TEAMCENTER

BMIDE for Data Model Design

Teamcenter 2412

SIEMENS

Unpublished work. © 2025 Siemens

This Documentation contains trade secrets or otherwise confidential information owned by Siemens Industry Software Inc. or its affiliates (collectively, “Siemens”), or its licensors. Access to and use of this Documentation is strictly limited as set forth in Customer’s applicable agreement(s) with Siemens. This Documentation may not be copied, distributed, or otherwise disclosed by Customer without the express written permission of Siemens, and may not be used in any way not expressly authorized by Siemens.

This Documentation is for information and instruction purposes. Siemens reserves the right to make changes in specifications and other information contained in this Documentation without prior notice, and the reader should, in all cases, consult Siemens to determine whether any changes have been made.

No representation or other affirmation of fact contained in this Documentation shall be deemed to be a warranty or give rise to any liability of Siemens whatsoever.

If you have a signed license agreement with Siemens for the product with which this Documentation will be used, your use of this Documentation is subject to the scope of license and the software protection and security provisions of that agreement. If you do not have such a signed license agreement, your use is subject to the Siemens Universal Customer Agreement, which may be viewed at **https://www.sw.siemens.com/en-US/sw-terms/base/uca/**, as supplemented by the product specific terms which may be viewed at **https://www.sw.siemens.com/en-US/sw-terms/supplements/**.

SIEMENS MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS DOCUMENTATION INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF INTELLECTUAL PROPERTY. SIEMENS SHALL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, CONSEQUENTIAL OR PUNITIVE DAMAGES, LOST DATA OR PROFITS, EVEN IF SUCH DAMAGES WERE FORESEEABLE, ARISING OUT OF OR RELATED TO THIS DOCUMENTATION OR THE INFORMATION CONTAINED IN IT, EVEN IF SIEMENS HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

TRADEMARKS: The trademarks, logos, and service marks (collectively, "Marks") used herein are the property of Siemens or other parties. No one is permitted to use these Marks without the prior written consent of Siemens or the owner of the Marks, as applicable. The use herein of third party Marks is not an attempt to indicate Siemens as a source of a product, but is intended to indicate a product from, or associated with, a particular third party. A list of Siemens’ Marks may be viewed at: **www.plm.automation.siemens.com/global/en/legal/trademarks.html**. The registered trademark Linux® is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a world-wide basis.

About Siemens Digital Industries Software

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

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

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

Contents

Your data model and the BMIDE 1-1

Basic tasks and sequence for configuring a business data model 2-1

Start the Business Modeler IDE 3-1

Install the Business Modeler IDE

Prerequisites for installing the Business Modeler IDE 4-1
Choose a Business Modeler IDE installation type 4-2
Install the Business Modeler IDE using TEM 4-3
Install the Business Modeler IDE using Deployment Center - connected 4-7
Install the Business Modeler IDE using Deployment Center - standalone 4-9
Add the Business Modeler IDE to an existing Eclipse SDK environment 4-13
Allocate memory to the Business Modeler IDE 4-16
Uninstall a Business Modeler IDE 4-17

Configure the Business Modeler IDE

Extension files 5-1
- Introduction to extension files 5-1
- Add an extension file 5-1
- Set the active extension file 5-2
- Move a custom element 5-3

Set Business Modeler IDE preferences 5-6
Add a server connection profile 5-9
Login to a Teamcenter server 5-11
Back up project data 5-12

Upgrade the Business Modeler IDE

Business Modeler IDE upgrade process 6-1
Upgrade a template project to the current data model format 6-1
Refactor postactions on create operations 6-3

Learning about the Business Modeler IDE

Business Modeler IDE interface 7-1
- Business Modeler IDE interface overview 7-1
- Business Modeler IDE menu commands 7-3
- Set favorites in the Business Modeler IDE 7-9
- Filter visible elements in the Business Modeler IDE 7-10

