –outfile=model.xml
```

10. Do not copy the edited XML file or its definitions back into the BMIDE as this XML format will not work in the BMIDE client. Note that the master copy of the LOV values is still in the BMIDE. Each time you want to make a change to the LOV order, do it in the BMIDE and repeat the above steps.

10.4 Best practices for Form Storage Class Swapping

The form storage class swapping on COTS forms is blocked from Tc2007.1.5 (or later) and Tc8 (or later) releases. This section explains the reasons for current practice of swapping, issues associated with swapping and best ways to achieve the same without swapping. It also explains the impact to existing customers' who might have already done swapping and are in the process of upgrading to Tc2007.1.5 (or later) or Tc8 (or later).

10.4.1 Background on Teamcenter Data Model and UML diagrams

In Teamcenter business objects are the fundamental objects to model business data. Business objects were formerly known as types in Teamcenter Engineering. Business objects are used as placeholders for the methods associated with a class. Class definitions define the static state of an object and the basic behavior of the class. Sub-business objects are additional business objects that are mapped to the same POM class as the parent business object and have the same behavior as the parent business object. These additional business objects must use another name and are considered sub-business objects or specializations of the POM class and primary business object. You can define business objects in a hierarchical manner. An automatic hierarchy of primary business objects is generated for subclasses of each POM_object.

The diagram Figure 48 shows some basic objects in the Teamcenter data model provided by the foundation template. The left half of the diagram shows Business Objects and the right hand side shows classes. Each business object is a blue rectangle that shows the name of the Business Object at the top. There is also a blue "B" icon to designate the figure as a business object. For example, the *POM_object* is a business object. Note business objects are also known as types. The left side is arranged into a hierarchy. Each business object is arranged above or below another object with an arrow. The arrow indicates that the object inherits from another object. For example *POM_application_object* inherits from *POM_object*. Each business object is associated with a class. The class hierarch is shown in yellow boxes with circled c on top of the box indicating a class. This type of modeling and relationship diagram is called UML diagram. For more information about UML diagrams see http://www.uml.org/. Each business object is based on a POM class. The persistent properties on business objects are nothing but the attributes of the class it is based on. In the diagram, the arrows going from left to right from classes to business objects illustrate this point. Note: Here only one attribute is shown for example. The classes in his example have more than one attribute.

Such diagrams will be used to explain some customization concepts in the next sections.

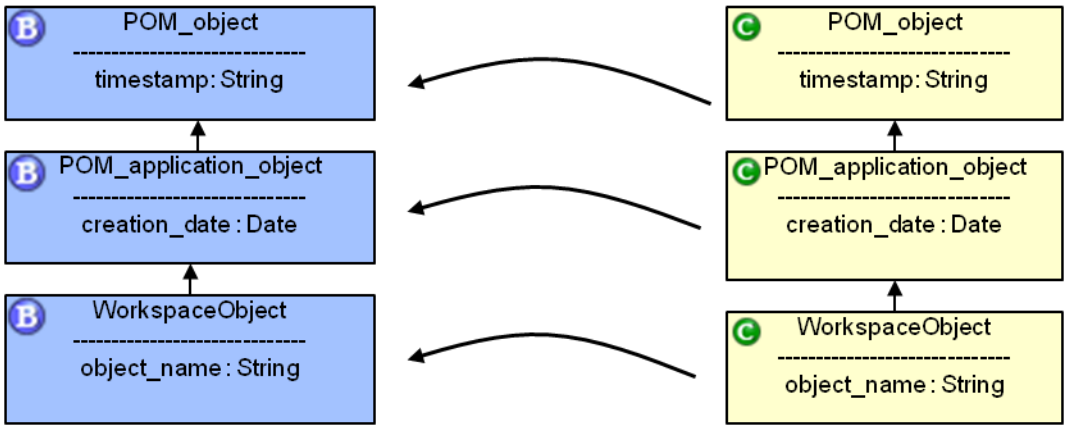


Figure 48 : Basic Data Model UML

10.4.2 How do Forms get their properties

Form business objects manage underlying product information and control how this information is displayed to users. Form storage classes are defined to manage product information using form business objects. By default, forms are displayed as a simple list of properties. Properties on forms are derived from 2 places, persistent properties from *Form* business object and attributes of its form storage class. Diagram Figure 49 shows through red arrows how the properties on *ItemRevision Master* are derived from *Form* business object and *ItemVersionMaster* storage class.

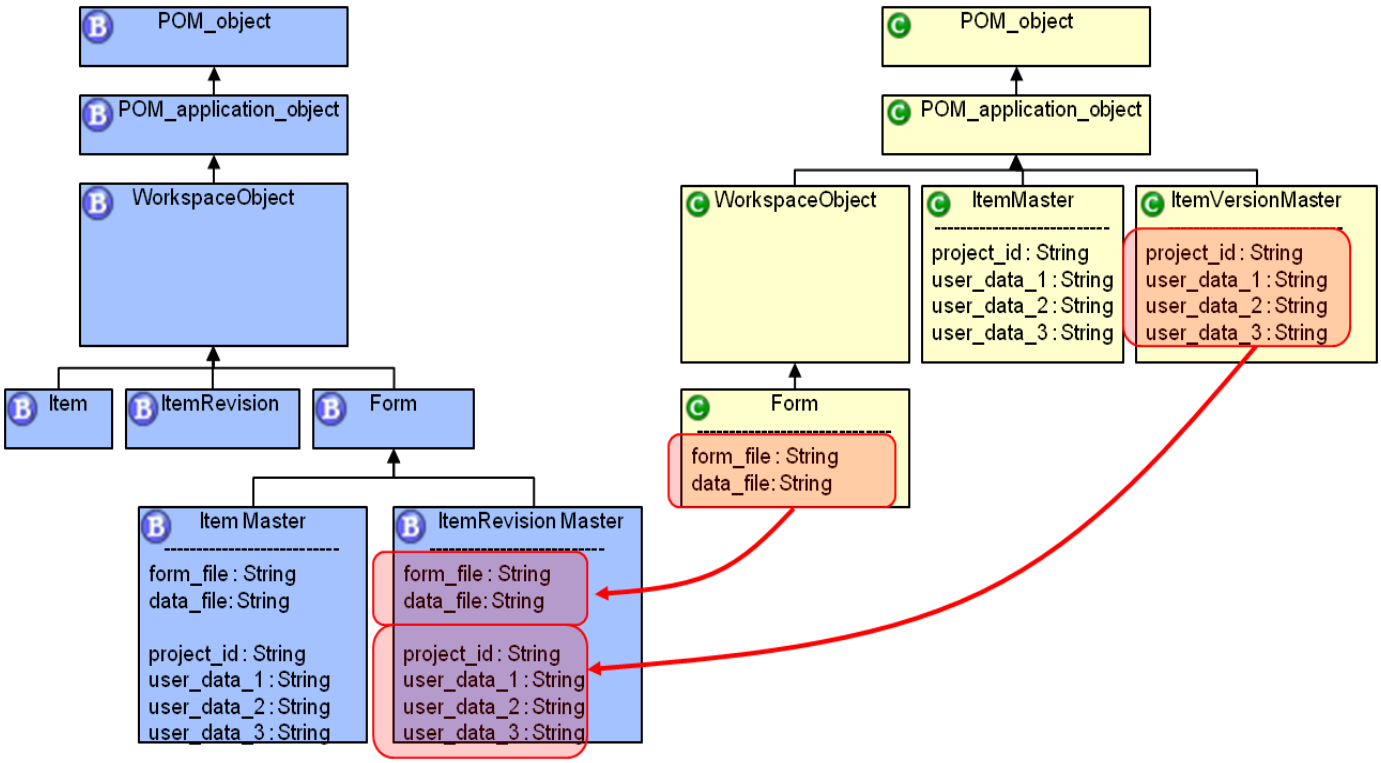


Figure 49 : ItemRevision Master properties

10.4.3 Current practice to customize form storage class

Typically customers customize the OOTB form types in order to view only a specific set of properties which serve the business need or to suppress certain out of the box properties. This is achieved through adding new properties or hiding some existing properties. For adding new properties, today customers add a new class parallel to the COTS storage class with required attributes. The OOTB form storage class is then switched to this new class, so that the attributes from the COTS storage class are not shown and attributes on the new class are exposed as properties on COTS form. To illustrate this let us say, customer wants to customize the OOTB *ItemRevision Master* form to add new properties called 'material' and 'weight' and hide properties from *ItemVersionMaster* class which are project_id, user_data_1, user_data_2, user_data_3. See Figure 50. To achieve this today customer typically creates a new class, say *MyClass* with attributes 'material' and 'weight'. This class is usually a sibling of 'ItemVersionMaster' and not a subclass of *ItemVersionMaster*. Note that the new class doesn't inherit the behavior or data of *ItemVersionMaster* class. The *ItemRevision Master* is then customized to point to this new class *MyClass* as its storage class. See below how the form now shows only the new properties 'material' and 'weight'.

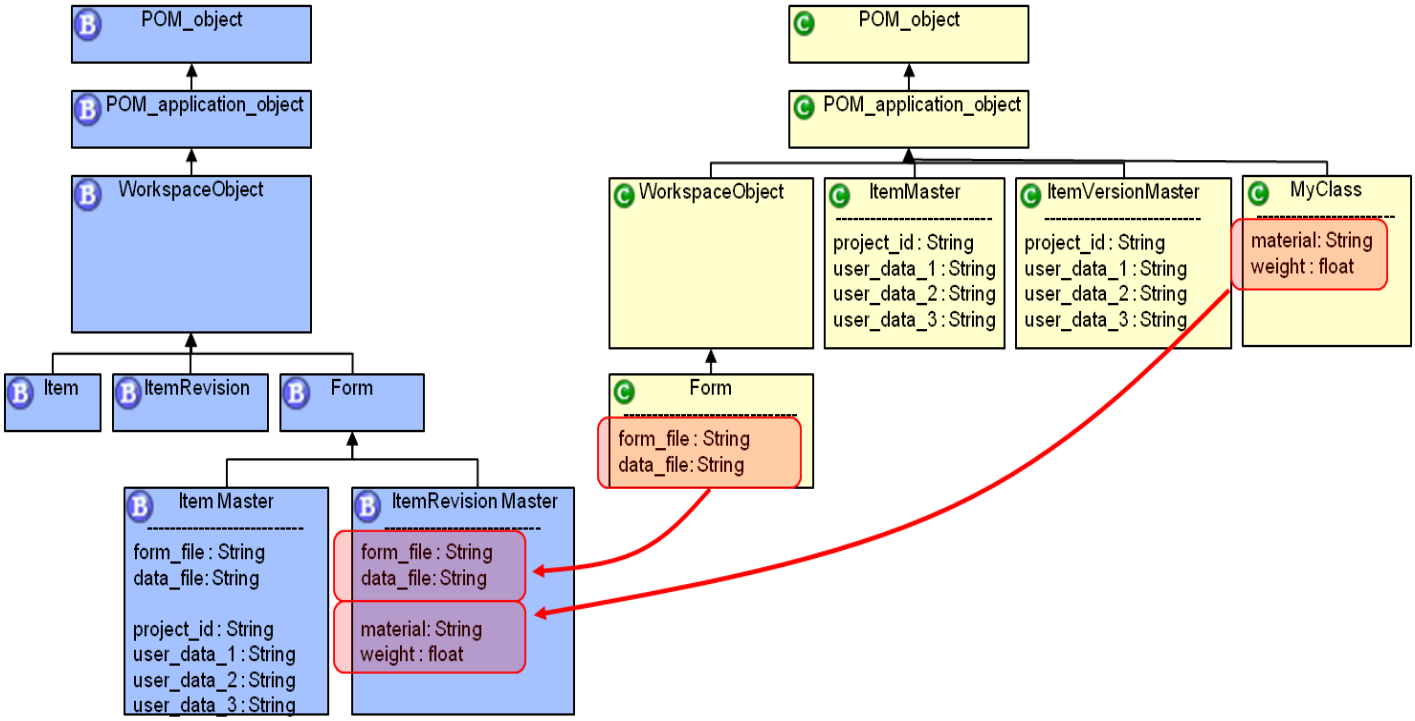


Figure 50 : Customizing ItemRevision Master

10.4.3.1 Issues with the current practice

The current practice of swapping the storage classes on COTS forms to custom classes is causing problems. Let us see how. Let us say that Teamcenter product has introduced some business rules that expect specific attributes to be available on the *ItemRevision Master* business object. Example: an LOV attach on *project_id* property of *ItemRevision Master* and an Extension Rule on the save method of the *user_data2* property. See Figure 51 for how these rules reference the properties on *ItemRevision Master*. If the customer swaps the storage class from *ItemVersionMaster* to *MyClass*, the properties *project_id* or *user_data_2* will no more be there on *ItemRevision Master*. As a result the database will now have dangling LOV attach and the extension rule referencing to nonexistent properties. See Figure 52. The invalid extension rule will also cause any save operation on *ItemRevision Master* form to fail.

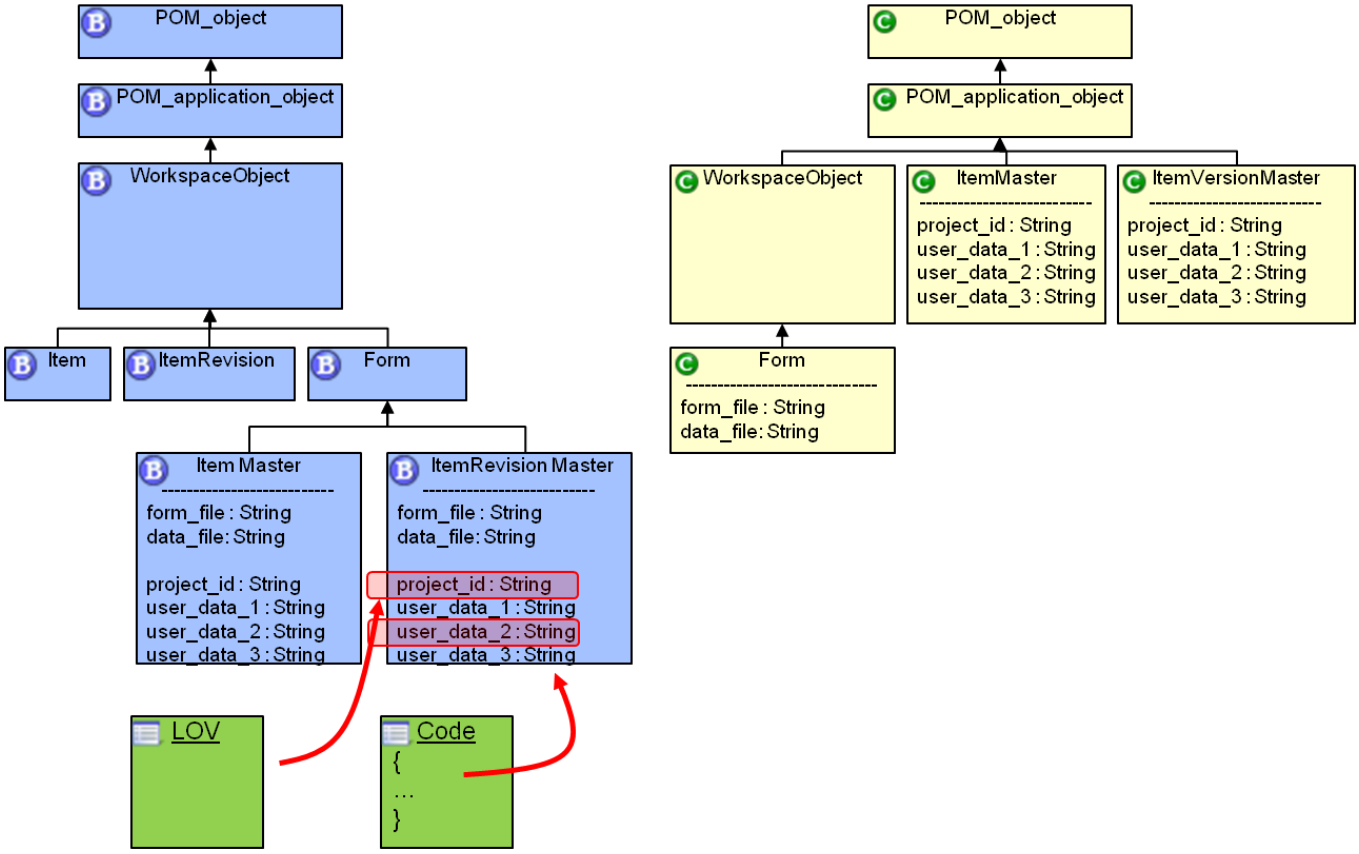


Figure 51: Issues with current best practices

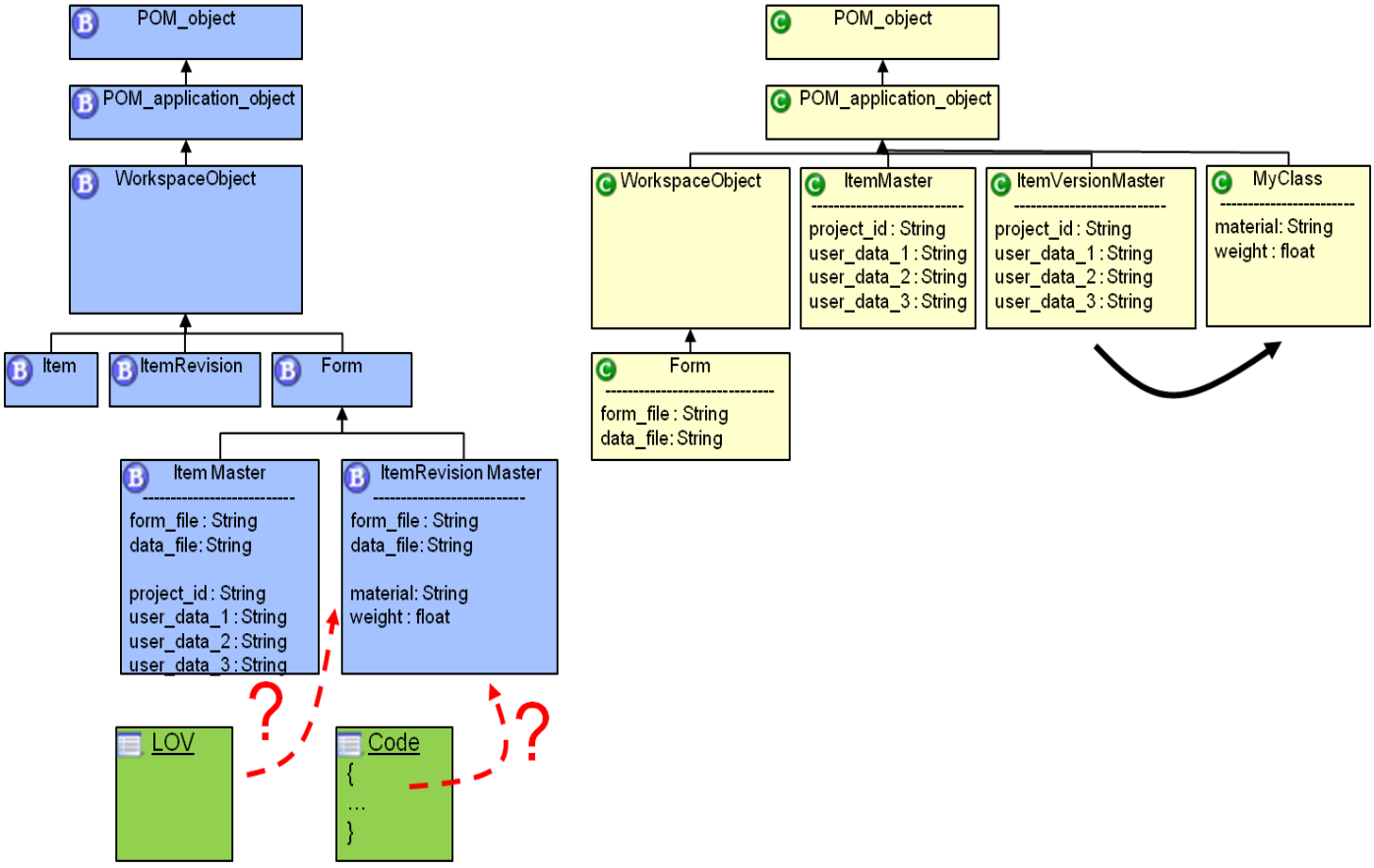


Figure 52: Issues with current best practices

The other problem is with current practice of customizing the master forms. Typically when new Item business objects are created, the new business objects are children of Item and ItemRevision. If say Item called *MyItem* is created, *MyItem* and *MyItemRevision* are children of *Item* and *ItemRevision*. However the Master forms, *MyItem Master* and *MyItemRevision Master* are typically created as siblings to *Item Master* and *ItemRevision Master*. In other words they are not children of current Item Master or Revision master forms, but are siblings. If the form storage classes are also customized, then the form storage classes on *MyItemMaster* and *MyItemRevision Master* are also siblings to the OOTB storage classes of Item master and Item revision master. See Figure 53. The green arrows indicate that *MyItem* and *MyItem Revision* inherit the behavior of *Item* and *ItemRevision*, as they are children. But the behavior or attributes from parent master forms of Item Master or ItemRevision Master is not picked up by the custom master and revision master forms. This results in a loss of desired functionality, faulty behavior, and data corruption issues. The red arrows indicate where the custom classes or form business objects should have actually been children of the Item master forms to avoid these issues.

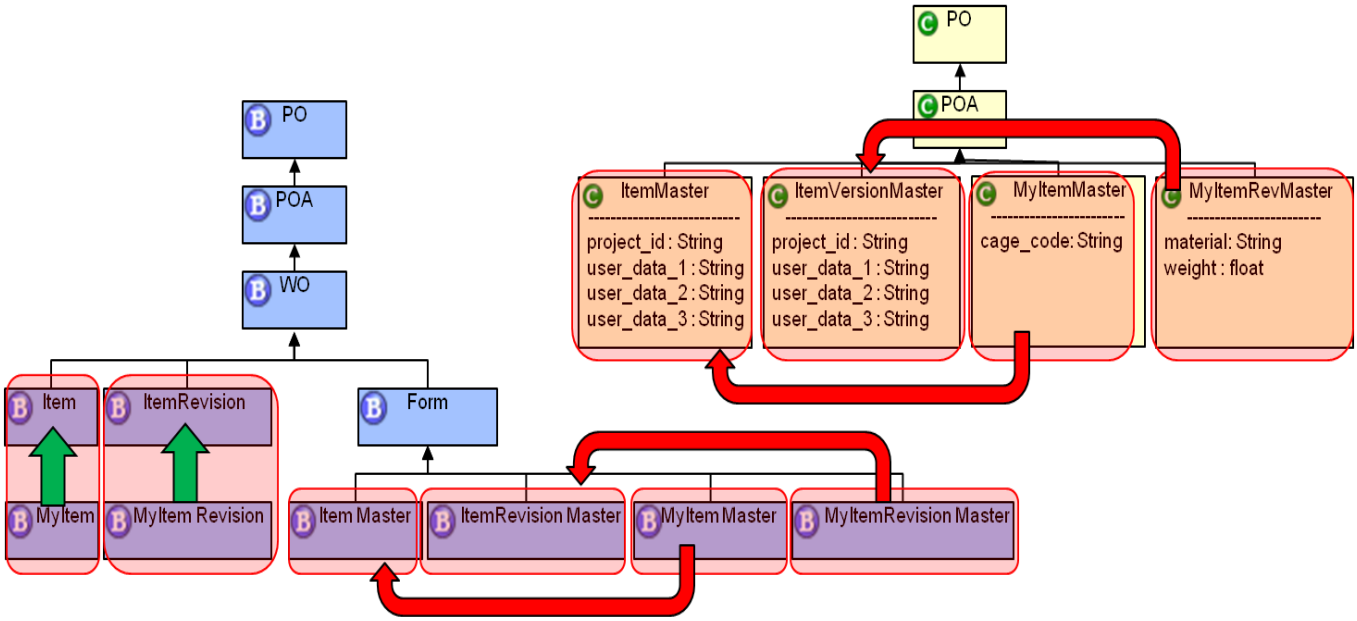


Figure 53: Issues with current best practices

10.4.4 New Best Practice for customizing forms

Because of the issues with the swapping of storage class, from Tc2007.1.5 and TcUA8, BMIDE client will not allow any more swapping of storage class on COTS form types. This is also documented in Teamcenter Release Bulletin: Chapter 3 as:*"Switching storage class of a Form business object no longer allowed".* In order to achieve the business needs of the users of adding and hiding properties on forms, a new practice is recommended. This section will explain this in detail.

10.4.4.1 What about my existing Forms that have already been swapped? Do I need to un-swap or migrate data?

No.

If you are upgrading to Tc2007.1.5 or later and there are already some COTS forms swapped for form storage classes, there is no need to do any work on these forms. There is no work needed for the swapped form for upgrade to be successful. After the upgrade is done, there may be some work needed if the import of the custom template in BMIDE client throws errors on the swapped forms explained in section 10.4.4.2

10.4.4.2 Fix errors on Swapped Forms

Threshold Properties is the term introduced for the attributes which are expected to be on the COTS business object. These properties may have LOV attaches, business rules etc attached to them. If customers have master forms with swapped storage classes, the threshold properties should be restored so that the rules and code doesn't break. To enforce this BMIDE has been augmented to throw an error to determine if there is a missing threshold property. These errors will be shown in the BMIDE console. This will typically occur when BMIDE template project is upgraded to BMIDE tc2007.1.5 or TCUA 8 or later. Or importing the custom template in BMIDE template project post engineering upgrade to Tc2007.1.5 or Tc8 or later. For more information see the release bulletin Chapter 3 for topic "Threshold properties must exist on the new Form storage classes" and error message "*The Form Storage Class on the form type form-name cannot be changed to the class MyClass unless the class has the following threshold*

properties: project_id, user_data_2. Add the missing properties". The release note will instruct you to add the threshold attributes to your new form storage class, so that the rules and code pointing to the missing properties will not have dangling references. For all the errors shown, add the threshold properties like attributes on the custom storage class. Let us take the example of threshold properties which are missing explained in section Issues with the current practice. The missing threshold properties are *project_id* and *user_data_2*. Now fix these properties post upgrade add the attributes called "project_id" and "user_data_2" to the custom storage class *MyClass*. See Figure 54 below. The threshold properties *project_id* and *user_data_2* are now available on *ItemRevision Master* form.

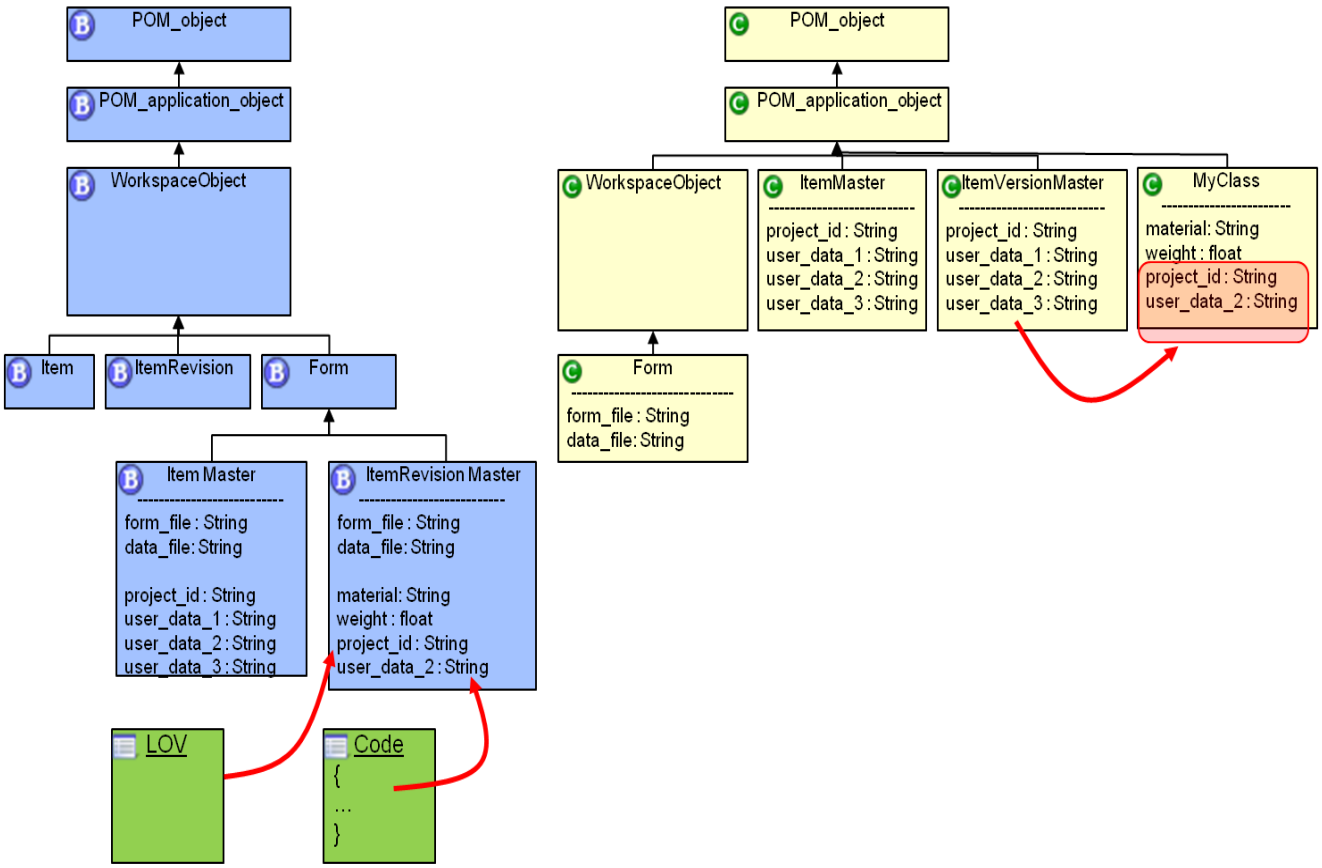


Figure 54: Threshold Properties

10.4.4.3 New Practice for hiding form properties

From Tc2007.1.5 or Tc8, follow a new practice to hide form properties. In order to hide the existing properties on a form, go to the property editor of the property in BMIDE client and make the property constant V*isible* value to false. Say that user_data_1 is to be hidden on *ItemRevision Master* form. To do this do the following BMIDE client.

1. Open the custom template project in BMIDE
2. Open the *ItemRevision Master* business object
3. Go to the Properties tab, select the property to hide, in this use case, user_data_1. See Figure 55

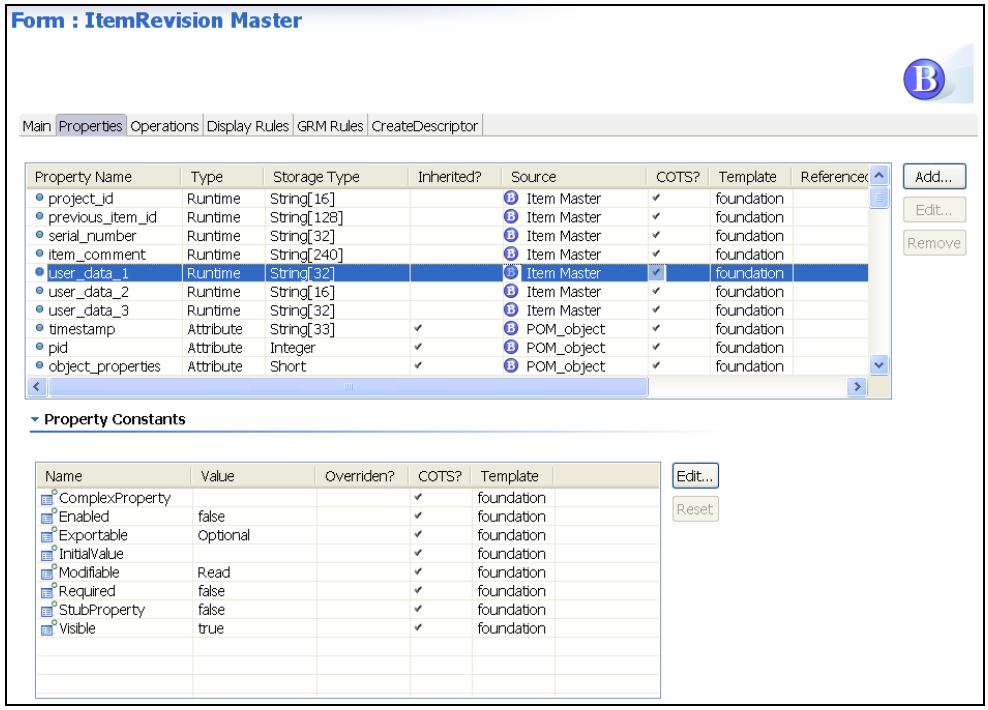


Figure 55: Best Practice for hiding properties

4. In the property constants section, notice that the value of constant Visible is equal to true. Click the edit button and change the *Visible* to false.
5. For this to take effect the updated template must be deployed and the server should be restarted.

10.4.4.4 New Practice for adding new form properties

To add new properties on an existing form, there is no need to actually create a new class and swap the storage class. Instead the new properties can be added on the form itself. Let us say two new persistent properties called weight and material are required to be added on *ItemRevision Master* form. For persistent properties, two new attributes can be added on the existing storage class itself. See the data model in Figure 56, which shows how the two new properties are custom attributes on the existing storage class *ItemVersionMaster*.

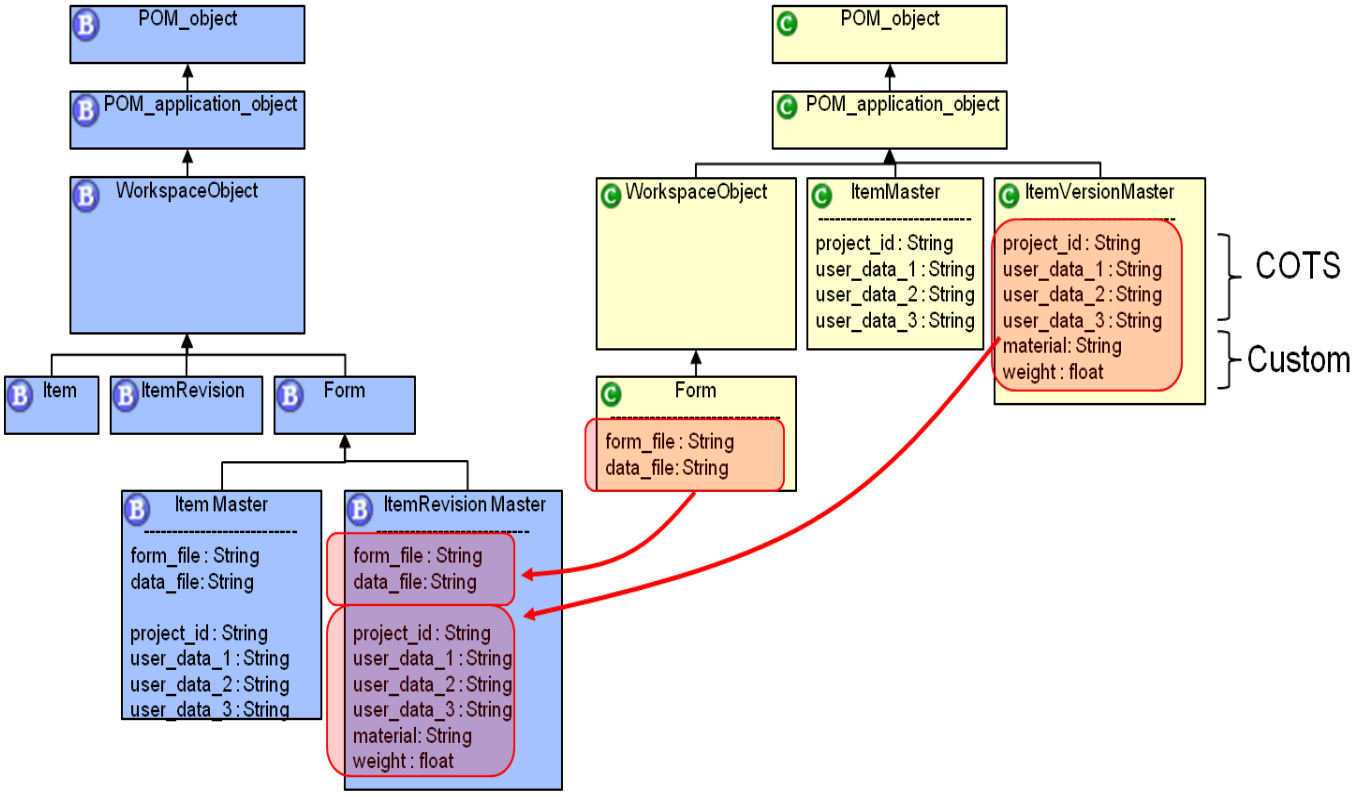


Figure 56: New best practice for adding new form propeties

In BMIDE, do the following steps to add new properties on existing form

Follow the following steps to do this in BMIDE client

1. Open the Template Project in BMIDE
2. Go to the properties tab of the business object *ItemRevision Master*
3. Click the add button which will take to the property wizard to add a new property. See Figure 57
4. Figure 58 shows the property wizard to add a new property. Add the weight and material properties through this wizard.

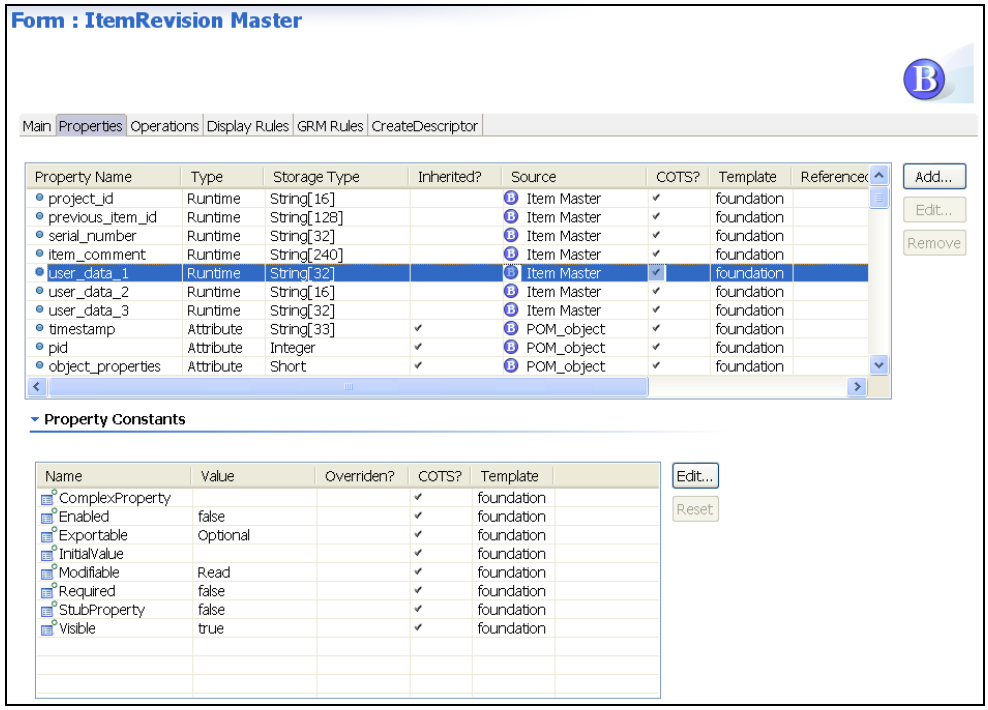


Figure 57

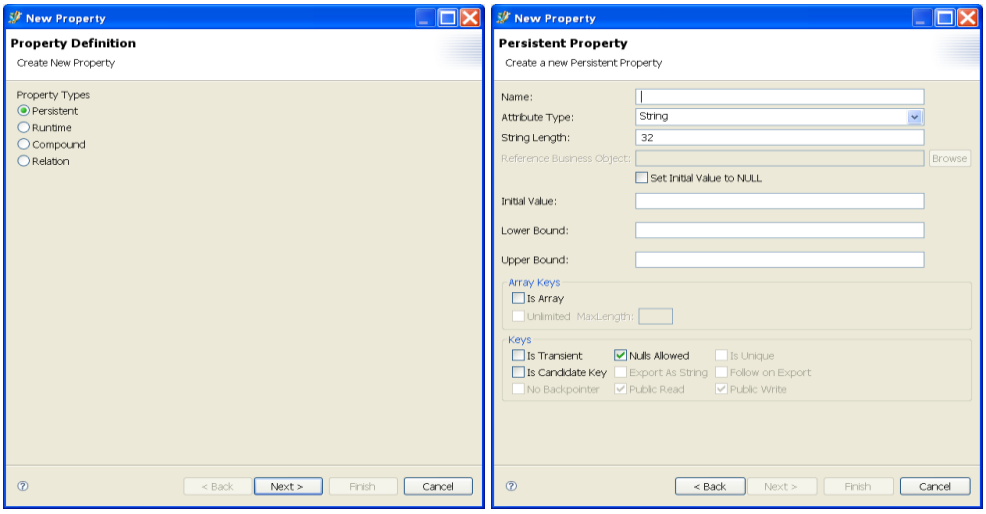


Figure 58

10.4.4.5 New best practice for parenting

When a new Item type say MyItem is created, verify that the new types *MyItem* and *MyItemRevision* are children of *Item* and *ItemRevision* so that they pick up their behavior. *MyItem Master* and *MyItemRevision Master* should be children of *Item Master* and *ItemRevision* Master. In other words **all the new types should be children of the parent type and not siblings**.

If you need to add new persistent properties to the custom master form, you may add them to the current storage class. If there is a need to create a new storage class, sub class the original storage class of the parent form so that the properties of the parent master storage are also inherited. See Figure 59. This way any rules on the original

storage class properties will not be broken. **This approach of adding new properties to the custom master form storage class should be taken under special circumstances**. It is recommended you add custom properties directly to your custom Item or custom ItemRevision. Please refer to the section: New Best Practices for Custom Item for further details.

The aforementioned behavior of parenting is being enforced in BMIDE when creating new Item business objects. Note that any forms customized prior to Tc2007.1.5 or Tc8 which don't fall in this pattern will still work and remain in the same state.

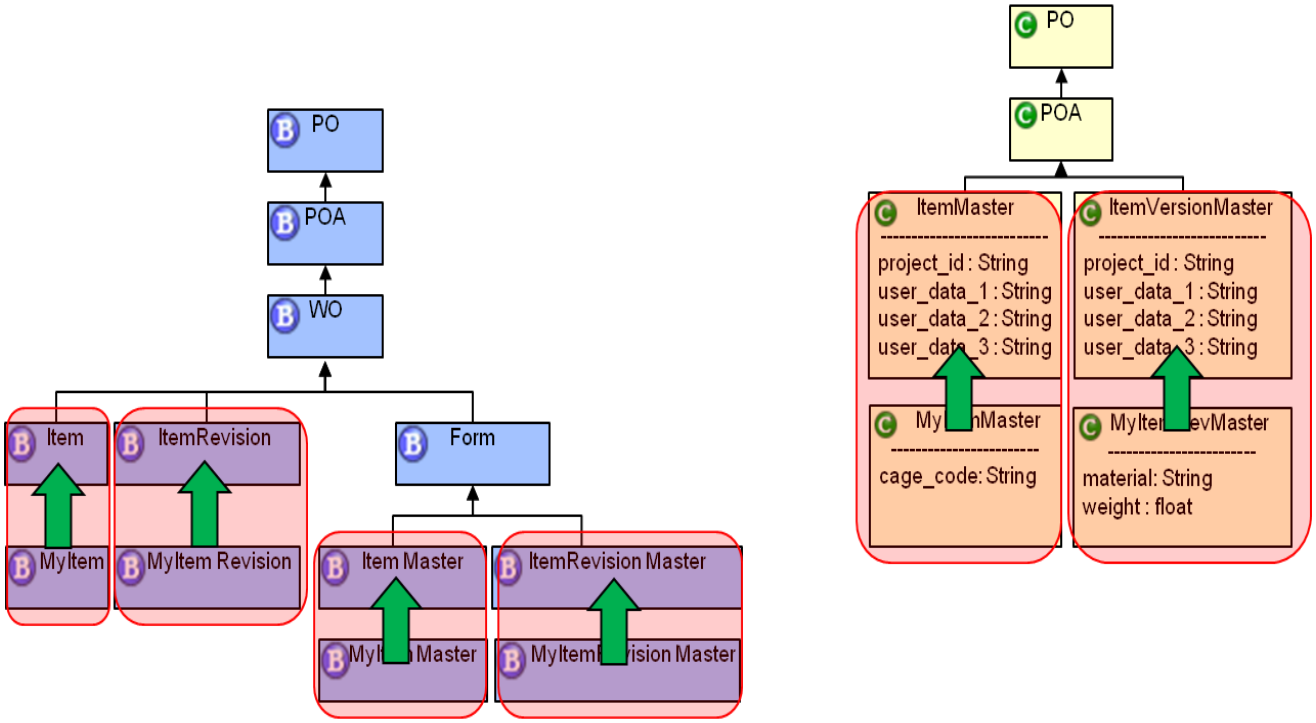


Figure 59: Best Practice For reparenting

10.5 Best practices for subclassing Form Storage Classes

This section discusses the best practices for creating a custom form storage class. **For a related topic, please also refer to the section:New Best Practices for Custom WorkspaceObject as Opposed to Custom Form.**

Starting from Teamcenter 8.0.0, you are not allowed to create a Form with form storage class as WorkspaceObject or subclasses of WorkspaceObject. Only a subclass of POM_object or POM_application_object can be used as form storage class.This restriction is enforced for the same reasons that are explained in sections 10.4.2 and 10.4.3 of this document. This section explains the issues with Item Master and ItemRevision Master forms perspective but the same explaination holds good for any generic form.

Example: This figure shows the **incorrect** way of creating the Form Type. In this example, the BusinessObject MyCustomForm has a Form Storage Class 'MyCustomFormStorage' which is a subclass of WorkspaceObject.

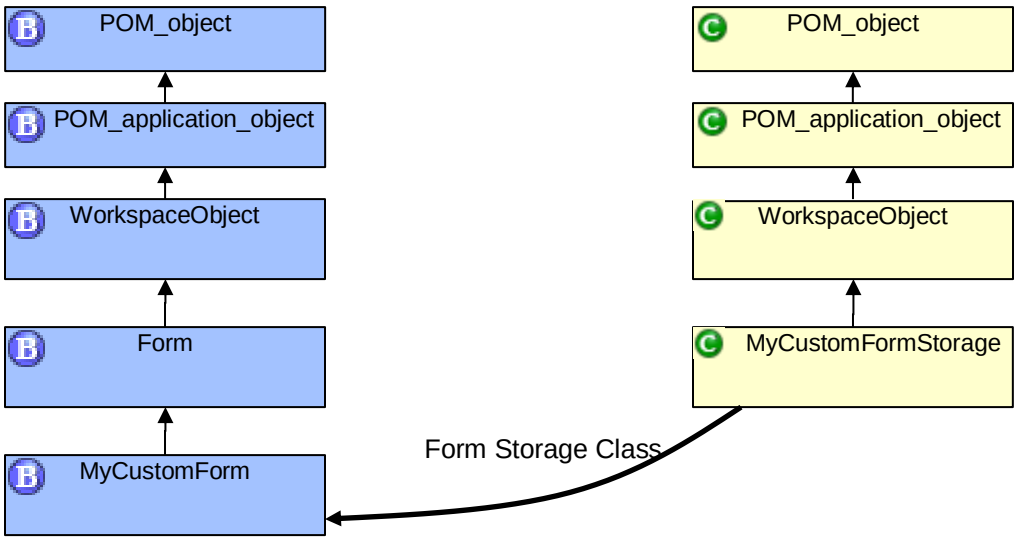


Figure 60 – Invalid Best Practice for Form Storage Class

Example: This figure shows the **correct** way of creating a Form Type. In this example, the BusinessObject 'MyCustomForm' has a Form Storage Class 'MyCustomFormStorage' which is a subclass of POM_application_object and not WorkspaceObject.

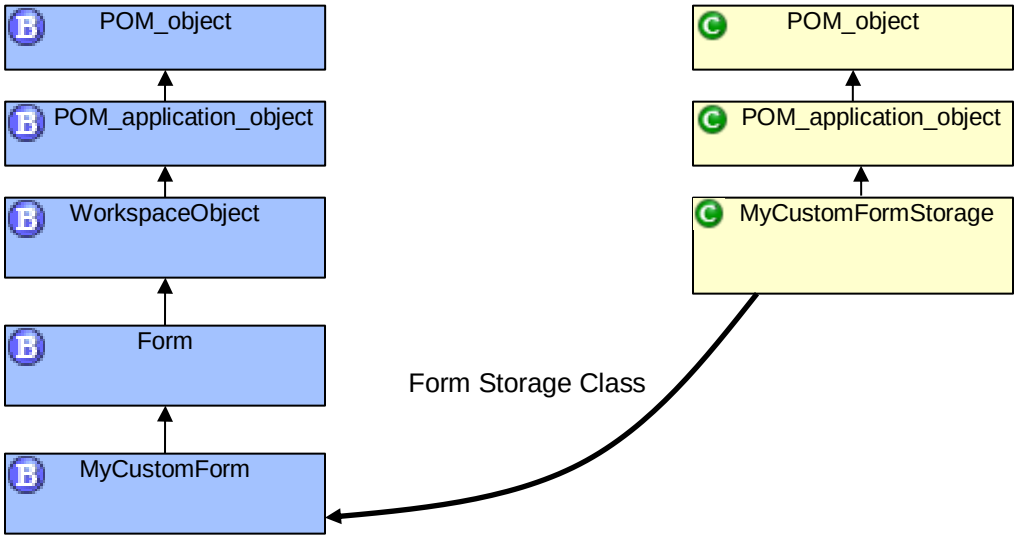


Figure 61 - Best Practice for Form Storage Class

Starting from Teamcenter 8.3.0, BMIDE UI has been updated to present an error message when you attempt to define a new Form type with form storage class as WorkspaceObject or subclasses of WorkspaceObject.

10.6 New Best Practices for Custom WorkspaceObject as Opposed to Custom Form

Starting from Teamcenter 8, it is recommended you should define a custom Business Object primary sub-class of WorkspaceObject rather than a custom Form when you need to create a sub-object to carry specific properties. All the functionalities supported by Form can be achieved with a custom WorkspaceObject Business Object.

1. In BMIDE, you can define a custom WorkspaceObject Business Object, add custom attributes to it, configure its Operation Descriptors, and add any other business rules for it.
2. Your custom WorkspaceObject Business Object will be automatically displayed in File->New->Other… dialog for creation. You can create your custom object using this create dialog without writing any custom code. You can configure stylesheets for your custom business object if required.
3. If necessary, you can also implement codeful extensions for your custom Business Object, such as implementing custom behavior for its create operations, implementing custom getter and setter operations for its properties, etc.

A custom WorkspaceObject Business Object is more efficient than a custom Form. A Form is represented in terms of two classes: Form class and its Storage class. As a result, retrieving the form properties derived from its storage

requires loading two objects: Form object and its storage object, thus causing performance overhead. This dual class model also causes unnecessary complexity and overhead in managing business rules, extensions and inheritance behaviors between Form and its Storage class. None of these issues are applicable to the single class model for a custom WorkspaceObject Business Object.

Going forward, you should define a custom WorkspaceObject Business Object instead of a custom Form for the sake of performance and easy maintainance. The implementations based on existing forms will continue to be supported.

10.7 New Best Practices for Custom Item

Starting from Teamcenter 8, BMIDE enforces the following behavior of parenting when you define a new custom Item, e.g. *MyItem*

1. *MyItem* and *MyItemRevision* are children of *Item* and *ItemRevision respectively*;
2. *MyItemMaster* and *MyItemRevisionMaster* are children of *Item Master* and *ItemRevision Master respectively*;
3. The custom master form storage classes: *MyItemMasterS* and *MyItemRevMasterS* are children of *ItemMaster* and *ItemVersionMaster* respectively.

All the aforementioned new custom types inherit the behavior from their parent type respectively.

After creating a new custom Item, i.e. *MyItem*, you should add custom properties directly to *MyItem* or *My ItemRevision* instead of custom master forms as often done in the past. This approach is more efficient.

1. All the business rules and behavior are modeled on one type, i.e. *MyItem* or *MyItemRevision*; All the business data are represented on one single class/object, e.g. *MyItem* object instead of the combination of *MyItem* object and *MyItemMaster* object.
2. Retrieving custom properties is straightforward and requires loading one object of *MyItem* or *MyItemRevision.* Retrieving master form properties would require loading multiple objects, e.g. master form relation, form and storage object without mentioning various overheads such as multiple layers of object model traversal, master form access delegation, etc.
3. You do not need to add a compound property on *MyItem* to represent a custom master form property. Retrieving this compound property is as expensive as retrieving the custom master form property itself as described at (2).

With this customization approach, the users do not need to interact with custom master forms.

It should be noted a master form is still created (without its storage object) by the system when an item or item revision is created. This is because a master form could be required by some Teamcenter applications and 3rd party solutions. Use the “Visible” property constant or a Stylesheet to hide the custom master forms in the UI.

11 Upgrading from Teamcenter Engineering to Teamcenter

Detailed information about how to upgrade form Teamcenter Engineering to Teamcenter is now documented in the Upgrade Guide. See the “Upgrade Guide” in the standard documentation set.

Additional information about upgrades is listed below.

11.1.1 Prerequisite tasks before starting an upgrade

1. **Check Process Templates on Change Types:** Before starting the upgrade from Teamcenter Engineering, please inspect all customer change types from the Type application in Teamcenter Engineering Portal and ensure that at least one process template is assigned to each change type.

   The Type application in Teamcenter Engineering is shown in the screenshot below. To check if a change type has at least one process template assigned to it, expand the Change node in the Types tree to view all change types. Double click on each Change Type and check if the selected process templates list is empty. If it is empty, then select one process template from the “Defined process templates” list and click on the add button (+) to add it to the “Selected process templates” list. Click on the “Modify” button at the bottom to complete the change.

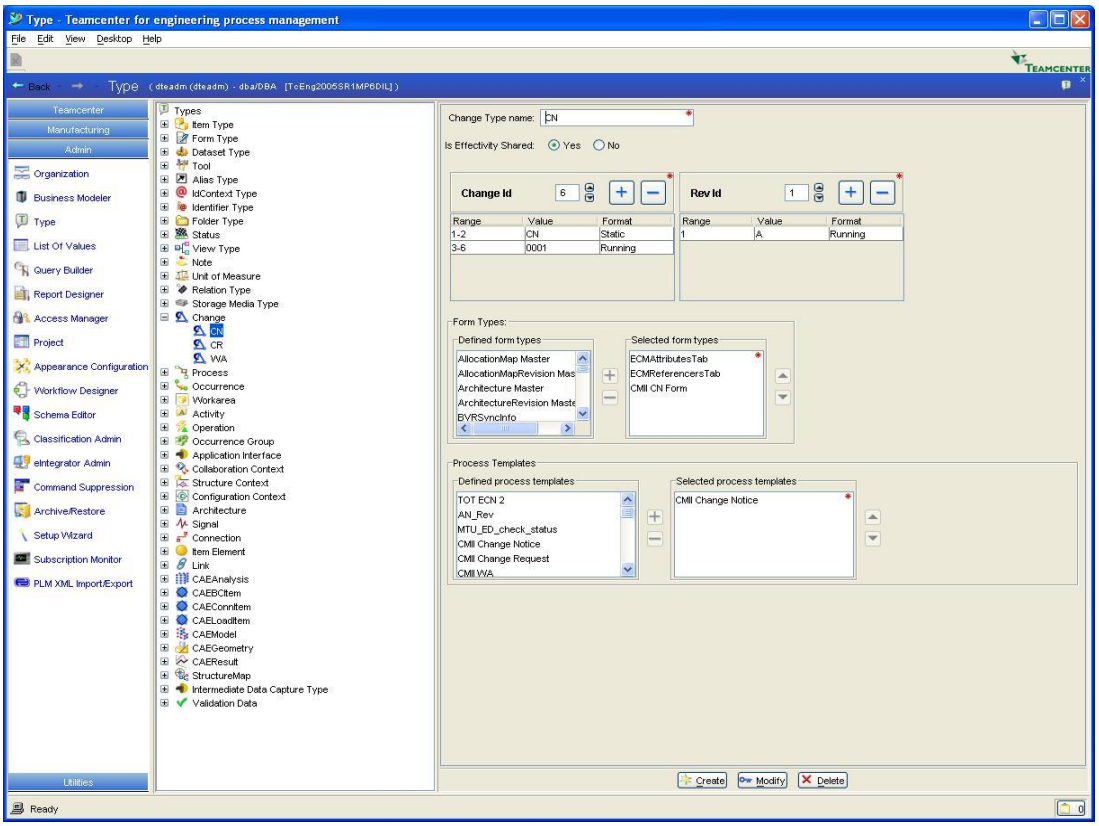


Figure 62: Prerequisite tasks - Check Process Templates on Change Types

2. **Correction of Form Storage class on custom Form types:** Before starting the upgrade from Teamcenter Engineering, please inspect your existing Form types to ensure none of the custom Form types has WorkspaceObject or its subclasses as form storage class. For example, say you have a custom Form named,

"MyCustomForm" and it is associated with the form storage class name "MyCustomFormStorage". The expectation is that the class "MyCustomFormStorage" does not inherit from WorkspaceObject or its subclasses. If there are any Form types that are created with WorkspaceObject or its subclasses as form storage class, then you are required to follow these steps to correct the data in your Teamcenter Engineering database before starting database upgrade.

a. In Schema Editor, create a new custom class under POM_object or POM_application_object with same definition of the existing form storage class except for its parent class.

b. In Type Admin Application, modify the existing custom Form type to use the new custom class as form storage class.

Note: Your database can contain instances of the Form Type that uses the old form storage class. For example, there could be instances of "MyCustomForm" that uses "MyCustomFormStorage". These instances will continue to work as before even after changing your form storage for "MyCustomForm". Since you have these instances, you cannot remove "MyCustomFormStorage" data model definition from your database. All new instances created from here on of the type "MyCustomForm" will start using the form storage class "MyNewCustomFormStorage".

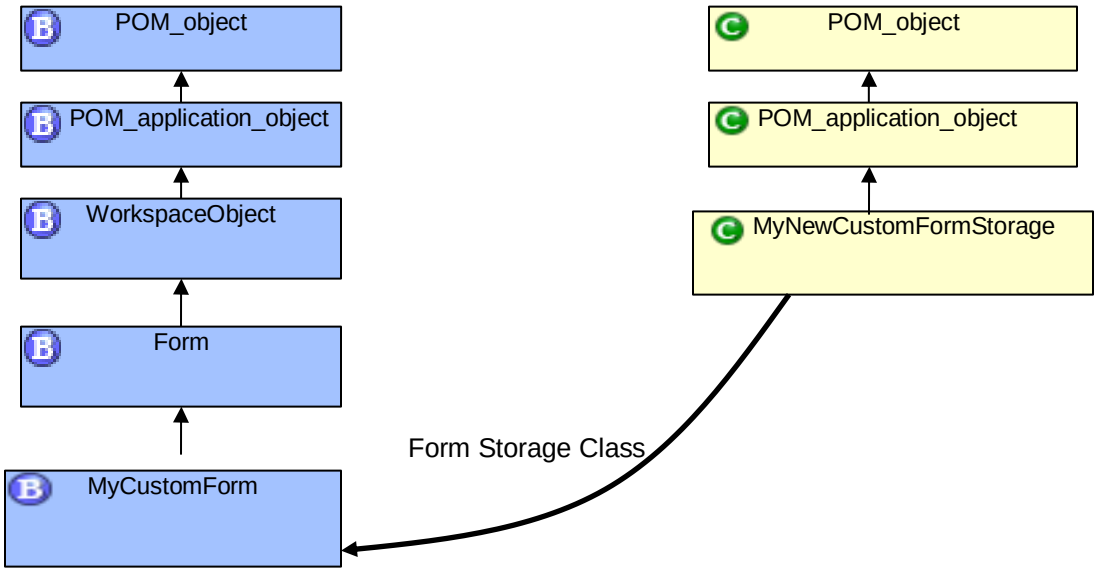


Figure 63 - Corrected Form Storage Class structure

11.2 Template Match Tags

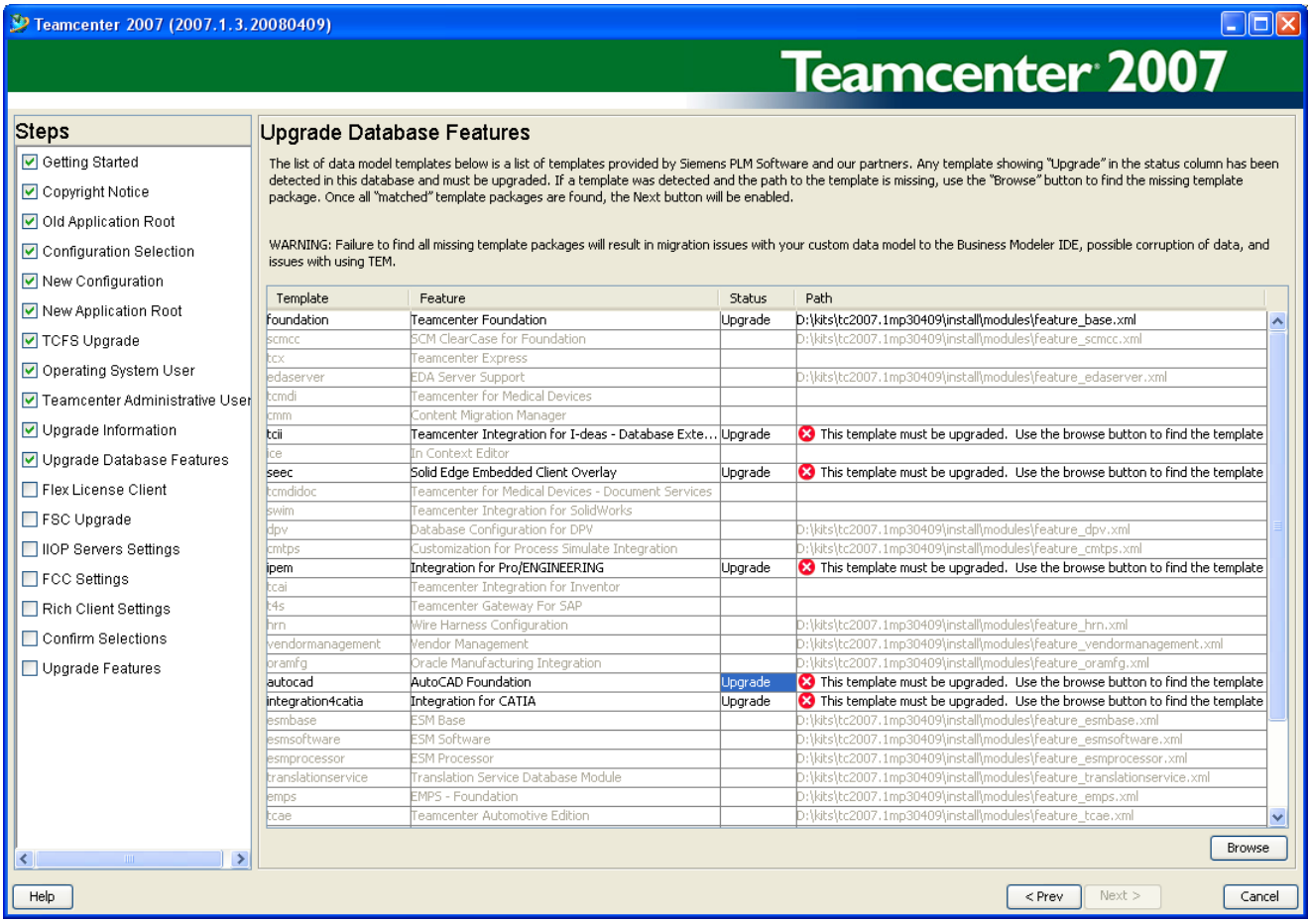


Figure 64 TEM Upgrade - Upgrade Database Features page

TEM uses template match tags to determine which templates are required for upgrade. The results of the match are shown in the Upgrade Database Features panel in Figure 64. These tags are only used by TEM to match templates that were available in Engineering. Therefore new templates introduced only for Teamcenter unified architecture do not have template match tags. If you plan to use your BMIDE template to upgrade other Teamcenter Engineering databases, then read on. Otherwise you may skip this section.

The TEM panel in Figure 64 shows a list of all templates that were available in the Engineering timeframe. Each template provides two or three template match tags. Each tag tells TEM to search in the database for a specific data model definition that should be in the database if the template's data model had been installed into Teamcenter Engineering. If all template match tags are found, then the template is a match in the database and the template is marked for upgrade. If any of the template tags are not matched, then the template is not marked for upgrade. Typically the template match tags are used to look for a class or type definition since these are the most common extensions in any given template. However, other data model definitions can be used as long as they are unique to your template and are not also defined in another template.

Warning: Do not attempt to edit another template's template match tags. These tags are selected for a specific reason by the owners of the template. If you believe that you have installed a template into Engineering that TEM is not matching please contact GTAC for assistance. Altering the tags will cause issues with the upgrade and extraction of custom data model definitions to the BMIDE.

Not only are template match tags used by all of the templates delivered by Siemens PLM Software and our partners, the template match tags can also be used by customers. The following is a list of reasons why you should use

template match tags. If any of these scenarios apply to you, then continue reading, otherwise you may skip this section.

- You upgrade any database from Engineering to Teamcenter unified architecture in order to extract out a custom BMIDE template. Then you use the package wizard to build a template feature. Then you intend to use this packaged template to upgrade other database sites from Engineering to Teamcenter unified architecture.
- You are developing data model customizations for an add-on solution to Teamcenter. Example: 3rd party developer. You have customers who are still using Engineering and they need to upgrade to Teamcenter unified architecture.

If you need to create template match tags, then we suggest that you create two or three minimum if possible. The template match tags are placed into the feature.xml file for your template. If you are assured that your data model definitions are unique to your add-on solution, then you can assume that TEM will use the template_match tags to match your template and not someone else's template.

Since a template defines data model definitions, you should only use template match tags to match those data model definitions which should exist in an Engineering database that has your solution installed. Do not match something that was optional, as all the template match tags must match in order for TEM to match the template. For example if you gave your customer the option to install a Class, LOV or Naming Rule, then you cannot assume that all databases have that Class, LOV, or Naming Rule. Therefore these optional elements are not candidates for a template match tag. Additionally do not try to match instances of data model classes. For example do not try to match an Item ID, or a Form name or dataset name entered by a user; you can only match a data model definition.

Good candidates for template match tags are those data model definitions that were added by your solution and existed in all supported Engineering versions (V913, V10, V10 SR1). Examples include

- Class name
- Type name
- Dataset type name
- LOV name
- Naming Rule name
- GRM rule
- Tool name
- View Type

Note that all data model definitions when loaded into a database are stored as an instance of a class. For example, every Class that is added to a database is stored as an instance of the POM_class class (in addition to the fact that it also adds its own table in the database). Every Type that is added to a database is stored as an instance of the ImanType class. Therefore when we build template match tags we want to see if the database table for Classes (POM_class) has a class by a given name. Or see if the data table for Types (ImanType) has a type by a given name.

Here is the format for template match tags. We need three things:

- The class name that stores the data model definition (database table)
- The attribute name that stores the name of the definition (database table column)
- The value of the attribute

Template match tags are written into the feature.xml file in the following format. Note that there are no spaces directly before or after the comma; however it is okay to have a space inside the value field if the value has a space in it. Also

note that each template match tag increments the property name value: template_match1, template_match2, template_match3, template_matchN.

<property name="template_match_1" value="ClassName,AttributeName,value"/>

<property name="template_match_2" value="ClassName,AttributeName,value"/>

<property name="template_match_3" value="ClassName,AttributeName,value"/>

If your template added two classes, here is what the template match tags would look like.

<property name="template_match_1" value="POM_class,name,ClassA"/>

<property name="template_match_2" value="POM_class,name,ClassB"/>

In this example, the first field is “POM_class” and this is the name of a database class (database table) that stores Teamcenter class definitions. The second field “name” is thge attribute name (database table column) of the “POM_class” (table) that stores the name of the class. “ClassA” and “ClassB” are Teamcenter class names and each of these two values should be found as a value of one of the instances of the specified database table and column.

Let us try another one. If your template added two types TypeA and TypeB, here is what the template match tags would look like.

<property name="template_match_1" value="ImanType,type_name,TypeA"/>

<property name="template_match_2" value="ImanType,type_name,TypeB"/>

Where “ImanType” is the database class that holds type definitions. “type_name” is the attribute name (database table column) that stores the type names. And TypeA and TypeB are two values that should be found in this table and column.

To match Dataset type definitions DatasetName1 and DatasetName2, use the following format

<property name="template_match_1" value="WorkspaceObject,object_name,DatasetName1"/>

<property name="template_match_2" value="WorkspaceObject,object_name,DatasetName2"/>

Note that the above two match tags use the WorkspaceObject class and object_name attribute to match because the dataset type definitions are stored in the Dataset class which is a subclass of WorkspaceObject. The dataset name is an attribute that is defined on the WorkspaceObject class (not defined on Dataset class). Since these template match tags use a SQL query to find the match, we have to specify the exact table that has the instance. Therefore we use WorkspaceObject for the first field.

To match Tool definitions, use the following format

<property name="template_match_1" value="WorkspaceObject,object_name,Tool1"/>

<property name="template_match_2" value="WorkspaceObject,object_name,Tool2"/>

As in the case of Dataset type definitions, Tool objects are stored in the Tool class which is a subclass of Workspace object. The tool’s name is stored in the object_name attribute which is defined on WorkspaceObject. Therefore we use WorkspaceObject in the first field.

To match LOV names, use the following format

<property name="template_match_1" value="ListOfValues,lov_name,LOV 1"/>

<property name="template_match_2" value="ListOfValues,lov_name,LOV 2"/>

<property name="template_match_3" value="ListOfValues,lov_name,LOV 3"/>

The LOV definitions are stored in the ListOfValues class using the lov_name attribute. Note also that each of the value fields has a space in the LOV name. This is a demonstration to show that it is okay to have a space inside the value field if the value has a space in it. Many type names, LOV names, and other data model elements can have spaces.

To match View Type definitions, use the following format

<property name="template_match_1" value="PSViewType,name,View Name 1"/>

<property name="template_match_2" value="PSViewType,name,View Name 2"/>

The View Type definitions are stored in the PSViewType class using the name attribute.

You can also use a combination of classes and types or other data model objects:

<property name="template_match_1" value="POM_class,name,ClassA"/>

<property name="template_match_2" value="ImanType,type_name,TypeB"/>

<property name="template_match_3" value="ListOfValues,lov_name,LOV1"/>

Your feature file does not have any template match tags by default; therefore, you must add them. They also must be added in the proper section of your feature.xml file. To add them, launch the BMIDE and go to the Navigator view. Find and double-click your feature_<template_name>.xml file. This will open the file into an editor. Find the section in the file that has the following line:

<property name="template_file" value="${template_name}_template.xml"/>

Then add your template match tags immediately after it as shown in Figure 65 at the bottom.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
Siemens and the Siemens logo are registered trademarks of Siemens AG.
UGS, Teamcenter and UGS Teamcenter are trademarks or registered trademarks of UGS or its subsidiaries.
This software and related documentation are proprietary to UGS Corp. (c)2007 UGS Corp. All rights reserved.
-->
<!--
    Document   : feature_dpv.xml
    Description: This XML is used by TEM to install or upgrade the dpv solution.
-->

<feature>

    <name value="Database Configuration for DPV"/>
    <property name="template_name" value="dpv"/>
    <depends name="foundation" value="8C061DD51E13E0CB9DC4687B1A3348BE"/>
    <size value="2"/>
    <removable value="false"/>
    <root value="true"/>
    <singular value="false"/>
    <group value="foundation"/>
    <bundle value="${template_name}Bundle.xml"/>
    <description value="${template_name}.description"/>
    <guid value="2D41613B6F9C42A08A7E6A966C712AB9"/>


    <files>
        <code>
            <unzip file="tc/${template_name}_template.zip"/>
            <unzip file="tc/${template_name}_install.zip"/>
        </code>
    </files>

    <property name="template_file" value="${template_name}_template.xml"/>
    <property name="template_match_1" value="ImanType,type_name,MEAttribute"/>
    <property name="template_match_2" value="ImanType,type_name,MEFeature"/>
    <property name="template_match_3" value="ImanType,type_name,MEInspection "/>
```

Figure 65 - Sample feature XML file

You should test your template match tags to ensure that they work correctly. Use the Package wizard to generate your template package and then use your template package to upgrade a test Teamcenter Engineering database that includes your solution. The testing should include each version of your add-on solution with each version of Teamcenter Engineering that is currently supported by your solution (V913, V10, V10 SR1).

During testing, when TEM displays the Upgrade Database Features (Figure 64) panel, you will need to click on the Browse button to locate your template's feature.xml file. Once TEM locates it, TEM will read the template match tags in the feature.xml file. If all template match tags match, TEM will display your template in a new row in the table and mark it for upgrade. If TEM does not match all of your tags, then TEM will display a message saying that the template was not installed in the database. If this happens, carefully review your template match tags and ensure that they are accurate or check that the database has the data model definitions that you are trying to match. Perhaps these data model definitions were optional instead of required.

11.3 Obsolete ITK and Utilities

In Teamcenter Engineering, it was very common to use ITK (Integrator Toolkit), *PLMXML* scripts, *sb*, and *install_types* utilities for creating data model definitions and business behaviors in a database. Once you have upgraded your database to Teamcenter unified architecture, the only supported method for extending the data model and behaviors is to use the BMIDE client and a custom template. All of the aforementioned utilities, ITKs and PLMXML files that supported data modeling are obsolete. To restate, they are not deprecated, they are obsolete. You cannot use them.

The full list of obsolete ITKs and utilities is listed in the Release Bulletin in the section titled “Obsolete utilities”. See this list for full details. Note that not all ITKs, PLMXML, and utilities are obsolete; just the ones related to the objects that the BMIDE manages.

Note that most of these utilities are still available in the code base and documented in the user guides. However, they are for internal use only. If you attempt to use any of these API or utilities you will risk corrupting your product data, corrupting the data model definitions, causing issues for upgrade, and possibly rendering the BMIDE inoperable until the system can be returned to its previous state. The reason these utilities and API have are obsolete is because the BMIDE XML now holds the master copy of the data model. If any changes are made outside of this, the BMIDE will not be aware of it and will undue any changes that you make exterior to the BMIDE.

11.4 Post Upgrade Steps (Dataset Merges)

The following are some steps that you may want to perform after all upgrade steps are completed.

Recall that prior to the upgrade; you used the Type Analysis Tool to generate a report of your data model in the database. This report included information about type name collisions, missing types, etc. One of the sections titled “Detailed Analysis of Dataset Merges” gives details about any custom Datasets that you own that will be merged with the COTS dataset definition.

If you had any Datasets in this section and you chose to let the dataset merge with the COTS definition, then you have to more optional tasks to complete.

- First you may have custom tools, custom references, and custom tool actions were merged (blended) into the COTS tools, references, and tool actions. This will leave the Dataset definition with two sets references, View Tools and Edit Tools. Users may be asked to choose between the two tools each time they open or edit the dataset.
    - If you feel that you would like to remove your custom tools, references, and tool actions, and go with the COTS tools, references, and tool actions, you can follow the steps in section 17.3.
- Second, if you plan to use this BMIDE template to upgrade other database sites from Engineering to Teamcenter unified architecture, you will need to clean up your baseline.xml file.
    - Open up your *<Project>/<template_name>_tcbaseline.xml*
    - Search and remove any of the following types of XML tags that include a reference to your merged dataset name. The following examples demonstrate various Dataset and Business rule XML tags that can occur in your file that could include the merged type name. If any of these are found to use your merged type name, then remove the XML tag. The examples below demonstrate a “PDF” dataset merge. Note do not clean up these definitions from your custom template source files; you should only cleanup your baseline.xml file.

        - TcDatasetReferenceAttach

```
<TcDatasetReferenceAttach datasetType="PDF">
     <TcDatasetReference name="PDF">
       <TcDatasetReferenceInfo template="*.pdf" format="BINARY"/>
</TcDatasetReference>
```

        - TcDatasetToolActionAttach

```
<TcDatasetToolActionAttach datasetType="PDF"
     tool="Adobe" action="Open">
     <TcToolActionReference export="true" referenceName="PDF"/>
     <TcToolParameter name="$PDF"/>
</TcDatasetToolActionAttach>
```

- TcDatasetToolAttach

```
<TcDatasetToolAttach datasetType="PDF">
  <TcDSEditTool name="Adobe"/>
</TcDatasetToolAttach>
```

- TcDeepCopyRule

```
<TcDeepCopyRule conditionName="isTrue" copyRelationAttributes="true"
      copyType="CopyAsObject" isRequired="false"
      isTargetPrimary="true" objectTypeName="FullText"
      operationName="ITEM_create_from_rev"
      relationTypeName="IMAN_specification" secured="false"
      typeName="PDF"/>
```

- TcExtension

```
<TcExtension cannedExtension="false" description="" internal="false"
      languageType="CPlusPlus" libraryName="libqsearch"
      name="CreateUpdateBBoxAndTSO">
     <TcExtensionValidity parameter="TYPE:PDF:AE_save_dataset:3"/>
 </TcExtension>
```

- TcCompoundPropertyRule

```
<TcCompoundPropertyRule destPropertyName="work_remaining"
   destTypeName="ScheduleTask" isReadOnly="false"
   pathToSource="REF(exec_form_info,PDF).work_remaining"
   sourcePropertyName="work_remaining"
   sourceTypeName="TaskExecutionFormInfo"/>
```

- TcPropertyConstantAttach

```
<TcPropertyConstantAttach constantName="Modifiable"
      propertyName="comments"
      typeName="PDF" value="Write"/>
```

- TcGRMRule

```
<TcGRMRule attachability="WriteAccessReq" changeability="Changeable"
      detachability="WriteAccessReq" primaryCardinality="-1"
      primaryTypeName="PDF" relationTypeName="HasParticipant"
      secondaryCardinality="-1" secondaryTypeName="Participant"
      secured="true"/>
```

- TcNamingRuleAttach

```
<TcNamingRuleAttach case="Mix" conditionName="isTrue" description=""
      override="false" propertyName="item_id"
      ruleName="Default Requirement ID Rule"
      ruleType="NamingRule"
      typeName="PDF"/>
```

11.5 Baseline file cleanup

After Teamcenter Engineering upgrade to Teamcenter 8.x, there are some MetaModel elements created for your custom business objects. These MetaModel elements are not expected to reside in the tcbaseline file since the tcbaseline file should represent only your Engineering customizations.

Follow the below steps to clean up the MetaModel elements from your tcbaseline file.

1. Launch the BMIDE client with your custom project and go to the Navigator view.
2. Open the tcbaseline file under <project> directory in a text editor.
3. Remove all XML tags named <OperationInputType>.

   Example:

   ```
   <OperationInputType typeName="TestTifCreI" parentTypeName="DatasetCreI"
   typeClassName="TestTifCreI" isAbstract="false" artifactName="TestTifCreI"/>
   ```

4. Remove all XML tags named <TcTypeConstantAttach>.

   Example:

   ```
   <TcTypeConstantAttach constantName="CreateInput" typeName="TEST_data"
   value="TEST_dataCreI"/>
   ```

5. Save the tcbaseline file.

12 Upgrading from Teamcenter to a new release of Teamcenter

Detailed information about how to upgrade form Teamcenter to Teamcenter is now documented in the Upgrade Guide. See the “Upgrade Guide” in the standard documentation set.

Additional information is listed below that you may need to consider during your upgrade.

12.1 Correction of Form Storage class on custom Form types

Before starting the upgrade from Teamcenter unified architecture, you are required to ensure that none of the custom Form types has WorkspaceObject or its subclasses as form storage class. For example, say you have a custom Form named, “MyCustomForm” and it is associated with the form storage class name “MyCustomFormStorage”. The expectation is that the class “MyCustomFormStorage” does not inherit from WorkspaceObject or its subclasses. If there are any Form types that are created with WorkspaceObject or its subclasses as form storage class, then you are required to follow these steps to correct the data in your Teamcenter database before starting database upgrade.

1. Run BMIDE client from existing source installation. For example, if you are at Teamcenter 2007.1.5 and are planning to upgrade to Tc8.0.0, then you must launch the BMIDE in your existing Teamcenter 2007.1.5 installation.
2. Get a list of all Form types having form storage classes that inherit from WorkspaceObject or its subclasses.
3. For each of these form storage classes, create a custom class under POM_object or POM_application_object with the same definition as the existing form storage class.
4. Choose File -> Save Data Model to save the data model changes.
5. Open the XML file under extensions folder in Navigator view and manually edit the XML file to manually change “formStorageClassName” attribute value to the new custom class for each of the custom Form types having the form storage classes which are subclasses of WorkspaceObject or its subclasses.

   For example:

   Before editing the XML file

   ```
   <TcForm typeName="MyCustomForm" parentTypeName="Form"
     typeClassName="Form" formStorageClassName="MyCustomFormStorage"/>
   ```

   After editing the XML file:

   ```
   <TcForm typeName="MyCustomForm" parentTypeName="Form"
     typeClassName="Form" formStorageClassName="MyNewCustomFormStorage"/>
   ```

2. Save the modified XML file.
3. In the Navigator view reload the data model by selecting RMB -> Reload data model.
4. After the BMIDE reloads, ensure that there are no errors reported in the Console view.
5. Open your changed custom Form type in the Business Objects view and check that the form storage class is now pointing to the new custom
6. Package and deploy the template to the Teamcenter 2007.1.5 database.

Note: Your database can contain instances of the Form Type that uses the old form storage class. For example, there could be instances of “MyCustomForm” that uses “MyCustomFormStorage”. These instances will continue to work as before even after changing your form storage for “MyCustomForm”. Since you have these instances, you cannot remove “MyCustomFormStorage” data model definition from your database. All new instances created from here on of the type “MyCustomForm” will start using the form storage class “MyNewCustomFormStorage”.

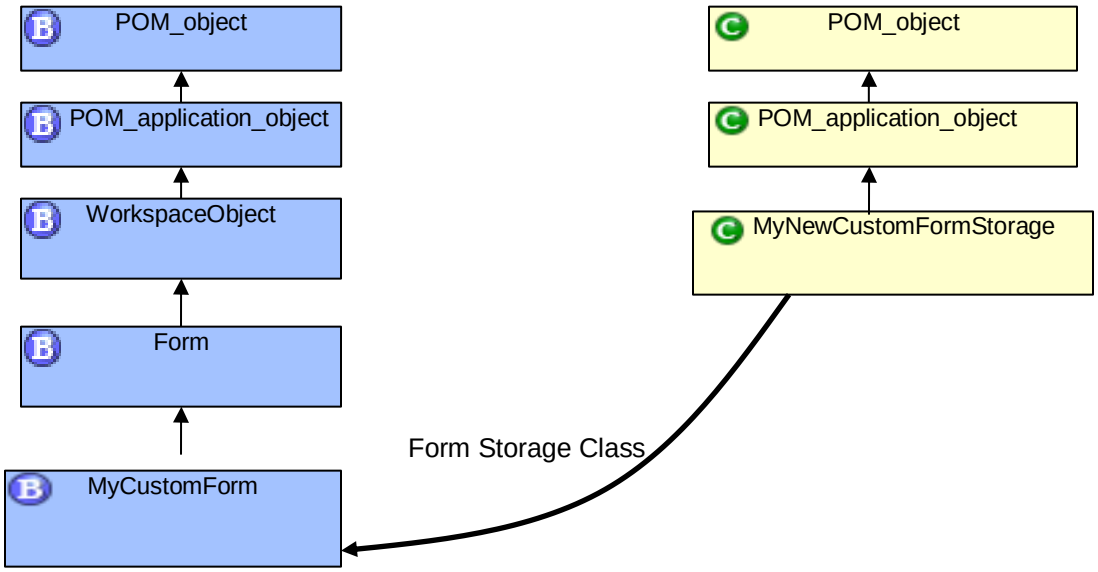


Figure 66 - Corrected Form Storage Class structure

13 Troubleshooting BMIDE Validation errors

This section discusses how to troubleshoot BMIDE validation errors. These errors can show up within the BMIDE Console View whenever a BMIDE template project is loading the data model. The BMIDE loads the data model for the following events: BMIDE startup, Import of a project, import of a file, or Reload Data Model.

The following checklist should be followed if you see an error in the BMIDE Console View:

1) Did the error occur right after synchronizing the custom template project in the BMIDE client with a Source Control Management (SCM) system? If yes, then the issue may be related to a file merge where the changes between two users were merged incorrectly. Or two users added the same definition or conflicting definitions. Work with the two developers who submitted the changes to correct the XML file.

2) Did the error occur while upgrading the custom template project in the BMIDE client? If yes, this means that something new was added to one of latest dependent templates that conflicts with one of the custom definitions in your BMIDE template. These issues fall into two groups:

   a. Type Name Collisions:

      A type name collision is where a new type or class definition has been added to one of the dependent templates and you also have the same type or class name in your custom template. Since Teamcenter cannot support two types or classes with the same name, your custom type will have to be fixed.

      A type name collision error message would look like this in the BMIDE Console View.

      ```
      ------------------------------------------------------------
      Model Error: file.xml Line:8 Column:16 A Business Object is already
      defined with the name "PDF". Choose another name.
      ------------------------------------------------------------
      ```

      A class name collision would look like this in the BMIDE Console View.

      ```
      ------------------------------------------------------------
      Model Error: file.xml Line:11 Column:57 A Class is already defined with
      the name "MyClass". Choose another name.
      ------------------------------------------------------------
      ```

      Type name collisions fall into one of three categories.

      i. A collision is occurring because of known issues with orphaned types. See section 17.2 to see if any of the collisions you have in your template are related to any known orphan type.
      ii. If your custom type is a dataset, see section 13.2. (A dataset is any type/class that is a subtype of the “Dataset” type/class).
      iii. If your custom type is NOT a dataset, see section 13.1. (This is any type/class that is not a subtype of the “Dataset” type/class).

   b. Other issues:

Other data model issues can occur in almost any object managed by the BMIDE. Each must be handled on a case by case basis. If you have an issue in this area, please contact GTAC for assistance. The following are some examples of these types of error messages, though this is not a complete list of error messages. Note that the file name, line number and column number have been removed for brevity.

```
-------------------------------------------------------------
Delete TcForm is not supported.

Delete TcStandardType is not supported.

Delete TcDataset is not supported.

Delete TcLOV is not supported.

Delete TcLOVAttach is not supported.

Delete TcAttribute is not supported.

Delete TcPropertyConstantAttach is not supported.

Change TcClass is not supported.

Change TcAttribute is not supported.

Change TcStandardType is not supported.

An LOV is already defined with the name "User Names". Choose another name.

Can not attach another LOV "User Names" to Property "owning_user" in
object "WorkspaceObject".

Undefined Dataset ZIP in TcDatasetToolAttach.
-------------------------------------------------------------
```

13.1 Regular Type name collisions

If your custom type is colliding with a COTS type name then follow these directions.

A collision will prevent a COTS type from being added to the data model. Since there is a collision and the COTS type must be allowed to be added to the database data model, your custom type must be renamed. Therefore you must fix this collision before you upgrade your database. If you have been following the process for upgrading outlined in this document, you would have discovered this collision while upgrading your custom BMIDE template but before upgrading the database. If however, you upgraded your database first, please revert the database back to the former version of Teamcenter unified architecture before following these steps.

1) Download the Change Type Name utility from GTAC
   - Go to http://ftp.ugs.com/
   - To the right of the Product Updates section, pull down the menu and select Teamcenter.
   - Click on the MP link.
   - Click on the appropriate platform.
   - Click on the tools link.

- Click on the ChangeTypeName link
- Download the following files:
    - ChangeTypeNameDocumentation.pdf
    - ChangeTypeName_TcEng913_<platform>.zip
    - ChangeTypeName_TcEng2005_SR1_2007_MP2r_<platform>.zip
    - ChangeTypeName_TcEng2005_SR1_2007_MP7_<platform>.zip

2) Read the Change Type Name utility documentation completely.
    a. Follow the steps outlined in the documentation on how to change your colliding type name to a new type name in the database and other areas.
    b. For assistance on selecting a new type name, try to use a prefix to avoid any future collisions. See section 9.
3) Rename the type name in the custom BMIDE template
    a. Launch the BMIDE client and go to the Navigator View
    b. Open each source file that is in your <project>/extensions folder into a text editor.
    c. Manually search for the old type name and replace it with the new type name. Note that there could be more than one occurrence of the type name.
        i. For example the name could be used in only the type definition, but could also appear in a class definition too if this is a primary type.
        ii. The name may also appear in rules such as Deep Copy Rules, Type Display Rules, Compound Property Rules, Extension Rules, GRM Rules, etc.
        iii. The name may also appear in attachments for LOVS, or Constants.
        iv. Review each XML element where you find the old name and carefully decide if the name should be replaced with the new type name.
    d. Save and close all source files.
    e. In the “Navigator” view, select your template project icon, right click and choose “Reload Data Model”.
    f. The customer project should now successfully load and the Console view should not report any errors. If any errors do exist, open the source file where the error occurred and investigate. Then Reload the Data Model again to verify for errors.
4) After the colliding type is renamed, resume with the upgrade steps.

13.2 Dataset type name collisions

Dataset collisions fall into a special category where a customer can choose to rename, replace, or merge the colliding custom dataset with the COTS dataset. To help you determine which option to pick, you will want to answer the following questions. To help you answer these questions, you should study both the COTS and custom dataset definitions. To study the COTS definition, find the dataset in the BMIDE Business Objects View and open it. To study the custom definition, open the source file that contains the definition in a Text Editor.

- Does my custom dataset have the same purpose as the COTS dataset?
    - For example, if your custom dataset is called “PDF” and the COTS dataset is called “PDF”, were both of these for managing “Portable Document Format” files? Or was one of the acronyms used to manage a different kind of file?
- Look at the tools used to view and edit each of the dataset definitions. Are they the same? Does the tool support the same type of file format? Can you use the COTS dataset’s tools for viewing and editing your dataset files instead of your custom tools?
- Look at the tool reference names of each dataset definition. Are they the same or different? Will it be okay to use the COTS dataset reference names for your instances of the datasets?

- Look at the tool actions for each dataset definition. Are they the same or different? Will it be okay to use the COTS tool actions for your datasets instead of your custom tools?

Here are the three options to choose from:

1. Rename the custom dataset
    a. If your analysis shows that the two dataset definitions are not for the same purpose, then you should rename your definition so that it does not collide with the COTS definition.
    b. Follow the steps outlined in section 13.1 to rename this type to a new type name.
2. Replace the custom dataset with the COTS dataset
    a. If your analysis shows that your custom dataset definition has the same purpose as the COTS dataset definition, then you should replace your custom definition with the COTS definition.
    b. This is the recommended approach if you can do this option. The reason is that it is best to have one set of Tools, references, and tool actions rather than two or more sets that do the same thing. With two or more sets, the user will be presented with a choice to select the tool for each dataset they open or edit.
    c. Follow these steps to fix the collision
        i. If your custom dataset is not spelled the same (case-sensitive) as the COTS dataset, your custom dataset must be renamed to match the COTS name.
            1. For example, if your custom dataset is spelled exactly the same (case-sensitive) as the COTS dataset, you do not need to rename it to match.

                Example: Custom dataset = “PDF” : COTS Dataset = “PDF”.
            2. However, if your custom dataset is spelled with a different case as the COT dataset, then you must rename it to align with the COTS name.

                Example 1: Custom dataset = “Pdf” : COTS Dataset = “PDF”.

                Example 2: Custom dataset = “pdf” : COTS Dataset = “PDF”.

                To rename your dataset to match the COTS name, follow the steps outlined in section 13.1.
        ii. Remove the colliding type definition from the custom template by following section 13.2.1
        iii. Fix the existing instances of your dataset in the database by following section 13.2.2.
        iv. In the BMIDE, use the Package Template Extensions Wizard to create a template feature by following section 6.2.
        v. Return to your upgrade steps in this document. During the upgrade of your database, TEM will ask for your custom template. Use the Browse button on the TEM panel to select the template feature created in the step above. As TEM applies your template to the database, the data model definitions and instances of the dataset will be fixed.
        vi. After the upgrade completes, you may want to validate that the instances of the dataset were corrected by following section 13.2.3.
3. Merge the custom dataset into the COTS dataset definition
    a. If your analysis shows that your custom dataset definition has the same purpose as the COTS dataset definition however you would like to keep some of your tools, references, or tools actions, then you should merge your custom definition with the COTS definition.
    b. Follow these steps to fix the collision

i. If your custom dataset is not spelled the same (case-sensitive) as the COTS dataset, your custom dataset must be renamed to match the COTS name.

1. For example, if your custom dataset is spelled exactly the same (case-sensitive) as the COTS dataset, you do not need to rename it to match.

   Example: Custom dataset = “PDF” : COTS Dataset = “PDF”.

2. However, if your custom dataset is spelled with a different case as the COT dataset, then you must rename it to align with the COTS name.

   Example 1: Custom dataset = “Pdf” : COTS Dataset = “PDF”.

   Example 2: Custom dataset = “pdf” : COTS Dataset = “PDF”.

   To rename your dataset to match the COTS name, follow the steps outlined in section 13.1.

ii. Remove the colliding type definition from the custom template by following section 13.2.1

iii. Launch the BMIDE.

1. Go to the Business Objects View, find the COTS dataset definition, open it, and review all the tools, references, and tool actions.
2. Review the backup copy of your custom dataset from the previous step. Determine which tools, references, or tool actions you want to keep and those you wish to replace with the COTS definitions.
3. Use the BMIDE client to add the custom tools, references, or tool actions to the COTS Dataset that you decided to keep.
   a. For adding custom tools that you decided to keep
      i. Open the COTS dataset in its editor
      ii. Add the edit tools and view tools to the respective tables
   b. For adding custom dataset tool actions that you decided to keep
      i. Open the COTS dataset in its editor
      ii. Go to the tab named “Tool Actions”
   c. For adding custom dataset references that you decided to keep
      i. Open the COTS dataset in its editor
      ii. Go to the tab named “References”
      iii. Add the dataset references that you decided to keep

      Note: If you are working in BMIDE of Teamcenter 8.3.0 or later, when you create the dataset reference you will be asked to name your reference with a prefix. Go ahead and provide the reference name with prefix. We will change it after saving the data model.
4. Go to File -> Save Data Model to save the changes to your template.