Business Modeler IDE workshops 7-11
- What are the BMIDE workshops? 7-11

© 2025 Siemens

Prerequisites for the BMIDE workshops 7-12
Workshop 1: Create a template project 7-12
Workshop 2: Explore the user interface 7-14
Workshop 3: Create a new item type 7-17
Workshop 4: Create custom properties 7-21
Workshop 5: Display custom properties in the client user interface 7-26
Workshop 6: Create lists of values 7-34
Workshop 7: Add a naming rule 7-41
Workshop 8: Create a form 7-45
Workshop 9: Add a relationship rule 7-49
Workshop 10: Add deep copy rules 7-53
Workshop 11: Add a business object display rule 7-59
Workshop 12: Add a predefined extension rule 7-65
Workshop 13: Add a compound property 7-69
Workshop 14: Save your data model changes 7-78
Business Modeler IDE process 7-79
Business Modeler IDE process overview 7-79
Develop and test extensions 7-79
Deploy a template to a production site 7-81
Edit extensions in a live production site 7-82
Control Business Modeler IDE elements that can be updated live 7-82
Edit live data 7-83
Incorporate live data updates from the production site 7-83
Development environments 7-85
Single production database environment 7-85
Test and production database environment 7-85
User testing environment 7-86
Multiple developer environment 7-86
SCM system 7-88
Using a source control management (SCM) system to manage files 7-88
Selecting a source control management (SCM) system 7-88
Getting started with a source control management system (SCM) 7-89
Business Modeler IDE help 7-91
Accessing help in the Business Modeler IDE 7-91
Configure help on Linux systems 7-91
Add a topic to the Business Modeler IDE online help 7-92
Create a new online help book in the Business Modeler IDE 7-93
Serve your custom help to Business Modeler IDE users 7-96

Data model concepts

Teamcenter data model overview 8-1
POM schema 8-2
Viewing POM schema 8-4
Schema versus non-schema objects 8-6
Class structure and attribute inheritance 8-7

Managing the data model

© 2025 Siemens

Find objects in the Business Modeler IDE 9-1
Open objects in the Business Modeler IDE 9-1
Delete objects in the Business Modeler IDE 9-1
Modifying objects in the Business Modeler IDE 9-2
Deprecating objects in the Business Modeler IDE 9-2
Naming objects in the Business Modeler IDE 9-3
Reloading the data model 9-4
Import and export the data model 9-4
Import a Business Modeler IDE template package 9-4
Import a Business Modeler IDE template project 9-7
Import a backed-up project 9-9
Import project preferences 9-13
Import a template file 9-14
Import a live update project 9-16
Export project preferences 9-17
Export a TC XML schema file 9-19
Data model reports 9-21
Run data model reports 9-21
Run a report comparing the data model from two different sources 9-22
Run a condition usage report 9-25
Run a report on create operation overrides 9-26
Run a report of individual types of data model 9-27
Run a report of all data model 9-29
Run a property group usage report 9-30
Graphically represent the data model in the UML editor 9-32
Introduction to the UML editor 9-32
Open a class or business object in the UML editor 9-32
Create a new UML diagram 9-34

Creating, deploying, and packaging templates

Introduction to templates 10-1
Understanding custom versus COTS templates 10-2
Template process in the Business Modeler IDE 10-2
Create a Business Modeler IDE template project 10-3
Enable admin data in an existing project 10-9
Add admin data to project 10-10
Create a Business Modeler IDE composite software project 10-11
Add a Business Modeler IDE project template to a composite software project 10-13
Deploying templates 10-13
Introduction to deploying templates 10-13
Deploy a template 10-14
Retrieve deployment archive files 10-18
Generate a software package for distribution 10-19
Versioning software packages 10-22
Install a template using TEM 10-25
Update a template using TEM 10-28

© 2025 Siemens

Install or update a template using the tem utility 10-30
Live updates 10-31
Introduction to live updates 10-31
Single administrator versus multiple administrators in a live updates environment 10-32
Live updates for a single administrator 10-34
Live updates for multiple administrators 10-49
Localization and live update 10-51
Live updates reference 10-52
Merge samples 10-59
Basic template tasks 10-59
Add a template to Business Modeler IDE reference templates 10-59
Add a dependent template to a Business Modeler IDE project 10-61
View Business Modeler IDE template project properties 10-62
Edit the template feature file 10-69
Push a template to the reference directory 10-70
Improving performance by closing and opening projects 10-71
Deleting projects 10-71
Templates reference 10-71
Templates overview 10-71
Creating a template project 10-72
Aligning Siemens Digital Industries Software template development and customer template development 10-72
Adding extensions to the template 10-73
Synchronize all data directories with the latest templates 10-73
Remove a template 10-74
Files in a template project 10-82
Templates installation reference 10-86
Template artifacts 10-99