   NOTE: If you are working with BMIDE of Teamcenter 8.3.0 or later and you added custom dataset reference to the COTS dataset then you must do the following:

   a. Go to the Navigator view
   b. Manually search and replace the dataset references that you added with prefix to a name without prefix. You will do this in the XML files located in the folder *<project>/extensions*

i. For example if your prefix is “Rx7” and you added a dataset reference “*Rx7customPDFReference*”, then search for all occurrences of “*Rx7customPDFReference*” in all XML files under *“<project>/extensions*” folder and replace it with just “customPDFReference”.

ii. The reason we do this is to ensure that the dataset reference name in your template matches with the reference that is already deployed to your database (in an older release).

iv. For those tools and references that you have decided to replace with the COTS definition, you can fix the existing instances of your dataset in the database by following section 13.2.2.

v. In the BMIDE, use the Package Template Extensions Wizard to create a template feature by following section 6.2.

vi. Return to your upgrade steps in this document. During the upgrade of your database, TEM will ask for your custom template. Use the Browse button on the TEM panel to select the template feature created in the step above. As TEM applies your template to the database, the data model definitions and instances of the dataset will be fixed.

vii. After the upgrade completes, you may want to validate that the instances of the dataset were corrected by following section 13.2.3.

13.2.1 Remove the colliding Dataset type definition from the custom template

In the following steps, we will show you how to manually remove the custom dataset definition from the template source file. Then we will show you how to validate the custom template data model to ensure that you performed the edit correctly.

1. Launch Business Modeler IDE with your customer project.
2. Go to the “Navigator” view.
3. Locate the extension file under *<project>/extensions* folder containing the colliding dataset definition.
4. Open the extension file in a Text editor within Business Modeler IDE.
5. Make a backup copy of the colliding customer dataset definition into a temporary file. A sample dataset definition is shown below.

```
<TcDataset typeName="PDF" parentTypeName="Dataset" typeClassName="Dataset">
     <TcDSViewTool name="CustomPDFTool"/>
     <TcDSEditTool name="CustomPDFTool"/>

     <TcDatasetReference name="CustomPDF_Reference1">
        <TcDatasetReferenceInfo template="*.pdf" format="BINARY"/>
     </TcDatasetReference>
     <TcDatasetReference name="CustomPDF_Reference2">
        <TcDatasetReferenceInfo template="*.pdf" format="BINARY"/>
     </TcDatasetReference>

     <TcDatasetToolAction tool="CustomPDFTool" action="Open">
        <TcToolActionReference export="true" referenceName="CustomPDF_Reference"/>
        <TcToolParameter name="$CustomPDF_Reference"/>
     </TcDatasetToolAction>
     <TcDatasetToolAction tool="CustomPDFTool" action="OpenUsing">
        <TcToolActionReference export="true" referenceName="CustomPDF_Reference"/>
        <TcToolParameter name="$CustomPDF_Reference"/>
     </TcDatasetToolAction>
     <TcDatasetToolAction tool="CustomPDFTool" action="Print">
```

```
            <TcToolActionReference export="true" referenceName="CustomPDF_Reference"/>
            <TcToolParameter name="$CustomPDF_Reference"/>
        </TcDatasetToolAction>
    </TcDataset>
```

6. Manually remove the conflicting customer dataset definition from the project's extension file.
7. Save the XML file.
8. In the "Navigator" view, select your template project icon, right click and choose "Reload Data Model".
9. The customer project should now successfully load and the Console view should not report any errors.
10. Optional Step. If you think that the view and edit tools used in your removed dataset definition are also not required anymore, then you can use the Business Modeler IDE UI to remove the tools.
    a. For example, in the above XML sample, the same tool called "CustomPDFTool" is used for both viewing and editing the PDF dataset, but it is not used for any other Dataset.
    b. Go to the Extension View in the BMIDE, find the tool called "CustomPDFTool".
    c. Right Click and select "Delete" to delete the custom tool.
    d. Save the data model through File->Save Data Model.

13.2.2 Fix the dataset instances in the database

In this section, we will show you how to prepare an XML file that will be used to convert the existing dataset instances in the database. A command that executes the file will be added to your template's upgrade script so that all existing dataset instances will be converted to use the COTS definition during the upgrade.

To convert the existing instances, we need to know the old (existing) and new tool names, and the old (existing) and new reference names. This information can be read from the BMIDE templates.

1. Launch Business Modeler IDE with your customer project.
2. Create a file named *FixCollidingDatasetInfo.xml* in the *<project>/install directory*. This file will contain the data to be used during upgrade to fix all dataset instances that were previously referring to custom dataset's references and tools.
3. This XML file will be used as input to the change_datasets utility. For more information about this utility see the utilities guide.
4. The XML file format is shown below and works as follows.
    a. Enter the name of the dataset to change in the "DatasetName" field.
    b. Enter the custom Tool name in the "OldToolName" field.
    c. Enter the COTS Tool name in the "NewToolName" field.
    d. Enter the Custom reference name in the "OldReferenceName1" field.
    e. Enter the COTS reference name in the "NewReferenceName1" field.
    f. Repeat the <TcDatasetReference> tag for multiple reference names that need to be changed.

```
<TcDatasets>
    <TcDataset name="DatasetName">
```

```
        <TcTool fromName="OldToolName" toName="NewToolName"/>
        <TcDatasetReferences>
            <TcDatasetReference fromName="OldReferenceName1" toName="NewReferenceName1"/>
            <TcDatasetReference fromName="OldReferenceName2" toName="NewReferenceName2"/>
        </TcDatasetReferences>
    </TcDataset>
</TcDatasets>
```

5. To construct the contents of the *FixCollidingDatasetInfo.xml,* you should refer to the custom and COTS dataset definitions. Below are examples of the custom and COTS dataset definition. By reading the XML definition from the BMIDE templates, you will be able to determine how to fill out the fields. Below the import sections of the BMIDE dataset definitions are marked in bold lettering.

    a. The following is an example of a custom PDF dataset definition taken from the backup information created in a previous step.

```
<TcDataset typeName="PDF" parentTypeName="Dataset" typeClassName="Dataset">
      <TcDSViewTool name="CustomPDFTool"/>
      <TcDSEditTool name="CustomPDFTool"/>

      <TcDatasetReference name="CustomPDF_Reference1">
         <TcDatasetReferenceInfo template="*.pdf" format="BINARY"/>
      </TcDatasetReference>
      <TcDatasetReference name="CustomPDF_Reference2">
         <TcDatasetReferenceInfo template="*.pdf" format="BINARY"/>
      </TcDatasetReference>

      …
</TcDataset>
```

    b. The following is an example of the COTS PDF dataset definition found in the foundation_template.xml file.

```
<TcDataset parentTypeName="Dataset" typeClassName="Dataset" typeName="PDF">
    <TcDSViewTool name="PDF_Tool"/>
    <TcDSEditTool name="PDF_Tool"/>

    <TcDatasetReference name="PDF_Reference">
      <TcDatasetReferenceInfo format="BINARY" template="*.pdf"/>
    </TcDatasetReference>

    …
</TcDataset>
```

6. Based on the custom and COTS dataset definitions, the resulting *FixCollidingDatasetInfo.xml* file should appears as follows.

    a. This file will convert all instances of “PDF” that use the “CustomPDFTool” tool to use the “PDF_Tool” tool.

    b. This file will convert all instances of “PDF” that use the “CustomPDF_Reference1” reference to use the “PDF_Reference” reference.

    c. This file will convert all instances of “PDF” that use the “CustomPDF_Reference2” reference to use the “PDF_Reference” reference.

```
<TcDatasets>
    <TcDataset name="PDF">
```

```
        <TcTool fromName="CustomPDFTool" toName="PDF_Tool"/>
        <TcDatasetReferences>
            <TcDatasetReference fromName="CustomPDF_Reference1" toName="PDF_Reference"/>
            <TcDatasetReference fromName="CustomPDF_Reference2" toName="PDF_Reference"/>
        </TcDatasetReferences>
    </TcDataset>
</TcDatasets>
```

7. Next you must add a command inside your templates' upgrade script. This command will be executed during an upgrade when your template is applied to the database through TEM.
    a. Open the file <*project*>*/install/upgrade_*<*template*>*_v2007.default*.
    b. Add the command "change_datasets" in the section <**post-non-schema-change**> to update all the dataset instances in the database.
    c. Below is an example of the section comments and section tags. The text in bold is the text that you should add to this file.

```
#................................................................................
# SECTION: Post Non-Schema Changes
#................................................................................
#    Use this section to add any commands that should migrate data after
#    the BMIDE tools have changed any existing Business Objects, Business Rules, LOVs,
#    Change objects, Validation Data, Tools, etc.
#................................................................................
<post-non-schema-change>
    change_datasets -u=infodba -p=${TC_USER_PASSWD} -g=dba
    -ds_info=${TC_INSTALL_DIR}/${template_name}/FixCollidingDatasetInfo.xml
</post-non-schema-change>
```

13.2.3 Validating the dataset instances in database

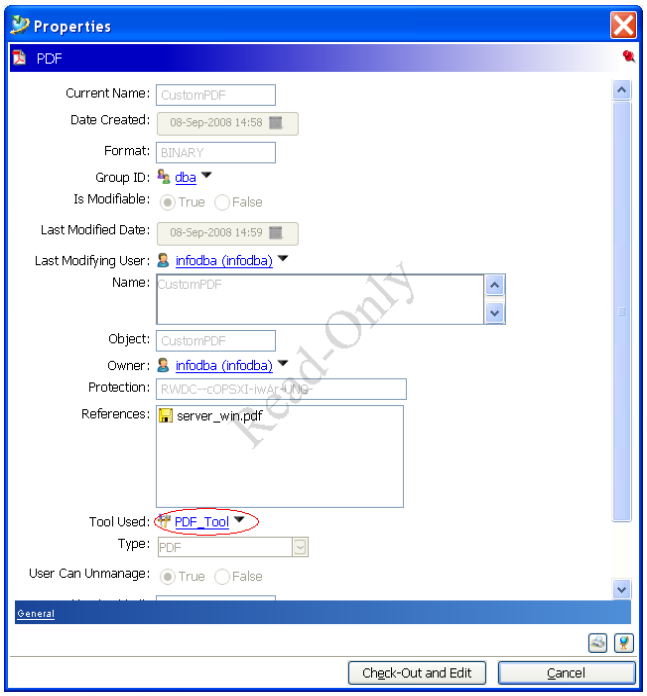
After completing the upgrade, you may want to verify that the existing instances of the dataset were converted to use the COTS dataset tool and reference names. Here is how to verify.

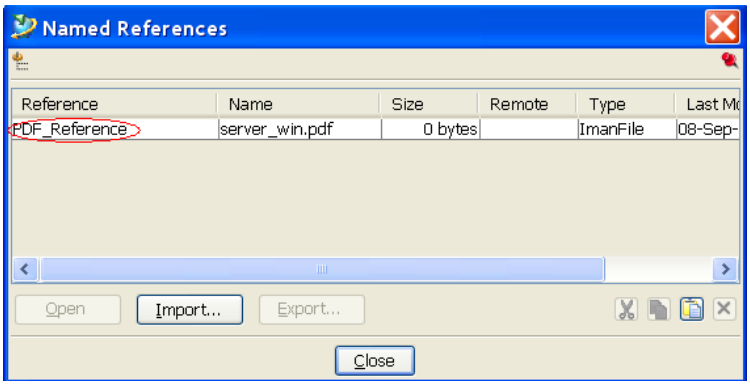
1. First, you should view the log file "*%temp%/ChangeDatasetInfo_*<*timestamp*>*.log*" to ensure that there are no errors.
    a. If there are errors, then look at the log files named "*TC_ROOT/logs/change_dataset*.syslog*" to understand the reason for failure.
    b. One of the common reasons for the utility to fail is if the Teamcenter DBA user (say infodba) does not have sufficient privileges to update dataset instances created by other users. The error message for this is shown below.

```
Failed to lock instances 21608. Error is = 515001
```

    c. If this is the case, go to the Access Manager application and add the follow rule. Then move the rule up in the tree just below the "Has Bypass" rule.
        Condition = "Is SA"
        Value = "true"
        ACL Name = "Bypass"
    d. After executing the upgrade, remove this rule from the Access Manager.

2. If there are no errors reported in the log file, launch the Rich Client Application and log into the upgraded database.
3. Query for the existing dataset instances.
4. Select a few dataset instances and verify that the tool and the named references are set correctly to the new one.
    a. To view the tool, select the instance, Right-Click -> Properties. Look at the “Tool Used” field. See the screenshot below for an example of the Properties panel showing the “Tool Used” field.
    b. To view the reference name, select the instance, Right-Click -> Named References. Look at the Reference column. See the screenshot below for an example of the Named Reference panel showing the “Reference” column.





14 Planning Upgrades

The follow sections will cover frequently asked questions about upgrades. Additionally this section will present various upgrade scenarios that you should consider when planning an upgrade for your databases.

14.1 When can I upgrade my Production Environment?

Scenario: A customer has Engineering installed into a Production Environment with the following Add-On solutions: Foundation, Wire Harness, Translation Services, and Pro/Engineering Integration. This customer wants to upgrade to Teamcenter unified architecture, however, the Pro/Engineering Integration template that supports Teamcenter has not been released yet.

Question: Can this customer upgrade their database?

Answer: No.

Reason: There is a common misperception that you can upgrade your Teamcenter database in phases. For example, upgrade Foundation, Wire Harness, and Translation Services now because they are available; then upgrade Pro/Engineering Integration template later when it is available. The issue with this is that there will be a period of time where the database and server will be operating with incompatible data models and libraries. Three of the four templates will have the latest data model and C++ libraries working with the base Teamcenter libraries. However the Pro/Engineering Integration template's data model, scripts, utilities and libraries are from a previous product version. The Pro/E template data model will still be present in the database after the upgrade, and this data model was intended for the previous product version. It has never been tested with the latest version. Additionally there may be business rules, like extension rules or user exits that execute custom written C++ code. This current C++ code was compiled and linked against former versions of the product. Operating mismatched code like this can cause the product to crash for linking issues or may cause irreparable data corruption.

Therefore this customer must wait until all required templates are released before they can upgrade. If the customer operates their database and server in this state and issues occur, GTAC will ask them to rollback their database to the former state and wait for all templates to be released. For a list of templates that are not released with the Teamcenter kit see section 14.2.

14.2 List of templates released asynchronously

The following is a list of templates that are released asynchronously to the Teamcenter unified architecture kit. If your database has any of these, you should check on their availability when planning an upgrade.

- Ideas Manager
- Teamcenter Integration for I-deas - Database Extensions
-
- Solid Edge Embedded Client Overlay
- AutoCAD Foundation
- In Context Editor
- Factory CAD Integrated Resource Management
- Integration for CATIA
- Integration for Pro/Engineering
- Solid Works
- Teamcenter for Medical Devices - Document Services
- Teamcenter for Medical Devices

- Teamcenter Gateway For SAP
- Teamcenter Integration for Inventor
- Content Migration Manager
- BCT aClass
- BCT aClass Exits
- BCT aClass Templates
- BCT CheckIt
- BCT Compliance
- BCT Exits
- BCT Inspector
- BCT Mail2TcEng
- BCT Material
- BCT ME10Manager
- BCT TechDoc
- Content Migration Manager
- Retail Foundation
- RTT DeltaGen Integration
- Teamcenter Integration for SolidWorks
- Teamcenter Gateway for Oracle E-Business Suite
- Teamcenter Gateway For SAP
- Teamcenter for Environmental Compliance
- Teamcenter for Medical Devices
- Teamcenter for Medical Devices - Document Services

14.3 When can I upgrade my Test Environment?

Scenario: A customer has Engineering installed into a test environment with the following Add-On solutions: Foundation, Wire Harness, Translation Services, and Pro/Engineering Integration. This customer wants to upgrade to Teamcenter unified architecture 2007.1, however, the Pro/Engineering Integration template that supports 2007.1 has not been released yet. Same scenario as section 14.1 except that this is test database as opposed to a production database.

Question: Can this customer upgrade their database?

Answer: No.

Reason: The reasons are the same as 14.1. The purpose of upgrading a test system is to do a trial run and evaluate the software. Since it is known that there will be incompatibility issues, the product will not be supported in this state. Therefore this customer must wait until all required templates are released before they can upgrade the test database. If the customer operates their database and server in this state and issues occur, GTAC will ask them to rollback their database to the former state and wait for all templates to be released.

14.4 How do I test upgrade without all the required templates?

Question: If all of my required templates are not released, then how do I support my customer's request to stand up a new installation of Teamcenter unified architecture for evaluation with our product data and data model?

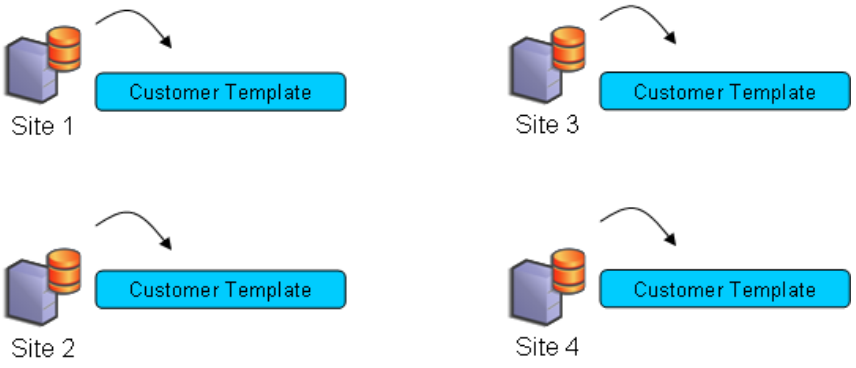
Answer: Since you cannot perform an upgrade, this is not possible. Instead you should perform a new installation of Teamcenter unified architecture and install the various features that are available and evaluate them. Later when all required templates are available, you can perform a sample upgrade.

14.5 How do I upgrade multiple database sites that all use the same set of data model extensions?

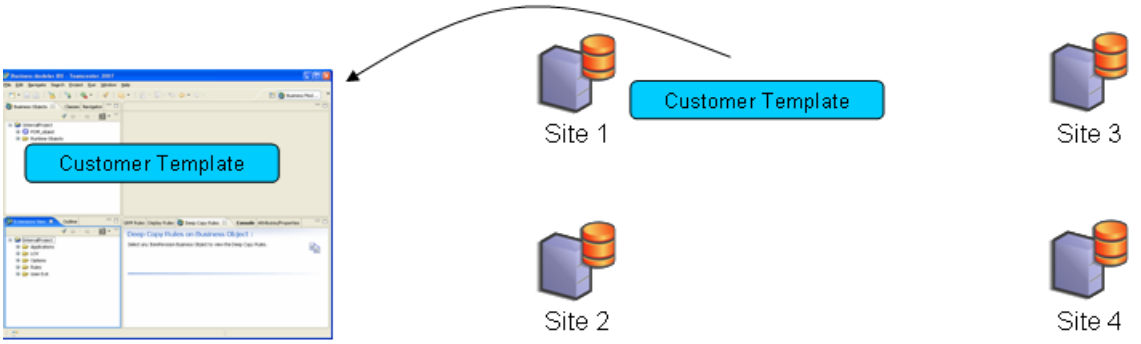
Question: A customer has Teamcenter Engineering installed in multiple data base sites. Each data base site has the exact same data model extensions and behaviors. What process should I follow for upgrading?

Answer: Since each site has the exact same data model extensions only one custom BMIDE template will need to be created. There are two options to accomplish this.

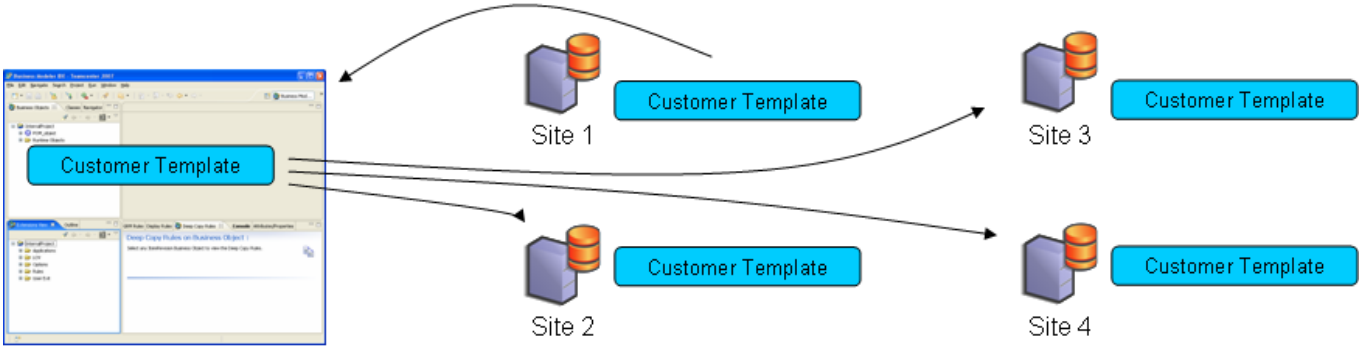
Option 1: You could do the upgrade at each site simultaneously. Run the TEM upgrade to upgrade each site. This will upgrade the database, extract out the custom template. Import the template into the BMIDE at one site only and maintain it there. Distribute updates to the BMIDE template to the other sites from the main site.



Option 2: Or you could upgrade one site first. Run the TEM to upgrade this site. Execute the post upgrade steps for extracting the custom template. Import the template into a BMIDE client.



Since you will be using this template to upgrade other Engineering sites, you will need to add template match tags. See section 11.2. After adding the template match tags, use the Package wizard to generate the BMIDE template package and distribute this package to the other sites for use with upgrade. Each site administrator should use TEM to include the custom template package you distributed during the upgrade. Use the Browse button on the Upgrade Database Features page (Figure 64) to locate the custom template feature file. After the TEM upgrade is complete, the TC_DATA model directory will have all required templates including your custom template. Therefore there is no need to execute the "bmide_postupgradetotc" command since the custom template is already installed.

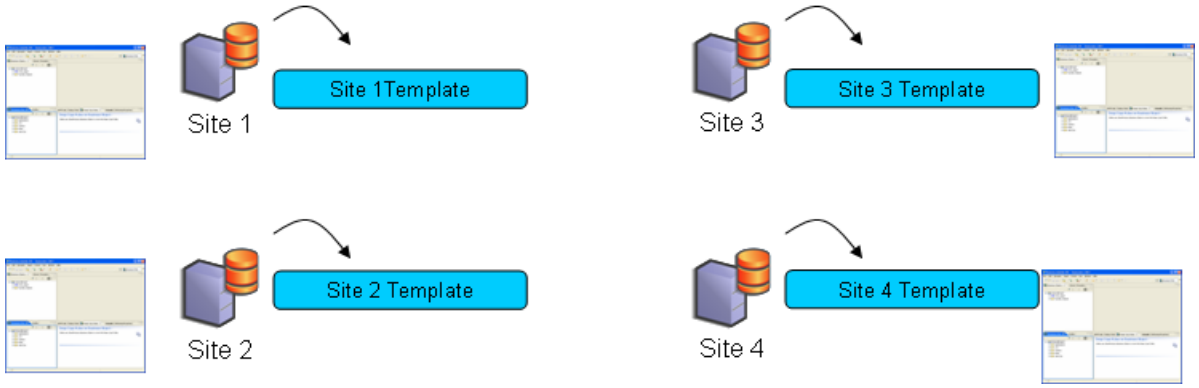


Commentary: Many customers with multiple sites believe that all sites have the same data model extensions. Often times we have found that in fact each database could have some minor differences between each site. This scenario described above will only work if you are certain that the sites are all exactly the same. If they are not, then you should consider section 14.6 or section 14.7.

14.6 How do I upgrade multiple database sites that all use a different set of data model extensions?

Question: A customer has Teamcenter Engineering installed in multiple data base sites. Each data base site has a uniquely different set of data model extensions and behaviors. What process should I follow for upgrading?

Answer: Since each site has uniquely different data model extensions each site will need to create its own custom BMIDE template and maintain it independent of the other templates. Each site should be assigned a unique template name so that there is no confusion. To accomplish this, let each site administrator run through the upgrade steps. This will upgrade the database, extract out the custom template specific for the site. Then each site administrator should import their site template into their own BMIDE client at the site. Since each site has its own template, it should be maintained at the site and never distributed to the other sites.



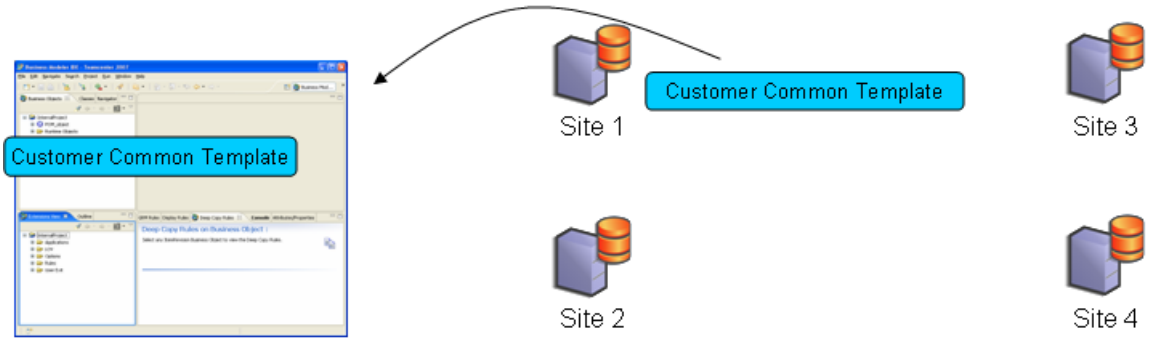
Commentary: It is not likely that all sites within a company have completely independent data models. However this scenario was given to demonstrate that you could separate management of the data model this way. The down side of this configuration is that if there are any common definitions that all sites share, these common definitions would have to be maintained at each site and someone would need to ensure they are synchronous. Since this scenario is not very likely to happen, it might be more likely that each site shares some common definitions and also has some additional extensions beyond the common ones. If this is the case, see scenario 14.7.

14.7 How do I upgrade multiple database sites that all use a similar set of data model extensions?

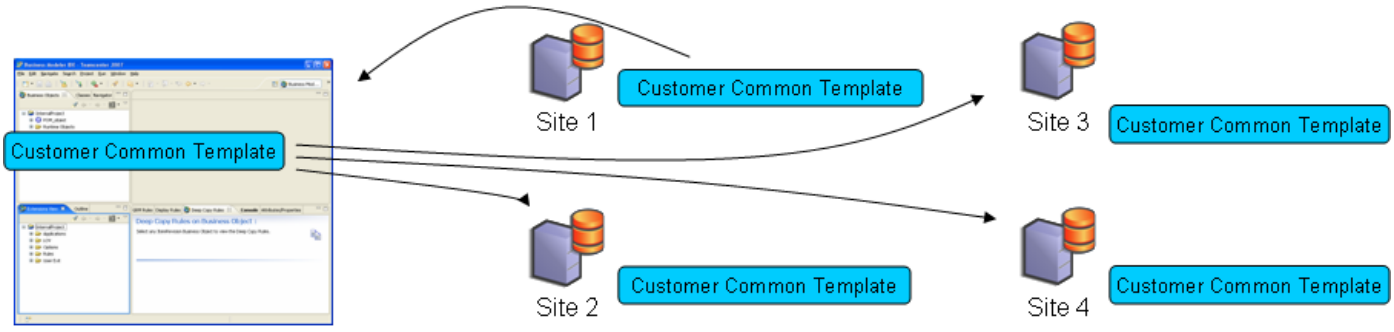
Question: A customer has Teamcenter Engineering installed in multiple data base sites. Each data base site shares a similar set of common data model extensions and behaviors. Each site may also have some additional extensions beyond the common site that are specific to the site's requirements. What process should I follow for upgrading?

Answer: Since each site has a common set of data model extensions we will need to create a common template that contains these extensions. The common template will be edited in one place and distributed to all sites for consistency. Additionally we will create a site specific template at each site to capture the addition extensions created at the site above and beyond the common template.

To accomplish this, first upgrade one site only. Select a site that has the common set of definitions. Run the TEM upgrade. Execute the post upgrade steps for extracting the custom template. As a best practice, use the word “Common” in your template name so that it designates this template as being the common one shared among the sites. Import the extracted common template into a BMIDE client at the same site and maintain it there.

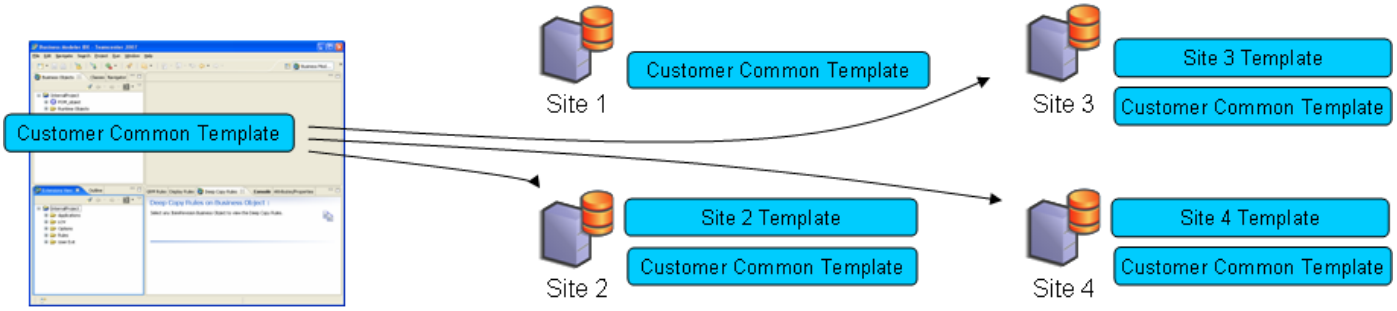


Since you will be using this template to upgrade other Engineering sites, you will need to add template match tags. See section 11.2. After adding the template match tags, use the Package wizard to generate the BMIDE template package and distribute this common template package to the other sites for use with upgrade.



During an upgrade each site administrator should use TEM to include the common custom template package you distributed. Use the Browse button on the Upgrade Database Features page (Figure 64) to locate the common

template feature file. After the TEM upgrade is complete, the TC_DATA/model directory will have all required COTS templates plus the common template. Next each site should extract out the site specific customizations to a site template using the “bmide_postupgradetotc” command. Ensure that each site is assigned a unique template name so that there is no confusion. Example: SiteATemplate, SiteBTemplate, SiteCTemplate, etc. Then each site administrator should import their site template into their own BMIDE client. Make sure that this site template within the BMIDE is dependent upon on the common template. See the Business Modeler IDE Guide for information on how to set template dependencies.

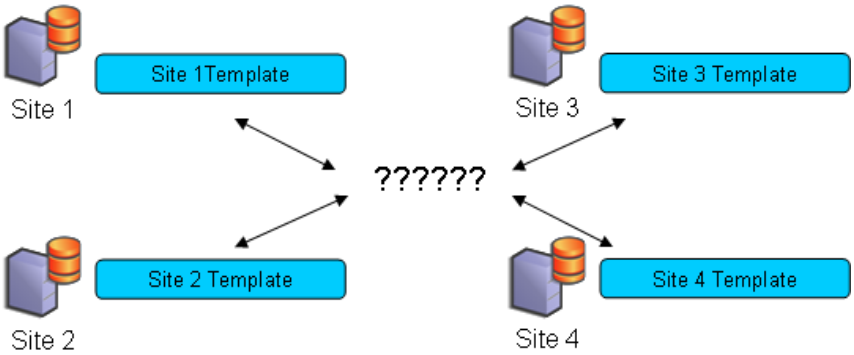


From this point forward the common data model definitions should be maintain and distributed to the other sites from a single site administrator. Each of the other sites should maintain their own templates and install them only into their own database site.

Commentary: For customers with multiple database sites, this is probably the most common scenario. Maintaining a common set of definitions at one site reduces work and maintenance costs and ensures consistency at all other sites.

14.8 How can I tell if my sites have the same or different data models?

Question: A customer has Teamcenter Engineering installed in multiple data base sites. Each data base site is supposed to have the same data model; however, the administrators are not certain. How can I tell if my sites have the same or different data models?

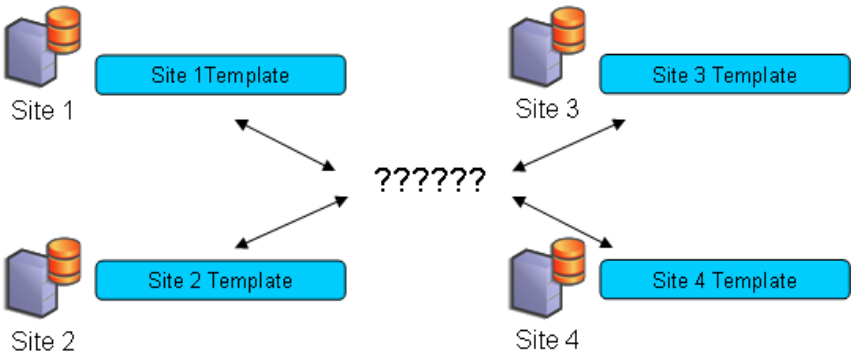


Answer: There is no way in Teamcenter Engineering to determine this. Some people have asked can the “database_verify” utility could be used. This utility only compares schema, types, tools, release status, and unit of measures between two databases. It does not compare the many other definitions that the BMIDE manages like Naming Rules, Deep Copy Rules, LOVs, Constants, etc. Therefore it will not provide a complete picture of everything that can be extracted into a BMIDE template.

Only Teamcenter unified architecture has the tools for comparing data models. Therefore to do this, you must perform a test upgrade on each site and compare the extracted templates. Once each site template has been extracted you can either compare these files manually or by using BMIDE command line tools.

14.9 How do I use the BMIDE command line tools to compare database models?

Question: A customer wants to perform a test upgrade on all Engineering database sites to determine if the databases models are the same or different. After I perform the test upgrade at each site to Teamcenter unified architecture, how do I use the BMIDE command line tools to help me with the comparison?



Answer: After the upgrade, use the BMIDE extractor command line tool to extract the entire data model from a given database site. Example output file -> site1model.xml

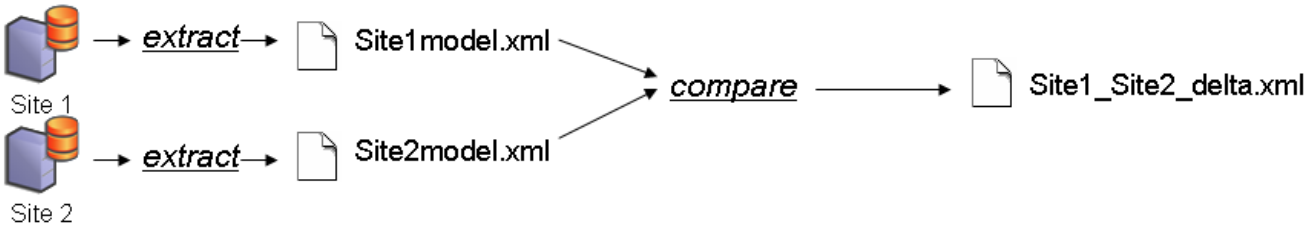
business_model_extractor –u=infodba –p=<infodba password> -g=dba –outfile=C:\site1model.xml –log=C:\site1db.log

Next upgrade another database and use the BMIDE extractor command line tool to extract the entire data model from this database. Example output file -> site2model.xml

business_model_extractor –u=infodba –p=<infodba password> -g=dba –outfile=C:\site2model.xml –log=C:\site1db.log

Next use the BMIDE comparator tool to compare the two model files.

bmide_comparator -all –old= C:\site1model.xml –new=C:\site2model.xml –delta= C:\site1_site2_delta.xml –log= C:\site1_site2_delta.log



The output file from the compare tool will list in XML format the additions, deletions, and changes between the two files. If the delta file is empty, then there are no differences between the two sites. If the delta file is not empty, each tag must be interpreted as follows.

An <Add> tag indicates that all the elements within the <Add></Add> tags exist in the “new” model but do not exist in the “old” model.

An <Delete> tag indicates that all the elements within the <Delete></Delete> tags do not exist in the “new” model but do exist in the “old” model.

An <Change> tag indicates that all the elements within the <Change></Change> tags exist in both models, however there are differences. To determine the changes, you may either inspect the contents of the changed element, or view the differences for this element in the old model file versus the new model file.

14.10 Testing Upgrades

Question: Is it safe to upgrade my production database first without first testing the upgrade on a test system?

Answer: It is strongly suggested that you perform a test upgrade first before doing the production system upgrade.

Reason: Product upgrades require the system to be taken down and users forced to stay off until the upgrade completes. Typically this is done over a weekend to reduce the impact to the business. If any issues occur that require GTAC assistance or Product Development attention, the turnaround time may not be quick enough to flush out the issue, repair it, resume the upgrade, and get the system up and running again.

As a best practice you should always test the upgrade of your production data base in a test system. If the data base upgrade passes, then you are clear to perform the production upgrade. Otherwise you must flush out any issues with GTAC and allow ample time to gather up the fixes and retest the upgrade again.

14.11 Can I use the template created from my test environment upgrade for a production upgrade?

In this question the person is asking, can I take a test environment where I loaded the same data model from my production environment, and then upgrade my test environment to extract out the custom template. Then when it is time to do the real upgrade on the production environment, can I use that template that I extracted from the test environment, to upgrade my product environment?

This will work successfully if the data model, rules, etc are exactly the same in both the test environment and the production environment. Any minor difference will result in differences in the extracted template that is used for the upgrade. It is very hard to be sure that both environments are the same. The databases must be at the same patch levels, and have the same configuration settings for preferences, etc. Note that some preferences cause changes in the data model with respect to runtime properties and relation properties. You must also ensure that all data model scripts, utilities, programs were loaded into the test database.

The best way to achieve this is to dump the production database and load it into a test database. Then do the upgrade to extract out the custom template. Then use that template to upgrade the actual production database. Note this will work as long as you ensure that no one makes any data model or behavior changes to the production database between the time that you dumped the database and when it is finally upgraded.

15 BMIDE Install Script and Upgrade Scripts

This section covers information about the commands that can be added to the install and upgrade scripts provided with the BMIDE template.

15.1 What Goes Into the Installation and Upgrade Scripts?

Each template project provides one install script and four upgrade scripts. These scripts are provided so that your template can execute additional commands during the install or upgrade of your template. These commands can be used to create additional data for your system. Since the BMIDE XML is managing the data model and behavior definitions, these extra commands should be used to add non-BMIDE objects. Non-BMIDE refers to any definitions or objects that are not managed by the BMIDE client. The list below provides examples of objects that are typically loaded through commands in these scripts:

- Process templates
- Export transfer modes
- Import transfer modes
- Preferences
- Filter rules
- Saved queries
- Property sets
- Closure Rules
- Workflow actions
- Workflow action handlers

Some of the definitions within your BMIDE template will refer to non-BMIDE objects in the database; therefore you will want to ensure that both are loaded whereever your template is loaded. For example, transfer modes are used to configure “AppInterface” business objects. Since the BMIDE takes care of loading the AppInterface definitions, then you will need commands in the install and upgrade scripts to ensure that the transfer model definitions are loaded before the AppInteface definitions are loaded.

Likewise, process templates are used to configure the Change definitions. If you have Change objects that use process templates then you need commands in the install and upgrade scripts to ensure that the process templates are loaded before the change management objects are loaded from the BMIDE XML.

To accommodate this type of synchronization between loading BMIDE and non-BMIDE objects, both the install and upgrade scripts are partitioned into sections. As the install or upgrade progresses, TEM will load parts of the data model from the BMIDE template and then give the install or upgrade script a chance to load its own data.

15.2 Install Script

This script is executed by TEM when your template feature is installed through TEM. See section 6.3. Every template must have an install script. This install script has very specific content and ordering of commands to get your BMIDE extensions installed into the database. You can only add your commands to the clearly marked areas. Do not change the order or remove any of the existing commands provided by Teamcenter.

15.2.1 Sample Install Script

```
---------------------------------------------------------------------------------------
#!/bin/sh
# install_customer.default:
#
# This software and related documentation are proprietary to Siemens PLM Software.
# COPYRIGHT 2007 Siemens PLM Software.  ALL RIGHTS RESERVED
#
# This script installs the Data Model for Customer solution.
# ____________________________________________________________________________
#
# ____________________________________________________________________________
# SECTION: Database Preparation
# ____________________________________________________________________________
#     THIS SECTION IS RESERVED for the utilities that will prepare
#     the database for an installation.
#     Do Not add any utilities commands in this section.
# ____________________________________________________________________________
install -regen_schema_file infodba ${TC_USER_PASSWD} dba
install -lock_db infodba ${TC_USER_PASSWD} dba
clearlocks -assert_all_dead infodba ${TC_USER_PASSWD} dba




# ____________________________________________________________________________
# SECTION: Schema Additions
# ____________________________________________________________________________
#     THIS SECTION IS RESERVED for the BMIDE tools to add new Classes and Attributes.
#     Do Not add any utilities commands in this section.
# ____________________________________________________________________________
business_model_updater -u=infodba -p=${TC_USER_PASSWD} -g=dba -update=schema -mode=upgrade -
file=${TC_DATA_MODEL}/delta.xml
install -regen_schema_file infodba ${TC_USER_PASSWD} dba




# ..........................................................................
# SECTION: Utilities
```

```
# ..........................................................................
#    Use this section if your solution needs to install any workflow templates,
#    transfer modes, or any other data that must be installed before the BMIDE tools install
#    Business Objects (Items, Datasets, Forms, Item Element, App Interface, IntDataCapture),
#    Business Rules, LOVs, Change Objects, Validation Data, Tools.
# ..........................................................................
#
```

```
# ____________________________________________________________________________
# SECTION: Non-Schema Additions
# ____________________________________________________________________________
#    THIS SECTION IS RESERVED for the BMIDE tools to install the Business Objects,
#    Business Rules, LOVs, Options, etc. which are managed by BMIDE.
#    Do Not add any utilities commands in this section.
# ____________________________________________________________________________
business_model_updater -u=infodba -p=${TC_USER_PASSWD} -g=dba -update=non_schema -mode=upgrade -
file=${TC_DATA_MODEL}/delta.xml
```

```
# ..........................................................................
# SECTION: Utilities - Post BMIDE
# ..........................................................................
#    Use this section if your solution needs to install any solution objects
#    that must be installed after the BMIDE tools install
#    Business Objects (Items, Datasets, Forms, Item Element, App Interface, IntDataCapture),
#    Business Rules, LOVs, Change Objects, Validation Data, Tools.
# ..........................................................................
```

```
# ____________________________________________________________________________
# SECTION: Database Finalization
# ____________________________________________________________________________
#    THIS SECTION IS RESERVED for those commands that finish up the
#    install.
```

```
#    Do Not add any utilities commands in this section.
# _____________________________________________________________________________
install -regen_schema_file infodba ${TC_USER_PASSWD} dba
clearlocks -assert_all_dead infodba ${TC_USER_PASSWD} dba
install -gen_xmit_file infodba ${TC_USER_PASSWD} dba
install -unlock_db infodba ${TC_USER_PASSWD} dba

-------------------------------------------------------------------------------------
```

15.2.2 File Description

The install script is separated into sections labeled with the *# SECTION:* comment and the name of the section. TEM needs to install the template's data model extensions in a specific order. Therefore, these sections must be kept in the order and the content provided by Teamcenter should not be removed or changed.

The solid lines and dotted lines that separate the sections are intentional and should not be changed. The solid lines indicate the sections that contain utilities that cannot be altered. These are required so that the BMIDE utilities can install the solution template data model. The dotted line sections are areas where the template owners can add additional utilities to support the installation of the template.

If you need to add any utilities, add them only in the appropriate (dotted) sections. Be sure to carefully read each section so that you can determine the correct placement of your utility with respect to the installation of the pieces of the data model.

Installation Script Sections		
SECTION	**Type**	**Description**
Database Preparation	Restricted	This section is used to prepare the database for a new template installation. This section is restricted and you should not add any utilities to this section.
Schema Additions	Restricted	This section is used to add the classes and attributes from your template to the database. This section is restricted and you should not add any utilities to this section.
Utilities	Editable	Use this section to add utilities that are required after the schema has been updated with the latest classes and attributes.
Non-Schema Additions	Restricted	This section is used to add the nonschema definitions from your template to the database—for instance, Types, LOVs, Business Rules, Extension Points, and so on. This section is restricted and you should not add any utilities to this section.
Utilities – Post BMIDE	Editable	Use this section to add utilities that are required after the nonschema data model objects have been updated—for example,

		Types, LOVs, Business Rules, Extension Points, and so on.
Database Finalization	Restricted	This section is used to finalize the database after a new template installation. This section is restricted and you should not add any utilities to this section.

Table 1: Installation script sections

15.3 Upgrade Scripts

Now that you have a BMIDE template, you must also support an upgrade for any database that has your template installed. TEM will enforce that any template previously installed will be required during the upgrade. This capability ensures that all data model templates are updated to the latest version simultaneously. During upgrade TEM will take your template and update the data model by comparing all the new extensions in all templates to what currently exists in the database. Then TEM begins to process the differences. Since we know that you may have to add other non-BMIDE type objects while your data model is being processed, a partitioned upgrade script is provided for you to add commands.

Every template project includes multiple upgrade scripts, one for each product version of upgrade supported to the current release. The file names indicate the version name that it supports. When an upgrade is performed, TEM uses the appropriate script to add additional objects to the database. By default, four upgrade scripts are provided because those are the four upgrades supported by Teamcenter 2007.1. If your template needs to support customers who may be upgrading from any of the former product versions, then you should keep all of the upgrade scripts. If your template does not support a given upgrade path, you can remove the file from the BMIDE.

Examples of upgrade scripts:

upgrade_customer_v913.default

upgrade_customer_v1000.default

upgrade_customer_v1001.default

upgrade_customer_tc2007.default

By default, each upgrade script is partitioned into sections and each section is initially empty. You can add your commands between the section tags. During upgrade, TEM upgrades your data model in partitions, working on schema first, and then non-schema next. Between each section, TEM calls the commands in your upgrade script section if any are present. Below is a sample file.

15.3.1 Sample upgrade Script

```
--------------------------------------------------------------------------------

# upgrade_customer_v1000.default:

#

# This software and related documentation are proprietary to Siemens PLM Software.

# COPYRIGHT 2007 Siemens PLM Software.  ALL RIGHTS RESERVED

#

#
```

```
#_____
#  Steps for upgrading Customer from Teamcenter <version> to latest Teamcenter release
#_____


#.....
# SECTION: Pre Schema Additions
#.....
#    Use this section to add any commands that should migrate data before
#    the BMIDE tools have added the new classes and attributes.
#.....
<pre-schema-add>

</pre-schema-add>



#.....
# SECTION: Post Schema Additions
#.....
#    Use this section to add any commands that should migrate data after
#    the BMIDE tools have added the new classes and attributes, but before
#    the BMIDE tools modify or delete any existing classes and attributes.
#.....
<post-schema-add>

</post-schema-add>



#.....
# SECTION: Post Schema Changes
#.....
#    Use this section to add any commands that should migrate data after the
#    the BMIDE tools have changed any existing classes and attributes,
```

```
#    but before the BMIDE tools deletes any existing classes and attributes
#    and before the BMIDE adds, changes, or deletes non-schema objects.
#    The creation of types for pre-tc2007 releases should be added in this section.
#.....................................................................
<post-schema-change>

</post-schema-change>

#.....................................................................
# SECTION: Pre Non-Schema Additions
#.....................................................................
#    Use this section to add any commands that should migrate data after
#    the BMIDE tools have added the main Business Objects and BMF Rules
#
#    Use this section if your solution needs to install any Workflow templates,
#    Transfer Modes, or any other data that must be installed before the BMIDE tools install
#    Business Objects (Items, Datasets, Forms, Item Element, App Interface, IntDataCapture),
#    Business Rules, LOVs, Change Objects, Validation Data, Tools.
#.....................................................................
<pre-non-schema-add>

</pre-non-schema-add>


#.....................................................................
# SECTION: Post Non-Schema Additions
#.....................................................................
#    Use this section to add any commands that should migrate data after the
#    the BMIDE tools have added the Business Objects, Business Rules, LOVs,
#    Change objects, Validation Data, Tools, etc.
#.....................................................................
<post-non-schema-add>

</post-non-schema-add>
```

```
#.....................................................................
# SECTION: Post Non-Schema Changes
#.....................................................................
#    Use this section to add any commands that should migrate data after the
#    the BMIDE tools have changed any existing Business Objects, Business Rules, LOVs,
#    Change objects, Validation Data, Tools, etc.
#.....................................................................
<post-non-schema-change>

</post-non-schema-change>


#.....................................................................
# SECTION: Post Non-Schema Deletions
#.....................................................................
#    Use this section to add any commands that should migrate data after the
#    the BMIDE tools have deleted any existing Business Objects, Business Rules, LOVs,
#    Change objects, Validation Data, Tools, etc.
#.....................................................................
<post-non-schema-delete>

</post-non-schema-delete>



#.....................................................................
# SECTION: Post Schema Deletions
#.....................................................................
#    Use this section to add any commands that should migrate data after the
#    the BMIDE tools have added/changed/deleted Business Objects, Business Rules, LOVs,
#    Change objects, Validation Data, Tools, etc.
#.....................................................................
<post-schema-delete>

</post-schema-delete>
```

15.3.2 File Description

The upgrade script is separated into sections labeled with the # *SECTION:* comment and the name of the section. TEM needs to upgrade the template's data model extensions in a specific order; therefore, these sections must be kept in the order that is provided. Additionally TEM is looking for the commented sections so they should not be altered.

Each section in the upgrade script is separated using dotted lines. As in the installation script, this means that each of these sections is editable by the template owners and you can add specific utilities in these sections. Because all sections are separated with dotted lines, it means that all sections can have utilities added.

Each section has two parts. There are the comments as shown below:

#.....

SECTION: Pre Schema Additions

#.....

Use this section to add any commands that should migrate data before the

the BMIDE tools have added the new classes and attributes.

#.....

And then there are the tags as shown below:

< pre-schema-add >

</ pre-schema-add >

Whenever you add utilities, they must go between the specific opening and closing tags. Any commands outside these tags are not executed.

Also, do not alter the tags in any way. The TEM upgrade runner looks specifically for these tags and does not recognize any new ones or tags by another name or spelling.

Upgrade Script Sections		
SECTION	**Type**	**Description**
SECTION: Pre Schema Additions	Editable	Use this section to add any commands to migrate data before the BMIDE tools have added the new classes and attributes.
SECTION: Post Schema Additions	Editable	Use this section to add any commands to migrate data after the BMIDE tools have added the new classes and attributes, but before the BMIDE tools modify or delete any existing classes and attributes.
SECTION: Post Schema Changes	Editable	Use this section to add any commands to migrate data after the BMIDE tools have changed any existing classes and attributes, but before the BMIDE tools

		delete any existing classes and attributes and before the BMIDE adds, changes, or deletes non-schema objects. The creation of types for pre-tc2007 releases should be added in this section.
SECTION: Pre Non-Schema Additions	Editable	Use this section to add any commands to migrate data after the BMIDE tools have added the main Business Objects and BMF Rules. Use this section if your solution needs to install any workflow templates, transfer modes, or any other data that must be installed before the BMIDE tools install business objects (items, datasets, forms, item elements, application interfaces (IntDataCapture), business rules, LOVs, change objects, validation data, tools.
SECTION: Post Non-Schema Additions	Editable	Use this section to add any commands to migrate data after the BMIDE tools have added the Business Objects, Business Rules, LOVs, Change objects, Validation Data, Tools, and so on.
SECTION: Post Non-Schema Changes	Editable	Use this section to add any commands to migrate data after the BMIDE tools have changed any existing Business Objects, Business Rules, LOVs, Change objects, Validation Data, Tools, and so on.
SECTION: Post Non-Schema Deletions	Editable	Use this section to add any commands to migrate data after the BMIDE tools have deleted any existing Business Objects, Business Rules, LOVs, Change objects, Validation Data, Tools, and so on.
SECTION: Post Schema Deletions	Editable	Use this section to add any commands to migrate data after the BMIDE tools have added/changed/deleted Business Objects, Business Rules, LOVs, Change objects, Validation Data, Tools, and so on.

Table 2: Upgrade script sections

15.4 How do I add other scripts or data files?

If your template requires additional scripts and or data files to be executed within the install or upgrade scripts, then you need to include these scripts or data files with the bundled template feature. For example, if your commands use PLMXML files or data files, then place these files in the *<Project>/install* directory adjacent to the install and upgrade scripts. Subsequently whenever you use the package wizard, this wizard will bundle up these additional files automatically and place them into the generated zip file.

15.5 Does your add-on solution need to support both Teamcenter Engineering and Teamcenter unified architecture?

For most customers, once you have upgraded your database to Teamcenter unified architecture, your data model and behaviors will be managed in the BMIDE from that point moving forward. However there are some cases where an add-solution provider has some customer databases in Teamcenter Engineering and others in Teamcenter unified architecture. As you recall the BMIDE only supports Teamcenter unified architecture. Therefore the XML definitions in your template cannot be used to install or update an Engineering database.

This is a typical situation for many of the templates which are delivered asynchronous to the release of the Teamcenter kits. See Figure 3 for examples of the asynchronous templates. This situation may also apply to you if you are providing an add-on solution for partners or customers to use.

If this scenario describes your situation, then you will have to maintain your data model and behavior definitions in both the Teamcenter Engineering format and the Teamcenter unified architecture format. This will require careful handling. Any definitions added to the Engineering scripts should also be added to the BMIDE template to keep the models consistent. However, keeping the data models consistent is not enough. Eventually your customers may want to upgrade from Engineering to Teamcenter unified architecture. We have found in most cases that our partners who supply data models typically have customers who are on various versions of their add-on solution. For example you may have version 1 of your solution that supports Engineering V10 SR1. Later you may have released version 2 of your solution that also supports Engineering V10 SR1. The version 2 will most likely have additional rules and data model elements.

The difficulty with this is that when you created your BMIDE template (by performing your own upgrade and extracting a BMIDE template), the post template extraction step created your template and a tcbaseline.xml file. The tcbaseline.xml file is a snapshot of your data model taken at the time of the upgrade. This may have been version 1 or version 2. The tcbaseline file is used to compute upgrades from Teamcenter Engineering to Teamcenter unified architecture and assumes that that the data model in the Engineering database is always consistent with the contents of the tcbaseline.xml file. If it is not, then the upgrade will not result in a consistent data model. If this is the case then some additional commands will need to be added to the upgrade scripts which will ensure that the upgraded data model results to same expected model.

The BMIDE template and upgrade scripts provide a framework for you to handle this scenario. It will require some consultation from Siemens PLM Software. Please contact GTAC for assistance if you have this need.

16 Testing

Testing of your BMIDE template is critical. During this transition period from Teamcenter Engineering to the new BMIDE format, it is possible that your BMIDE template does not reflect all the data model elements that are getting added to the database through the installation of your template. If there are any discrepancies, then these can cause issues for installation, upgrade, and the BMIDE client. To validate that your template does not cause any data model issues, we suggest the following tests.

There are two types of testing that are preformed with the templates: installation testing and upgrade testing. Both tests are designed to discover and eliminate any data model accuracy issues in your templates and your installation and upgrade scripts. These inaccuracies can occur for many reasons. The following is a list of some of the more common reasons:

- Installation script added/changed/deleted some definitions that the upgrade script does not, or vice versa.
- There are runtime properties added through ITK that are not reflected in the XML template.
- There are runtime types added through ITK that are not reflected in the XML template.
- An attribute's string storage length was changed in a former upgrade script but this change was not reflected in a new installation.
- An attribute's read or write status was changed by running a script that was not included in the original solution.
- Misspellings between installation and upgrade scripts.
- An attribute's nulls allowed flag was changed in a former release's installation script but the upgrade script did not reflect the same change.
- An attribute's initial value was changed in a former release's installation script but the upgrade script did not reflect the same.
- Classes become obsolete, so they were removed from the installation scripts. New installations of the solution do not have the class but older installations do have the class. This means there is a class in the database that the BMIDE has no knowledge of. The class must be either added to the XML or deleted from the database.
- A new type was added by the solution and later the same type was added by a COTS template. If a name collision occurs, the customer must rename the type and provide migration tools to move all instances to the new type.

The items listed in this list are some of the more common data model accuracy issues that can occur. The following sections will teach you how to detect these issues and others.

16.1 Install Testing

Installation testing will test the data model for accuracy issues on a newly installed database. In this test, you install Teamcenter and your template, extract the data model from the database, and compare to the XML definitions from the templates. If there are no differences, the test passes.

The following is a list of steps you should perform to test:

1. Use TEM to install the Corporate Server and your BMIDE template. If your template requires any dependent templates, then you should install them too. If you do use dependent templates from a third party, then coordinate with them to ensure that they have tested their template before you test yours. Otherwise, your analysis will reveal issues with their template.
2. Extract the full data model from the database into an XML file. To do this, open a Teamcenter command prompt and execute the following command:

business_model_extractor –u=infodba –p=<infodba password> -g=dba –outfile=C:\
data_accuracy\testdb_model.xml –log=C:\data_accuracy \extractor_testdb.log

3. Copy the TC_DATA/model/model.xml file to the C:\data_accuracy directory. This file represents the consolidation of your template definitions and all dependent template definitions.
4. Compare the extracted data model with the expected model in TC_DATA/model directory. Open a Teamcenter command prompt and execute the following command. It is important that you do not rearrange the order of the old and new directives below. If the elements are flipped, the resulting delta file will have all elements flipped and this may confuse your analysis.

   bmide_comparator -all

   –old= C:\data_accuracy\model.xml

   –new=C:\data_accuracy\testdb_model.xml

   –delta= C:\data_accuracy\delta_testdb_install.xml

   –log= C:\data_accuracy\ bmide_comparator_testdb.log
5. Analyze the C:\data_accuracy\delta_testdb_install.xml file. See section 16.3 for more information.
6. If any issues are found, fix the issues either in the XML file or by adding commands in the install/upgrade scripts included with the template. Then repeat all the steps above until you get a delta file that has zero differences.

16.2 Upgrade Testing

Upgrade testing will test for data model accuracy issues on an upgraded database. In this test, you install a former version of Teamcenter Engineering and your solution, and then do an upgrade that includes your template. Extract the data model from the database and compare to the XML definitions from the templates. If there are no differences, the test passes.

For upgrade testing, you should test all versions of the product that your template supports. Teamcenter 2007.1 supports the following upgrade paths:

- Teamcenter Engineering v9.1.3 to Teamcenter 2007.1
- Teamcenter Engineering v10 to Teamcenter 2007.1
- Teamcenter Engineering v10 SR1 to Teamcenter 2007.1

Your template may only support one, two, or all three paths. You should test all supported paths because issues can occur from any and all of these versions.

The following is a list of steps you should perform to test:

1. Install Teamcenter Engineering version X (from the list above). Include your add-on solution and any dependent add-on solutions.
2. Upgrade the Teamcenter Engineering database to Teamcenter 2007.1 using TEM. Be sure to include your custom template and any dependent templates. If you do use dependent templates from a third party, then coordinate with them to ensure that they have tested their template before you test yours. Otherwise, your analysis will reveal issues with their template.
3. Extract the full data model from the database into an XML file. To do this open a Teamcenter command prompt and execute the following command:

   business_model_extractor –u=infodba –p=<infodba password> -g=dba –outfile=C:\
   data_accuracy\testdb_model.xml –log=C:\data_accuracy \extractor_testdb.log
4. Copy the TC_DATA/model/model.xml file to the C:\data_accuracy directory. This file represents the consolidation of your template definitions and all dependent template definitions.

5. Compare the extracted data model with the expected model in TC_DATA/model directory. To do this, open a Teamcenter command prompt and execute the following command. It is important that you do not rearrange the order of the old and new directives below. If the elements are flipped, the resulting delta file will have all elements flipped and this may confuse your analysis.

   bmide_comparator -all

   –old= C:\data_accuracy\model.xml

   –new=C:\data_accuracy\testdb_model.xml

   –delta= C:\data_accuracy\delta_testdb_install.xml

   –log= C:\data_accuracy\bmide_comparator_testdb.log

6. Analyze the C:\data_accuracy\delta_testdb_install.xml file. See section 16.3 for more information.
7. If any issues are found, fix the issues either in the XML file or by adding commands in the install/upgrade scripts included with the template. Then repeat all the steps above until you get a delta file that has zero differences.

16.3 Analyzing Issues

The “bmide_comparator” tool produces a delta file that lists the differences between the old model and the new model. For testing, always set up the old model to be the expected XML definitions from the templates and the new model to be the extracted model from the database.

If the delta file has zero elements in it, then this means there are no differences between the expected and actual models. Therefore the test passes. If however, there are differences, then these differences should be analyzed. The XML files are very straightforward and easy to read.

The delta file is divided into three main sections:

<Add>

<Change>

<Delete>

16.3.1 <Add> Elements

Any element that appears in the <Add> section is an element that exists in the “new” model but not in the “old” model. It means that the <Add> elements are in the extracted database model but not in the expected template XML model. Therefore, there are utilities/commands run as a part of installation/upgrade that are adding elements that are not reflected in the XML template. Either remove the commands or add the elements to the template.

For example, suppose the delta file has the following <Add> element:

<Add>

<TcStandardType **typeName**="**TraceLink**" parentTypeName="ManagedRelation" typeClassName="TraceLink"/>

</Add>

This means that a type called “TraceLink” was added to the database but is not listed in the template XML.

16.3.2 <Delete> Elements

Any element that appears in the <Delete> section is an element that does not exist in the “new” model but does exist in the “old” model. It means that the <Delete> elements are not in the extracted database model but are in the expected template XML model. Therefore, there are utilities/commands that are missing in the installation/upgrade scripts that should be adding the elements. Either add the commands or remove the definitions from the template.

For example, suppose the delta file has the following <Delete> elements:

```
<Delete>
   <TcAttributeAttach className="CAEAnalysisRevMasterStore">
     <TcAttribute attributeName="solution_step" attributeType="POM_string" maxStringLength="32"
        isArray="false" followOnExport="false" isNullsAllowed="true" isUnique="false"
        isPublicRead="false" isPublicWrite="false" isCandidateKey="false" arrayLength="0"
        isTransient="false" exportAsString="false" noBackpointer="false" initialValue="NULL"/>
   </TcAttributeAttach>
   <TcGRMRule primaryTypeName="CAEAnalysisRevision" secondaryTypeName="CAEResultRevision"
      relationTypeName="TC_CAE_Results" primaryCardinality="1" secondaryCardinality="1"
      secured="false" attachability="WriteAccessReq" changeability="Changeable"
      detachability="WriteAccessReq"/>
</Delete>
```

There are two elements in the delete section above. The first element “CAEAnalysisRevMasterStore” is a class and has an attribute called “solution_step” that is missing on the class. Either add the attribute in the installation/upgrade script or remove the definitions from the XML template.

The second element is a GRM rule. The GRM rule has been removed from the database. Therefore, either add a command to add the GRM rule, or remove the XML definition of the GRM rule from the template.

16.3.3 <Change> Elements

Any element that appears in the <Change> section is an element that exists in both the “new” model and in the “old” model. However, one of the features of the definition has been changed. In this case, it means that the <Change> elements are in the extracted database model in one state and the expected template model has it in another state. Therefore, either a command needs to be added in the install/upgrade script to reconcile the difference, or the XML template definitions need to be corrected.

For example, suppose the delta file has the following <Change> element:

```
<Change>
   <TcAttributeAttach className="Architecture">
     <TcAttribute attributeName="has_basedon_preexist_elemnt" attributeType="POM_logical"
        maxStringLength="0" isArray="false" followOnExport="false" isNullsAllowed="false"
        isUnique="false" isPublicRead="false" isPublicWrite="false" isCandidateKey="false"
        arrayLength="0" isTransient="false" exportAsString="false" noBackpointer="false"/>
   </TcAttributeAttach>
</Change>
```

When a <Change> element is written, the entire element is rewritten to file rather than just the changed feature of the element. Therefore, to understand what has changed, you need to compare the changed element in the delta file to the same element definition in the expected XML file. The following shows the same definition in the XML template file:

<TcAttribute arrayLength="0" attributeName="**has_basedon_preexist_elemnt**" attributeType="POM_logical" exportAsString="false" followOnExport="false" initialValue="false" isArray="false" isCandidateKey="false" **isNullsAllowed="true"** isPublicRead="true" isPublicWrite="false" isTransient="false" isUnique="false" maxStringLength="0" noBackpointer="false"/>

Notice that the isNullsAllowed attribute has been changed. Either a command should be added to reconcile the difference or the XML template should be corrected to match the database.

Note: Ignore any <Change> elements that include <TcTool> elements. Tools are stored in the BMIDE templates with no release date in the tool definition. However, when they are installed into the database, they are automatically assigned a date. The comparison will reveal that there is a date change between the original definition (no date) and the extracted definition, therefore the changed Tool definitions are written to the delta file and can be ignored.

17 Appendix

17.1 Definitions and Terms

Business Analyst

One of the primary targeted users of the Business Modeler IDE. A business analyst uses the IDE to configure the system to meet company business objectives.

BMIDE

See Business Modeler IDE

Business Modeler IDE

A business analyst tool for extending Teamcenter using published extension points in the system.

Eclipse

Eclipse is an open-source platform for building integrated development environments. The BMIDE is built on top of this platform. See www.ecipse.org.

SCM

A Source Control Management system manages source files. Examples: Clear Case, CVS, Subversion, and Perforce.

17.2 Orphaned Type Issues

(This appendix is a replica of text that appears in the Teamcenter Release Bulletin beginning with the Tc2007.1.4 release).

During upgrade testing of some older Engineering databases, product development found the existence of some classes and type definitions that were no longer managed by Teamcenter. Analysis revealed that these classes and types were formerly added by Teamcenter Engineering in releases before Engineering V9.1.3, but have since been obsolete. These classes and type definitions were removed from the install scripts so that they were no longer added during a fresh install; however no commands were provided in the upgrade scripts to remove the same definitions during an upgrade. As a result databases that originally existed prior to Engineering V9.1.3 may have some of these classes and types still lingering in the database, while new databases will not have them. This has lead to frustration for customers who try to align all the data models at all sites.

Product development has resolved this issue for Tc2007.1 MP4. The resolution involved three methods for cleaning this up so that all data models are consistent regardless of the upgrade or install path taken to get to the release.

- Deleted 4 specific types and 1 class
- Reinstate 69 types to the foundation template
- Try to delete 16 specific types if the database does not have any instances. If the database does have instances of these types, then leave the type definitions and instances and allow the upgrade to convert this type to a custom type. These types will result into the customer template during upgrade.

There are two action items that you may have to take with respect to your BMIDE templates. Please continue to read and follow the actions if the scenarios below apply to you.

Action Item 1:

If you upgraded your database from Teamcenter Engineering to any release of Teamcenter unified architecture, you may notice the presence of some type definitions in your custom BMIDE template that you did not know that you owned.

Below is a list of types that Teamcenter development no longer needs and considers them to be obsolete. Since these were obsolete many releases ago and these type definitions were never removed from some databases, we cannot be sure if these types are still in use by customers. Therefore we added an upgrade utility which will try to delete these type definitions if there are no instances. If there are no instances, we assume that the customer is not using these types, so the type definition will be permanently deleted. If there are instances, we assume that the customer is using these types; so we do not delete them and the type definition remains in the database. After the upgrade from Teamcenter Engineering to Teamcenter unified architecture, the customer executes the “post_bmideupgradetotc” utility which extracts out any custom definitions in the customer’s BMIDE template. If the types were deleted during the upgrade, the types will not appear in the customer template. If the types were not deleted, the types will result as custom definitions into the customer template.

Your action item is to inspect your custom template and look for the existence of any of these types listed below. Whether you find them or not, there is nothing to do. Teamcenter and the BMIDE will work correctly. The purpose of this note is to raise awareness that if you find these types, the ownership of them has been transferred from Teamcenter to your custom BMIDE template. You may do with them as you need.

List of type definitions that may be deleted during an upgrade or may result in your custom BMIDE template:

```
 <TcStandardType typeName="IMAN_analysis" parentTypeName="ImanRelation"
   typeClassName="ImanRelation"/>
<TcStandardType typeName="IMAN_basis" parentTypeName="ImanRelation"
   typeClassName="ImanRelation"/>
<TcStandardType typeName="IMAN_derivation" parentTypeName="ImanRelation"
   typeClassName="ImanRelation"/>
<TcStandardType typeName="IMAN_evolution" parentTypeName="ImanRelation"
   typeClassName="ImanRelation"/>
<TcStandardType typeName="IMAN_markup" parentTypeName="ImanRelation"
   ypeClassName="ImanRelation"/>
<TcStandardType typeName="IMAN_master_model" parentTypeName="ImanRelation"
   typeClassName="ImanRelation"/>
<TcStandardType typeName="IMAN_model" parentTypeName="ImanRelation"
   typeClassName="ImanRelation"/>
<TcStandardType typeName="IMAN_nc_data" parentTypeName="ImanRelation"
   typeClassName="ImanRelation"/>
<TcStandardType typeName="IMAN_revision" parentTypeName="ImanRelation"
   typeClassName="ImanRelation"/>
<TcStandardType typeName="IMAN_structure" parentTypeName="ImanRelation"
   typeClassName="ImanRelation"/>
<TcStandardType typeName="IMAN_structure_model" parentTypeName="ImanRelation"
   typeClassName="ImanRelation"/>
<TcForm typeName="Item Revision Master" parentTypeName="Form"
   typeClassName="Form" formStorageClassName="ItemVersionMaster"/>
<TcForm typeName="Equipment" parentTypeName="Form"
   typeClassName="Form"/>
<TcForm typeName="Equipment Master" parentTypeName="Form"
   typeClassName="Form"/>
<TcForm typeName="EquipmentRevision" parentTypeName="Form"
```

```
    typeClassName="Form"/>
<TcForm typeName="EquipmentRevision Master" parentTypeName="Form"
    typeClassName="Form"/>
```

Action Item 2:

If you upgraded your database from Engineering to one of the following releases (Tc2007.1.0, Tc2007.1.1, Tc2007.1.2, or Tc2007.1.3), please read this section.

Your custom BMIDE template will most likely have some or all of the 69 reinstated classes and types and 4 deleted types and classes. This is because the upgrade process required you to extract out a custom template using the "post_bmideupgradetotc" utility. Since these classes and types existed in your database, yet none of these had any XML representation in the foundation template, the utility pulled them into your template because it thought that you owned them. There is no harm with this. The BMIDE and Teamcenter will work correctly. If you plan to continue using one of these releases of Teamcenter (Tc2007.1.0, Tc2007.1.1, Tc2007.1.2, or Tc2007.1.3) please do not try to remove these XML definitions from your custom template. You will fix these once you move to a later release as listed below.

When you upgrade this database to the next version of Teamcenter Tc2007.1.4 or beyond, these reinstated classes and types (listed below) will have XML representation in the foundation template. Since both your custom template and the foundation template have the same definitions, these names will collide. Typically you will see these collisions occur during an upgrade, when installing a new feature into the server using TEM, or when using the BMIDE. To fix this, you will need to remove these definitions manually from the XML source files in your custom BMIDE template. The directions for how to remove these are listed below.

Likewise some or all of the deleted classes and types (listed below) will need to have their XML definitions removed from your custom template. The directions for how to remove these are listed below.

Below is a list of all the XML definitions that you should remove from your XML source files. Please carefully remove them. Since there are so many, the easiest way to remove them is to perform the following steps:

Reinstated Types and Classes Cleanup

- Install the new version of the BMIDE client and the COTS templates that you require from the latest release kit. For example if you are upgrading Tc2007.1 MP4 then install the Tc2007.1 MP4 BMIDE client and templates.
- Import the older BMIDE custom template project into the new BMIDE client.
- During the import the custom data model will be validated against the new COTS template definitions. During this time any collisions will be shown in the Console View in the BMIDE client. A collision in the BMIDE for a class or type would look something like this below:
    - `Model Error: business_objects.xml Line:19 Column:57 A Class is already defined with the name "POM_index". Choose another name.`
    - `Model Error: business_objects.xml Line:45 Column:111 A Business Object is already defined with the name "CORCoreDList". Choose another name.`
- For each collision shown in the Console View, first check to see if the colliding definition is in the list of reinstated types below. Note that name collisions can occur for other reasons, such as defining a custom type that has the same name as a COTS type. These kinds of collisions that you created must be fixed in a different manner. If the collision listed in the Console View matches one of the definitions listed below, then manually remove the XML definition from your source file.
- Note that not all the XML definitions listed below will appear in your template. You may have none, or some combination, or all.
- When finished with removing all collisions, do a "Reload Data Model" in the BMIDE client. This command will reload the custom data model and validate it against the COTS template data model.

- If any issues occur they will be displayed in the Console view in the BMIDE. Fix the errors then use the "Reload Data Model" again. Repeat these steps until all errors are fixed. Contact GTAC if you need assistance.

Deleted Types and Classes Cleanup

- After all the errors related the reinstated types and classes are fixed, the custom XML entries for the deleted types should also be removed if any exist.
- See the list below for the list of the deleted types and class definitions.
- For each of the types listed in the list (UserInBox, BOMViewRevision, IMAN_drawing, ExternalObject), repeat steps below.
    - In the BMIDE, go to the Business Objects view
    - Find the business object with exact same name and case for each deleted type name
    - If business object does not exist , there is nothing to do, proceed to next business object
    - Right mouse click on this business object and click delete. Note that in the case of UserInBox, the class called UserInBox will also be deleted when the type is deleted.
    - Do, File-> Save Data Model
- Before returning to complete the upgrade, use the Package Wizard to generate a template package with the latest XML. Use this new package to perform the upgrade.

Reinstated XML Definitions

List of XML reinstated definitions that were reinstated in the Foundation template:

```
   <TcStandardType typeName="CORCoreDList" parentTypeName="DistributionList"
typeClassName="DistributionList"/>

   <TcStandardType typeName="CNDList" parentTypeName="DistributionList" typeClassName="DistributionList"/>

   <TcStandardType typeName="CNRevDList" parentTypeName="DistributionList"
typeClassName="DistributionList"/>

   <TcStandardType typeName="WACSAppDList" parentTypeName="DistributionList"
typeClassName="DistributionList"/>

   <TcStandardType typeName="WAIACoreDList" parentTypeName="DistributionList"
typeClassName="DistributionList"/>

   <TcStandardType typeName="ANDList" parentTypeName="DistributionList" typeClassName="DistributionList"/>

   <TcStandardType typeName="CTDList" parentTypeName="DistributionList" typeClassName="DistributionList"/>

   <TcStandardType typeName="CPRCoreDList" parentTypeName="DistributionList"
typeClassName="DistributionList"/>

   <TcStandardType typeName="WAIAAppDList" parentTypeName="DistributionList"
typeClassName="DistributionList"/>

   <TcStandardType typeName="WACSCoreDList" parentTypeName="DistributionList"
typeClassName="DistributionList"/>

   <TcStandardType typeName="CORAppDList" parentTypeName="DistributionList"
typeClassName="DistributionList"/>

   <TcStandardType typeName="CNAppDList" parentTypeName="DistributionList"
typeClassName="DistributionList"/>

   <TcStandardType typeName="CPRAppDList" parentTypeName="DistributionList"
typeClassName="DistributionList"/>

   <TcStandardType typeName="ITEMAppDList" parentTypeName="DistributionList"
typeClassName="DistributionList"/>
```

```
   <TcStandardType typeName="CM_CR_approver_list" parentTypeName="ImanRelation"
typeClassName="ImanRelation"/>
   <TcStandardType typeName="CM_list" parentTypeName="ImanRelation" typeClassName="ImanRelation"/>
   <TcStandardType typeName="CM_state" parentTypeName="ImanRelation" typeClassName="ImanRelation"/>
   <TcStandardType typeName="Change Order" parentTypeName="ImanRelation" typeClassName="ImanRelation"/>
   <TcStandardType typeName="CM_decision" parentTypeName="ImanRelation" typeClassName="ImanRelation"/>
   <TcStandardType typeName="ReferenceFolder" parentTypeName="Folder" typeClassName="Folder"/>
   <TcStandardType typeName="CM Context" parentTypeName="Folder" typeClassName="Folder"/>
   <TcStandardType typeName="CM Default Order Data" parentTypeName="Folder" typeClassName="Folder"/>
   <TcStandardType typeName="CM Setup routing" parentTypeName="Folder" typeClassName="Folder"/>
   <TcStandardType typeName="CP ITEM" parentTypeName="Folder" typeClassName="Folder"/>
   <TcStandardType typeName="CM SETUP" parentTypeName="Folder" typeClassName="Folder"/>
   <TcStandardType typeName="CM Default Proposal Data" parentTypeName="Folder" typeClassName="Folder"/>
   <TcStandardType typeName="CM_Routing Slips" parentTypeName="Folder" typeClassName="Folder"/>
   <TcStandardType typeName="CM LIST" parentTypeName="Folder" typeClassName="Folder"/>
   <TcStandardType typeName="CP PACKAGE" parentTypeName="Folder" typeClassName="Folder"/>
   <TcStandardType typeName="CO REFERENCE" parentTypeName="Folder" typeClassName="Folder"/>
   <TcStandardType typeName="CP Item Folder" parentTypeName="Folder" typeClassName="Folder"/>
   <TcStandardType typeName="CO ITEM REVISION" parentTypeName="Folder" typeClassName="Folder"/>
   <TcStandardType typeName="CO Package" parentTypeName="Folder" typeClassName="Folder"/>
   <TcStandardType typeName="CM Context Job" parentTypeName="Folder" typeClassName="Folder"/>
   <TcStandardType typeName="CP REFERENCE" parentTypeName="Folder" typeClassName="Folder"/>
   <TcStandardType typeName="CO WHERE USED" parentTypeName="Folder" typeClassName="Folder"/>
   <TcStandardType typeName="CM Context Routing" parentTypeName="Folder" typeClassName="Folder"/>
   <TcStandardType typeName="CM Package" parentTypeName="Folder" typeClassName="Folder"/>
   <TcStandardType typeName="IMAN_wolf_link" parentTypeName="ImanRelation" typeClassName="ImanRelation"/>
   <TcStandardType typeName="IMAN_CCContent" parentTypeName="ImanRelation" typeClassName="ImanRelation"/>
   <TcStandardType typeName="NameCounters" parentTypeName="POM_application_object",
typeClassName="NameCounters"/>
   <TcStandardType typeName="NameFields" parentTypeName="POM_application_object"
typeClassName="NameFields"/>
   <TcStandardType typeName="NameRules" parentTypeName="POM_application_object" typeClassName="NameRules"/>
   <TcStandardType typeName="POM_rdbms_dd" parentTypeName="POM_object" typeClassName="POM_rdbms_dd"/>
   <TcStandardType typeName="POM_table" parentTypeName="POM_rdbms_dd" typeClassName="POM_table">
   <TcStandardType typeName="POM_view" parentTypeName="POM_rdbms_dd" typeClassName="POM_view"/>
   <TcStandardType typeName="POM_column" parentTypeName="POM_rdbms_dd" typeClassName="POM_column"/>
   <TcStandardType typeName="POM_index" parentTypeName="POM_rdbms_dd" typeClassName="POM_index"/>
   <TcStandardType typeName="EIM_external_backpointer" parentTypeName="POM_object",
typeClassName="EIM_external_backpointer"/>
   <TcStandardType typeName="CMMV_transform" parentTypeName="POM_object" typeClassName="CMMV_transform"/>


   <TcForm typeName="CMSForm" parentTypeName="Form" typeClassName="Form" formStorageClassName="CMIDNum"/>
```

```
   <TcForm typeName="CORForm" parentTypeName="Form" typeClassName="Form"
formStorageClassName="ChangeReviewData"/>

   <TcForm typeName="ChargeList" parentTypeName="Form" typeClassName="Form"/>

   <TcForm typeName="CNRForm" parentTypeName="Form" typeClassName="Form"/>

   <TcForm typeName="WCForm" parentTypeName="Form" typeClassName="Form"
formStorageClassName="ChangeReviewData"/>

   <TcForm typeName="COForm" parentTypeName="Form" typeClassName="Form"
formStorageClassName="ChangeOrderData"/>

   <TcForm typeName="CPForm" parentTypeName="Form" typeClassName="Form"
formStorageClassName="ChangeProposalData"/>

   <TcForm typeName="WIForm" parentTypeName="Form" typeClassName="Form"
formStorageClassName="ChangeReviewData"/>

   <TcForm typeName="ReasonList" parentTypeName="Form" typeClassName="Form"/>

   <TcForm typeName="CPRForm" parentTypeName="Form" typeClassName="Form"
formStorageClassName="ChangeReviewData"/>

   <TcForm typeName="CNForm" parentTypeName="Form" typeClassName="Form"
formStorageClassName="ChangeNotificationData"/>

   <TcForm typeName="Task Form" parentTypeName="Form" typeClassName="Form"/>


   <TcDataset typeName="IMANReportData" parentTypeName="Dataset" typeClassName="Dataset">

      <TcDSViewTool name="Report Viewer"/>

      <TcDSEditTool name="Report Viewer"/>

   </TcDataset>

   <TcDataset typeName="IMANReportDesign" parentTypeName="Dataset" typeClassName="Dataset">

      <TcDSViewTool name="Report Design"/>

      <TcDSEditTool name="Report Design"/>

   </TcDataset>


   <TcClass className="NameCounters" parentClassName="POM_application_object" isExportable="true"

         isUninstantiable="false" isUninheritable="false" applicationName="INFOMANAGEV200">

      <TcAttribute attributeName="counter_name" attributeType="POM_string" maxStringLength="32"

         isArray="false" followOnExport="false" isNullsAllowed="true" isUnique="false"

         isPublicRead="false" isPublicWrite="false" isCandidateKey="false" arrayLength="0"

         isTransient="false" exportAsString="false" noBackpointer="false"/>

      <TcAttribute attributeName="no_of_chars" attributeType="POM_int" maxStringLength="0"

         isArray="false" followOnExport="false" isNullsAllowed="false" isUnique="false"

         isPublicRead="false" isPublicWrite="false" isCandidateKey="false" arrayLength="0"

         isTransient="false" exportAsString="false" noBackpointer="false"/>

      <TcAttribute attributeName="start_pos" attributeType="POM_int" maxStringLength="0"

         isArray="false" followOnExport="false" isNullsAllowed="false" isUnique="false"

         isPublicRead="false" isPublicWrite="false" isCandidateKey="false" arrayLength="0"

         isTransient="false" exportAsString="false" noBackpointer="false"/>

      <TcAttribute attributeName="char_type" attributeType="POM_string" maxStringLength="1"

         isArray="false" followOnExport="false" isNullsAllowed="true" isUnique="false"
```

```
      isPublicRead="false" isPublicWrite="false" isCandidateKey="false" arrayLength="0"
      isTransient="false" exportAsString="false" noBackpointer="false"/>
   <TcAttribute attributeName="init_value" attributeType="POM_string" maxStringLength="32"
      isArray="false" followOnExport="false" isNullsAllowed="true" isUnique="false"
      isPublicRead="false" isPublicWrite="false" isCandidateKey="false" arrayLength="0"
      isTransient="false" exportAsString="false" noBackpointer="false"/>
   <TcAttribute attributeName="max_value" attributeType="POM_string" maxStringLength="32"
      isArray="false" followOnExport="false" isNullsAllowed="true" isUnique="false"
      isPublicRead="false" isPublicWrite="false" isCandidateKey="false" arrayLength="0"
      isTransient="false" exportAsString="false" noBackpointer="false"/>
   <TcAttribute attributeName="next_id" attributeType="POM_string" maxStringLength="32"
      isArray="false" followOnExport="false" isNullsAllowed="true" isUnique="false"
      isPublicRead="false" isPublicWrite="false" isCandidateKey="false" arrayLength="0"
      isTransient="false" exportAsString="false" noBackpointer="false"/>
</TcClass>

<TcClass className="NameFields" parentClassName="POM_application_object" isExportable="true"
    isUninstantiable="false" isUninheritable="false" applicationName="INFOMANAGEV200">
    <TcAttribute attributeName="type_name" attributeType="POM_string" maxStringLength="32"
       isArray="false" followOnExport="false" isNullsAllowed="false" isUnique="false"
       isPublicRead="false" isPublicWrite="false" isCandidateKey="false" arrayLength="0"
       isTransient="false" exportAsString="false" noBackpointer="false"/>
    <TcAttribute attributeName="property_name" attributeType="POM_string" maxStringLength="32"
       isArray="false" followOnExport="false" isNullsAllowed="false" isUnique="false"
       isPublicRead="false" isPublicWrite="false" isCandidateKey="false" arrayLength="0"
       isTransient="false" exportAsString="false" noBackpointer="false"/>
    <TcAttribute attributeName="case" attributeType="POM_int" maxStringLength="0"
       isArray="false" followOnExport="false" isNullsAllowed="false" isUnique="false"
       isPublicRead="false" isPublicWrite="false" isCandidateKey="false" arrayLength="0"
       isTransient="false" exportAsString="false" noBackpointer="false"/>
    <TcAttribute attributeName="rule_tag" attributeType="POM_typed_reference" maxStringLength="0"
       isArray="false" followOnExport="false" isNullsAllowed="true" isUnique="false"
       isPublicRead="false" isPublicWrite="false" isCandidateKey="false" arrayLength="0"
       isTransient="false" exportAsString="false" noBackpointer="false" typedRefClassName="NameRules"/>
</TcClass>

<TcClass className="NameRules" parentClassName="POM_application_object" isExportable="true"
    isUninstantiable="false" isUninheritable="false" applicationName="INFOMANAGEV200">
    <TcAttribute attributeName="rule_name" attributeType="POM_string" maxStringLength="32"
       isArray="false" followOnExport="false" isNullsAllowed="false" isUnique="true"
       isPublicRead="false" isPublicWrite="false" isCandidateKey="false" arrayLength="0"
```

```
        isTransient="false" exportAsString="false" noBackpointer="false"/>
     <TcAttribute attributeName="pattern" attributeType="POM_string" maxStringLength="240"
        isArray="true" followOnExport="false" isNullsAllowed="true" isUnique="false"
        isPublicRead="false" isPublicWrite="false" isCandidateKey="false" arrayLength="-1"
        isTransient="false" exportAsString="false" noBackpointer="false"/>
     <TcAttribute attributeName="autogen" attributeType="POM_logical" maxStringLength="0"
        isArray="false" followOnExport="false" isNullsAllowed="false" isUnique="false"
        isPublicRead="false" isPublicWrite="false" isCandidateKey="false" arrayLength="0"
        isTransient="false" exportAsString="false" noBackpointer="false"/>
     <TcAttribute attributeName="counter_tags" attributeType="POM_typed_reference"
        maxStringLength="0" isArray="true" followOnExport="false" isNullsAllowed="true"
        isUnique="false" isPublicRead="false" isPublicWrite="false" isCandidateKey="false"
        arrayLength="-1" isTransient="false" exportAsString="false" noBackpointer="false"
        typedRefClassName="NameCounters"/>
</TcClass>

<TcClass className="POM_rdbms_dd" parentClassName="POM_object" isExportable="false"
   isUninstantiable="true" isUninheritable="false" applicationName="POM">
   <TcAttribute attributeName="dbms" attributeType="POM_string" maxStringLength="33"
      isArray="false" followOnExport="false" isNullsAllowed="false" isUnique="false"
      isPublicRead="true" isPublicWrite="false" isCandidateKey="false" arrayLength="0"
      isTransient="false" exportAsString="false" noBackpointer="false"/>
   <TcAttribute attributeName="mapped" attributeType="POM_logical" maxStringLength="0"
      isArray="false" followOnExport="false" isNullsAllowed="false" isUnique="false"
      isPublicRead="true" isPublicWrite="false" isCandidateKey="false" arrayLength="0"
      isTransient="false" exportAsString="false" noBackpointer="false"/>
</TcClass>

<TcClass className="POM_table" parentClassName="POM_rdbms_dd" isExportable="false"
   isUninstantiable="false" isUninheritable="false" applicationName="POM">
   <TcAttribute attributeName="name" attributeType="POM_string" maxStringLength="33"
      isArray="false" followOnExport="false" isNullsAllowed="false" isUnique="false"
      isPublicRead="true" isPublicWrite="false" isCandidateKey="false" arrayLength="0"
      isTransient="false" exportAsString="false" noBackpointer="false"/>
   <TcAttribute attributeName="columns" attributeType="POM_typed_reference" maxStringLength="0"
      isArray="true" followOnExport="false" isNullsAllowed="false" isUnique="false"
      isPublicRead="true" isPublicWrite="false" isCandidateKey="false" arrayLength="-1"
      isTransient="false" exportAsString="false" noBackpointer="false" typedRefClassName="POM_column"/>
</TcClass>

<TcClass className="POM_view" parentClassName="POM_rdbms_dd" isExportable="false"
```

```
   isUninstantiable="false" isUninheritable="false" applicationName="POM">
   <TcAttribute attributeName="name" attributeType="POM_string" maxStringLength="33"
      isArray="false" followOnExport="false" isNullsAllowed="false" isUnique="false"
      isPublicRead="true" isPublicWrite="false" isCandidateKey="false" arrayLength="0"
      isTransient="false" exportAsString="false" noBackpointer="false"/>
</TcClass>

<TcClass className="POM_column" parentClassName="POM_rdbms_dd" isExportable="false"
   isUninstantiable="false" isUninheritable="false" applicationName="POM">
   <TcAttribute attributeName="name" attributeType="POM_string" maxStringLength="33"
      isArray="false" followOnExport="false" isNullsAllowed="false" isUnique="false"
      isPublicRead="true" isPublicWrite="false" isCandidateKey="false" arrayLength="0"
      isTransient="false" exportAsString="false" noBackpointer="false"/>
   <TcAttribute attributeName="object" attributeType="POM_typed_reference" maxStringLength="0"
      isArray="false" followOnExport="false" isNullsAllowed="false" isUnique="false"
      isPublicRead="true" isPublicWrite="false" isCandidateKey="false" arrayLength="0"
      isTransient="false" exportAsString="false" noBackpointer="false" typedRefClassName="POM_rdbms_dd"/>
   <TcAttribute attributeName="type" attributeType="POM_int" maxStringLength="0"
      isArray="false" followOnExport="false" isNullsAllowed="false" isUnique="false"
      isPublicRead="true" isPublicWrite="false" isCandidateKey="false" arrayLength="0"
      isTransient="false" exportAsString="false" noBackpointer="false"/>
   <TcAttribute attributeName="length" attributeType="POM_int" maxStringLength="0"
      isArray="false" followOnExport="false" isNullsAllowed="false" isUnique="false"
      isPublicRead="true" isPublicWrite="false" isCandidateKey="false" arrayLength="0"
      isTransient="false" exportAsString="false" noBackpointer="false"/>
   <TcAttribute attributeName="nulls_allowed" attributeType="POM_logical" maxStringLength="0"
      isArray="false" followOnExport="false" isNullsAllowed="false" isUnique="false"
      isPublicRead="true" isPublicWrite="false" isCandidateKey="false" arrayLength="0"
      isTransient="false" exportAsString="false" noBackpointer="false"/>
</TcClass>

<TcClass className="POM_index" parentClassName="POM_rdbms_dd" isExportable="false"
   isUninstantiable="false" isUninheritable="false" applicationName="POM">
   <TcAttribute attributeName="name" attributeType="POM_string" maxStringLength="33"
      isArray="false" followOnExport="false" isNullsAllowed="false" isUnique="false"
      isPublicRead="true" isPublicWrite="false" isCandidateKey="false" arrayLength="0"
      isTransient="false" exportAsString="false" noBackpointer="false"/>
   <TcAttribute attributeName="columns" attributeType="POM_typed_reference" maxStringLength="0"
      isArray="true" followOnExport="false" isNullsAllowed="false" isUnique="false"
      isPublicRead="true" isPublicWrite="false" isCandidateKey="false" arrayLength="-1"
      isTransient="false" exportAsString="false" noBackpointer="false" typedRefClassName="POM_column"/>
```

```
   <TcAttribute attributeName="table" attributeType="POM_typed_reference" maxStringLength="0"
      isArray="false" followOnExport="false" isNullsAllowed="false" isUnique="false"
      isPublicRead="true" isPublicWrite="false" isCandidateKey="false" arrayLength="0"
      isTransient="false" exportAsString="false" noBackpointer="false" typedRefClassName="POM_table"/>
   <TcAttribute attributeName="unique" attributeType="POM_logical" maxStringLength="0"
      isArray="false" followOnExport="false" isNullsAllowed="false" isUnique="false"
      isPublicRead="true" isPublicWrite="false" isCandidateKey="false" arrayLength="0"
      isTransient="false" exportAsString="false" noBackpointer="false"/>
</TcClass>

<TcClass className="EIM_external_backpointer" parentClassName="POM_object" isExportable="false"
   isUninstantiable="false" isUninheritable="false" applicationName="EIM">
   <TcAttribute attributeName="to_reference" attributeType="POM_untyped_reference"
      maxStringLength="0" isArray="false" followOnExport="false" isNullsAllowed="false"
      isUnique="false" isPublicRead="false" isPublicWrite="false" isCandidateKey="false"
      arrayLength="0" isTransient="false" exportAsString="false" noBackpointer="false"/>
</TcClass>

<TcClass className="CMMV_transform" parentClassName="POM_object" isExportable="false"
   isUninstantiable="false" isUninheritable="false" applicationName="CMMV">
   <TcAttribute attributeName="transform" attributeType="POM_double" maxStringLength="0"
      isArray="true" followOnExport="false" isNullsAllowed="false" isUnique="false"
      isPublicRead="false" isPublicWrite="false" isCandidateKey="false" arrayLength="16"
      isTransient="false" exportAsString="false" noBackpointer="false"/>
</TcClass>

<TcTool toolName="Report Design" toolMimeType="" toolSymbol="ReportDesign" toolVersion="1.0"
   toolReleaseDate="">
   <TcToolInputFormat formatName="BINARY"/>
   <TcToolOutputFormat formatName="BINARY"/>
</TcTool>

<TcTool toolName="Report Viewer" toolMimeType="" toolSymbol="ReportViewer" toolVersion="1.0"
   toolReleaseDate="">
   <TcToolInputFormat formatName="BINARY"/>
   <TcToolOutputFormat formatName="BINARY"/>
</TcTool>
```

Deleted XML Definitions

List of XML deleted definitions that may end up in your custom template:

```
<TcClass className="UserInBox" parentClassName="POM_object" isExportable="false"
          isUninstantiable="false" isUninheritable="false" applicationName="INFOMANAGEV200">
        <TcAttribute attributeName="inbox_user" attributeType="POM_typed_reference"
            maxStringLength="0" isArray="false" followOnExport="true" isNullsAllowed="false"
            isUnique="false" isPublicRead="true" isPublicWrite="false" isCandidateKey="false"
            arrayLength="0" isTransient="false" exportAsString="false" noBackpointer="false"
            typedRefClassName="User"/>
        <TcAttribute attributeName="inbox_delegate" attributeType="POM_untyped_reference"
            maxStringLength="0" isArray="false" followOnExport="true" isNullsAllowed="true"
            isUnique="false" isPublicRead="true" isPublicWrite="false" isCandidateKey="false"
            arrayLength="0" isTransient="false" exportAsString="false" noBackpointer="false"/>
        <TcAttribute attributeName="start_date" attributeType="POM_date" maxStringLength="0"
            isArray="false" followOnExport="false" isNullsAllowed="true" isUnique="false"
            isPublicRead="false" isPublicWrite="false" isCandidateKey="false" arrayLength="0"
            isTransient="false" exportAsString="false" noBackpointer="false"/>
        <TcAttribute attributeName="end_date" attributeType="POM_date" maxStringLength="0"
            isArray="false" followOnExport="false" isNullsAllowed="true" isUnique="false"
            isPublicRead="false" isPublicWrite="false" isCandidateKey="false" arrayLength="0"
            isTransient="false" exportAsString="false" noBackpointer="false"/>
        <TcAttribute attributeName="inbox_resourcepools" attributeType="POM_typed_reference"
            maxStringLength="0" isArray="true" followOnExport="true" isNullsAllowed="false"
            isUnique="false" isPublicRead="true" isPublicWrite="false" isCandidateKey="false"
            arrayLength="-1" isTransient="false" exportAsString="false" noBackpointer="false"
            typedRefClassName="ResourcePool"/>
    </TcClass>

    <TcStandardType typeName="UserInBox" parentTypeName="POM_object" typeClassName="UserInBox"/>
    <TcStandardType typeName="BOMViewRevision" parentTypeName="PSBOMViewRevision"
                    typeClassName="PSBOMViewRevision"/>
    <TcStandardType typeName="IMAN_drawing" parentTypeName="ImanRelation"
                    typeClassName="ImanRelation"/>
   <TcStandardType typeName="ExternalObject" parentTypeName=" WolfObject "
                   typeClassName=" WolfObject"/>
```

17.3 How to remove custom tools, references, and tool actions from a COTS Dataset

This section covers how to remove any custom tools, references, and tool actions added by your BMIDE template to a COTS Dataset definition.