Creating data model elements

Add a new model element 11-1
Business Modeler IDE administration tasks 11-2
Business objects 11-3
Create business objects 11-3
Business object icons 11-46
Supporting create, save as, and revise operations in clients 11-59
Business object constants 11-71
Changing the name of custom business objects 11-77
Convert secondary business objects to primary 11-86
TC_WorkContext business object reference 11-88
Classes 11-89
Introduction to creating classes 11-89
Add a new class 11-90
Class attributes 11-93
Properties 11-101
Introduction to properties 11-101
Properties table in the BMIDE 11-102

© 2025 Siemens

Hide properties on a form business object 11-106
Add properties to a custom form business object 11-107
Add properties 11-108
Property constants 11-136
Compound properties 11-141
Atrribute groups 11-145
Using property formatters to configure display format in the Active Workspace 11-153
Lists of values 11-167
What are lists of values (LOVs)? 11-167
Create classic lists of values 11-168
Batch LOVs 11-172
Dynamic LOVs 11-177
Attach an LOV to a property 11-209
Add values to an existing classic LOV 11-212
Create a filter LOV 11-212
Create a cascading LOV 11-214
Create an interdependent LOV attachment 11-215
Attaching LOVs with conditions 11-219
Attaching LOVs conditionally based on object properties 11-224
Display LOVs based on a project 11-226
LOV value types 11-231
LOV usage types 11-232
Special considerations for LOVs 11-232
Creating options 11-233
About the Options folder 11-233
Add an ID context 11-233
Note types 11-234
Add an occurrence type 11-236
View types 11-237
Add a status type 11-239
Add a storage media option 11-242
Add a tool 11-243
Linked Data Framework 11-245
Introduction to Linked Data Framework 11-245
Create a new service catalog 11-247
Create a new Linked Data Framework service 11-249
Create a Linked Data Framework service operation 11-252
Working with applications 11-258
Teamcenter Component objects 11-260
What is a Teamcenter Component object? 11-260
Create a Teamcenter Component object 11-261
Create a verification rule 11-264
Global constants 11-265
What are global constants? 11-265
Create a global constant 11-266
Change the value of a global constant 11-268

Creating business rules

© 2025 Siemens

Introduction to business rules 12-1
Inheritance of business object rules 12-1
Naming rules 12-1
Introduction to naming rules 12-1
Add a naming rule 12-2
Add a revision naming rule 12-5
Assign a baseline suffix naming rule 12-9
Attach a naming rule to a property 12-9
Attach naming rules with conditions 12-13
Override a COTS naming rule 12-14
Naming rule patterns 12-15
Naming rules reference 12-22
Business object display rules 12-29
Add a business object display rule 12-29
Business object display rules reference 12-33
Generic Relationship Manager (GRM) rules 12-34
Introduction to GRM rules 12-34
Add a GRM rule 12-36
Generic Relationship Manager 12-42
Evaluation order for GRM rules 12-42
Impact of business object inheritance on GRM rules 12-42
Maintain stable relation IDs 12-43
Deep copy rules 12-44
Add a deep copy rule 12-44
Understanding the impact of inheritance on deep copy rule behavior 12-50
Restrictions on the use of deep copy rules 12-52
Deep copy API 12-53
Deep copy rule conditions 12-54
Propagation rules 12-64
Propagating security-related property values 12-64
Example of property value propagation 12-65
Propagation groups and styles 12-66
Create a propagation rule 12-72
Configure a property for propagation 12-77
Propagate from Item to BOM View and from Item Revision to BOM View Revision 12-80
Disable propagation of a property 12-81
Identify and fix propagation rules with potential traversal risk 12-81
Propagation rules preferences 12-85
Migrate propagation preference rules 12-86
Extension rules 12-87
Introduction to application extensions 12-87
Add an application extension point 12-88
Add an application extension rule 12-92
Add a business context 12-96
Sample application extension APIs 12-98
Attaching extensions to operations 12-105
Alternate ID rules 12-126