This situation can occur through several scenarios:

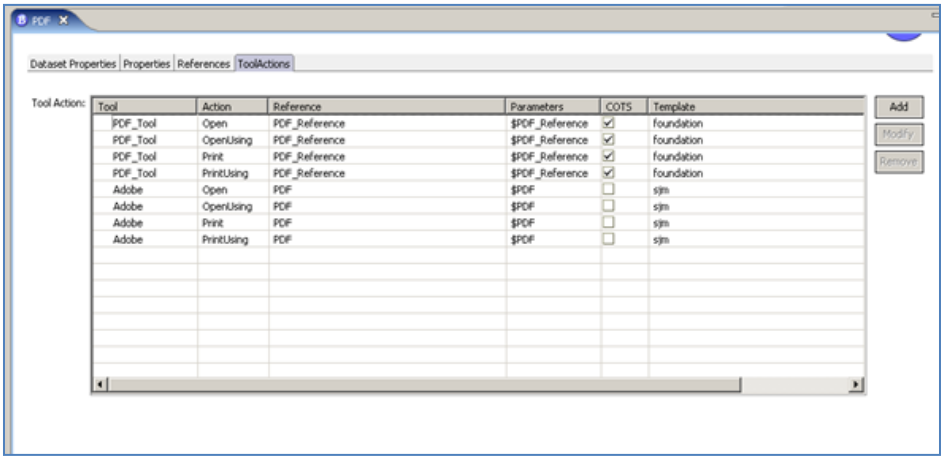
- It is permissible in the BMIDE to extend a COTS dataset by adding additional custom view tools, edit tools, references, and tool actions.

- During an upgrade from Teamcenter Engineering to Teamcenter unified architecture, you had defined some custom Datasets who names collided with a COTS Dataset definition. If you had run the Type Analysis Tool on the Engineering database before upgrading, the generated report would have advised you of this situation. If you chose to allow your custom dataset to be merged with the COTS dataset during the upgrade, then you would have ended up with a merged dataset. This will commonly occur with PDF and Zip Datasets.
- During an upgrade from Teamcenter unified architecture to Teamcenter unified architecture, you found that one of your custom Dataset definitions collided with a new COTS Dataset with the same name. Following the upgrade instructions for handling type name collisions, you chose to merge your dataset definition with the COTS definition. This will commonly occur with PDF and Zip Datasets.

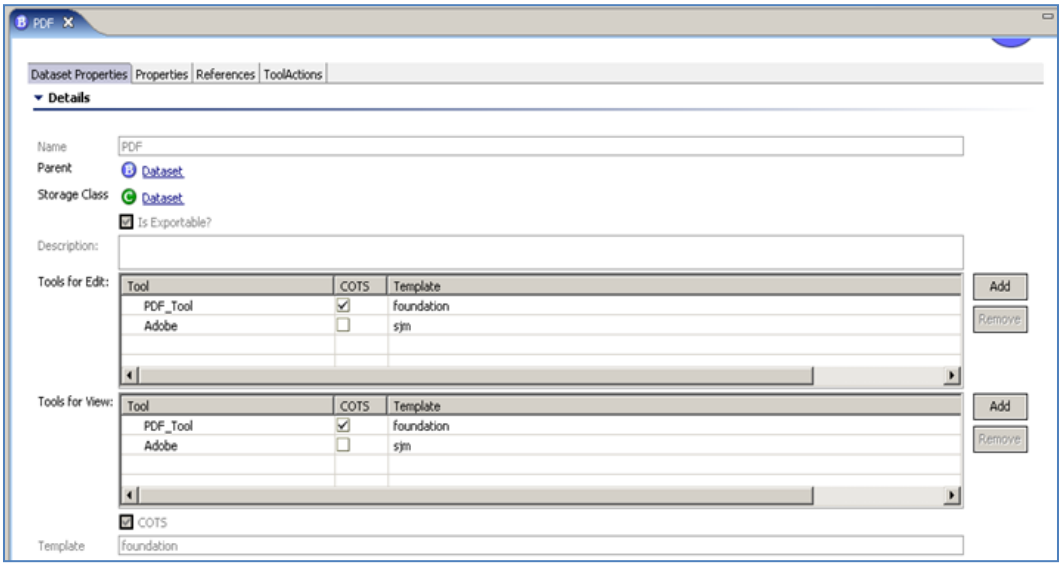
If any of these three scenarios occurred, it will not cause any issues for your database. However, you can remove the custom tools, references, and tool actions from the COTS dataset. One reason for wanting to remove the custom extensions is that if there are two sets of View Tools and Edit Tools, users will be asked to choose between the two tools each time they open or edit the dataset in a client.

Follow these steps to remove any custom extensions from a COTS Dataset

1. Launch the new BMIDE client
2. Go to the Business Objects View and find the Dataset. Example: PDF
3. Select the Dataset and right click->Open
4. In the Dataset editor click on the “Tool Actions” tab. (See below)
    a. Review the list of tool actions. Note the ones which are COTS and custom.
    b. Select the custom tool actions to remove and click the “Remove” button. The custom tool action should be removed from the list of tools.
    c. Go to File->Save Data Model to save the changes to the template source files.



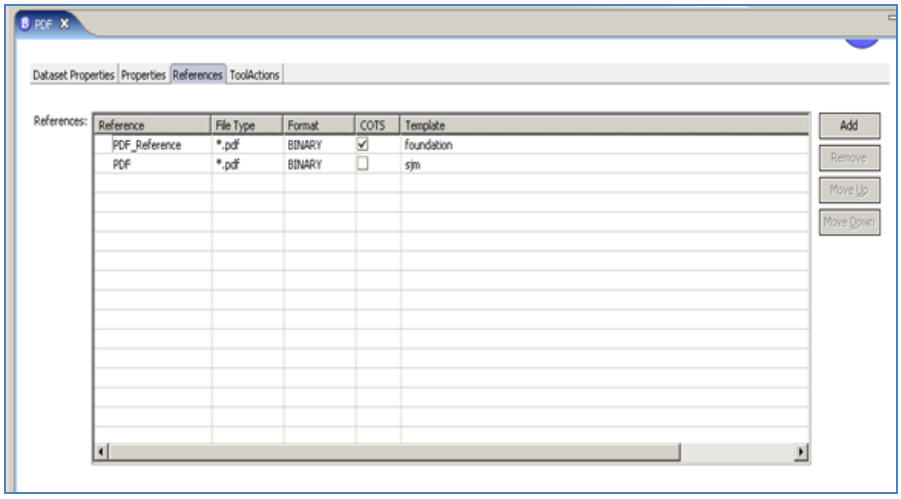
5. In the Dataset editor click on the “Dataset Properties” tab (see below).
    a. Review the Tools for Edit and Tools for View. Note the ones which are COTS and custom.
    b. For any tool that you want to remove, write down the name of the tool so it can be used in a later step. In the example below, the custom tool name is “Adobe”.
    c. Additionally write down the name of the COTS tool. In the example below the COTS tool is “PDF_Tool”.
    d. Select the tool to remove and click the “Remove” button. The custom tool should be removed from the list of tools.
    e. Go to File->Save Data Model to save the changes to the template source files.



6. In the Dataset editor click on the “References” tab. (See below)
    a. Review the list of references. Note the ones which are COTS and custom.
    b. For any reference that you want to remove, write down the name of the reference so it can be used in a later step. In the example below, the custom reference name is “PDF”.
    c. Additionally write down the name of the COTS reference. In the example below the COTS reference is “PDF_Reference”.
    d. Select the reference to remove and click the “Remove” button. The custom reference should be removed from the list of tools.
    e. Go to File->Save Data Model to save the changes to the template source files.
    f. Note: in some versions of the BMIDE there is a known bug where the “Remove” will not properly remove the XML definition from the source file during the save of the data model.
        i. To check, open up the XML source files in your BMIDE and look for the definition below. In the example below, the XML definition adds the custom reference “PDF” to the “PDF” dataset. You may have a different Dataset name and reference name but the XML tags will look the same.
        ii. If you find this XML definition for your custom reference, then manually remove it from the file.

```
<TcDatasetReferenceAttach datasetType="PDF">
   <TcDatasetReference name="PDF">
      <TcDatasetReferenceInfo template="*.pdf" format="BINARY"/>
   </TcDatasetReference>
</TcDatasetReferenceAttach>
```

        iii. After removing the XML definition save the XML file.
        iv. Go to the Navigator View; select the Project icon, right-click -> Reload Data Model. This will validate that the manual changes to the XML file are correct. If there are issues these errors will appear in the Console view. If there are errors, fix the XML file and reload the data model again. Repeat until there are no errors remaining.



7. Create a file named *FixCollidingDatasetInfo.xml*. This file will contain the data to be used to migrate all dataset instances that were previously referring to custom dataset's references and tools to the COTS references and tools. This XML file will be used as input to the change_datasets utility. For more information about this utility see the utilities guide. The XML file format is show below and works as follows.
    a. Enter the name of the dataset to change in the "DatasetName" field.
    b. Enter the custom Tool name in the "OldToolName" field.
    c. Enter the COTS Tool name in the "NewToolName" field.
    d. Enter the Custom reference name in the "OldReferenceName1" field.
    e. Enter the COTS reference name in the "NewReferenceName1" field.
    f. Repeat the <TcDatasetReference> tag for multiple reference names that need to be changed.

```
<TcDatasets>
    <TcDataset name="DatasetName">
        <TcTool fromName="OldToolName" toName="NewToolName"/>
        <TcDatasetReferences>
            <TcDatasetReference fromName="OldReferenceName1" toName="NewReferenceName1"/>
            <TcDatasetReference fromName="OldReferenceName2" toName="NewReferenceName2"/>
        </TcDatasetReferences>
    </TcDataset>
</TcDatasets>
```

8. To construct the contents of the *FixCollidingDatasetInfo.xml,* you should refer back to the names of the tools and references that you wrote down from the previous steps. Based on the example that we have provided, the resulting file would be shown below.
    a. The configuration file below states that we will be converting any "PDF" Dataset instances that use the "Adobe" tool to the "PDF_Tool".
    b. And will convert any "PDF" Dataset instances that use the "PDF" reference name to the "PDF_Reference" name.

```
<TcDatasets>
    <TcDataset name="PDF">
        <TcTool fromName="Adobe" toName="PDF_Tool"/>
        <TcDatasetReferences>
            <TcDatasetReference fromName="PDF" toName="PDF_Reference"/>
        </TcDatasetReferences>
    </TcDataset>
</TcDatasets>
```

9. Open up a Teamcenter command line prompt and execute the clearlocks command to ensure there are no others users logged onto the system
10. In the command prompt execute the following command:

    change_datasets –u=infodba –p=<password> –g=dba –ds_info=FixCollidingDatasetInfo.xml

11. You should view the log file "*%temp%/ChangeDatasetInfo_<timestamp>.log*" to ensure that there are no errors.

    a. If there are errors, then look at the log files named "*TC_ROOT/logs/change_dataset*.syslog*" to understand the reason for failure.

    b. One of the common reasons for the utility to fail is if the Teamcenter DBA user (say infodba) does not have sufficient privileges to update dataset instances created by other users. The error message for this is shown below.

       Failed to lock instances 21608. Error is = 515001

    c. If this is the case, go to the Access Manager application and add the follow rule. Then move the rule up in the tree just below the "Has Bypass" rule.
       Condition = "Is SA"
       Value = "true"
       ACL Name = "Bypass"

    d. After executing the utility, remove this rule from the Access Manager.

12. Note that if you want this utility to be automatically run, so that it cleans up a database during an upgrade, you should also add the "change_datasets" command listed above and the corresponding XML file to your custom template project in the BMIDE. See section 13.2.2 for more details about how to add this command and XML file to the upgrade script(s) for your template.
13. Package the BMIDE template using the "Package Template Extensions" wizard in the BMIDE.
14. Use TEM to deploy the packaged template to the database from the previous step. This will correct the data model definitions for the Dataset in the database.

15. After the upgrade completes, you may want to validate that the instances of the dataset were corrected by following section 13.2.3.

17.4 How to remove COTS elements causing data model errors from custom BMIDE template

This section is applicable only if you are upgrading your custom BMIDE template in the BMIDE client to Teamcenter 2007.1.4 (or later) release and are seeing data model errors when upgrading the custom template.

Note: *These instructions should not be used to remove any other element that is causing data model errors in the BMIDE client. Please contact GTAC for help with such issues.*

Below is a list of elements that Teamcenter development has reinstated in the COTS templates. Your action item is to inspect your custom template and look for the existence of any of these COTS elements listed below and remove them manually.

List of COTS elements that may result in the custom BMIDE template that should be deleted manually from the custom BMIDE template XML files:

```
<TcOperationAttach operationName="PROP_ask_value_string" extendableElementName="MEActivity"
    extendableElementType="Type" propertyName="long_description">
  <TcExtensionPoint extensionPointType="PreCondition" isOverridable="true"/>
  <TcExtensionPoint extensionPointType="PreAction" isOverridable="true"/>
  <TcExtensionPoint extensionPointType="BaseAction" isOverridable="false"/>
  <TcExtensionPoint extensionPointType="PostAction" isOverridable="true"/>
</TcOperationAttach>
<TcOperationAttach operationName="PROP_set_value_string" extendableElementName="MEActivity"
    extendableElementType="Type" propertyName="long_description">
  <TcExtensionPoint extensionPointType="PreCondition" isOverridable="true"/>
  <TcExtensionPoint extensionPointType="PreAction" isOverridable="true"/>
  <TcExtensionPoint extensionPointType="BaseAction" isOverridable="false"/>
  <TcExtensionPoint extensionPointType="PostAction" isOverridable="true"/>
</TcOperationAttach>
<TcExtension name="MEActivity:long_description:PROP_ask_value_string" internal="true"
    cannedExtension="false" languageType="CPlusPlus">
  <TcExtensionValidity parameter="PROPERTY:MEActivity:long_description:PROP_ask_value_string:4"/>
</TcExtension>
<TcExtension name="MEActivity:long_description:PROP_set_value_string" internal="true"
    cannedExtension="false" languageType="CPlusPlus">
  <TcExtensionValidity parameter="PROPERTY:MEActivity:long_description:PROP_set_value_string:4"/>
</TcExtension>
<TcExtensionAttach extensionName="MEActivity:long_description:PROP_ask_value_string"
    operationName="PROP_ask_value_string" isActive="true" propertyName="long_description"
    extendableElementName="MEActivity" extendableElementType="Type" extensionPointType="BaseAction"/>
<TcExtensionAttach extensionName="MEActivity:long_description:PROP_set_value_string"
    operationName="PROP_set_value_string" isActive="true" propertyName="long_description"
    extendableElementName="MEActivity" extendableElementType="Type" extensionPointType="BaseAction"/>
```

17.5 How to add Smart Folder capability to Teamcenter without having to install the CPG Template

Smart folders provide a mechanism for configuring arbitrary subdivisions of data within a project or program based on functional and sub functional units. Smart folders are pseudo folders configured in a hierarchical structure; they are not physical folder objects in Teamcenter.

Capability to add/show smart folders in Teamcenter Project is now available without CPG.

In Teamcenter 10.1 global constant (DefaultProjectSmartFolders) definition moved from “Packaging and Artwork” template to foundation. The value for this constant is the LOV name that holds the default smart folder hierarchy for a project.

17.5.1 For pre Teamcenter 10.1 customers (from Teamcenter 8.3.3 up to Teamcenter10)

This is not applicable if you have “Packaging and Artwork” template deployed in your database as it’s already have this constant defined.

17.5.1.1 How to create this global constant in their custom template

Assumption: You must have working setup of Business Modeler IDE which contains loaded custom template.

Step 1 : How to manually edit/add global constant-

- Go to the Navigator view, open the extension file (example - default.xml)
- Add the XML into the file between the <Add> tags as shown below

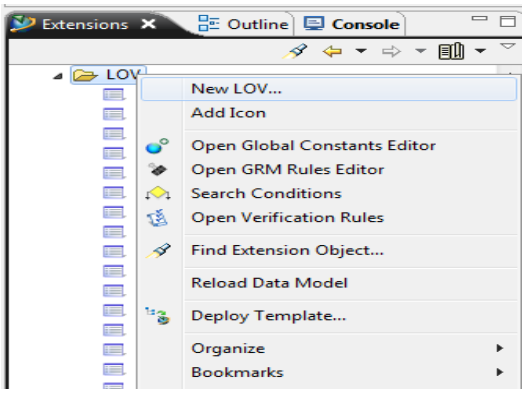
```
<Add>
  <TcGlobalConstant name="DefaultProjectSmartFolders"
dataType="String" defaultValue=""
   description="DefaultProjectSmartFolders" isMultiValued="false"
isValueAttachmentSecured="false"
   allowOpsDataUpdate="false" allowOpsDataUpdateOverride="false"/>
</Add>
```

- Save and close the file
- Reload Data Model

Step 2 : How to override global constant (DefaultProjectSmartFolders)

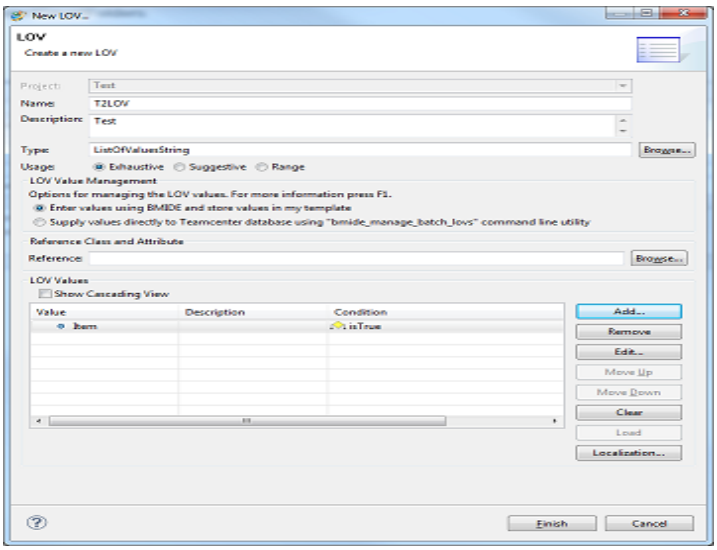
How to add LOVs

- Select “LOV” Folder under Extension Tab in Business Model IDE

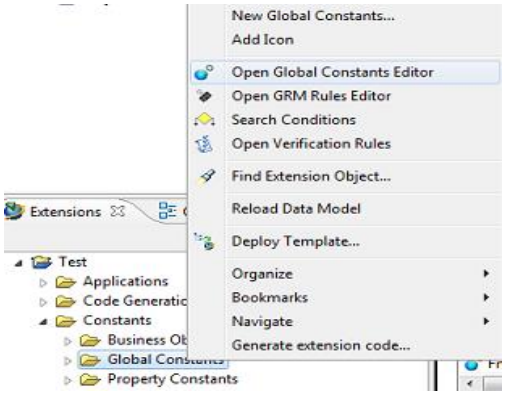


- Click on “New LOV…” menu option on RMB it will open a new wizard to create LOV.
- In wizard fill the required values.
- Click on “Add” button to add your Business Object Names.

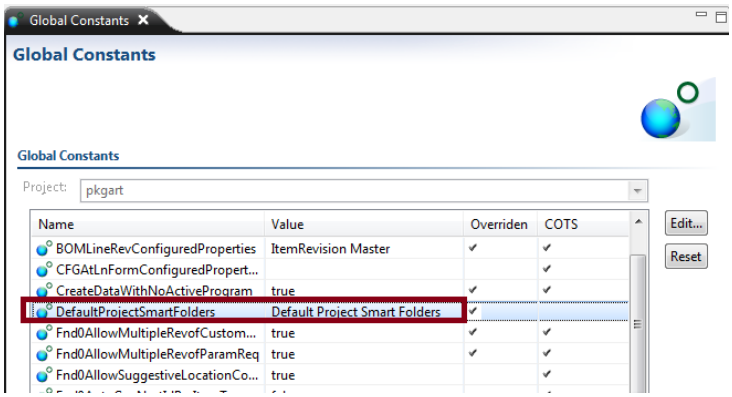
- Click on “Finish” to complete.



- Open Global Constant Editor to attach LOV created in above step.

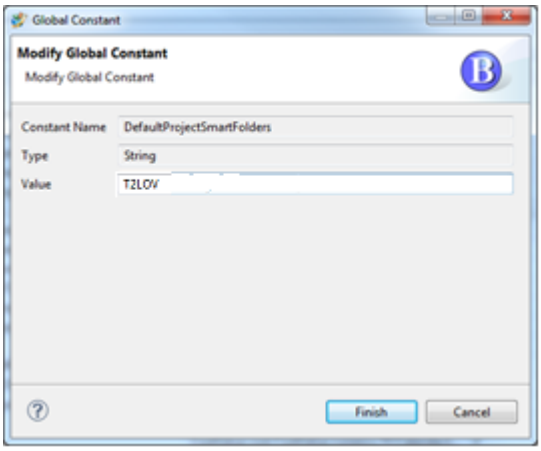


- Select “**DefaultProjectSmartFolders**” from List of Global Constants and click on “Edit…” button.



- In Value filed enter the name of LOV created above (i.e. T2LOV).

- Click on “Finish” to complete.



- Save Data Model.
- Deploy Template.
- These steps above will generate a **TcGlobalConstantAttach** statements in the extension file similar to the one shown below-

```
<TcGlobalConstantAttach constantName="DefaultProjectSmartFolders"
value="Default Project Smart Folders" />

<TcLOV name="T2LOV" lovType="ListOfValuesString" usage="Exhaustive"
description="">
  <TcLOVValue value="Item" description="" conditionName="isTrue" />
  <TcLOVValue value="Specifications" description=""
conditionName="isTrue" />
  <TcLOVValue value="Products" description="" conditionName="isTrue"
/>
</TcLOV>
```

17.5.2 For customers upgrading to Teamcenter 10.1

For those customers who follow the steps listed above to add the Global Constant to your own template, you will have some additional steps to follow when you upgrade to Teamcenter 10.1. Since Teamcenter 10.1 also has the DefaultProjectSmartFolders Global Constant defined in the Foundation template, the system will not allow it to be defined twice, once in the Foundation and template and again in your template. You will need to remove the definition from your template during the upgrade. Follow these steps below as a pre-upgrade activity..

17.5.2.1 You already have global constant DefaultProjectSmartFolders defined in your custom template

In a case your template already has definition of **DefaultProjectSmartFolders** constant you will get error during process of template migration for duplicate constant definition. You need to remove this from custom template to avoid any conflict.

Assumption: - You must have working setup of Business Modeler IDE which contains loaded custom template.

How to resolve the error :

You need to execute below steps before you start upgrade.

- Go to the Navigator view, find the source file where you have **DefaultProjectSmartFolders** defined
- Double click to open it
- Delete Global Constant and its attachment-

```
<TcGlobalConstant name="DefaultProjectSmartFolders" dataType="String"
defaultValue=""
description="DefaultProjectSmartFolders" isMultiValued="false"
isValueAttachmentSecured="false"
allowOpsDataUpdate="false" allowOpsDataUpdateOverride="false"/>
```

You need to delete **TcGlobalConstantAttach** and **TcLOV** as well from extension file (where this attachment was defined), delete below-

```
<TcGlobalConstantAttach constantName="DefaultProjectSmartFolders"
value="Default Project Smart Folders"/>

<TcLOV name="T2LOV" lovType="ListOfValuesString" usage="Exhaustive"
description="">
  <TcLOVValue value="Item" description="" conditionName="isTrue" />
  <TcLOVValue value="Specifications" description=""
conditionName="isTrue" />
  <TcLOVValue value="Products" description="" conditionName="isTrue"
/>
</TcLOV>
```

- Save and close the extension file
- Reload Data Model
- Package your templates and use them for the upgrade.

17.5.2.2 Post Upgrade Steps

After successful upgrade to Teamcenter 10.1, please refer section 1.1.2 of this document to add your custom LOVs and override the **DefaultProjectSmartFolder** global constant again.

17.5.2.3 Special scenarios

If you have multiple custom templates, you need to clean up the **DefaultProjectSmartFolders** global constant definition and attachments from all those templates. For instructions please refer section 2.1.

18 How to rename a template

This section describes the process to rename a BMIDE template. You can use this process to rename a template that you own. However, you cannot use this process to rename a template that is shipped in the Teamcenter kit or templates shipped by Teamcenter integrations and third parties.

The intention of renaming a template is to change the internal name of your template. Let's ensure you understand what this means. Every template has a "Template name" and "Template display name". For example in the Figure 67 the "Template name" is "seplanning" and the "Template display name" is "SE Planning".

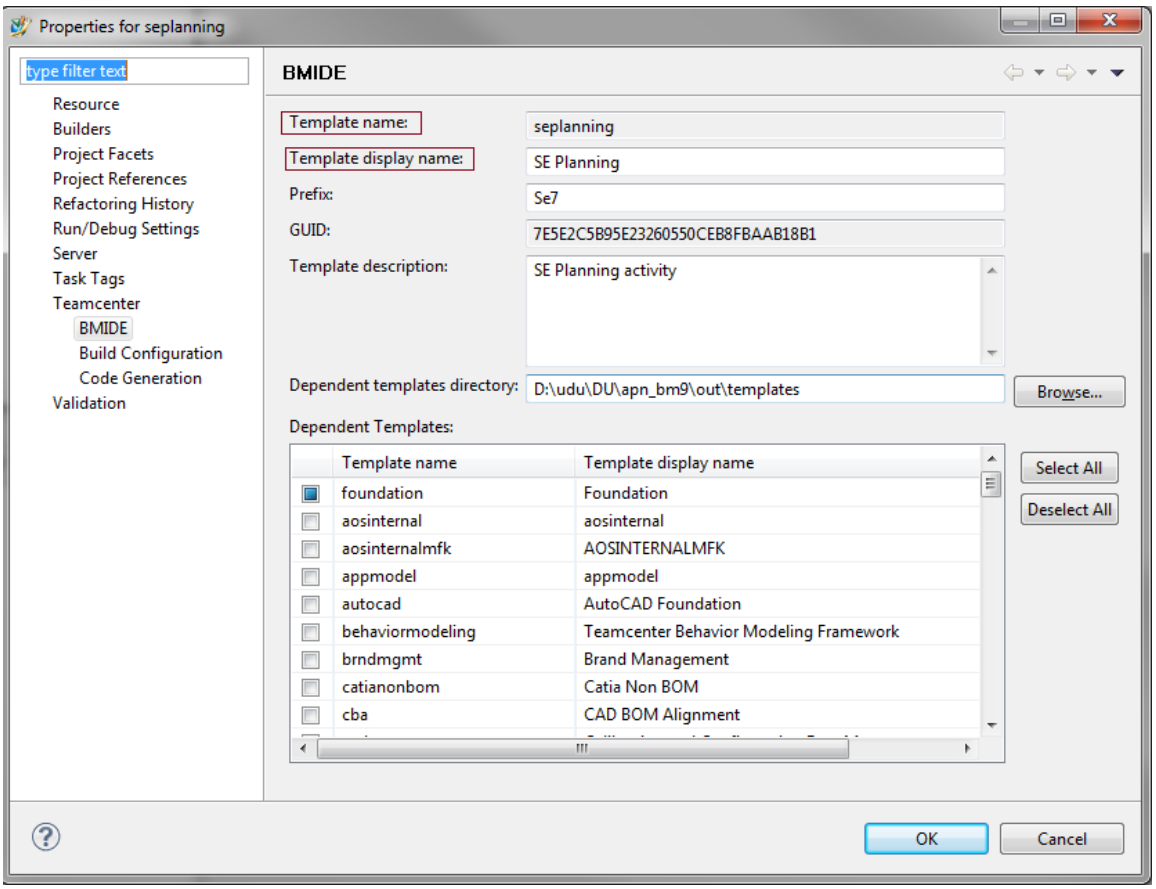


Figure 67 Template Project Properties Page

The "Template name" is an internal unique name for your template. This name is used by BMIDE and its utilities for various reasons like in creation of your template package for installation and distribution, registering your template in the database, templates that dependent on your template refers to it by this internal name. The reason for this behavior is because we do not expect the internal name of a template to change. The "Template display name" on the other hand is what is visible to users during TEM installation, update and upgrade and can be changed at any time to suite your business needs.

The process described in this section is to help you change the "Template name" field entry which is currently disabled in the BMIDE.

An alternative to changing the "Template name" is to just change the "Template display name". It is easy to change the "Template display name" for your template without incurring an overhead of testing and validation. If you intend to change the "Template display name" then you can do the following –

1) Open your project in BMIDE

2) Go to the Navigator view
3) Select your project, RMB select Properties→Teamcenter→BMIDE
4) Change the “Template Display Name” field
5) Click “OK”. This will update the display name for your template and the new display name will show up in your future TEM install, update or upgrade.

If your business needs require you to change the “Template name” then you must exercise caution while renaming your template. Here is the list of cautionary items-

- There is no automated support to rename a template. It is a manual process.
- Renaming a template will require system downtime.
- After you rename your template in BMIDE, you must update the template name in all your Developer Environments, Integration Test Environments, Production Environments, User Acceptance Testing Environments, etc.
- You must also update other templates that you own and are dependent on the template being renamed.
- You must share your renamed template package with your partners who have installed your template and request them to update their Developer Environments, Integration Test Environments, Production Environments, User Acceptance Testing Environments, etc.
- If your partners have created additional templates that are dependent on your renamed template, then you must inform them to update all such templates to use the new name of your template.
- If you have shared your template with partners then do not include new data model in your renamed template. Ensure that the data model contents of your renamed template matches with the one that was previously distributed to your partners. It is not wise to distribute a template that is both renamed and has new data model changes. Since renaming is a manual process, including new data model changes in a renamed template increases the debugging complexity if we encounter issue with deploying your renamed template.
- If you have shared your template with partners, then the renamed template that you distribute must be in the same release as that of your partner. For example, if you shared your template belonging to Teamcenter 8.3 with your partner, then renaming of your template must also be done in the same Teamcenter 8.3 release. You cannot rename your template in a release that is newer to Teamcenter 8.3 or older to Teamcenter 8.3.

While describing the renaming steps, the following conventions are used:

- **oldtemplate**
  - This is the name used to indicate the current name of your template
- **newtemplate**
  - This is the name used to indicate the new name for your template

18.1 Renaming your template project in BMIDE

The first step in the renaming process is to rename your template project in BMIDE and ensure it loads successfully within BMIDE. The following are the steps to rename your template project in BMIDE.

1) Ensure your **oldtemplate** is in sync with the database.
   - This means if you have any new data model changes in your template that is not yet deployed to the database, ensure you deploy them before renaming your template. After renaming your template, the final step would be to deploy your renamed template to the database. So if you include deployment of new data model and renaming of a template as a single process and say deployment fails, then it adds to the complexity of debugging such failed deployments.
   - But if you have shared your template with your partner then do not add new data model to your template. Ensure the data model matches with the one that was distributed to your partners.

2) Keep a backup copy of your **oldtemplate** project. In case you run into issues with renaming your template, we can always get back to your old state of the template.

3) Launch BMIDE and remove **oldtemplate** project from the BMIDE workspace.
   - You have to remove the oldtemplate project from the BMIDE workspace so that you can rename the project outside of BMIDE and then import it back into BMIDE. Follow the below steps to remove your project.

   a) Go to the Navigator view
   b) Select **oldtemplate** project, RMB select "Close Project"
   c) Next to remove the project from workspace, RMB select "Delete" on your closed project
      - Caution: In the conformation dialog, do not select the option "Delete project contents on disk (cannot be undone)". You should leave this option unchecked.
   d) Once the oldtemplate project is removed from the BMIDE workspace, shutdown the BMIDE client.

4) In the next few steps, we will make changes to **oldtemplate** project files in your file system. We will open certain files in your project to find and replace the word "oldtemplate" (which is the name of the current template that you are renaming) with the word "newtemplate" which is the new name for your template. During this process, please ensure that you are doing a case sensitive search and replace.

5) In your file system, go to the location where the **oldtemplate** project resides. For example <TC_ROOT>/bmide/workspace/<oldtemplate project>.

   a) **File: client_oldtemplate.properties**
      i) Open the file <project>/client_oldtemplate.properties, if it exists.
      ii) Search and replace all occurrences of **oldtemplate** with **newtemplate**
      iii) Rename the file client_oldtemplate.properties as client_newtemplate.properties

      File sample after changes:

```
#Business Modeler Client Properties
#Wed Mar 30 08:46:12 BST 2011
FullDeployCS=3B8550CF8913E7BEF8EE147FF005B803
client_id=Config5692VM0_newtemplate_56C0C7A8A783F6D153CF6E22E9CE733A
```

   b) **File: ProjectInfo.xml**
      i) Open the file <project>/ProjectInfo.xml
      ii) Search and replace all occurrences of **oldtemplate** with **newtemplate**

      File sample after changes:

```
<NameValue key="Codegen.Namespace" value="newtemplate"/>
```

   c) **File: .project**
      i) Open the file <project>/.project
      ii) Search and replace all occurrences of **oldtemplate** with **newtemplate**

      File sample after changes:

```
<?xml version="1.0" encoding="UTF-8"?>
<projectDescription>
<name>newtemplate</name>
<comment></comment>
<projects>
</projects>
<buildSpec>
</buildSpec>
<natures>
<nature>com.teamcenter.bmide.server.ui.TcBusinessDataProjectNature</nature>
<nature>com.teamcenter.bmide.server.ui.FileRepositoryProjectNature</nature>
```

```
<nature>com.teamcenter.bmide.server.ui.ExternalProjectNature</nature>
</natures>
</projectDescription>
```

d) **File: dependency.xml**
   i) Open the file <project>/extensions/ dependency.xml
   ii) Search and replace all occurrences of **oldtemplate** with **newtemplate**

   File sample after changes:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<TcBusinessDataIncludes ActiveRelease="tc9000.0.0" displayName="Template for planning
division" enableOpsDataDeploy="true" guid="399B9CA4D818408AD040E33034580C1C"
name="newtemplate" optional="true" prefixes="R4" teamcenterTemplate="false">
  <include file="foundation_template.xml"/>
</TcBusinessDataIncludes>
```

e) **File: Language files**
   i) This step is required only if you are working in Teamcenter 8.3 or a later release
   ii) Rename all the language files located at <project>/extension/lang
   iii) For example, rename "**oldtemplate**_template_en_US.xml" to "**newtemplate**_template_en_US.xml"

f) **File: feature_oldtemplate.xml**
   i) Open the file <project>/install/feature_oldtemplate.xml
   ii) Search and replace all occurrences of **oldtemplate** with **newtemplate**
   iii) Rename the file feature_oldtemplate.xml to feature_newtemplate.xml

   File sample after changes:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!--
    Document   : feature_newtemplate.xml
    Description: This XML is used by TEM to install or upgrade the "newtemplate"
solution.
-->
<feature>
    <name value="Template for planning division"/>
    <property name="feature_name" value="newtemplate"/>
    <group value="package"/>
    <guid value="74D6C1B94FAE0AE4158817B01ECB8AAD"/>
    <bundle value="${feature_name}Bundle.xml"/>
    <description value="${feature_name}.description"/>
    <include file="dataModelDependency.xml"/>
    <relation/>
   ...
   ...
</feature>
```

g) **File: install_oldtemplate.default**
   i) Open the file <project>/install/install_oldtemplate.default
   ii) Search and replace all occurrences of **oldtemplate** with **newtemplate**
   iii) Rename the file install_oldtemplate.default to install_newtemplate.default

   File sample after changes:

```
#!/bin/sh
# install_newtemplate.default:
# This script installs the Data Model for newtemplate solution.
# ____________________________________________________________________________
...
...
```

```
bmide_manage_templates -u=infodba -p=${TC_USER_PASSWD} -g=dba -option=add -
templates=newtemplate
```

h) **File: upgrade_oldtemplate_xxxx.default**
   i) Open all your upgrade scripts whose name is of the format <project>/install/upgrade_oldtemplate_v2007.default
   ii) Search and replace all occurrences of **oldtemplate** with **newtemplate**
   iii) Rename all upgrade scripts from upgrade_oldtemplate_xxxx.default to upgrade_newtemplate_xxxx.default

i) **File: TEM bundle files**
   i) Open the file <project>/install/bundle/oldtemplateBundle_en_US.xml
   ii) Search and replace all occurrences of **oldtemplate** with **newtemplate**
   iii) Rename the file "**oldtemplate**Bundle_en_US.xml " to "**newtemplate**Bundle_en_US.xml "
   iv) If you have any more TEM bundle files in other languages, then fix all these bundle files in the same manner.

   File sample after changes:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<IDMap>
    <TextID name="newtemplate.description" text="This template installs.."/>
</IDMap>
```

6) After you have fixed all files in the project, the next step is to rename your project folder
   a) In your File system, go to the folder <TC_ROOT>/bmide/workspace
   b) Rename the project folder name from "**oldtemplate"** to "**newtemplate**"
   c) After this step, your workspace would look like this -<TC_ROOT>/bmide/workspace/newtemplate

7) Import your renamed BMIDE project and validate
   a) Launch BMIDE
   b) Import **newtemplate** project using *File→Import→ Business Modeler IDE→Import a Business Modeler IDE Template Project*
      - This is the project that you fixed in the previous steps and would be located at <TC_ROOT>/bmide/workspace/newtemplate.
   c) After the project is imported, review the "Console View" and ensure there are no loading errors

8) Package your renamed template for distribution
   a) Go to Navigator view
   b) Select "newtemplate" project
   c) Select *File->New->Other->Package Template Extensions…* option to package the template

18.2 Updating Teamcenter installations where the renamed template is deployed

After you have renamed your "oldtemplate" to "newtemplate", you must update all Corporate Server environments where the "oldtemplate" was deployed. The "oldtemplate" could be deployed by you in your environments or it could be deployed by your partners with whom you have shared "oldtemplate". So the steps mentioned in this section must be applied by both you and your partners who have deployed the "oldtemplate".

The updating of Corporate Server environments involves three steps which are explain in detail in subsequent sections-

1) Update TEM related files in TC_ROOT
2) Update files in TC_DATA/model
3) Deploy the renamed template to the database

All these steps require system downtime. You have to bring down all Teamcenter servers before applying the above steps. Ensure that you apply these steps to all your environments. For example –

- Developer Environments
- Integration Test Environments
- Production Environment
- User Acceptance Testing Environments, etc.

18.2.1 Update TEM related files in your Corporate Server TC_ROOT

This section describes the steps to update the TEM related files in your Corporate Server <TC_ROOT> folder. Please note that if you have multiple <TC_ROOT> created, then you must apply these steps to fix all your <TC_ROOT> folders.

1) Before proceeding, you must obtain the template package for "**newtemplate**"

2) Keep a backup of the folder <TC_ROOT>/install

3) Delete the folder named "**oldtemplate**" from the directory <TC_ROOT>/install
   - The "oldtemplate" folder is not required any more. It is used as a staging folder for your template during TEM updates. We will be deploying your "newtemplate" shortly and hence the "oldtemplate" folder is not required. So you can safely delete it.

4) Update each of the TEM files listed below.

   a) **File: configuration.xml**
      i) Open the file <TC_ROOT>/install/configuration.xml
      ii) Search and replace all occurrences of **oldtemplate** with **newtemplate.** Please ensure you are performing a case sensitive and exact word match search and replace.

      File sample after changes:

```
<features>
...
...
   <installed feature="399B9CA4D818408AD040E33034580C1C"
        name="newtemplate/Data Model" />
    <spf feature="3FD330C0710D6D839C4F15B0868A5297" name="aldID_02" />
    <spf feature="E289526B3F58306609C4EB06BC9BEBB2" name="EDA Server and
        Rich Client Support" />
    <spf feature="74D6C1B94FAE0AE4158817B01ECB8AAD" name="newtemplate" />

...
...
</features>
```

   b) **File: uninstall.xml**
      i) Open the file <TC_ROOT>/install/uninstall.xml
      ii) Search and replace all occurrences of **oldtemplate** with **newtemplate.** Please ensure you are performing a case sensitive and exact word match search and replace.

      File sample after changes:

```
...
...

    <zip name="tc/newtemplate_template.zip" content="6283451792031056.txt">
      <feature code="399B9CA4D818408AD040E33034580C1C" config="tc0309" />
    </zip>
    <zip name="tc/newtemplate_install.zip" content="7707878700501186.txt">
      <feature code="399B9CA4D818408AD040E33034580C1C" config="tc0309" />
```

```
    </zip>
...
...
```

c) **File: feature_oldtemplate.xml**
   i) Delete the file named feature_oldtemplate.xml from the directory <TC_ROOT>/install/install/modules
   ii) Copy the file named feature_newtemplate.xml from your "newtemplate" template package to <TC_ROOT>/install/install/modules

d) **File: TEM bundle files**
   i) Delete the TEM bundles files related to your oldtemplate and copy the new bundle files from the newtemplate package.
   ii) For example,
      - Delete the file named "oldtemplateBundle_en_US.xml" from <TC_ROOT>/install/install/lang/en_US
      - Copy the file named "newtemplateBundle_en_US.xml" from your "newtemplate" template package to <TC_ROOT>/install/install/lang/en_US
   iii) If you have provided TEM bundle file translations in other languages, then you must perform similar changes to the remaining TEM bundle files. All these TEM bundle file translations are under <TC_ROOT>/install/install/lang folder.

18.2.2 Update files in TC_DATA/model

After the <TC_ROOT> folders are updated, the next step is to update the files in <TC_DATA>/model folder. Please note that if you have multiple <TC_DATA> created for your database then you must apply these steps to update all <TC_DATA> folders.

1) Update each of the files listed below. In these files, you will do a search and replace of "oldtempalate" with "newtemplate". Ensure that you are using a case sensitive and exact word match in your search and replace process.

   a) **File: client_oldtemplate.properties**
      i) Open the file <TC_DATA>/model/client_oldtemplate.properties, if it exists.
      ii) Search and replace all occurrences of **oldtemplate** with **newtemplate**
      iii) Rename client_oldtemplate.properties as client_newtemplate.properties

      File sample after changes:

```
#Business Modeler Client Properties
#Wed Mar 30 08:46:12 BST 2011
FullDeployCS=3B8550CF8913E7BEF8EE147FF005B803
client_id=Config5692VM0_newtemplate_56C0C7A8A783F6D153CF6E22E9CE733A
```

   b) **File: FileVersions.txt**
      i) Open the file <TC_DATA>/model/FileVersions.txt, if it exists.
      ii) Search and replace all occurrences of **oldtemplate** with **newtemplate**

      File sample after changes:

```
...
...
file=newtemplate_dependency.xml version=1
file=newtemplate_template.xml version=1
file=newtemplate_template_en_US.xml version=1
...
...
```

   c) **File: master.xml**
      i) Open the file <TC_DATA>/model/master.xml

ii) Search and replace all occurrences of **oldtemplate** with **newtemplate**

File sample after changes:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<TcBusinessDataIncludes>
    <include file="foundation_template.xml"/>
    ...
    <include file="newtemplate_template.xml"/>
    ...
</TcBusinessDataIncludes>
```

d) **File: oldtemplate_dependency.xml**
   i) Delete the file "oldtemplate_dependency.xml" from <TC_DATA>/model folder
   ii) Unzip "newtemplate_template.zip" and copy the file "newtemplate_dependency.xml" to <TC_DATA>/model

e) **File: oldtemplate_template.xml**
   i) Delete the file "oldtemplate_template.xml" from <TC_DATA>/model folder
   ii) Unzip "newtemplate_template.zip" and copy the file "newtemplate_template.xml" to <TC_DATA>/model

f) **File: Language related to oldtemplate**
   i) This step is required only if you are working with Teamcenter 8.3 or a later release.
   ii) Delete all language files related to oldtemplate from the folder <TC_DATA>/model/lang
   iii) Unzip "newtemplate_template.zip" and copy the language files for "newtemplate" into <TC_DATA>/model/lang folder.
   iv) For example, delete the file "oldtemplate_template_en_US.xml" from <TC_DATA>/model folder and copy newtemplate_template_en_US.xml from the package "newtemplate_template.zip"

2) Cleanup **oldtemplate** files from datasets
   - This step is required only if you are working with Teamcenter 8.3 or a later release
   - Starting at Teamcenter 8.3.0, files in <TC_DATA>/model folder are also stored in the database as datasets. The datasets that store these files are maintained in a folder named **Fnd0BMIDEResource.**This means the database contains a copy of your "oldtemplate" files in datasets and they must be removed from the database.
   - The below steps will help you clean up the "oldtemplate" related files from datasets.

   a) Launch Teamcenter Rich Client and query the folder of type **Fnd0BMIDEResource.** See Figure 68 for the results of the search.

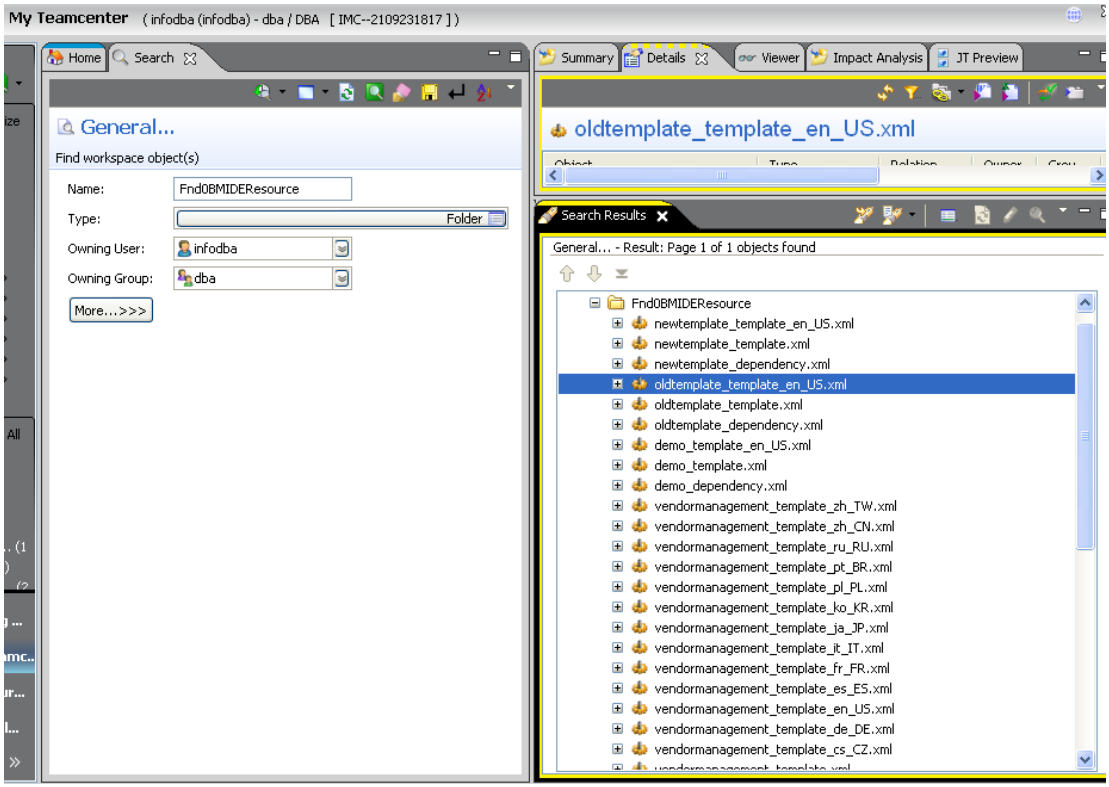


Figure 68 Results from query for folder contents

b) From the search results, delete all files related to oldtemplate as listed below:

- oldtemplate_template.xml
- oldtemplate_dependency.xml
- oldtemplate_template_en_US.xml. You will have to do the same for all language files that is applicable to your “oldtemplate”.

3) Upload all files in <TC_DATA>/model to the database
   - This step is required only if you are working in Teamcenter 8.3 or a later release
   - After the files related to “oldtemplate” have been removed from datasets. We must upload the “newtemplate” files to datasets. This will ensure that the <TC_DATA>/model and datasets are in sync.
   - The below steps will help you upload files to datasets.

   a) To upload “newtemplate” files, run the following command -
   `manage_model_files -u=<user name> -p=<user password> -g=dba -option=upload`

4) Rename the template name in <TC_DATA>/install/models.xml
   a. This step is required only if you are working in Teamcenter 9.1 or a later release
   b. Update the entries of “oldtemplate” to “newtemplate”

18.2.3 Deploy the renamed template to the database

After updating the <TC_ROOT> and <TC_DATA>/model folders, the next step is to deploy the renamed template “**newtemplate”** to the database. This ensures that the renaming process is successful and your BMIDE project and

database are in sync. You must deploy the template in all environments where you updated the TC_ROOT and TC_DATA/model folders.

1) Before updating the database with your renamed template, you have to first unregister your "**oldtemplate**" and register your "**newtemplate**" with the database and TEM. To do this, run the below commands-

   a) Unregister oldtemplate
   ```
   bmide_manage_templates -u=<user name> -p=<user password> -g=dba -
   option=remove -templates=oldtemplate
   ```

   b) Register newtempalte
   ```
   bmide_manage_templates -u=<user name> -p=<user password> -g=dba -option=add -
   templates=newtemplate
   ```

2) Update the database with your "newtemplate" package that was created in BMIDE
   a) Launch TEM in maintenance mode
   b) Select the option *Teamcenter Foundation→Update database(Full Model – System downtime required).* See Figure 69.

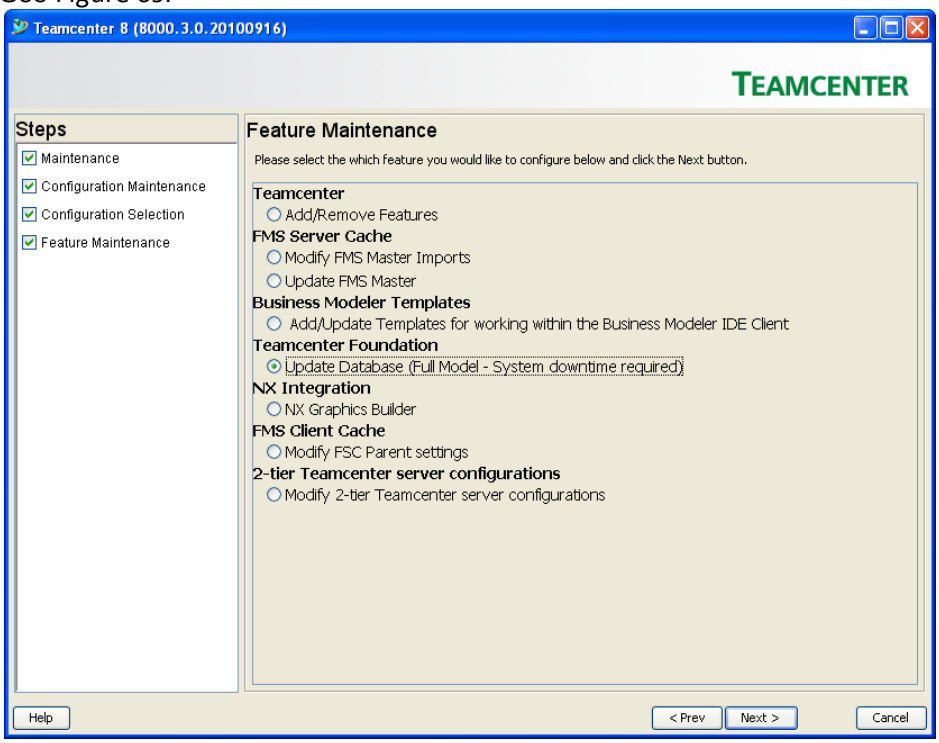


Figure 69 TEM panel with template update option

   c) When TEM prompts for the template package, you must browse and provide the "newtemplate" package. This is the template package you generated in BMIDE after renaming and importing your project. See Figure 70 showing the selected newtemplate.

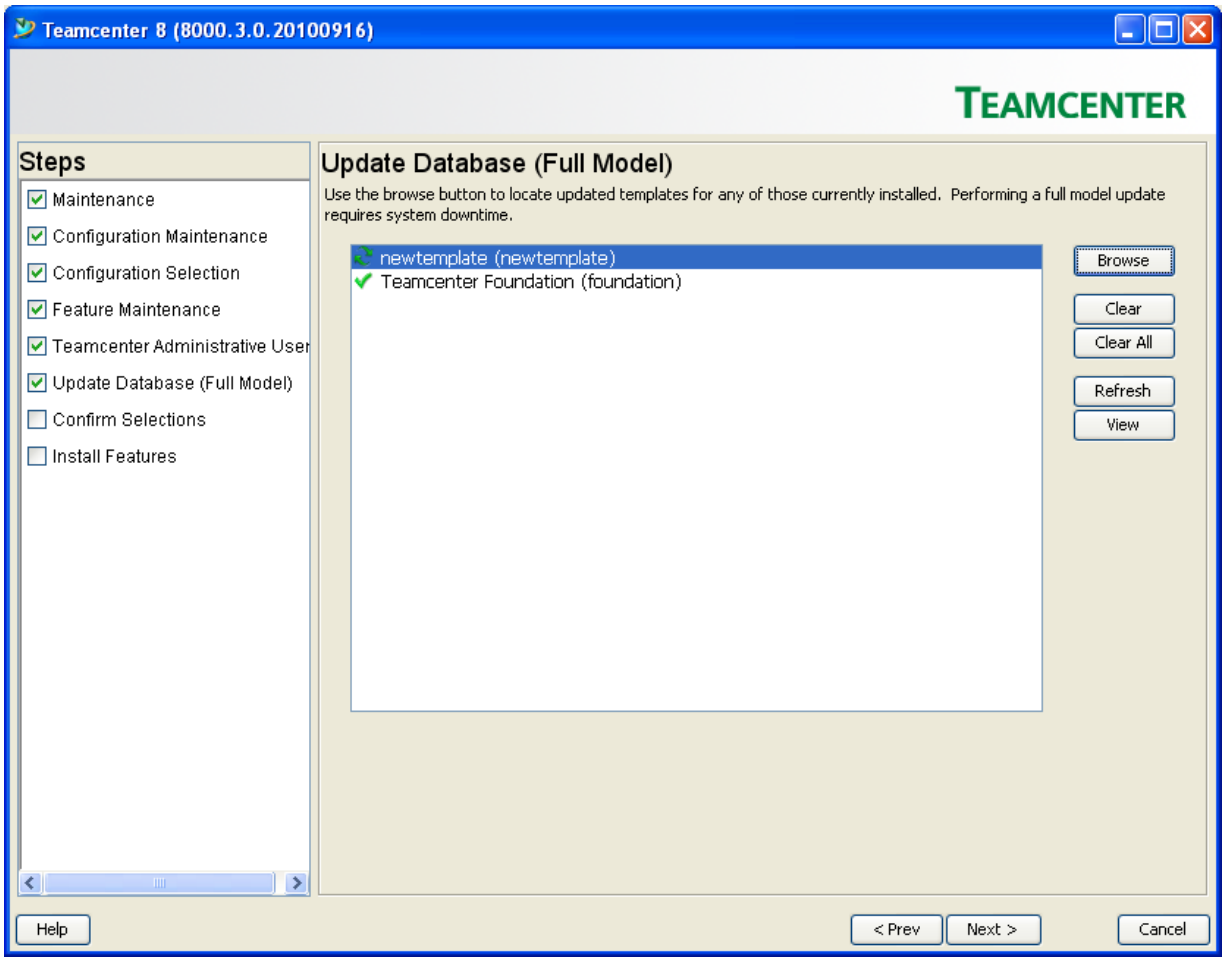


Figure 70 TEM panel where newtemplate is selected for update

d) Go through the rest of the TEM panels and update the database

NOTE: The TEM options shown here is from Teamcenter 8.3.0. Please ensure you are selecting the correct option in TEM to update your database in releases older than Teamcenter 8.3.0. For more details on how to update a template, please read the topic *Adding features→Installing a custom solution or third-party template→Update the database using TEM* in the Teamcenter installation guide.

At this point you have completed the renaming process of your “oldtemplate”. You can continue working on your “newtemplate”.

However, if you had shared the “oldtemplate” with your partners then you must request them to apply the renaming process described in section titled “Updating Teamcenter installations where the renamed template is deployed”.

If you have other templates dependent on “oldtemplate” that whether you or your partner owns then apply the steps described in the section titled “Update other templates that are dependent on the renamed template”.

18.3 Update other templates that are dependent on the renamed template

This section describes the steps to update templates that were dependent on your “oldtemplate”. These could be templates you own or your partners. Since the “oldtemplate” is renamed to “newtemplate” you must now update all templates that were dependent on “oldtemplate” to be now dependent on “newtemplate”.

Before proceeding with the steps described below, you must first get the template package for “newtemplate”. This is the package of the renamed template.

While describing the dependency change steps, the following conventions are used:

- **partner**
  - This is the name of the template in which the dependency must be changed from “oldtemplate” to “newtemplate”. It could be a template you own or your partner owns.

18.3.1 Update the dependency in your “partner” template project

In this section we will update the BMIDE template project that has a dependency on “oldtemplate”.

1) Ensure the “**partner”** template project in which the dependency has to be changed is in sync with the database. This means if you have any new data model changes in your template that is not deployed to the database, ensure you deploy them before the renaming process. After fixing the dependency on “oldtemplate”, the final step would be to deploy your template to the database. So if you include deployment of new data model and changing the dependency of your template as a single process and say deployment fails, then it adds to the complexity of debugging such failed deployments.

2) Close your project in BMIDE before changing its dependency
   a) Go the Navigator view
   b) Select “partner” project, RMB select “Close Project”
   c) This will close your project in BMIDE

3) Shutdown BMIDE

4) Install the “newtemplate” package for usage in your BMIDE client
   a) Go to <TC_ROOT> where your BMIDE Client is installed
   b) Launch TEM in maintenance mode
   c) Use the TEM option *“Add/Update templates for working within the Business Modeler IDE Client”* to add the “newtemplate”. See Figure 71.

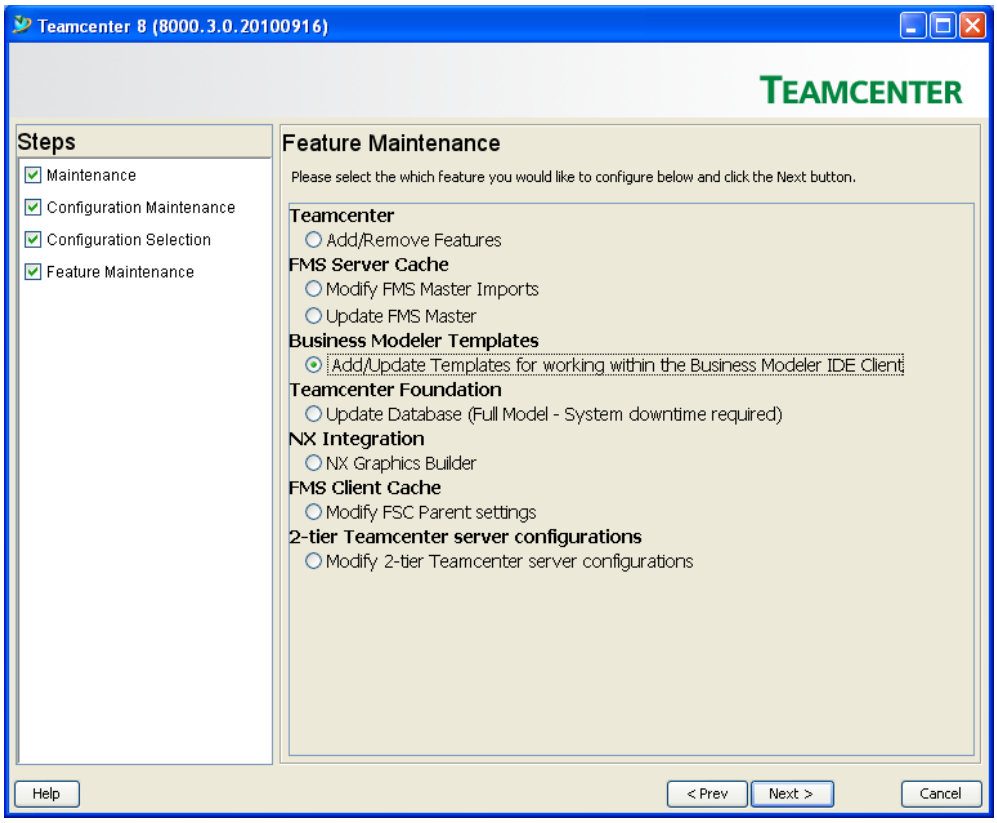


Figure 71 TEM option to update templates for usage in BMIDE

d) When TEM asks for the template package, browse and provide the "newtemplate" package which contains the renamed template.
   - NOTE: The TEM options shown here is from Teamcenter 8.3.0. Please ensure you are selecting the correct option of TEM in releases older than Teamcenter 8.3.0. For more details on adding a template for usage within BMIDE, see "*Add a template to a Business Modeler IDE Template project*" in the Business Modeler IDE User Guide.

e) Once TEM completes, verify that the newtemplate_template.xml and newtemplate_dependency.xml resides in <TC_ROOT>/bmide/templates folder.

5) Remove "oldtemplate" from <TC_ROOT>/bmide/templates
   - You are not expected to use "oldtemplate" anymore. Hence we should remove all references to this in <TC_ROOT>/bmide/templates. Since there is not UI support to remove a template, you must follow the below manual steps.

a) Go to <TC_ROOT>/bmide/templates and delete the following files
   i) oldtemplate_template.xml
   ii) oldtemplate_dependency.xml
   iii) Go to <TC_ROOT>/bmide/templates
   iv) Delete the following files from this folder

b) Go to <TC_ROOT>/bmide/templates/lang and delete the following files
   i) oldtemplate_template_en_US.xml
   ii) Delete all other language files related to "oldtemplate"

c) Remove "oldtemplate" entry from master.xml
   i) Open <TC_ROOT>/bmide/templates/master.xml

ii) Remove the entry for “oldtemplate”

6) Add dependency on “newtemplate” in the “partner” template project
   a) Update the **dependency.xml** file for “partner” template
      i) Go to the folder <TC_ROOT>/bmide/workspace/<partner>/extensions in your file system
      ii) Open the file dependency.xml located in this directory
      iii) Search and replace all occurrences of **oldtemplate** with **newtemplate**

File sample after changes:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<TcBusinessDataIncludes ActiveRelease="tc8000.3.0" displayName="Partner Template"
enableOpsDataDeploy="false" guid="ABAAF6855A497F72D251423600482831" name="partner"
optional="true" prefixes="F3" teamcenterTemplate="false">
    <include file="foundation_template.xml"/>
...
    <include file="newtemplate_template.xml"/>
...
</TcBusinessDataIncludes>
```

   b) Update the **feature_partner.xml** file for “partner” template
      i) Go to the folder <TC_ROOT>/bmide/workspace/<partner>/install in your file system
      ii) Open the file feature_partner.xml located in this directory
      iii) Search and replace all occurrences of **oldtemplate** with **newtemplate**

File sample after changes:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!--
    Document   : feature_partner.xml
    Description: This XML is used by TEM to install or upgrade the "partner"
solution.
-->
<feature>
    <name value="Partner Planning division"/>
    <property name="feature_name" value="partner"/>
    <group value="package"/>
    <guid value="ED0936452CC064BCE17D103FBEB1D5BB"/>
    <bundle value="${feature_name}Bundle.xml"/>
    <description value="${feature_name}.description"/>
    <include file="dataModelDependency.xml"/>
...
    <relation>
        <depends name="newtemplate" value="2BDE508357268235F1FF32C6C1E0DCFA"/>
    </relation>
...
</feature>
```

7) Load the “partner” template whose dependency was changed in BMIDE and validate there are no loader errors.
   a) Launch BMIDE
   b) Select “partner” project, RMB select “Open Project”
   c) This will open your project in BMIDE and load it
   d) Review the “Console” view and ensure there are no loading errors

8) Package the “partner” template for deployment
   a) Go to Navigator view
   b) Select “partner” project
   c) Select *File->New->Other->Package Template Extensions…* option to package the template

18.3.2 Update TC_ROOT where the "partner" template is deployed

If your "partner" template is installed in your database, then you must update the <TC_ROOT> folders in all your environments to rename all files related to "oldtemplate". In this step you will not manipulate any files related to "partner" template. You are only going to apply the process to rename "oldtemplate" to "newtemplate" in your TC_ROOT.

To update the TC_ROOT files, you can follow the steps outlined in section titled "Update TEM related files in your Corporate Server TC_ROOT".

Updating the TC_ROOT folders require system downtime. Hence you must bring down all Teamcenter servers. Please ensure that you update the TC_ROOT in all these environments-

- Developer Environments
- Integration Test Environments
- Production Environments
- User Acceptance Testing Environments, etc.

18.3.3 Update TC_DATA/model where the "partner" template is deployed

If your "partner" template is installed in your database, then you must update the <TC_DATA> folders in all your environments to rename all files related to "oldtemplate". In this step you will not manipulate any files related to "partner" template. You are only going to apply the process to rename "oldtemplate" to "newtemplate" in your TC_ROOT.

To update the <TC_DATA>/model files, you can follow the steps outlined in section titled "Update files in TC_DATA/model".

Updating the TC_DATA/model folders require system downtime. Hence you must bring down all Teamcenter servers. Please ensure that you update the TC_DATA/model in all these environments-

- Developer Environments
- Integration Test Environments
- Production Environments
- User Acceptance Testing Environments, etc.

18.3.4 Deploy the "newtemplate" and "partner" templates to database

Once the TC_ROOT and TC_DATA/model folders are updated to have reference to the "newtemplate", you must deploy both the "newtemplate" and "partner" template to the database. This ensures that the change of dependency from "oldtemplate" to newtemplate" is successful and your "partner" project in BMIDE and database are in sync.

1) Before updating the database, you have to first unregister your "oldtemplate" and register your "newtemplate" with the database and TEM. To do this, please run the below commands on command line:
   a) Unregister oldtemplate
   ```
   bmide_manage_templates -u=<user name> -p=<user password> -g=dba -
   option=remove -templates=oldtemplate
   ```

   b) Register newtemplate
   ```
   bmide_manage_templates -u=<user name> -p=<user password> -g=dba -option=add -
   templates=newtemplate
   ```

2) Obtain the “newtemplate” template package.

3) Obtain the “partner’ template package. This is the template whose dependency was changed in BMIDE from “oldtemplate” to “newtemplate”.

4) Update the database using TEM
   a) Launch TEM in maintenance mode
   b) Select the option *Teamcenter Foundation→Update database(Full Model – System downtime required).* *See* Figure 72*.*

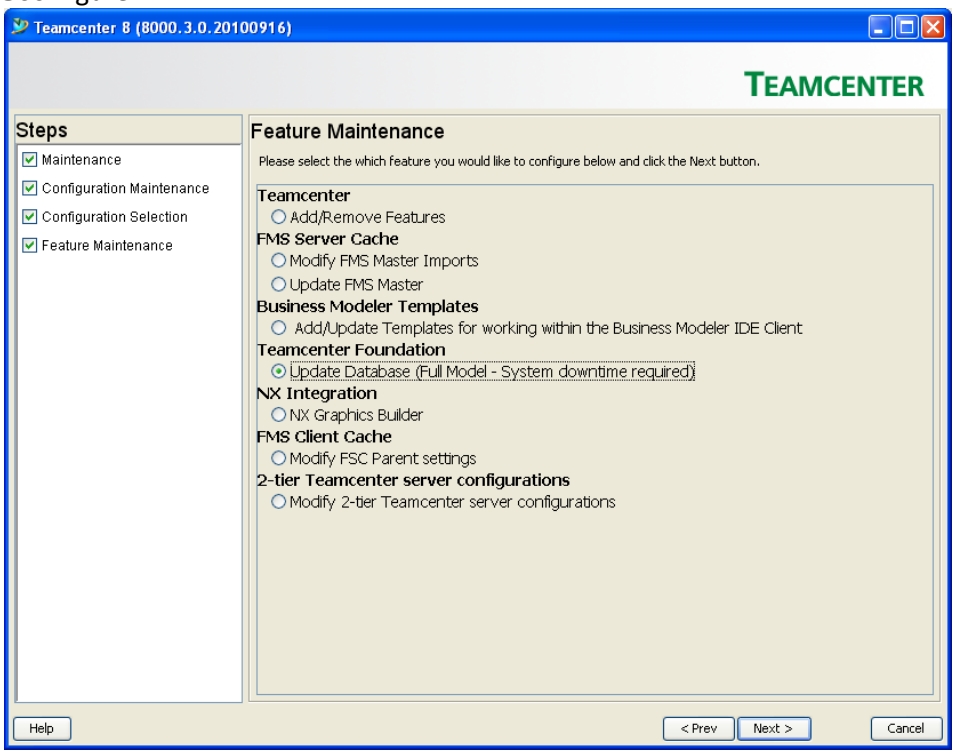


Figure 72 TEM option to update template in database

   c) When TEM prompts for the template package, you must browse and provide both the “newtemplate” and “partner” template packages. See Figure 73 showing the selected “newtemplate” and “partner” template for database udpate.

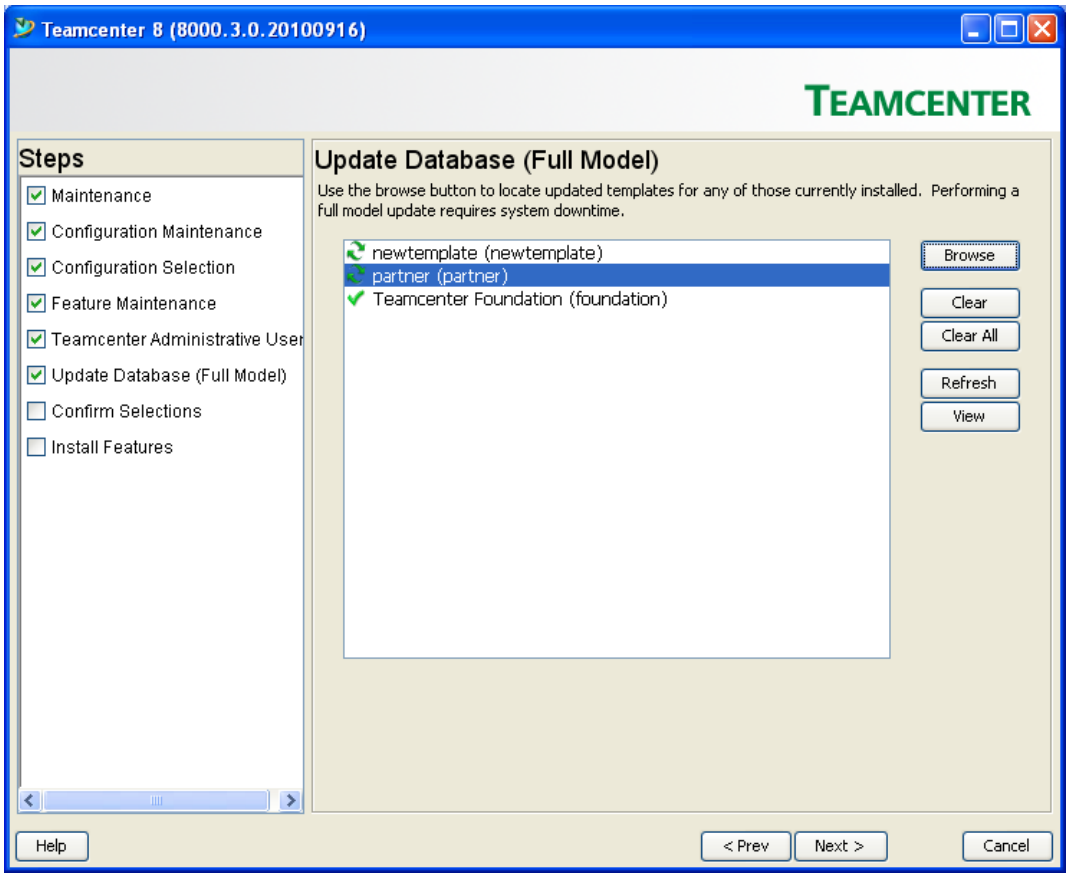


Figure 73 TEM panel showing the selected templates for database update

d) Go through the rest of the TEM panels and update the database.

NOTE: The TEM options shown here is from Teamcenter 8.3.0. Please ensure you are selecting the correct option of TEM in releases older than Teamcenter 8.3.0. For more details on how to update a template, please read the topic *Adding features→Installing a custom solution or third-party template→Update the database using TEM* in the Teamcenter installation guide.

At this point you have completed the renaming process of “oldtemplate” and fixing all templates that either you or your partner own and were dependent on “oldtemplate”. You can now continue to work on your “partner” template as before.

19 How to Use Custom Operation in a Condition Expression

The objective of this section is to provide an interim solution to make condition expression extensible so that a customer is able to use in a condition expression their own custom operation defined on a COTS business object. This solution consists of the following three steps:

1. **Create a Custom Persistent Business Object and Implement Custom Operation**
   For example, the customer can define a custom business object under POM_object or WorkspaceObject. The user can then implement a custom operation, e.g. checkObjects (tag_t objTag, bool* result).

2. **Add a Custom Runtime Property to UserSession**
   This custom runtime property is implemented so as to return an instance of the custom Business Object. In this example, the custom property name is “MyCustomObject”

3. **Define a Condition Expression using Custom Property of UserSession**
   Using the custom runtime property, the customer is able to use their own custom operation in a condition expression in the following manner:

   //ConditionName (Signature) → Expression

   isConditionExampleA(WorkspaceObject o, UserSession u) →

   u.**MyCustomObject**.**checkObjects** (o) = false

The details of each step are described as follows.

19.1 Create a Custom Persistent Business Object and Implement Custom Operation

Suppose the customer needs to define a custom operation on a COTS business object and use the custom operation in a condition expression. Currently, this use case is not supported. To overcome this limitation, the best alternative (without incurring overburden on the customer) is that the customer can create their own custom business object and define custom operations on the custom business object. The customer is then able to use the custom operation of their own custom business object in a condition expression.

In the following, let’s use the aforementioned custom operation: checkObjects example to explain this alternative solution. The summary of the use case example is as follows:

Ability to implement an operation: checkObjects on a COTS business object, e.g. WorkspaceObject. This operation returns a boolean as the check result.

Ability to use this operation in a condition expression, e.g.

//ConditionName (Signature) → Expression

isConditionExample(WorkspaceObject o) →

o.**checkObjects** ( ) = false

Currently, BMIDE does not allow the above custom operation to be implemented **directly** on the COTS WorkspaceObject. The customer can use the following alternative solution:

1. Define a custom business object e.g. CustomBusinessObject under POM_object or WorkspaceObject.
2. Add a custom operation to CustomBusinessObject and implement its base function in CustomBusinessObjectImpl class
   The signature of this custom operation is as follows:
   int checkObjects(tag_t objTag, bool* result)

Notice this operation takes the tag of the target object as input. The business logic codes will be written against the input target object as illustrated in the following code snippet:

```
int  CustomBusinessObjectImpl::checkObjectsBase( tag_t objTag, bool* result )
{

        //Use Teamcenter ITK and C++ APIs to access data of the target object identified by “objTag”

        //Implement business logic code to check the data of the target object

        //set *result to true or false depending on the validation result.

}
```

It should be noted the stub for the Impl class is generated the first time when the code generation is invoked. The details of implementing a custom operation and code generation can be found in **Business Modeler IDE Guide ->Using the Business Modeler IDE for codeful customization**.

Also note an operation that can be used in a condition expression must satisfy certain criteria. For example, the operation must return an int and must have exactly one output that is the last parameter in the operation signature. Please refer to **Business Modeler IDE Guide -> Business Modeler IDE Reference -> Rules reference -> Conditions reference -> Condition system -> Condition operation rules** for details.

19.2 Add a Custom Runtime Property to UserSession

Given a custom operation defined on a custom business object, the customer could use this custom business object and its custom operation directly in defining a condition expression as shown below. But, where is the target object input? That is, the actual use case is to define a condition function against the target object of WorkspaceObject instead of the custom business object.

```
//ConditionName ( Signature ) → Expression

isConditionExample(CustomBusinessObject  o) →

    o.checkObjects ( ) =  false
```

In this alternative solution, the custom business object is used as a gateway for a condition expression to access the custom operation through UserSession object. For this purpose, the customer needs to add a custom runtime property to the UserSession business object. The sole purpose of this custom runtime property is to create and return a singleton instance of the custom business object. The storage type of this property is TypedReference to the custom business object.

19.2.1 Define Getter Operation for Custom Runtime Property

Currently, BMIDE UI does not allow the customer to define the getter operation for a custom property defined on a COTS business object. To overcome this limitation, the customer can manually add the following BMIDE elements into their template to define the getter operation for their custom property.

In this example, the custom property name is “MyCustomObject”. This property is a single-value reference property. Therefore, the operation: PROP_ask_value_tag is used.

- TcOperationAttach to attach the getter operation: PROP_ask_value_tag to the property: MyCustomObject at UserSession

```
<TcOperationAttach operationName="PROP_ask_value_tag" extendableElementName="UserSession"
    extendableElementType="Type" propertyName="MyCustomObject"    description="">
  <TcExtensionPoint extensionPointType="PreCondition" isOverridable="true"/>
  <TcExtensionPoint extensionPointType="PreAction" isOverridable="true"/>
  <TcExtensionPoint extensionPointType="BaseAction" isOverridable="false"/>
  <TcExtensionPoint extensionPointType="PostAction" isOverridable="true"/>
```

</TcOperationAttach>

- TcExtension to define an extern function for the property: MyCustomObject. The name of the extern function is “getMyCustomObjectBase”, which must be unique.

  <TcExtension name="**getMyCustomObjectBase**" internal="false"

  cannedExtension="false" languageType="CPlusPlus" description="">

  <TcExtensionValidity parameter="PROPERTY:UserSession:**MyCustomObject**:PROP_ask_value_tag:4"/>

  </TcExtension>

- TcExtensionAttach to attach the extern function as BaseAction of the getter operation

  <TcExtensionAttach extensionName="**getMyCustomObjectBase**"

  operationName="PROP_ask_value_tag" isActive="true" propertyName="**MyCustomObject**"

  extendableElementName="UserSession" extendableElementType="Type" extensionPointType="BaseAction"

  conditionName="isTrue" description=""/>

19.2.2 Implement Getter Base Function for Custom Runtime Property

The getter base function for a property operation is declared in the following manner to avoid C++ name mangling. In this example, the library name is “MyLib”

```
#ifdef cplusplus
extern “C”
{
#endif
extern MYLIB_API int getMyCustomObjectBase(METHOD_message_t *, va_list args);
#ifdef cplusplus
}
#endif
```

The following is an example code snippet for the base function: “getMyCustomObjectBase”, which is expected to return the tag of the singleton instance of the custom business object. Please note the instance is created but not saved so that this custom business object is actually used as a “runtime” object, thus causing no impact on database data. Also note the tag of the instance is cached in a static variable. Using the static variable to cache the tag is to make sure only one instance is created.

```
int  getMyCustomObjectBase(  METHOD_message_t *, va_list args )
{
        va_list largs;
        va_copy( largs, args );
        va_arg( largs, tag_t ); /*Property Object tag_t  not used*/
        tag_t*  customObjTag     = va_arg( largs, tag_t* );
        va_end( largs );
```

```
    int ifail = ITK_ok;
    *customObjTag = NULLTAG;

    //Create and cache an instance of my custom business object
    static tag_t myCustomObjectTag = NULLTAG;
    if( myCustomObjectTag == NULLTAG )
    {
        static const char *  customBOName  = "MyCustomBOName";
        Teamcenter::CreateInput* creInput =
        dynamic_cast<Teamcenter::CreateInput*>(Teamcenter::BusinessObjectRegistry::instance().createInput
        tObject(customBOName, OPERATIONINPUT_CREATE));

        Teamcenter::BusinessObject*  obj  =
        Teamcenter::BusinessObjectRegistry::instance().createBusinessObject(creInput);

        myCustomObjectTag = obj->getTag();
    }
    *customObjTag = myCustomObjectTag;
    return ifail;
}
```

19.3 Define a Condition Expression using Custom Property of UserSession

Following the above steps, the customer has done the following:

1. Define a custom business object: CustomBusinessObject and a custom operation in BMIDE
   The custom operation signature follows the Condition operation rules defined in BMIDE IDE Guide.
   int checkObjects (tag_t objTag, bool* result)

2. Implement the base function for the custom operation in CustomBusinessObjectImpl.cxx.

3. Add a custom property: MyCustomObject to UserSession in BMIDE
4. Define BMIDE template data for the getter operation.
5. Implement the getter base function for the custom property that returns a singleton instance of CustomBusinessObject.

Suppose the condition signature is: isConditionExampleA(WorkspaceObject o, UserSession u). At this very last step, the customer is now able to start with UserSession argument "u" to build a condition expression using the custom operation "checkObjects". In "Expression:" field of "New Condition…" dialog,

- Enter u.
  Then, ContentAssist will automatically popup to display applicable properties and operations of UserSession to choose from or press Ctrl + space bar

- Select “MyCustomObject” property from the ContentAssist selection list
  u.MyCustomObject

- Enter . after MyCustomObject, i.e. u.MyCustomObject.
  Then, ContentAssist will automatically popup to display applicable properties and operations of CustomBusinessObject to choose from or press Ctrl + space bar

- Select “checkObjects” operation from the ContentAssist selection list and enter WorkspaceObject “o” argument as the input target object to “checkObjects” operation
  u.MyCustomObject.checkObjects(**o**) = false

  //ConditionName (Signature) → Expression

  isConditionExampleA(WorkspaceObject o, UserSession u) →

  u.**MyCustomObject**.**checkObjects** (o) = false

20 How to remove a template

The TEM wizard provides the ability to install BMIDE templates into a Teamcenter system. Currently in Teamcenter, there is no TEM support to remove a template deployed in the database. However a template can be removed manually from a Teamcenter installation.

This section describes how a template can be removed manually from a TcUA installation.

20.1 Who should read this section?

You should read this section if you are a Teamcenter Administrator and plan to remove a template from the Teamcenter installation.

Note: The user who is going to perform this task of removing template should have basic understanding of Teamcenter Administration, BMIDE templates, and deleting objects and references in Teamcenter.

20.2 Before removing a template from Teamcenter

Before proceeding with removing a template you must take a note of the following points

- You can remove any BMIDE template except the foundation template
- Instance data are runtime instances of the data model business objects definitions defined in the templates. These instances are stored in the Teamcenter database and rely on the business object definition to also be defined in the database. Teamcenter cannot function with instances of business object without the definitions also being present in the system. Likewise before a business object definition can be removed, the business object instances must be deleted. Hence you must first delete all instances of each definition in the template which is to be removed. If you cannot or do not want to delete the instances, you cannot remove the template. Refer section 22 for the steps involved in removing the business object instances.
- As per Teamcenter data model architecture, data model elements defined in any template can be inherited by creating sub-type data model definitions in a BMIDE template. Hence instance data of these sub-type data model definitions also needs to be deleted from the database just like actual parent type instance data. If you cannot or do not want to delete the instances, you cannot remove the template
- Only the data model defined in the template is removed through this process. The non-data model elements (Workflow templates, organization, libraries etc) added through the BMIDE template's install.default or upgrade.default scripts will not be removed by this documented process. If you desired to remove any other associated non-BMIDE type objects, you must perform this as well
- The runtime server libraries that are generated, built, and packaged through the BMIDE and deployed through TEM are not removed through this process. If you have libraries that are associated with the template you plan to delete, you must also remove these libraries from the Teamcenter installation.
- Before removing the template, it is strongly recommended that you back up your existing Teamcenter Data. This is recommended in case you encounter difficulty during the removal process and have to restore the data in the system to its former state. For more information, refer to the *Backup existing Teamcenter Data* section from *Upgrade Guide.* Keep a back up of following:
    - The Teamcenter application root directory on each installed workstation. (TC_ROOT)
    - The Teamcenter data directory for each configured database (TC_DATA)
    - The Teamcenter volume directories for each configured database
- Before removing the template it is also strongly recommended that you back up your existing Teamcenter Database. For more information, Refer to the *Export Oracle Databases* section from *Upgrade Guide*

If your business needs require you to remove the template from then you must exercise caution while removing your template. Here is the list of cautionary items-

- There is no automated support to remove a template. It is a manual process prior to Teamcenter 11.2.3 and is a semi-automated process from Teamcenter 11.2.3.
- If you miss executing a step in the template removal process, it may result in future deploy/upgrade future.
- Removing a template will require system downtime.
- After you remove the template dependency in BMIDE, you must update the same in all your Developer Environments, Integration Test Environments, Production Environments, User Acceptance Testing Environments, etc.
- You must also update other templates that you own and are dependent on the template being removed
- If your partners have created additional templates that are dependent on your template to be removed, then you must inform them to update all such templates to remove dependency on your template.
- If you have shared your template with partners then do not include new data model new data model in your template for which you removed the dependency on another template. Ensure that the data model definitions of your template matches with the one that was previously distributed to your partners, except the necessary changes to resolve the data model loading errors displayed in the console log. It is not wise to distribute a template that is both template dependency removal and has new data model changes. Since template removal is a manual process, including new data model changes in a template with dependencies removed increases the debugging complexity if we encounter issue with the deploying your template that has template dependency removal.
- If you have shared your template with partners, then your template with dependencies removed must be in the same release as that of your partner. For example, if you shared your template belonging to Teamcenter 8.3 with your partner, then removing of your template dependency must also be done in the same Teamcenter 8.3 release. You cannot remove your template dependency in a release that is newer to Teamcenter 8.3 or older to Teamcenter 8.3.

20.3 Removing a template that you own

Below are the instructions to remove a template owned by you named as "**customer**" from a Teamcenter Unified installation. Note that the steps to remove a template owned by you vary from the steps to remove a custom template that you own someone else, as the template project's source code for that template will available to you.

Note: If you intend to remove a template from Teamcenter 11.2.3 and onwards, refer the instructions in section 20.5. Refer sections 20.3 and 20.4 for an earlier release.

Assumptions:

- There is no instance data in the database for the data model defined in the template to be removed
- If there are any other templates that are dependent upon this template, they must be removed from the database first before this template can be removed.
- The contents of model directory are in sync with the database.
    - This means if you have any new data model changes in your template that is not yet deployed to the database, ensure you deploy them before removing template dependencies. If you include deployment of new data model and removing of template dependency as a single process and say deployment fails, then it adds to the complexity of debugging such failed deployments.
    - But if you have shared your template with your partner then do not add new data model to your template after removing the template dependency. Ensure the data model matches with the one that was previously distributed to your partners.

Note that the instructions below will assume that the template being removed is called “customer”. If you are removing a template with another name please substitute the name of your template everywhere you see the word “customer” in the following directions.

There are two possible scenarios of removing a template that you own:

A. No template is dependent on the template you want to remove

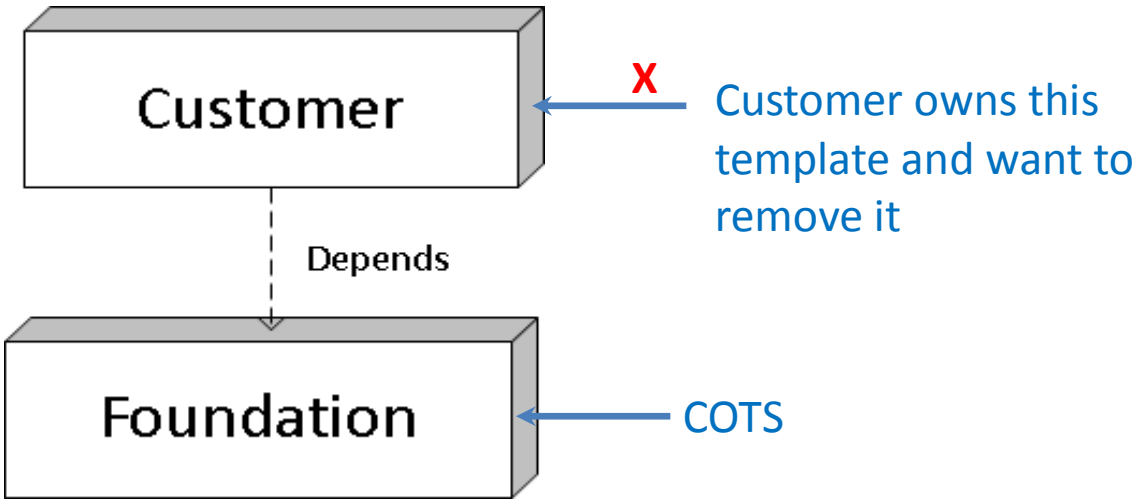


Figure 74 - No dependent templates

If none of the templates are dependent on the template to removed, you can proceed with the steps detailed below starting from Section 20.3.1.

B. Other templates are dependent on the template you want to remove

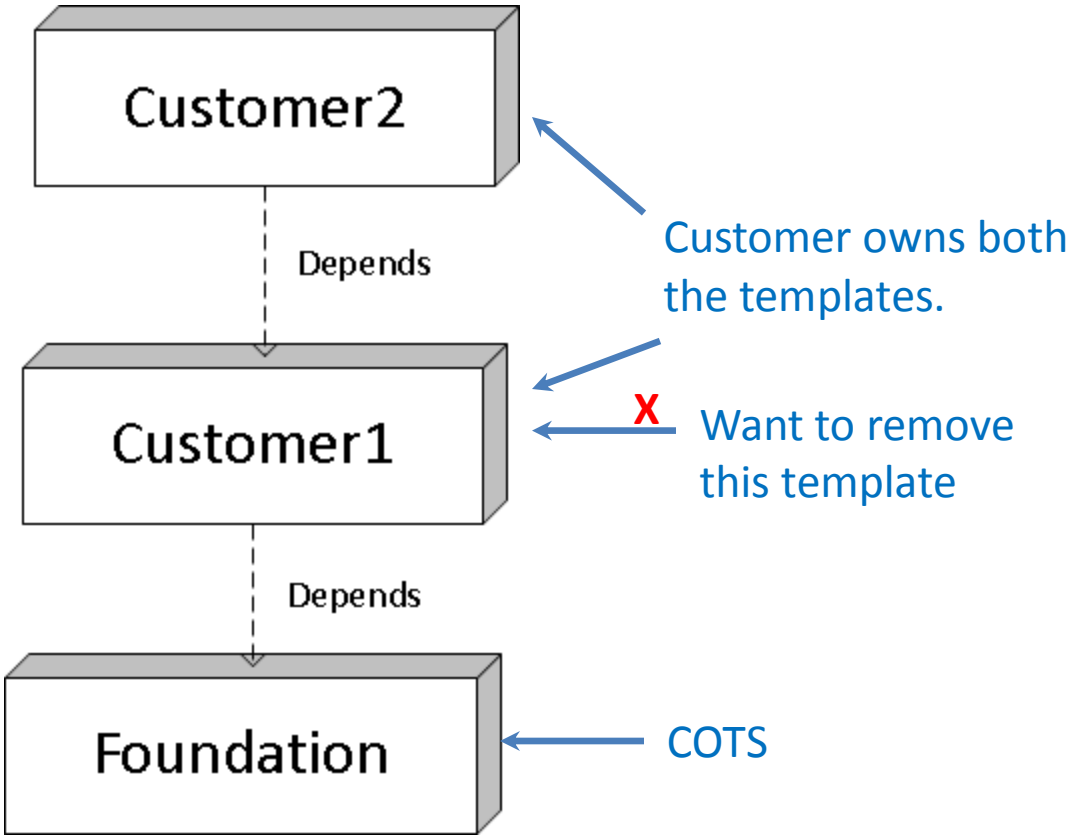


Figure 75 - dependent templates

If you have a custom template that is depended on this template, you will have to remove the dependency before removing this template. Execute the steps detailed in section 21 before executing the steps detailed in this section. After removing the dependency, you can proceed with the steps detailed below starting from Section 20.4.1.