© 2025 Siemens

Add an alternate ID rule 12-126
Alternate ID rule example 12-131
Alternate ID rules characteristics 12-134
Alias ID rules 12-135
Add an alias ID rule 12-135
Alias ID rules reference 12-137
Intelligent part numbering 12-138
Intelligent part numbering use case and example 12-138
Configure intelligent part numbering for a Create operation 12-144
Configure intelligent part numbering for a Save As operation 12-149
Multifield keys 12-151
Introduction to multifield keys 12-151
Creating multifield key definitions 12-152
Multifield key domains 12-155
Creating objects with the same item ID 12-156
Managing multifield keys 12-161
Analyzing multifield keys 12-162
Configure the displayed name of business object instances 12-165
Considerations for using multifield keys 12-168
Administration data candidate keys 12-170
Introduction to administration data candidate keys 12-170
Create administration data candidate keys 12-172
Conditions 12-174
Conditions overview 12-174
Add a condition 12-177
Search conditions 12-182
Condition examples 12-183
Condition system 12-188

Setting display names

What is a display name? 13-1
Change display names for business objects and option types 13-2
Set display names for properties 13-4
Set display names for lists of values (LOVs) 13-4
Validating localizations 13-5

Configure Teamcenter applications

Access Manager 14-1
Configure Access Manager using the Business Modeler IDE 14-1
Create a custom privilege 14-1
ADA License 14-2
Configure ADA License using the Business Modeler IDE 14-2
Add ADA License categories 14-2
Aerospace and Defense 14-7
Configure Aerospace and Defense using the Business Modeler IDE 14-7
Aerospace and Defense business objects 14-7
Audit Manager 14-9

© 2025 Siemens

Configure Audit Manager using the Business Modeler IDE 14-9
Create an event type 14-10
Create an event type mapping 14-11
Enable auditing workflow release status objects 14-12
Create an audit definition 14-13
Post-upgrade steps required for importing custom event types into a template project 14-18
Audit Manager data model objects 14-18
Automotive Edition 14-20
Configure Automotive Edition using the Business Modeler IDE 14-20
CAE Manager 14-20
Configure CAE Manager using the Business Modeler IDE 14-20
Change Management 14-20
Configure Change Management using the Business Modeler IDE 14-20
Change Management business objects 14-22
Create a new relation for use with a Change Management pseudofolder 14-25
Add a custom naming rule to standard Change Management objects 14-30
Change Management conditions 14-31
Defining deep copy rules for creating changes from another change 14-31
Define deep copy rules for copying options from an ECR to an ECN 14-32
Restrict actions based on the type of applied change object 14-34
Classification 14-36
Configure Classification using the Business Modeler IDE 14-36
Classification extensions 14-36
Service Lifecycle Management (SLM) 14-38
Configure Service Lifecycle Management (SLM) using the Business Modeler IDE 14-38
Consideration when creating a custom physical part 14-38
Manufacturing Process Planner 14-39
Configure Manufacturing Process Planner using the Business Modeler IDE 14-39
3D PDF business objects 14-39
Enable multi-unit configuration 14-40
Multi-Structure Manager 14-41
Configure Multi-Structure Manager using the Business Modeler IDE 14-41
Configure the automateAndLink extension 14-41
Using conditions with the automateAndLink extension 14-50
Create a condition asking whether to create a part 14-51
NX 14-57
Creating item types with required attributes for NX 14-57
Configure NX CAM Integration using the Business Modeler IDE 14-57
Organization 14-57
Configure Organization using the Business Modeler IDE 14-57
Add additional properties to users 14-57
Product and manufacturing information (PMI) 14-60
Configure product and manufacturing information (PMI) using the Business Modeler IDE 14-60
Schedule Manager 14-61
Configure Schedule Manager using the Business Modeler IDE 14-61
Create a custom status for Schedule Manager 14-61