20.3.1 Delete the data model definitions from the template to be removed in BMIDE

The first step in the template removal process is to delete the data model definitions from the template project in BMIDE and ensure it loads successfully within BMIDE. This section describes the steps to delete the data model definitions from the custom template to be removed and the database. Since this is a custom template, you would have access to the template project's source code which is required to perform these steps.

Following are steps to delete the data model definitions:

1) Keep a backup copy of your **customer** project. In case you run into issues with removing your template, we can always get back to your old state of the template.

2) Launch BMIDE and remove the data model definitions from the extension files.
   a. Go to the Navigator view, expand the <Project>/extensions to view the source files.
   b. Open each XML source file (except master.xml and dependency.xml) and remove the XML code between the two <Add></Add>, <Change></Change> and <Delete></Delete> tags. Save the files.

      File sample after changes:

```
<?xml version="1.0" encoding="UTF-8"?>
<TcBusinessData xmlns="http://teamcenter.com/BusinessModel/TcBusinessData"
    Date="" TcVersion="">
<Add>
</Add>
<Delete>
</Delete>
<Change>
</Change>
</TcBusinessData>
```

c. Repeat the same steps for source files below the <Project>/extensions/lang directory. Save the file.

File sample after changes:

```
<?xml version="1.0" encoding="UTF-8"?>
<TcBusinessDataLocalization
xmlns="http://teamcenter.com/BusinessModel/TcBusinessDataLocalization"
    Date="" TcVersion="">
<Add>
</Add>
</TcBusinessDataLocalization>
```

3) Package your template for distribution
   d) Select *File->New->Other->Package Template Extensions…* option to package the template

20.3.2 Removing the template from the Teamcenter installations

After you deleted the data model definitions from your template project “customer”, you must update all Corporate Server environments where the template “customer” was deployed to delete the data model definitions from the database.

The updating of Corporate Server environments involves three steps which are explained in detail in subsequent sections-

1) Delete the data model definitions of the template to be removed from the database
2) Update TEM related files in TC_ROOT
3) Update files in TC_DATA/model
4) Unregister the template from the database

20.3.2.1 Delete the data model definitions from the database

This section describes the steps to delete the data model definitions from the database.

5) Before proceeding, you must obtain the template package for “customer” template
6) Create a backup of the folder <TC_DATA>/model
7) Update the database with your “customer” package that was created in BMIDE
   e) Launch TEM in maintenance mode
   f) Select the option *Teamcenter Foundation→Update database (Full Model – System downtime required).* See Figure 76 TEM panel with template update optionFigure 76.

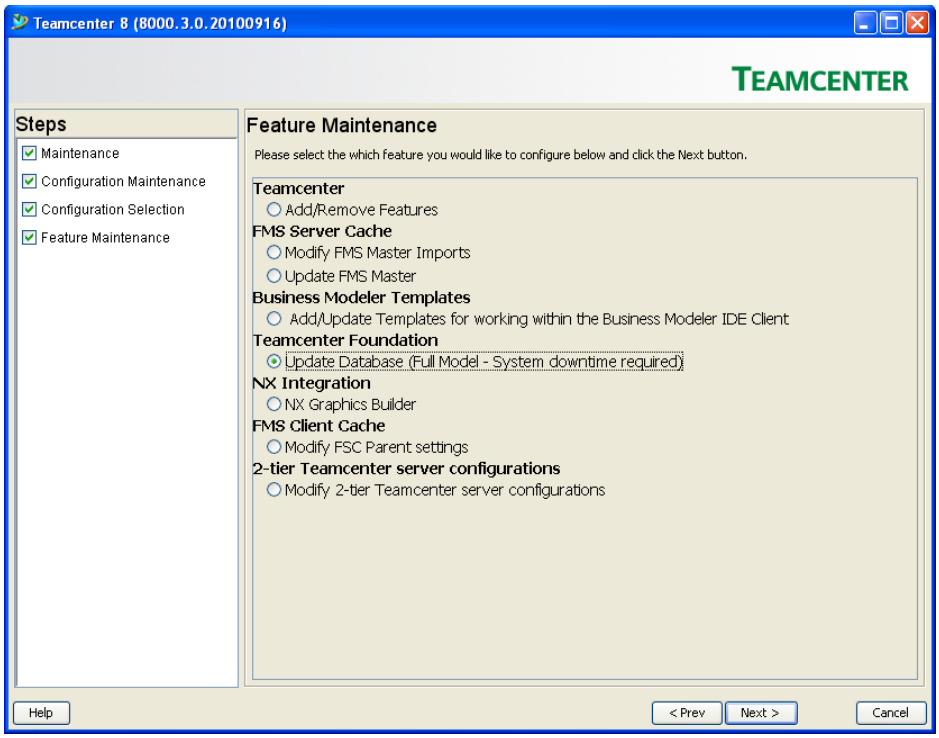


Figure 76 TEM panel with template update option

g) When TEM prompts for the template package, you must browse and provide the "customer" package.
h) Go through the rest of the TEM panels and update the database

NOTE: The TEM options shown here is from Teamcenter 8.3.0. Please ensure you are selecting the correct option in TEM to update your database in releases older than Teamcenter 8.3.0. For more details on how to update a template, please read the topic Adding features→Installing a custom solution or third-party template→Update the database using TEM in the Teamcenter installation guide.

At this point you have deleted the data model definition of the template to be removed from the database.

20.3.2.2 Update TEM related files in TC_ROOT

This section describes the steps to update the TEM related files in your Corporate Server <TC_ROOT> folder.

1) Create a backup of the folder <TC_ROOT>/install

2) Delete the folder named "**customer**" from the directory <TC_ROOT>/install.
   a. The directory name will be <TC_ROOT>/install/<template_name>. So if the template you want to remove is "customer", the directory to be deleted is <TC_ROOT>/install/customer.
   b. The "customer" folder is not required any more. It is used as a staging folder for your template during TEM updates. We will be removing your template "customer" shortly and hence this folder is not required. So you can safely delete it.

3) Update each of the TEM files listed below.

   e) **File: configuration.xml**
      iii) Open the file <TC_ROOT>/install/configuration.xml
      iv) Search and delete all occurrences of **customer.** Please ensure you are performing a case sensitive and exact word match search and delete.
      File sample with the lines to be deleted:

```
<features>
...
...
  <installed feature="2AC0751418AC21DE184FA2A6AF65BB87" name="customer/Data Model"/>
  <spf feature="393DCEC604D5232AB36BCEAB9300E623" name="customer" />
...
...
</features>
```

f) **File: uninstall.xml**

iii) Open the file <TC_ROOT>/install/uninstall.xml

iv) Search and delete all occurrences of **customer.** Please ensure you are performing a case sensitive and exact word match search and delete.

File sample with the lines to be deleted:

```
...
...
    <zip name="tc/customer_template.zip" content="02841667177685303.txt">
      <feature code="2AC0751418AC21DE184FA2A6AF65BB87" config="MYDB" />
    </zip>
    <zip name="tc/customer_install.zip" content="16279217094899623.txt">
      <feature code="2AC0751418AC21DE184FA2A6AF65BB87" config="MYDB" />
    </zip>
...
...
```

g) **File: feature_customer.xml**

iii) Delete the file named feature_customer.xml from the directory <TC_ROOT>/install/install/modules

h) **File: TEM bundle files**

iv) Delete the TEM bundles files related to your customer feature

v) For example,

- Delete the file named “customerBundle_en_US.xml” from <TC_ROOT>/install/install/lang/en_US

vi) If you have provided TEM bundle file translations in other languages, then you must perform similar changes to the remaining TEM bundle files. All these TEM bundle file translations are under <TC_ROOT>/install/install/lang folder.

20.3.2.3 Update files in TC_DATA/model

After the <TC_ROOT> folders are updated, the next step is to update the files in <TC_DATA>/model folder.

5) Create a backup of the folder <TC_DATA>/model.

6) Update each of the files listed below. In these files, you will do a search and delete the entries containing “customer”. Ensure that you are using a case sensitive and exact word match in your search and delete process.

g) **File: FileVersions.txt**

iii) Open the file <TC_DATA>/model/FileVersions.txt, if it exists.

iv) Search and delete all occurrences of **customer.**

File sample with the lines to be deleted:

```
...
...
file=customer_dependency.xml version=2
file=customer_template.xml version=2
```

file=customer_template_en_US.xml version=2...
...

h) **File: master.xml**
   iii) Open the file <TC_DATA>/model/master.xml
   iv) Search and delete all occurrences of **customer**

   File sample with the lines to be deleted:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<TcBusinessDataIncludes>
    ...
    <include file="customer_template.xml"/>
    ...
</TcBusinessDataIncludes>
```

i) **File: customer_dependency.xml**
   iii) Delete the file "customer_dependency.xml" from <TC_DATA>/model folder

j) **File: customer_template.xml**
   iii) Delete the file "customer_template.xml" from <TC_DATA>/model folder

k) **File: Languages related to customer template**
   v) Delete all language files related to customer template from the folder <TC_DATA>/model/lang

7) Update all the template-related files from the <TC_DATA>/model folder to datasets in the database.
   - This step is required only if you are working with Teamcenter 8.3 or a later release
   - From the TC command shell, run the manage_model_files utility pointing to the <TC_DATA>/model directory

```
manage_model_files –u=<user name> -p=<user password> -g=dba –option=upload
```

8) Delete **customer** template related files from datasets
   a) This step is required only if you are working with Teamcenter 8.3 or a later release
   b) Starting at Teamcenter 8.3.0, files in <TC_DATA>/model folder are also stored in the database as datasets. The datasets that store these files are maintained in a folder named **Fnd0BMIDEResource.**This means the database contains a copy of your "customer" files in datasets and they must be removed from the database.
   c) The below steps will help you delete the "customer" related files from datasets.

   i. Launch Teamcenter Rich Client and query the folder of type **Fnd0BMIDEResource**. See Figure 77 for the results of the search.

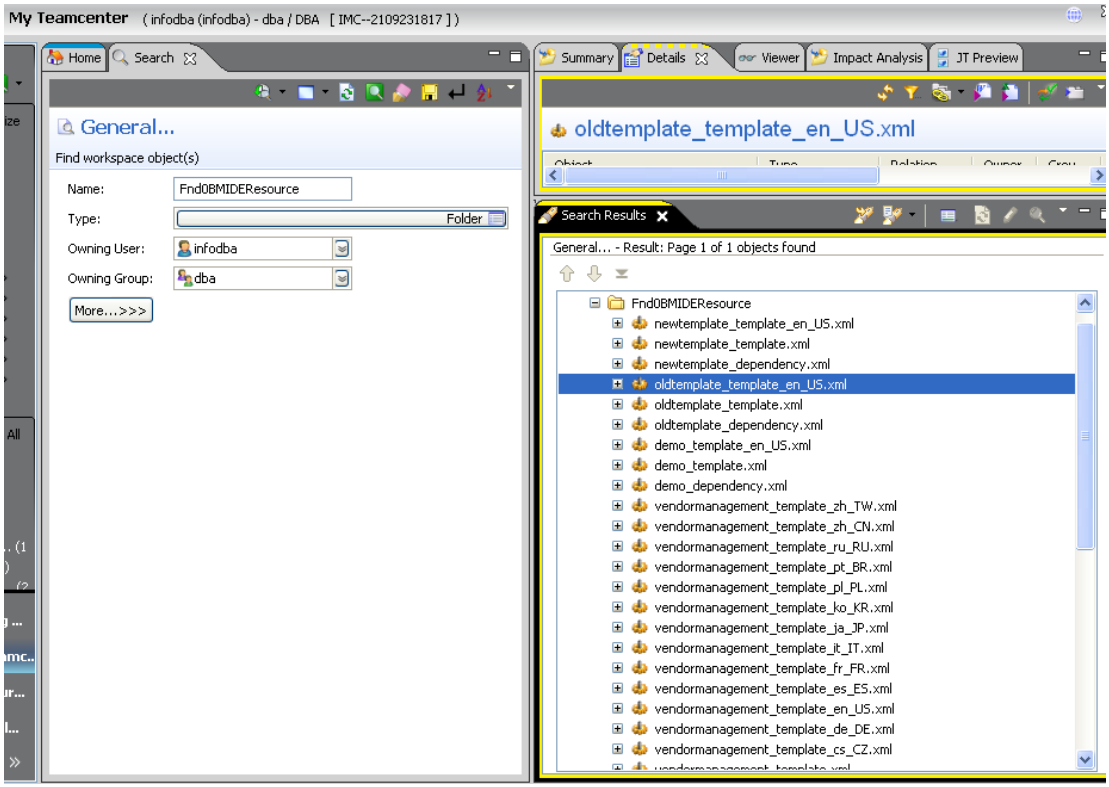


Figure 77 Results from query for folder contents

ii. From the search results, delete all files related to customer as listed below:

- customer_template.xml
- customer_dependency.xml
- customer_template_en_US.xml. You will have to do the same for all language files that is applicable to your “customer” template.

iii. Close RAC.

9) Remove the template name from <TC_DATA>/install/models.xml
   a. This step is required only if you are working in Teamcenter 9.1 or a later release
   b. Remove the entries of “customer” template.

20.3.2.4 Unregister the template from the database

3) After updating the <TC_ROOT> and <TC_DATA>/model folders, the next step is to unregister the template **“customer”** from the database. This ensures that the template removal process is successful and your model directory and database are in sync. To unregister the “**customer**” project from the database, run the below command from the Teamcenter command prompt-

```
bmide_manage_templates -u=<user name> -p=<user password> -g=dba -option=remove -templates=customer
```

At this point you have completed the removal process of your “customer” template.

20.4 Removing a template owned by someone else

Below are the instructions to remove a template owned by someone else named as "**tcii**" from a Teamcenter Unified installation. Note that the steps to remove a template owned by someone else vary from the steps to remove a custom template that you own, as the template project's source code for that template will not available to the customers. However, the packaged template is available and we will utilize that to remove the template from the database.

Note: If you intend to remove a template from Teamcenter 11.2.3 and onwards, refer the instructions in section 20.5. Refer sections 20.3 and 20.4 for an earlier release.

Assumptions:

- There is no instance data in the database for the data model defined in the template to be removed
- If there are any other templates that are dependent upon this template, they must be removed from the database first before this template can be removed.
- The contents of model directory are in sync with the database.
    - This means if you have any new data model changes in your template that is not yet deployed to the database, ensure you deploy them before removing template dependencies. If you include deployment of new data model and removing of template dependency as a single process and say deployment fails, then it adds to the complexity of debugging such failed deployments.
    - But if you have shared your template with your partner then do not add new data model to your template after removing the template dependency. Ensure the data model matches with the one that was previously distributed to your partners.

Note that the instructions below will assume that the template being removed is called "tcii". If you are removing a template with another name please substitute the name of your template everywhere you see the word "tcii" in the following directions.

There are two possible scenarios of removing a template that is not owned by you:

A. No template is dependent on the template you want to remove

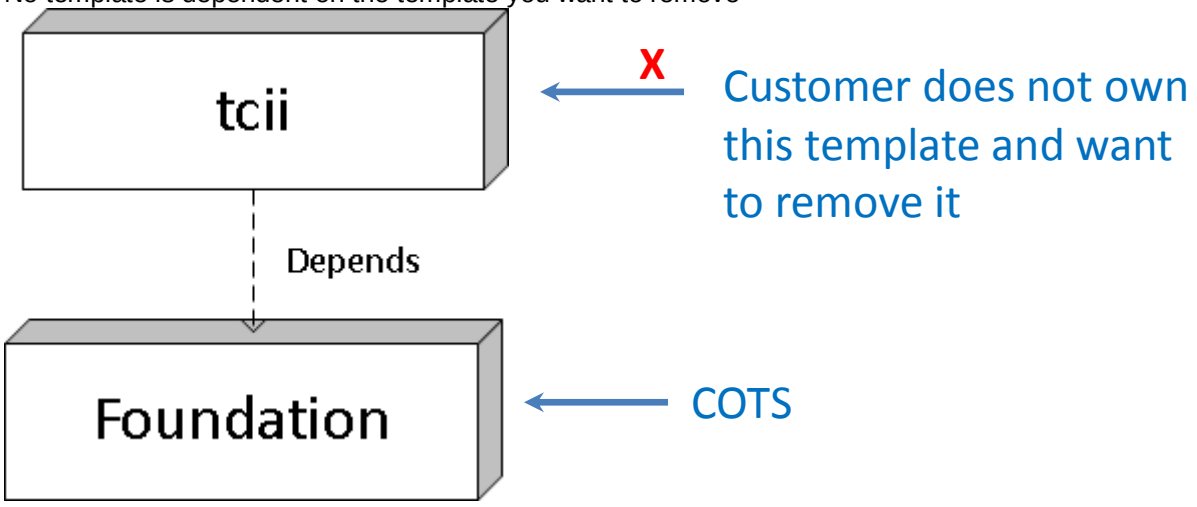


Figure 78 - No dependent templates

If none of the templates are dependent on the template to removed, you can proceed with the steps detailed below starting from Section 20.4.1.

B. Other templates are dependent on the template you want to remove

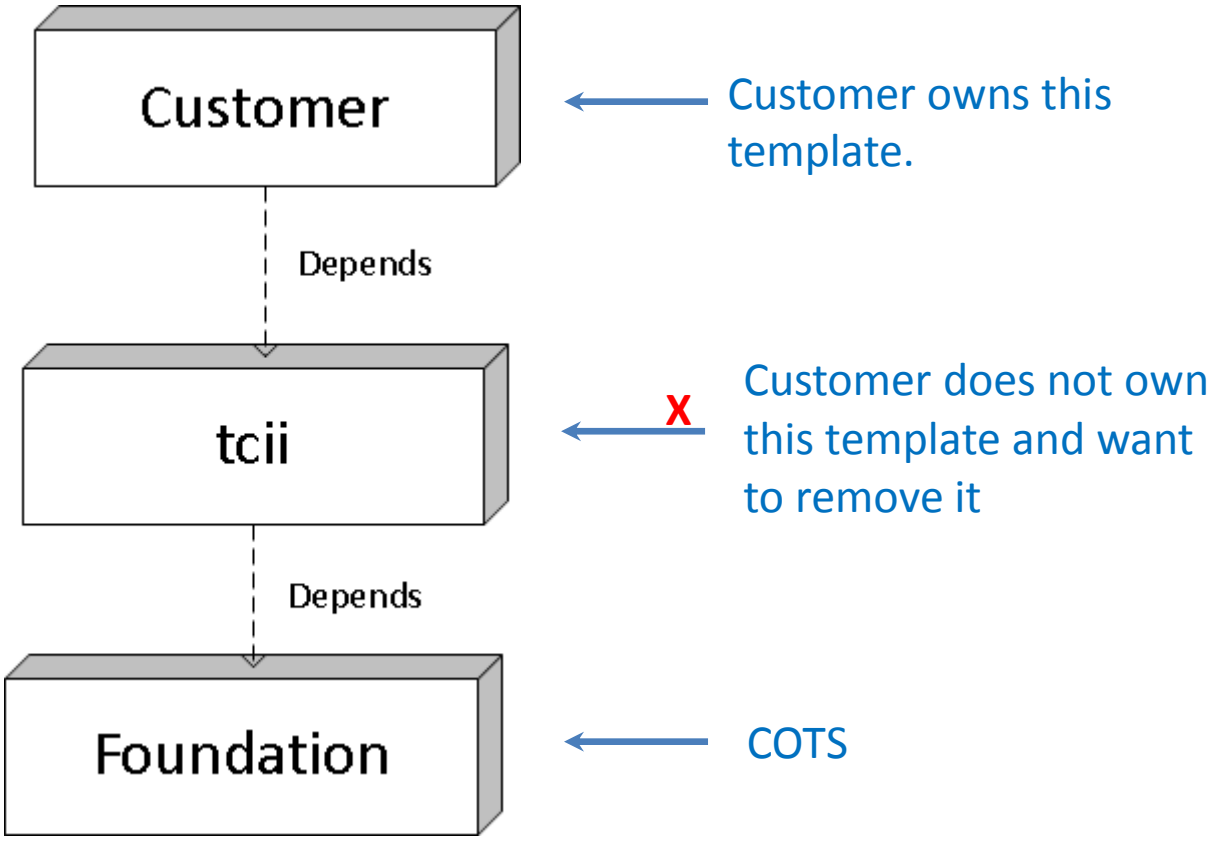


Figure 79 - dependent templates

If you have a custom template that is depended on this template, you will have to remove the dependency before removing this template. Execute the steps detailed in section 21 before executing the steps detailed in this section. After removing the dependency, you can proceed with the steps detailed below starting from Section 20.4.1.

20.4.1 Remove the dependency on the template to be removed

This section describes the steps to remove the removing the dependency on tcii template.

20.4.1.1 Remove the template from the list of templates in master.xml

1) Open the file <TC_DATA>/model/master.xml
2) Search and delete all occurrences of template name “tcii”
3) File sample with the lines to be deleted:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<TcBusinessDataIncludes>
    ...
    <include file="tcii_template.xml"/>
    ...
</TcBusinessDataIncludes>
```

20.4.2 Removing the template from the Teamcenter installations

After you removed the dependency on the “tcii” template, you must update all Corporate Server environments where the template “tcii” was deployed to delete the data model definitions from the database.

The updating of Corporate Server environments involves three steps which are explained in detail in subsequent sections-

1) Delete the data model definitions of the template to be removed from the database
2) Update TEM related files in TC_ROOT
3) Update files in TC_DATA/model
4) Unregister the template from the database

20.4.2.1 Delete the data model definitions of the template to be removed from the database

20.4.2.1.1 Use bmide_consolidator utility to consolidate the templates

The bmide_consolidator utility consolidates the data model of all templates listed in the master.xml into a single file.

a) From the TC command shell, run the bmide_consolidator utility pointing to the <TC_DATA>/model directory

```
bmide_consolidator -dir=%TC_DATA%/model -consolidate=all -
file=<path_to_consolidated_model.xml>
```

20.4.2.1.2 Extract the data model in the database using business_model_extractor

The business_model_extractor utility extracts the business data from the database into an xml file.

a) From the TC command shell, run business_model_extractor utility to extract the data model from the database.

```
business_model_extractor -u=<user name> -p=<user password> -g=dba -mode=all -
outfile=<path_to_extracted_model.xml>
```

20.4.2.1.3 Generate a delta using bmide_comparator

The bmide_comparator utility compares two Teamcenter model files and generates a differences file.

a) From the TC command shell, run the bmide_comparator utility to generate a delta file between the extracted file (generated in section 20.4.2.1.2) and consolidated model file (generated in section 20.4.2.1.1)

```
bmide_comparator -compare=all -old=<path_to_extracted_model.xml> -
new=<path_to_consolidated_model.xml> -delta=<path_to_delta.xml>
```

20.4.2.1.4 Update the database

The business_model_updater utility deploys the TC schema, types and business rules in to the database.

a) From the TC command shell, run business_model_updater to deploy the changes listed in the differences file generated in section 20.4.2.1.3.

```
business_model_updater -u=<user name> -p=<user password> -g=dba -mode=upgrade -update=all
-file=<path_to_delta.xml>
```

At this point you have deleted the data model definition of the tcii template to be removed from the database.

20.4.2.2 Update TEM related files in TC_ROOT

This section describes the steps to update the TEM related files in your Corporate Server <TC_ROOT> folder.

1) Create a backup of the folder <TC_ROOT>/install

2) Delete the folder named “tcii” from the directory <TC_ROOT>/install
   a) The “tcii” folder is not required any more. It is used as a staging folder for your template during TEM updates. We will be removing the “tcii” shortly and hence this folder is not required. So you can safely delete it.

3) Update each of the TEM files listed below.
   a) **File: configuration.xml**
      i) Open the file <TC_ROOT>/install/configuration.xml
      ii) Search and delete all occurrences of “tcii”**.** Please ensure you are performing a case sensitive and exact word match search and delete.

      File sample with the lines to be deleted:

```
<features>
...
...
    <installed feature="2AC0751418AC21DE184FA2A6AF65BB87" name=" tcii/Data Model" />
    <spf feature="393DCEC604D5232AB36BCEAB9300E623" name="tcii" />
...
...
</features>
```

   b) **File: uninstall.xml**
      i) Open the file <TC_ROOT>/install/uninstall.xml
      ii) Search and delete all occurrences of “tcii”. Please ensure you are performing a case sensitive and exact word match search and delete.

      File sample with the lines to be deleted:

```
...
...
    <zip name="tc/tcii_template.zip" content="02841667177685303.txt">
      <feature code="2AC0751418AC21DE184FA2A6AF65BB87" config="MYDB" />
    </zip>
    <zip name="tc/tcii_install.zip" content="16279217094899623.txt">
      <feature code="2AC0751418AC21DE184FA2A6AF65BB87" config="MYDB" />
    </zip>
...
...
```

   c) **File: feature_tcii.xml**
      i) Delete the file named feature_tcii.xml from the directory <TC_ROOT>/install/install/modules

   d) **File: TEM bundle files**
      i) Delete the TEM bundles files related to your customer feature
      ii) For example,
         - Delete the file named “tciiBundle_en_US.xml” from <TC_ROOT>/install/install/lang/en_US
      iii) If you have provided TEM bundle file translations in other languages, then you must perform similar changes to the remaining TEM bundle files. All these TEM bundle file translations are under <TC_ROOT>/install/install/lang folder.

20.4.2.3 Update files in TC_DATA/model

After the <TC_ROOT> folders are updated, the next step is to update the files in <TC_DATA>/model folder.

1) Keep a backup of the folder <TC_DATA>/model folder.

2) Update each of the files listed below. In these files, you will do a search and delete the entries containing “tcii”. Ensure that you are using a case sensitive and exact word match in your search and delete process.

a) **File: FileVersions.txt**
   i) Open the file <TC_DATA>/model/FileVersions.txt, if it exists.
   ii) Search and delete all occurrences of template1**.**

   File sample with the lines to be deleted:

```
...
...
file= tcii_dependency.xml version=2
file= tcii_template.xml version=2
file= tcii_template_en_US.xml version=2...
...
```

b) **File: tcii_dependency.xml**
   i) Delete the file “tcii_dependency.xml” from <TC_DATA>/model folder

c) **File: tcii_template.xml**
   i) Delete the file “tcii_template.xml” from <TC_DATA>/model folder

d) **File: tcii_tcbaseline.xml**
   i) Delete the file “tcii_template.xml” from <TC_DATA>/model/baselines folder

e) **File: tcii_icons.zip**
   i) This step is required only if you are working with Teamcenter 9.0 or a later release
   ii) If exists, delete the file “tcii_icons.zip” from <TC_DATA>/model/icons folder

f) **File: Languages related to tcii template**
   i) Delete all language files related to tcii template from the folder <TC_DATA>/model/lang

3) Update all the template-related files from the <TC_DATA>/model folder to datasets in the database.
   a) This step is required only if you are working with Teamcenter 8.3 or a later release
   b) From the TC command shell, run the manage_model_files utility pointing to the <TC_DATA>/model directory

```
manage_model_files –u=<user name> -p=<user password> -g=dba –option=upload
```

4) Delete template tcii related files from datasets
   a) This step is required only if you are working with Teamcenter 8.3 or a later release
   b) Starting at Teamcenter 8.3.0, files in <TC_DATA>/model folder are also stored in the database as datasets. The datasets that store these files are maintained in a folder named **Fnd0BMIDEResource.**This means the database contains a copy of your “tcii” files in datasets and they must be removed from the database.
   c) The below steps will help you delete the “tcii” related files from datasets.

   c) Launch Teamcenter Rich Client and query the folder of type **Fnd0BMIDEResource.** See Figure 80 for the results of the search.

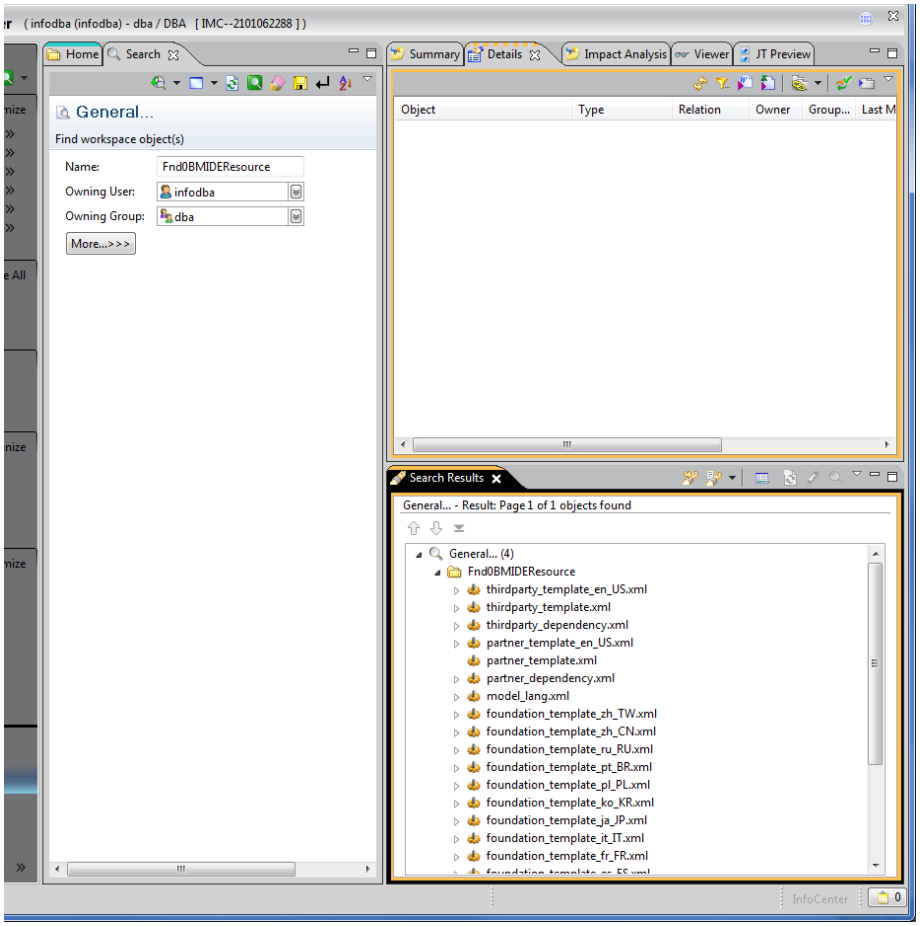


Figure 80 - Results from query for folder contents

d) From the search results, delete all files related to customer as listed below: (NO delete option when you right click. But del button works)

- tcii_template.xml
- tcii_dependency.xml
- tcii_template_en_US.xml. You will have to do the same for all language files that is applicable to your “tcii” template.
- tcii_icons.zip
- tcii_icons.zip.ref
- tcii_template.ref

e) Close RAC.

5) Remove the template name from <TC_DATA>/install/models.xml
   a. This step is required only if you are working in Teamcenter 9.1 or a later release
   b. Remove the entries of “tcii” template.

20.4.2.4 Unregister the template from the database

After updating the <TC_ROOT> and <TC_DATA>/model folders, the next step is to unregister the template **"**tcii**"** from the database. This ensures that the template removal process is successful and your model directory and database are in sync.

To unregister the "tcii" template from the database, run the below command from the Teamcenter command prompt-

```
bmide_manage_templates -u=<user name> -p=<user password> -g=dba -option=remove -
templates= tcii
```

To update the %TC_DATA%\model\installed_templates_info.txt, run the below command from the Teamcenter command prompt-

```
bmide_manage_templates -u=<user name> -p=<user password> -g=dba -option=list
```

At this point you have completed the removal process of your "tcii" template.

20.5 Removing a template – from Teamcenter 11.2.3 and onwards

Below are the instructions to remove a template named as "**tcii**" from a Teamcenter Unified installation from 11.2.3 and onwards.

Assumptions:

- There is no instance data in the database for the data model defined in the template to be removed.
- If there are any other templates that are dependent upon this template, they must be removed from the database first before this template can be removed.
- The contents of model directory are in sync with the database.
    - This means if you have any new data model changes in your template that is not yet deployed to the database, ensure you deploy them before removing template dependencies. If you include deployment of new data model and removing of template dependency as a single process and say deployment fails, then it adds to the complexity of debugging such failed deployments.
    - But if you have shared your template with your partner then do not add new data model to your template after removing the template dependency. Ensure the data model matches with the one that was previously distributed to your partners.
- All the backup steps mentioned in section 20.2 (Before removing a template from Teamcenter) are executed.

Note that the instructions below will assume that the template being removed is called "tic". If you are removing a template with another name please substitute the name of your template everywhere you see the word "tcii" in the following directions.

There are two possible scenarios of removing a template:

A. No template is dependent on the template you want to remove

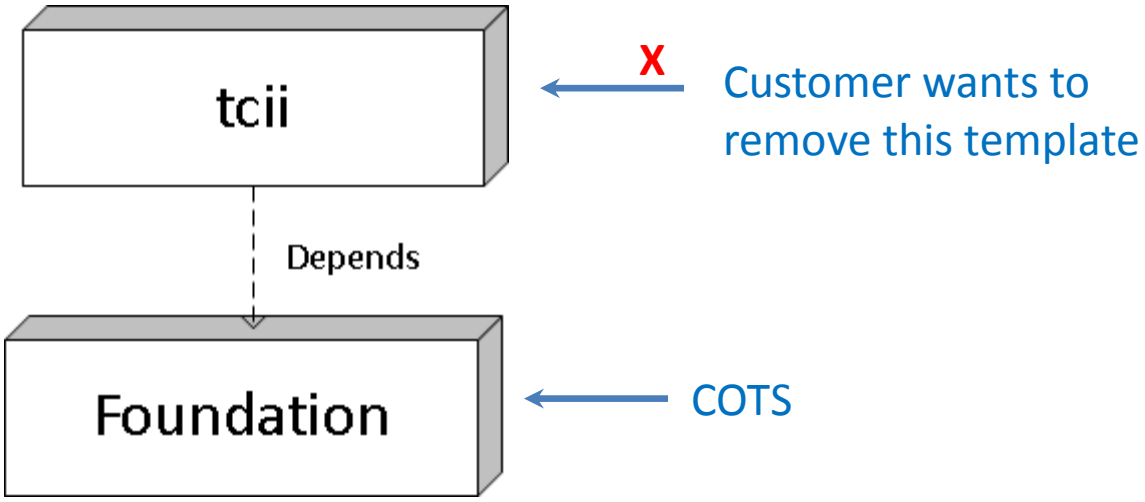


Figure 81 - No dependent templates

If none of the templates are dependent on the template to removed, you can proceed with the steps detailed below starting from Section 20.5.1.

B. Other templates are dependent on the template you want to remove

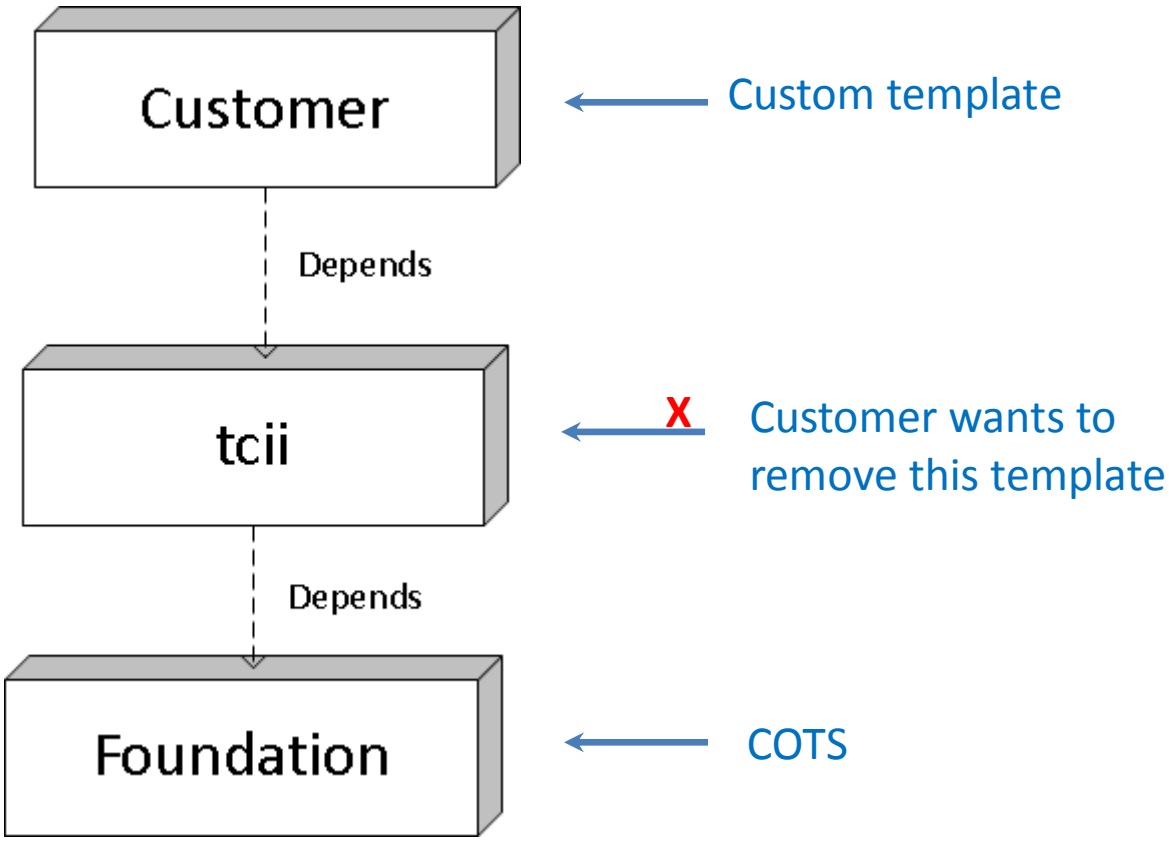


Figure 82 - dependent templates

If you have a custom template that is depended on this template, you will have to remove the dependency before removing this template. Note that the instructions below will assume that for template “customer”, you want to remove the dependency on “tcii”. If you are removing dependency on a template with another name please substitute the name of your template everywhere you see the word “tcii” in the following directions.

Following are steps to remove the dependency on “tcii”:

1) Ensure the “customer” template project in which the dependency has to be changed is in sync with the database. This means if you have any new data model changes in your template that is not deployed to the database, ensure you deploy them before the updating process. After removing the dependency on “tcii”, the final step would be to deploy your template to the database. So if you include deployment of new data model and changing the dependency of your template as a single process and say deployment fails, then it adds to the complexity of debugging such failed deployments.

2) Keep a backup copy of your template project. In case you run into issues with updating your template, we can always get back to your old state of the template.

3) Load the “customer” template project in BMIDE and remove the dependency on “tcii”
    a) In Standard perspective, select the “customer” project, RMB and select “Properties”. Alternatively, In Advanced perspective, Go to Navigator view, select “Properties”.
    b) In the “Properties for customer” dialog, select Teamcenter → BMIDE
    c) In the “Dependent Templates” table, uncheck the checkbox against “tcii” and click OK to remove the dependency

4) Select your template project icon, right click and choose “Reload Data Model”.
    a) Review the “Console” and ensure there are no loading errors. If there are no loading errors, the console view displays a success message as below:

```
------------------------------------------------------------------------
Loading project: "customer"...
Project "customer" load completed on Sun Mar 04 17:52:20 EST 2012.
```

    For information on loading errors, please refer Section 13 Troubleshooting BMIDE Validation errors and Section 6.7.4.2.1 Loader / Model errors.

5) Package the “customer” template for deployment
    a. Select File->New->Other->Package Template Extensions… option to package the template

6) Keep a backup of the folder <TC_DATA>/model

7) Update the data model definitions of “customer” template in the database
    a. Launch TEM in maintenance mode
    b. Select the option Teamcenter Foundation->Update database(Full Model – System downtime required). See Figure 76 TEM panel with template update option.
    c. When TEM prompts for the template package, you must browse and provide the “customer” package.
    d. Go through the rest of the TEM panels and update the database.

8) Package the “customer” template for deployment
    a. Select File->New->Other->Package Template Extensions… option to package the template

9) Update the data model definitions of “customer” template in the database
    a. Launch TEM in maintenance mode

b. Select the option Teamcenter Foundation->Update database(Full Model – System downtime required). See Figure 76 TEM panel with template update option.
c. When TEM prompts for the template package, you must browse and provide the “customer” package.
d. Go through the rest of the TEM panels and update the database.

Now, the project “customer” is not dependent on “tcii” anymore. After removing the dependency, you can proceed with the steps detailed below starting from Section 20.5.1.

20.5.1 Remove the data model and template

After you removed the dependency on the “tcii” template, you must update all Corporate Server environments where the template “tcii” was deployed to delete the data model definitions from the database.

The updating of Corporate Server environments involves three steps which are explained in detail in subsequent sections-

1) Delete the data model definitions of the template to be removed from the database
2) Update TEM related files in TC_ROOT
3) Update files in TC_DATA

20.5.1.1 Delete the data model definitions of the template to be removed from the database

20.5.1.1.1 Use bmide_remove_template utility to validate if the template can be removed

The bmide_remove_template utility in “dryrun” mode validates if a template can be removed. It reports if there are template dependencies or all the instances of the data model defined in the template including references of each instance.

a) From the TC command shell, run the following command:

```
bmide_remove_template -u=<user name> -p=<user password> -g=dba –mode=dryrun –
template=tcii
```

b) Review the “Analysis Results” section of bmide_remove_template log generated in TEMP folder and check if the template can be removed. If there are instances of the data model being removed, then the log provides a summary of the instances in the database, a detailed information of each instance, their references and whether they can be automatically deleted by the bmide_remove_template utility in the “instanceDelete” mode.

If there are no instances of the data model defined in the template being removed, then “Analysis Results” section displays a success message as below:

```
The template "xxx" can be deleted because there are no instances for the
definitions in the database for this template.
```

20.5.1.1.2 Use bmide_remove_template utility to delete the instances of the data model

If there are instances (of the data model being removed) reported in the dryrun log, then run the bmide_remove_template utility in “instanceDelete” mode to delete the instances. It identifies all the instances of the data model being removed and deletes the instances (along with their references) and report the results of the delete activity.

a) From the TC command shell, run the following command:

```
bmide_remove_template -u=<user name> -p=<user password> -g=dba –mode=instanceDelete –
template=tcii
```

b) Review the bmide_remove_template log generated in TEMP folder and check if the all the instances are deleted successfully. For the instances that could not be automatically deleted by the utility, you must manually delete this instance using an end user client like RAC or Active Workspace. You can use the instructions in Section 22.2 as a reference.

20.5.1.1.3 Use bmide_remove_template utility to remove the data model

If there are no instances (of the data model being removed) reported in the dryrun/instanceDelete log, then run the bmide_remove_template utility in "remove" mode to remove the data model definitions in the template from the database and unregister the template

a) From the TC command shell, run the following command:

```
bmide_remove_template -u=<user name> -p=<user password> -g=dba –mode=remove –
template=tcii
```

b) Review the "Execution Results" section of bmide_remove_template log generated in TEMP folder and check if the template is removed successfully.

c) If the template is removed successfully, then "Execution Results" section displays a success message as below:

```
This utility was successful in removing the "xxx" template from the database.
In order to remove the template, all data model

definitions and their instances were removed from the database. See the "Log
file" mentioned in the "Log Section" above for more details.
```

At this point you have deleted the data model definition of the tcii template to be removed from the database.

20.5.1.2 Update TEM related files in TC_ROOT

This section describes the steps to update the TEM related files in your Corporate Server <TC_ROOT> folder.

1) Create a backup of the folder <TC_ROOT>/install

2) Delete the folder named "tcii" from the directory <TC_ROOT>/install
   a) The "tcii" folder is not required any more. It is used as a staging folder for your template during TEM updates. We will be removing the "tcii" shortly and hence this folder is not required. So you can safely delete it.

3) Update each of the TEM files listed below.
   a) **File: configuration.xml**
      i) Open the file <TC_ROOT>/install/configuration.xml
      ii) Search and delete all occurrences of "tcii"**.** Please ensure you are performing a case sensitive and exact word match search and delete.

      File sample with the lines to be deleted:

```
<features>
...
...
```

```
      <installed feature="2AC0751418AC21DE184FA2A6AF65BB87" name=" tcii/Data Model" />
      <spf feature="393DCEC604D5232AB36BCEAB9300E623" name="tcii" />
  ...
  ...
  </features>
```

b) **File: uninstall.xml**

i) Open the file <TC_ROOT>/install/uninstall.xml

ii) Search and delete all occurrences of “tcii”. Please ensure you are performing a case sensitive and exact word match search and delete.

File sample with the lines to be deleted:

```
...
...
    <zip name="tc/tcii_template.zip" content="02841667177685303.txt">
      <feature code="2AC0751418AC21DE184FA2A6AF65BB87" config="MYDB" />
    </zip>
    <zip name="tc/tcii_install.zip" content="16279217094899623.txt">
      <feature code="2AC0751418AC21DE184FA2A6AF65BB87" config="MYDB" />
    </zip>
...
...
```

c) **File: feature_tcii.xml**

i) Delete the file named feature_tcii.xml from the directory <TC_ROOT>/install/install/modules

d) **File: TEM bundle files**

i) Delete the TEM bundles files related to your customer feature

ii) For example,

- Delete the file named “tciiBundle_en_US.xml” from <TC_ROOT>/install/install/lang/en_US

iii) If you have provided TEM bundle file translations in other languages, then you must perform similar changes to the remaining TEM bundle files. All these TEM bundle file translations are under <TC_ROOT>/install/install/lang folder.

20.5.1.3 Update files in TC_DATA

After the <TC_ROOT> folders are updated, the next step is to update the files in <TC_DATA> folder.

1) Remove the template name from <TC_DATA>/install/models.xml
   c. This step is required only if you are working in Teamcenter 9.1 or a later release
   d. Remove the entries of “tcii” template.

21 How to remove a dependency from your template

Below are the instructions to remove a dependency on a template “template1” from template “template2”.