© 2025 Siemens

Schedule Manager operations, extensions, and conditions used for statuses 14-64
Schedule Manager property operations 14-65
Schedule Manager extensions 14-67
Subscriptions 14-68
Configure subscription conditions using the Business Modeler IDE 14-68
Supplier Relationship Management 14-69
Configure Supplier Relationship Management using the Business Modeler IDE 14-69
Configure lists of values (LOVs) for Supplier Relationship Management 14-70
Systems Engineering 14-75
Configure Systems Engineering using the Business Modeler IDE 14-75
Add an application domain for diagrams 14-75
Creating an icon for use in the Systems Engineering and Requirements Management BOM view 14-79
Validation Manager 14-80
Configure Validation Manager using the Business Modeler IDE 14-80
Validation Manager business objects 14-80
Validation Manager properties 14-81
Wiring Harness Design Tools Integration 14-83
Configure Wiring Harness Design Tools Integration using the Business Modeler IDE 14-83
Workflow Designer 14-83
Configure Workflow Designer using the Business Modeler IDE 14-83
Create dynamic participants 14-84
Register custom workflow handlers 14-88
Use conditions to filter workflow template availability 14-94
Create a custom form that supports setting security classification in a workflow 14-97
Create a custom form that supports assigning project members in a workflow 14-99
Create a custom form that supports assigning projects in a workflow 14-100
Create a dynamic LOV to display project IDs 14-101
Add custom decision labels 14-103

Troubleshooting the Business Modeler IDE

BMIDE related log files 15-1
Deployment errors 15-1
Check the deployment log 15-1
Check your deployment setup 15-1
Cannot connect to the server error 15-2
Class is referenced error 15-3
Incompatible argument error 15-3
Deployment fails when business object names do not contain USASCII7 characters 15-4
Time zone error 15-4
Incorporate Latest Live Update Changes error 15-5
Business Modeler IDE has slow performance, an out-of-memory error, or does not launch 15-5
Could not create the Java virtual machine error 15-6
Workspace is locked error 15-7
Type name collision error 15-7

© 2025 Siemens

Backup and recovery following failed template deployment 15-8
Localized property with default value changes its master locale setting after transfer to a remote site 15-9
BASE-10001: ENCODING_VALIDATION_ERROR 15-9
Correct a transposed interdependent LOV attachment 15-12
FND-10001: MULTIPLE_INTERDEPENDENT_LOV_ATTACHMENT_ERROR 15-13
FND10002: INVALID_CONDITION_FOR_DEEP_COPY_RULE_ERROR 15-14
FND10003: THE_ELEMENT_HAS_BEEN_REMOVED_FROM_ITS _DEPENDENT_TEMPLATE 15-14
FND10004: CANNOT_SET_LOCALIZABLE_CONSTANT_ATTACHMENT_ERROR 15-15
Applying condition rules for custom change objects after a patch or upgrade 15-15

Business Modeler IDE reference

Business Modeler IDE utilities A-1
Business Modeler IDE utilities A-1
Obsolete utilities and APIs A-1
Business Modeler IDE perspectives, views, and editors A-2
Standard perspective A-2
Advanced perspective A-4
Business Modeler IDE editors A-7
Eclipse views used by the Business Modeler IDE A-38

© 2025 Siemens

1. Your data model and the BMIDE



Your business data model is the crux of your PLM system. A good model leverages product data, articulates business processes, and identifies how different groups can collaborate. Creating an effective model is key to realizing the benefits of PLM. These benefits include:

- Increased productivity
- Cost management
- Reduced risk
- Accelerated time to market
- Data sharing
- Increased revenue

In Teamcenter the data model is a structured, hierarchical mapping of objects. The model is built from template files that define the objects in a given Teamcenter application (also known as a solution). For example, the base Teamcenter application is the Foundation solution, and its objects are defined in the **foundation_template.xml** file.

Administrators, business analysts, and developers load relevant templates into the Business Modeler Integrated Development Environment (BMIDE) as needed to configure, extend, and maintain your data model so that your Teamcenter environment can accurately model your products and processes.

Configuring objects versus configuring operations

At a high level, the tasks related to configuring the Teamcenter data model fall into two categories:

High level task	Where can I find help?
Configure or add data model objects to the data model.	This kind of task does not require writing code, and is described here in *BMIDE for Data Model Design*.
Change or add core Teamcenter behavior.	This kind of customization task requires writing C or C++ code extensions to existing operations, or writing new operations, and is described in *Teamcenter Customization — Getting Started*.

Deploying template changes into a Teamcenter environment

When changes to templates are made in a BMIDE project, best practice is to first deploy the changes to a test Teamcenter environment where the developer or administrator can verify that the environment behaves as intended. After successful completion of testing, the developer or administrator can

© 2025 Siemens

package the BMIDE project for distribution. The package can be installed into a production Teamcenter environment using Teamcenter Environment Manager (TEM) or Deployment Center.