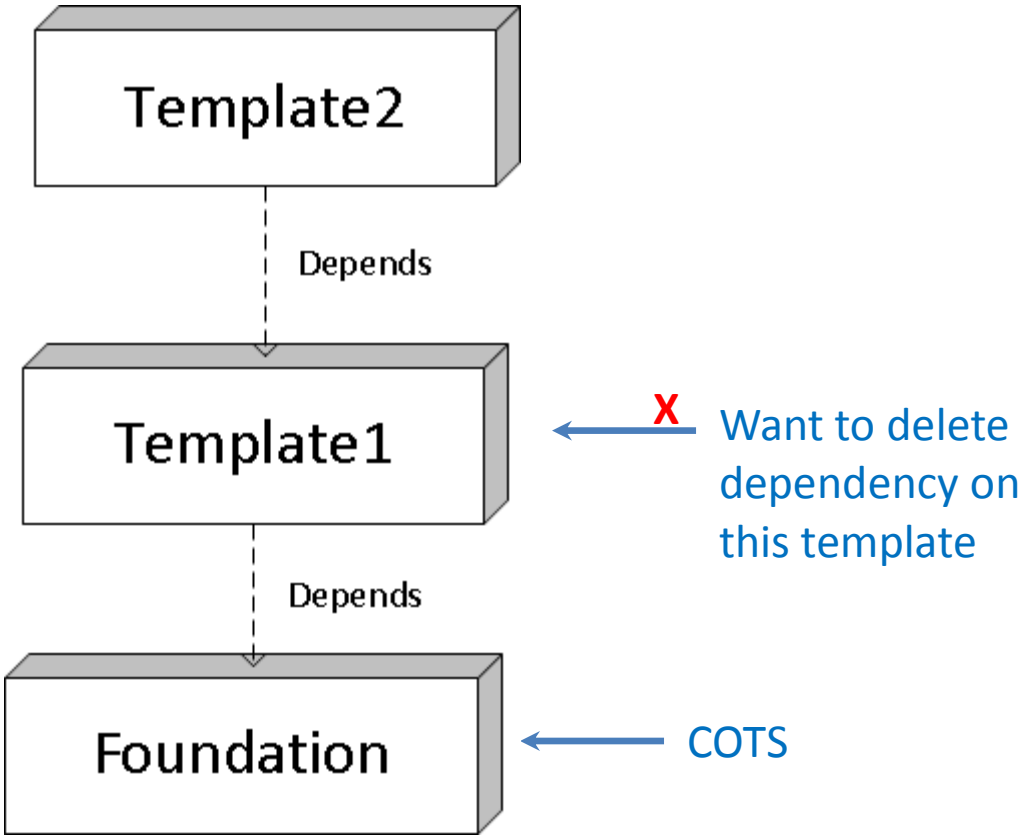


Figure 83 - dependent templates

21.1.1 Update the dependent template project

This section describes the steps to remove the dependency on a template in BMIDE. Note that the instructions below will assume that for template “template2”, you want to remove the dependency on “template1”. If you are removing dependency on a template with another name please substitute the name of your template everywhere you see the word “template1” in the following directions.

Assumptions:

- There is no instance data in the database for the data model defined in the template to be removed
- The contents of model directory are in sync with the database.
    - This means if you have any new data model changes in your template that is not yet deployed to the database, ensure you deploy them before removing template dependencies. If you include deployment of new data model and removing of template dependency as a single process and say deployment fails, then it adds to the complexity of debugging such failed deployments.
    - But if you have shared your template with your partner then do not add new data model to your template after removing the template dependency. Ensure the data model matches with the one that was previously distributed to your partners.

Following are steps to update up the data model definitions:

10) Ensure the “template2” template project in which the dependency has to be changed is in sync with the database. This means if you have any new data model changes in your template that is not deployed to the database, ensure you deploy them before the updating process. After removing the dependency on “template1”, the final step would be to deploy your template to the database. So if you include deployment

of new data model and changing the dependency of your template as a single process and say deployment fails, then it adds to the complexity of debugging such failed deployments.

11) Keep a backup copy of your template project. In case you run into issues with updating your template, we can always get back to your old state of the template.

12) Close your project in BMIDE before removing its dependency
    a) Go to Navigator view
    b) Select the “template2” project, RMB select “Close project”
    c) This will close your project in BMIDE

13) Shutdown BMIDE

14) Remove dependency on “template1” in the “template2” template project
    a. Update the dependency.xml file for “template2” template
        i. Go to the folder <TC_ROOT>/bmide/workspace/template2/extensions in your file system
        ii. Open the file dependency.xml located in this directory
        iii. Search and remove all occurrences of “template1”

        File sample after changes:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<TcBusinessDataIncludes ActiveRelease="tc8000.3.0" displayName="Template2"
enableOpsDataDeploy="false" guid="ABAAF6855A497F72D251423600482831"
name="template2" optional="true" prefixes="F3" teamcenterTemplate="false">

<include file="foundation_template.xml"/>
...
</TcBusinessDataIncludes>
```

    b. Update the feature_template2.xml file for “template1” template
        i. Go to the folder <TC_ROOT>/bmide/workspace/template2/ install in your file system
        ii. Search and remove all occurrences of “template1” in the feature file.

        File sample after changes:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!--
Document : feature_template2.xml
Description: This XML is used by TEM to install or upgrade the "template2"
solution.
-->
<feature>
<name value="Template2"/>
<property name="feature_name" value="template2"/>
<group value="package"/>
<guid value="ED0936452CC064BCE17D103FBEB1D5BB"/>
<bundle value="${feature_name}Bundle.xml"/>
<description value="${feature_name}.description"/>
<include file="dataModelDependency.xml"/>
...
</feature>
```

15) Load the “template2” template whose dependency was changed in BMIDE and validate there are no loader errors.
    a) Launch BMIDE
    b) Select “template2” project, RMB select “Open Project”
    c) This will open your project in BMIDE and load it
    d) Review the “Console” and ensure there are no loading errors. If there are no loading errors, the console view displays a success message as below:

```
----------------------------------------------------------------------
```

```
Loading project: "template2"...
Project "template2" load completed on Sun Mar 04 17:52:20 EST 2012.
```

For information on loading errors, please refer Section 13 Troubleshooting BMIDE Validation errors and Section 6.7.4.2.1 Loader / Model errors.

16) Package the “template2” template for deployment
    b. Select File->New->Other->Package Template Extensions… option to package the template

17) Keep a backup of the folder <TC_DATA>/model

18) Update the data model definitions of “template2” template in the database
    e. Launch TEM in maintenance mode
    f. Select the option Teamcenter Foundation->Update database(Full Model – System downtime required). See Figure 76 TEM panel with template update option.
    g. When TEM prompts for the template package, you must browse and provide the “template2” package.
    h. Go through the rest of the TEM panels and update the database.

Now, the project “template2” is not dependent on “template1” anymore.

22 How to delete business object instances from the Database

The most important prerequisite for removing a template is to delete the instances of the entire data model elements defined in the template from the database. This section provides you a quick overview of the steps involved in identifying the data model elements and deleting the instance data.

Note: If you intend to remove a template from Teamcenter 11.2.3 and onwards, you could use the bmide_remove_template utility in “instanceDelete” mode to delete the instances of standand types, datasets, forms and references of Tool and NoteType. See Teamcenter Help documentation for details on this utility.

22.1 Identify the data model definitions defined in the template you want to remove

The first step in deleting the instance data is to identify the data model elements, in the template to be removed, which can have the instances in the database. Below are steps to identify the custom data model definitions in the “customer” template:

1) Open the file <TC_DATA>/model/custom_template.xml
2) From the XML tags present in the template XML file, identify the data model elements that can possibly have instances in the database. For example, below are the XML tags to identify some of the data model elements from the TCII template.
   - Class:

```
<TcClass className="IdeasDrawingSetRefs" isExportable="true"
isUninheritable="false" isUninstantiable="false" parentClassName="POM_object">

      <TcAttribute arrayLength="-1" attributeName="target_dataset_version"
      attributeType="POM_typed_reference" exportAsString="false"
      followOnExport="false" isArray="true" isCandidateKey="false"
      isNullsAllowed="false" isPublicRead="false" isPublicWrite="false"
      isTransient="false" isUnique="false" maxStringLength="0"
      noBackpointer="false" typedRefClassName="Dataset"/>

</TcClass>
```

   - Type:

```
<TcStandardType parentTypeName="POM_object" typeClassName="IdeasFemReference"
typeName="IdeasFemReference"/>
```

- Relation Type:

```
<TcStandardType parentTypeName="ImanRelation" typeClassName="ImanRelation"
typeName="IDEAS_ACCONTEXT"/>
```

- Dataset:

```
<TcDataset parentTypeName="Dataset" typeClassName="Dataset"
typeName="IdeasAssemblyViewInfo">
      <TcDSViewTool name="PV"/>
      <TcDSEditTool name="PV"/>
      <TcDatasetReference name="IdeasAssemblyViewInfo">
            <TcDatasetReferenceInfo format="BINARY" template="*.vfz"/>
      </TcDatasetReference>
</TcDataset>
```

- NoteType:

```
<TcNoteType description="I-DEAS occurrence identifier" noteTypeName="IDEAS TMTID"/>
```

- Tool:

```
<TcTool toolMimeType="application/iman-ideas" toolName="Ideas" toolReleaseDate="15-
Feb-2007 14:19" toolSymbol="C:\SDRC" toolVersion="1.0">
      <TcToolInputFormat formatName="BIN"/>
      <TcToolOutputFormat formatName="BIN"/>
</TcTool>
```

Below table shows the data model elements defined in TCII template (tcii_template.xml):

Data Element Type	Data Element Name
Class	
	IdeasAssemblyData
	IdeasAuxFileRefs
	IdeasBoundingBox
	IdeasDrawingReferences
	IdeasDrawingSetRefs
	IdeasFemReference
	IdeasGuids
	IdeasMigratedItemGuid
	IdeasPartAcr

	IdeasSourceReference
	IdeasUserAttributes
Dataset	
	IdeasAssemblyViewInfo
	IdeasAssembly
	IdeasDrawing
	IdeasDrawingSet
	IdeasFem
	IdeasPart
Relation	
	IDEAS_ACCONTEXT
	IDEAS_DRAWING
	IDEAS_FEM
	IDEAS_SOURCE
	IDEAS_DWGSET
NoteType	
	I-DEAS occurrence identifier
Tool	
	Ideas

Table 3 Data Model elements in TCII template

22.2 Delete the business object instances from the Database

To delete the instance data, you can use the RAC applications like MyTeamcenter, ProductStructureManager, Workflow Viewer etc. These steps to delete instance data will be different for different templates. The sequence of deleting instances of types and the number of iterations will vary from template to template.

For example, Ideas datasets have named references which hold POM references to datasets and Item Revisions. In order to successfully delete the data, all datasets need to be deleted. You may have to delete the instances iteratively until all instances are deleted, because these instances may have multiple references to each other. And an instance cannot be deleted if it is referenced by another instance. So TCII team suggests the following sequence for dataset deletion:

1. IdeasDrawingSet
2. IdeasDrawing / IdeasFem
3. IdeasAssembly
4. IdeasPart
5. IdeasAssemblyViewInfo

This section provides you some examples to create the queries get the required business object instances for the data model elements defined in TCII template. It also provides you a sample list of commonly encountered issues while deleting the instance data and the corrective actions.

22.2.1 Deleting the Dataset instances

The parameters (Search class and Search Criteria) in defining a query may differ depending on the Business Object whose instances to you want to search. Below are steps to query and delete the instances of dataset business objects defined in TCII template. These will serve as an example for defining queries and deleting the instances of the business object defined in your template.

22.2.1.1 Deleting IdeasDrawing dataset instances

1) Create a query to find all the item revisions which has IdeasDrawing datasets attached with Manifestation relation
    i. Launch MyTeamcenter. Switch to *Admin -> Query Builder* application
    ii. Create a query (e.g. ItemRevsWithIdeasDrawings) as shown below:

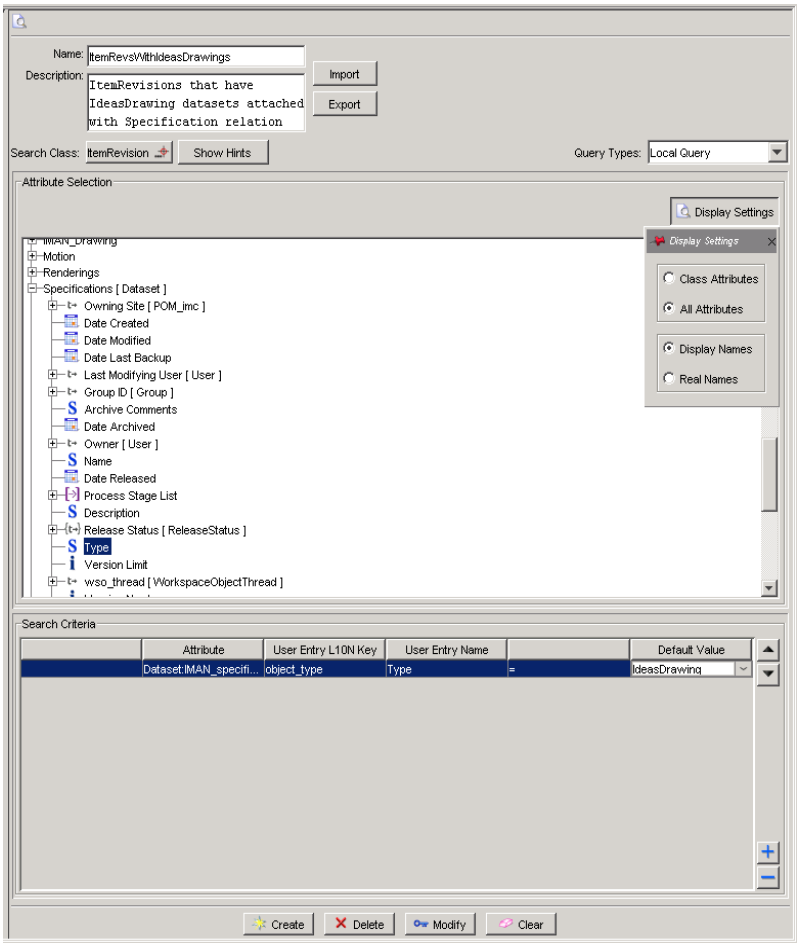


Figure 84 - Query Definition to delete IdeasDrawing dataset instances

2) Switch to *Teamcenter -> My Navigator* application, create a folder (e.g. ItemRevsWithIdeasDrawings) under NewStuff folder

3) Run the query created in step 1. This will list all the item revisions containing IdeasDrawing datasets attached with Manifestation relation. Copy all the item revisions and paste them under the folder created in step 2

4) Select all the item revisions under the ItemRevsWithIdeasDrawings folder, switch to *Details* panel, select *View -> Expand*

5) Select the all the expanded objects under the ItemRevsWithIdeasDrawings folder and sort the objects by *Type* in *Details* panel. This will list all the IdeasDrawing datasets together

6) Select all the IdeasDrawing datasets in *Details* panel and select *Edit -> Delete*

7) If the dataset deletion fails, check the error message, fix the issue and delete the dataset. Repeat this process until all datasets are deleted. Refer Section 22.2.2 for most commonly encountered issues and their corrective actions.

8) Follow the same steps as IdeasDrawing dataset for below dataset types:
    i. IdeasDrawingSet
    ii. IdeasFem
    iii. IdeasPart
    iv. IdeasAssemblyViewInfo

22.2.1.2 Deleting IdeasAssembly dataset instances

1) Launch MyTeamcenter. Create a query to find all the item revisions which has IdeasAssembly datasets attached with Specifications relation
    i. Switch to *Admin -> Query Builder* application
    ii. Create a query (e.g. ItemRevsWithIdeasAssemblies) as shown below:

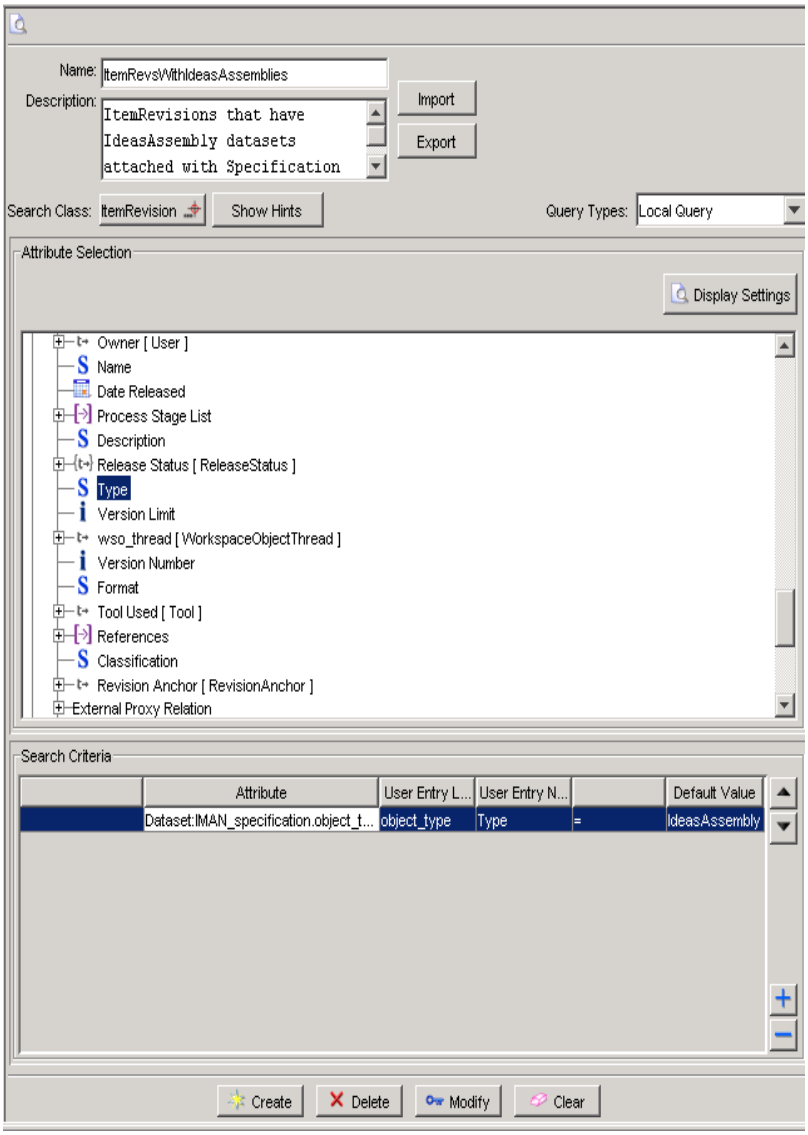


Figure 85 Query Definition to delete IdeasAssembly dataset instances

2) Switch to *Teamcenter -> My Navigator* application, create a folder (e.g. ItemRevsWithIdeasAssemblies) under NewStuff folder

3) Run the query created in step 1. This will list all the item revisions containing ItemRevsWithIdeasAssemblies datasets attached with Specifications relation. Copy all the item revisions and paste them under the folder created in step 2

4) Select all the item revisions under the ItemRevsWithIdeasAssemblies folder, switch to Details panel, and select *View -> Expand*

5) Select the all the expanded objects under the ItemRevsWithIdeasAssemblies folder and sort the objects by Type in Details panel. This will list all the IdeasAssemblies datasets together

6) Select all the IdeasAssembly datasets in Details panel and select *Edit -> Delete*. Repeat this step until two consecutive delete operations have nothing to delete

7) Perform the *General…* query with Type as IdeasAssembly. Make sure there are no objects left of this type. Delete the unreferenced objects, if any left

8) Repeat steps 6 and 7 till you delete all the datasets

9) If the dataset deletion fails, check the error message, fix the issue and delete the dataset. Refer Section 22.2.2 for most commonly encountered issues and their corrective actions.

10) Perform the *General…* query with Type as IdeasAssembly. Make sure there are no objects left of this type. Delete the unreferenced objects, if any left

22.2.1.3 Deleting NoteType instances

1) In MyTeamcenter, create a folder (e.g. AllBVRs) under NewStuff folder

2) Execute a *General…* query to find all instances of type *BOMView Revision*

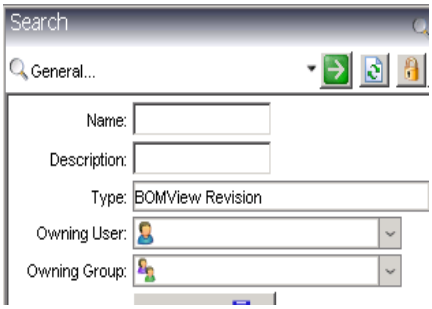


Figure 86 Query to identify BVRs

3) Copy all the BOMView Revisions and paste them under the folder created in step b

4) Double click on the BOMView Revision to launch PSM. Select *View -> Expand Below*

5) Double click on the BOMLines that have IDEAS TMTD note type in *All Notes* column

6) In the dialog, select IDEAS TMD in the Existing notes pull down menu and click *Remove* button. Click *OK*

7) Save and close the BOM Window

22.2.1.4 Deleting Ideas Type instances

1) Deleting the dataset instances should automatically delete the instance data of the following IDEAS types:
   i. Sub-types of POM_object:
      - IdeasAssemblyData
      - IdeasAuxFileRefs

- IdeasBoundingBox
- IdeasDrawingReferences
- IdeasDrawingSetRefs
- IdeasFemReference
- IdeasGuids
- IdeasMigratedItemGuid
- IdeasPartAcr
- IdeasSourceReference
- IdeasUserAttributes

ii. Sub-types of Relation:

- IDEAS_ACCONTEXT
- IDEAS_ACSOURCE
- IDEAS_DRAWING
- IDEAS_FEM
- IDEAS_SOURCE
- IDEAS_DWGSET

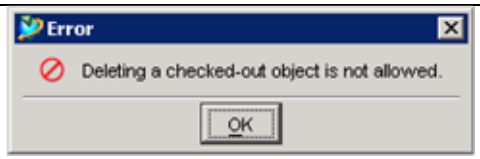
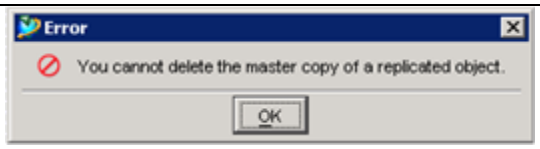
iii. Tools

- Ideas

2) After deleting the instances, search for the instances of above types and make sure there are no instances left.

22.2.2 Common error messages and Corrective actions:

Following are some of the most commonly encountered issues while deleting the business object instances and the proposed corrective actions.

Error Message	Corrective Action
	Check-in the dataset
	Delete the replicated dataset and then delete the master.

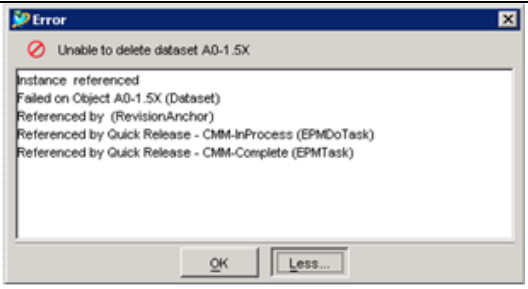
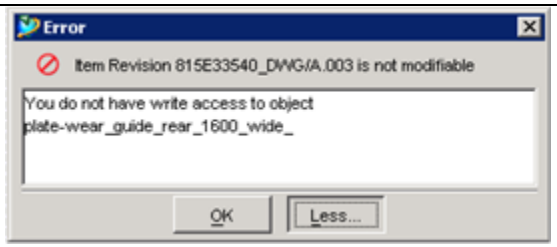
	• Select the dataset and switch to *Referencers* tab • Select the referencing workflow, *RMB -> SendTo -> Workflow* Viewer • Click on *Display the Task Attachments Panel* • Select the dataset from Targets attachments and click *Cut* • Switch back to *My Navigator*, select the dataset and click *Delete*
	• Verify the access privileges (Write) for the Item Revision • Modify/Remove the access rules through *Admin -> Access Manager* application that prohibit you to delete the dataset

Table 4 Common error messages and corrective actions

23 Codeful Extensions

The following is an example of how to add a Post Action extension rule to the “Folder” business object BMIDE. The result of adding will be that whenever a user creates an instance of the Folder business object, the code within the post action will be executed. Since this is an example, feel free to put your own code in the example where it says “Write custom business logic here”.

23.1 Adding a post-action on “Folder” BO consists of the following steps:

As an example, an extension named “***H2FolderPostAction***” will be defined and attached to business object “Folder” from BMIDE. Sample code for this extension will be provided as an example; customers have to write their business logic in the extension specific to their business needs. The following explains the steps with respect to ***H2FolderPostAction***.

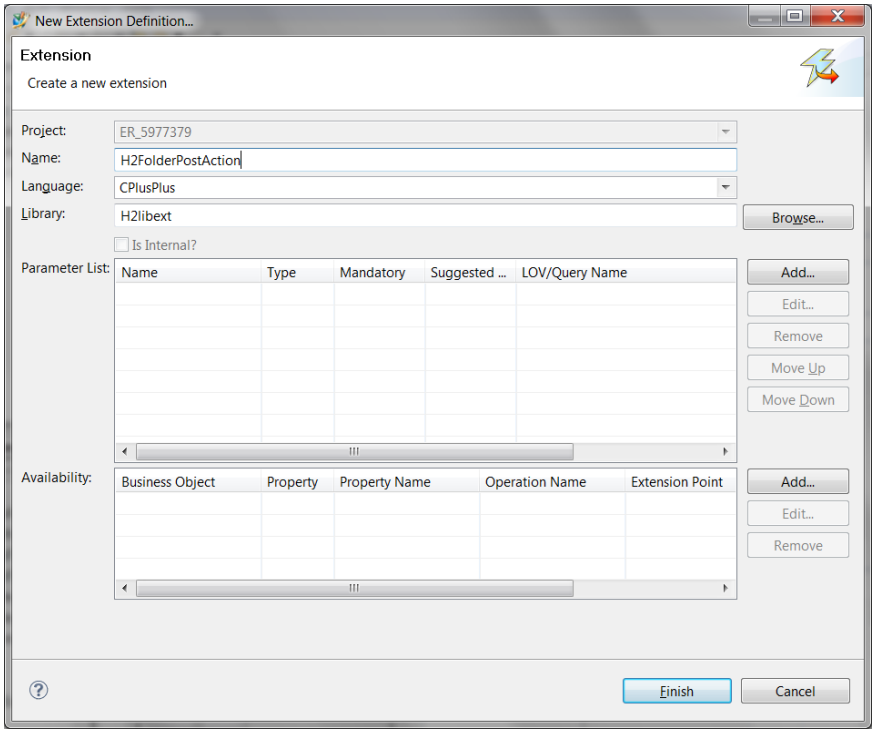
1. Defining the extension “***H2FolderPostAction”***. Since an extension(s) resides in a library, a library needs to be defined from BMIDE (see the BMIDE User Manual for details), if a custom library had already been defined in BMIDE, user can implement this extension in that library. In this example a library named “***H2libext***” has been defined from BMIDE. The extension has been defined as follows:

    a. In BMIDE, go to the Business Objects view select the project in which you want to add the extension. Right-click the project and choose Organize->Set active extension file. Select the file where you want to save the data model changes.

    b. In the BMIDE, expand the project and the **Rules→Extensions** folders.

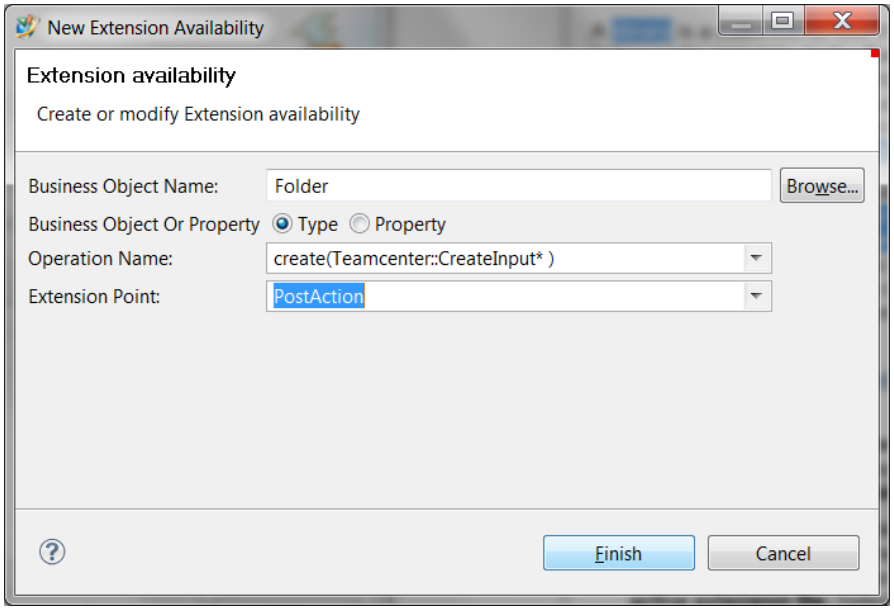
    c. Right-click the **Extensions** folder and choose **New Extension Definition**.

    The New Extension Definition wizard runs.

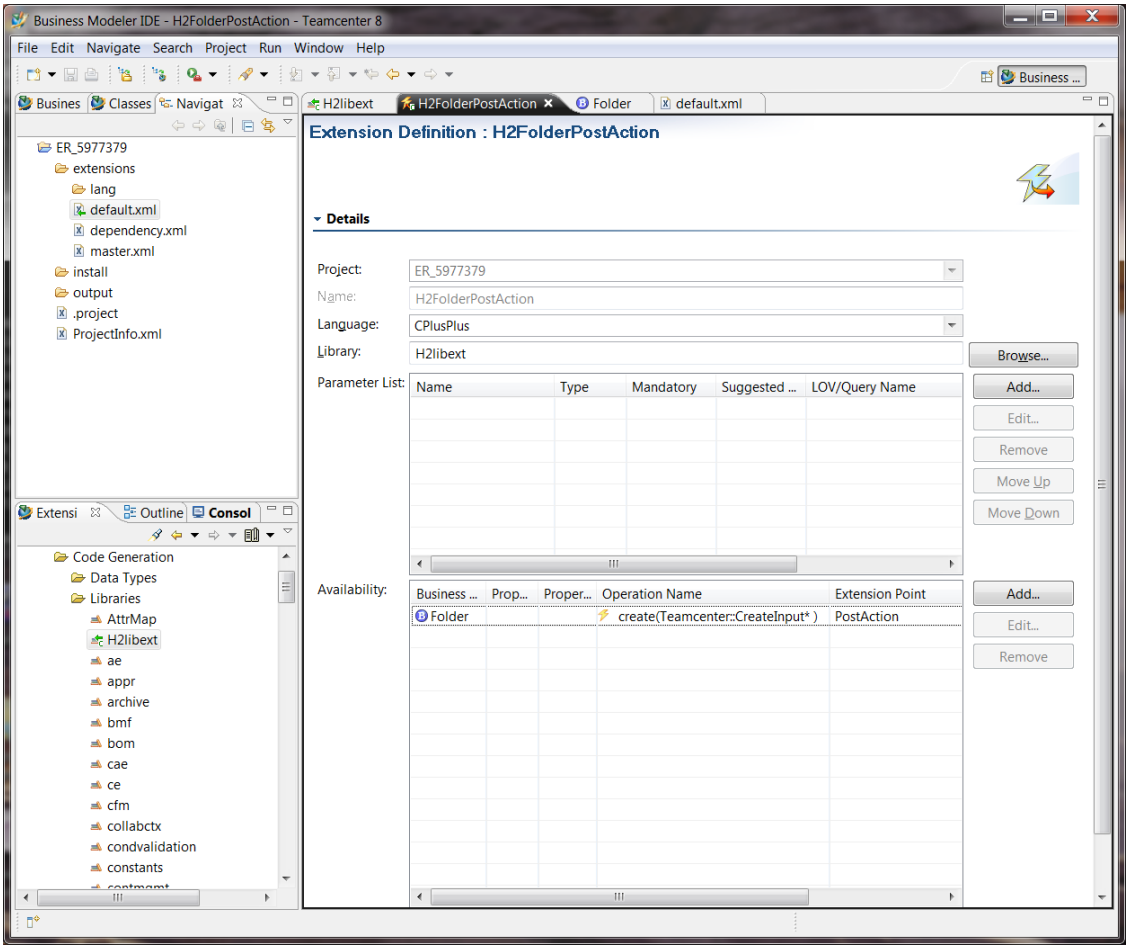


    d. Fill up the required fields

    e. In the Availability field, click “Add” button and choose the “Business Object Name”, “Operation Name” and “Extension Point”. See below.



When the “Finish” button is clicked, the extension definition will be created and the definition table will look like the following:

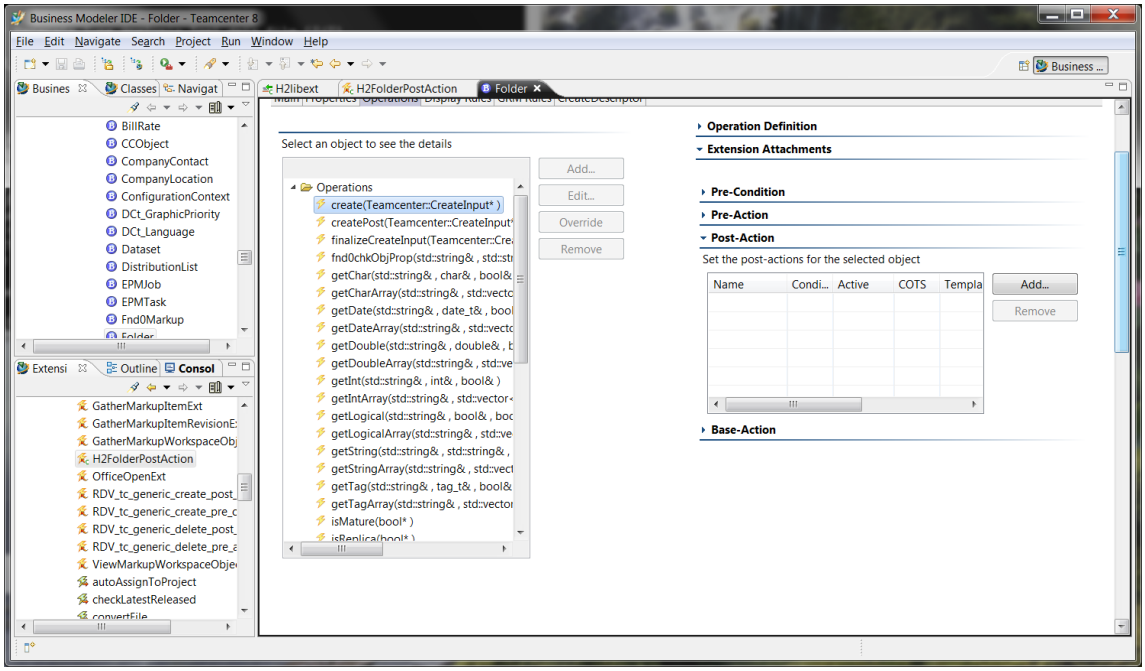


2. Attaching the extension "***H2FolderPostAction"*** to "Folder":

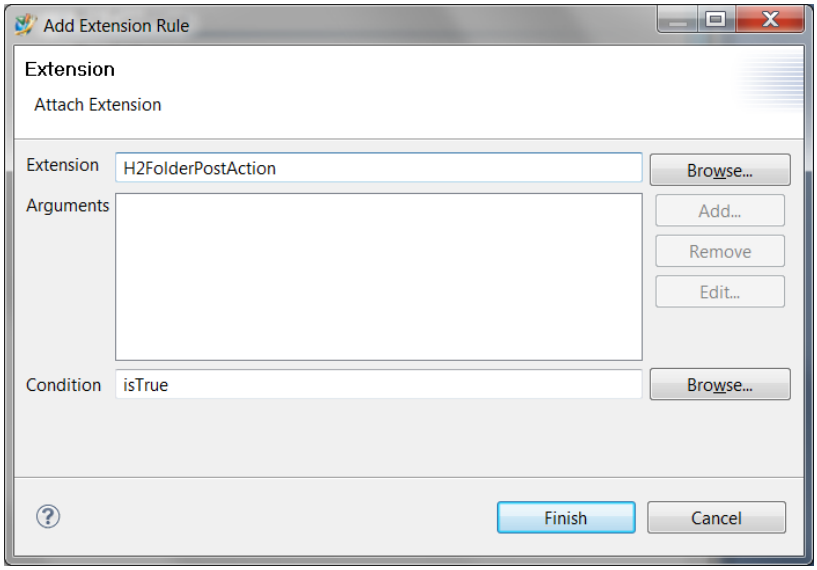
    a. In the **Business Objects** view, right-click the business object you have made available on the extension (Folder), choose **Open**,

    b. In the business object editor, click the **Operations** tab. In the operations list, find and select the operation you want to attach the post action too. Example: "create (Teamcenter::createInput*)".

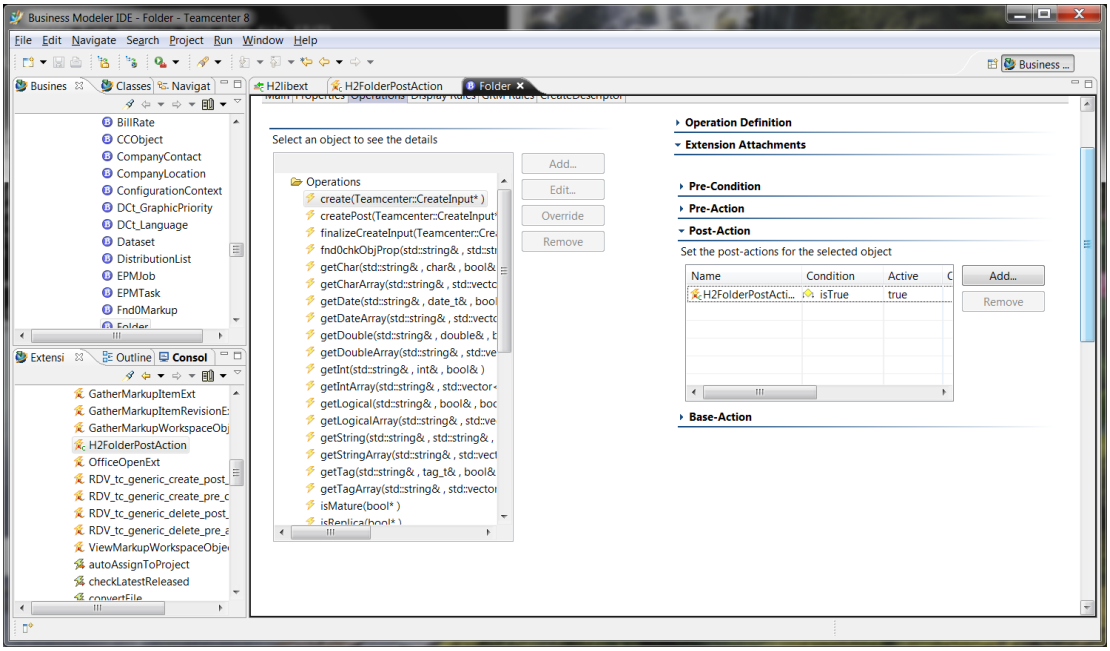
    c. In the Post Action table, click the "Add" button (see below):



d. Select "***H2FolderPostAction" extension.***



e. Hit the finish button, the Post-Action table will look the following:



Note that when File->Save Data Model is done, the xml entries will look like the following: Note that library name is saved as "***libH2libext***" *(*xml entries stores the exact name of the library without library extension (.dll or .so etc) which is different than what has defined earlier.

```
<Add>
   <TcExtension name="H2FolderPostAction" internal="false"
cannedExtension="false"
       languageType="CPlusPlus" libraryName="libH2libext" description="">
     <TcExtensionValidity
parameter="TYPE:Folder:create#Teamcenter::CreateInput,*:3"/>
   </TcExtension>
   <TcExtensionAttach extensionName="H2FolderPostAction"
operationName="create#Teamcenter::CreateInput,*"
 isActive="true" extendableElementName="Folder" extendableElementType="Type"
       extensionPointType="PostAction" conditionName="isTrue" description=""/>
</Add>
```

3. Implement the extension "***H2FolderPostAction"***

   Basically there are 2 options for placing the extension implementation files:

   a. This is a *new mechanism* where extension code can be placed and built from BMIDE See the "**Write extension code**" in BMIDE online help for details.

   b. This is a *legacy mechanism* where extension codes can be placed anywhere (not in BMIDE), use **Build the library for the example extension"** in BMIDE online help to built the library.

Sample Header file (userext.hxx):

```
#ifndef USER_EXT_H
#define USER_EXT_H

#include <method.h> // put customer specefic path of method.h
#include <libuserext_exports.h> // put customer specefic path of libuserext_exports.h
#ifdef __cplusplus

   extern "C"{
#endif

extern USER_EXT_DLL_API int H2FolderPostAction (METHOD_message_t* msg, va_list args);

#ifdef __cplusplus
}
#endif

#include <bmf/libuserext_undef.h>

#endif //USER_EXT_H
```

Sample cxx file (<filename>.cxx)

```
#include <userext.hxx>
#include <stdio.h>
#include <stdarg.h>
#include <ug_va_copy.h>
#include <CreateFunctionInvoker.hxx>
using namespace Teamcenter;

int H2FolderPostAction (METHOD_message_t* msg, va_list args)
{
        std::string  name;
        std::string description;
        bool isNull;
        // printf("\t ++++++++++++  In H2FolderPostAction()  ++++++++++\n");

        va_list local_args;
        va_copy( local_args, args);

        CreateInput *pFoldCreInput = va_arg(local_args, CreateInput*);

        pFoldCreInput->getString("object_name",name,isNull);

        pFoldCreInput->getString("object_desc",description,isNull);
        // Write custom business logic here

   va_end( local_args );

   return 0;
}
```

4. Build the library (*libH2libext.dll)* for Extension "***H2FolderPostAction"***
   a. Use the "**Write extension code**" in BMIDE online help to build the library if extension codes have been placed in BMIDE. "**Build the library for the example extension"** (not from BMIDE) can also be used to build the library in legacy way.

b. Follow **"Build the library for the example extension"** in BMIDE online help if extension code has been placed in legacy way.

Run the "dumpbin" (win) or "nm" (UNIX) to see if the "H2FolderPostAction" extension symbol has been exported correctly. The symbol should not be mangled (C++).

dumpbin /EXPORTS <path_of_libH2libextr>*libH2libext.dll*

The output should look similar to the following:

71 00002BE0 H2FolderPostAction = _H2FolderPostAction

Put the library *libH2libext.dll* in %TC_ROOT%\bin or from RAC, set the **BMF_CUSTOM_IMPLEMENTOR_PATH** to the path of the library (see Teamcenter Help Documentation for details for this preference).

5. Deploy your extension definition to your database by deploying your custom template "***H2FolderPostAction"*** (use BMIDE User Guide for more details)
6. Test the extension "***H2FolderPostAction"*** by creating a folder object in RAC. The extension will be executed.

23.2 Relation Navigation from Secondary to Primary Object (Custom Runtime Property):

Problem: It is possible to navigate from primary object to secondary object through relation properties from RAC/Thin client's properties panels or from summery view. Out of the box it is not possible to navigate from secondary object to primary object similar to relation properties. This custom solution provides a way to navigate from secondary object to primary objects through customization.

Example: ItemRevision (object IR001) is attached with Document (object Doc001) with IMAN_specification relation. Currently in RAC/web client summary/ properties page it is possible to navigate from IR001 to Doc001 through IMAN_sepcification relation properties. It is **not** possible to navigate from Doc001 to IR001.

Solution: This solution consists of the following steps:
Primary Business Object (say Item) is related to secondary Business Object (say Document) using relation **H2MyRelation** (Custom Relation Business Object).
Note: This sample code assumes that cardinality is 1:1; for any other cardinality like (*:1, * :*) use similar pattern but appropriate property messages.

i) Add a Custom Runtime Property:

Use BMIDE to attach a run time property say **h2myRelationToPrimary (<RelationName>ToPrimary)** on **Document** business object. Save the data model. Note this is a just a convention used in the sample code and not a mandatory naming convention.

ii) Attach custom operation and extensions manually:

Currently, BMIDE UI does not allow the customer to define the getter operation for a custom property defined on a COTS business object. To overcome this limitation, the customer can manually add the following BMIDE elements into the template file to define the getter operation for the custom property. Add following lines manually in xml file of custom template file.

```
<TcOperationAttach operationName="PROP_ask_value_tag" extendableElementName="Document"
   extendableElementType="Type" propertyName="h2MyRelationToPrimary"
```

```
        description="">
    <TcExtensionPoint extensionPointType="PreCondition" isOverridable="true"/>
    <TcExtensionPoint extensionPointType="PreAction" isOverridable="true"/>
    <TcExtensionPoint extensionPointType="BaseAction" isOverridable="false"/>
    <TcExtensionPoint extensionPointType="PostAction" isOverridable="true"/>
</TcOperationAttach>

<TcExtension name="getH2MyRelationToPrimary" internal="false"
 cannedExtension="false" languageType="CPlusPlus" libraryName="myLibrary"  description="">
 <TcExtensionValidity
   parameter="PROPERTY:Document:h2MyRelationToPrimary:PROP_ask_value_tag:4"/>
</TcExtension>

<TcExtensionAttach extensionName="getH2MyRelationToPrimary"
 operationName="PROP_ask_value_tag" isActive="true" propertyName="h2MyRelationToPrimary"
 extendableElementName="Document" extendableElementType="Type"
 extensionPointType="BaseAction" conditionName="isTrue" description=""/>
```

Reload BMIDE custom project to make sure there are no typographical errors in console. Deploy the template.

iii) Implement getter C++ function for h2MyRelationToPrimary property:

The getter function for a property operation is declared in the following manner to avoid C++ name mangling. In this example, the library name is "**myLibrary**"

```
#ifdef __cplusplus
extern "C"{
#endif

      extern  MY_API int getH2MyRelationToPrimary(
            METHOD_message_t * msg /* <I> */,
            va_list  args /* <I> */);

#ifdef __cplusplus
}
#endif
```

The following is an example code snippet for the function "`getH2MyRelationToPrimary`", which is expected to return the tag of primary business object.

```
int getH2MyRelationToPrimary (
            METHOD_message_t * message,  va_list  args )
{
    int ifail = ITK_ok;

    tag_t   secondaryTag  = message->object_tag;

    va_list         largs;
    va_copy( largs, args);
    va_arg( largs, tag_t );
    tag_t*  primaryTag  = va_arg( largs, tag_t*);
    va_end( largs );

    if( secondaryTag != NULLTAG )
    {
       static tag_t relType = NULLTAG;
       if( relType == NULLTAG )
       {
            // include tctype.h for following ITK
```

```
            ifail = TCTYPE_find_type("H2MyRelation", 0, &relType);
            if( ifail != ITK_ok ) return ifail;
        }
        if( relType != NULTAG )
        {
           int count = 0;
           // include grm.h for following structure and ITK
           GRM_relation_t * primaries = 0;
           ifail = GRM_list_primary_objects(secondaryTag , relType,
                                                 &count, &primaries);
           if( ifail != ITK_ok ) return ifail;
           if(count > 0)
           {
               *primaryTag = primaries[0].primary;
           }
           SM_free( primaries );
         }// end if( relType != NULTAG )
    }// end if( secondaryTag != NULLTAG )
    return ifail;
}
```