© 2025 Siemens

2. Basic tasks and sequence for configuring a business data model

The following sequence lists the basic tasks performed in a Business Modeler IDE during configuration of a business data model.

1. **Create a template project to hold your custom data model**.

   A BMIDE project manages your Teamcenter data model configuration and extensions. The project contains folders and files that are used to organize your template XML files and to package your template for deployment.

2. Create data model objects to represent objects in Teamcenter.

   A typical starting point is to create new business objects as children of the COTS **Part** business object to represent your product parts, and children of the **Design** business object to represent designs.

   Additional information:

   - **Create business objects**.
   - **Create properties on the business objects**.
   - **Create lists of values for setting object property values**.
   - **Create rules for the business objects**.

3. Save your work.

   Choose **BMIDE**>**Save Data Model**, or on the main toolbar click **Save Data Model**  .

   > Note:
   >
   > To check for data model errors, right-click your project and select **Reload Data Model**. See the **Console** view for errors.

4. Verify your extensions by deploying them to a Teamcenter test server.

   Choose **BMIDE**>**Deploy Template** on the menu bar.

5. After you verify your extensions, you can **package your data model into a template** that can be installed on a production server.

© 2025 Siemens

6. **Install the template to a production server**.

© 2025 Siemens

3. Start the Business Modeler IDE

Start a Business Modeler IDE in one of several ways, depending on the installation type:

Installation type	Platform	Procedure to start Business Modeler IDE
BMIDE Standalone, 2-tier, or 4-tier	Windows	Click the **Start** button and choose **All Programs**>**Teamcenter** [*version*]>**Business Modeler IDE**. This runs the **bmide.bat** file.
	Linux	Run the **bmide.sh** file in the *install-location*/**bmide/client** directory.
Eclipse environment to which BMIDE plug-ins have been added	Windows	Navigate to the directory where Eclipse is installed and execute the **Eclipse.exe** command. `Eclipse.exe -vmargs -Xmx2024M` To ensure that you have enough memory to run Eclipse, run the command with a virtual memory argument. In the example, the argument increases virtual memory to 2 GB.
	Linux	Navigate to the directory where Eclipse is installed and execute the **Eclipse** command. `Eclipse -vmargs -Xmx2024M`

For BMIDE operations that require connection to the Teamcenter server, users of the BMIDE must be members of the Teamcenter database administrators (**dba**) group. To add a user to the **dba** group, in the Teamcenter rich client use the Organization perspective.

If a perspective fails to open, it could be that not enough memory is allocated to the Business Modeler IDE.

© 2025 Siemens

© 2025 Siemens

4. Install the Business Modeler IDE

Prerequisites for installing the Business Modeler IDE

For certified versions of software mentioned in the following list, see the Hardware and Software Certifications knowledge base article on Support Center.

Supported operating system

Business Modeler Integrated Development Environment (BMIDE) is supported on Microsoft Windows, SUSE Linux, and Red Hat Linux.

Memory

One GB of RAM dedicated to BMIDE. If you perform live updates, then the system must have a minimum of 2 GB of RAM.

After installing BMIDE, you **allocate memory to BMIDE** in a **BMIDE_SCRIPT_ARGS** environment variable and in the *[Teamcenter install location]***/bmide/client/BusinessModelerIDE.ini** file.

Eclipse

An existing Eclipse installation is a prerequisite only if you intend to **add BMIDE plugins to your existing Eclipse SDK environment**.

Java

Java Development Kit (JDK) is required if you want to create Teamcenter services. In that case, before installing BMIDE, set the environment variable **JDK_HOME**.

Example:

```
JDK_HOME=C:\Program Files\Java\jdk-10.0.1
```

Compiler

A C++ compiler for compiling code.

When using Windows, install Microsoft Visual Studio with the **Microsoft Foundation Classes for C++** option. After installing BMIDE, edit the **bmide.bat** file and add a call to the Visual Studio **vcvarsall.bat** file. Place the call before the **PATH** statement.

Example:

```
call "C:\apps\MVS12\VC\vcvarsall.bat" x64
set PATH=%JDK_HOME%\bin;%JRE_HOME%\bin;TC_ROOT\lib;
%FMS_HOME%\bin;%FMS_HOME%\lib;%PATH%;
```


© 2025 Siemens

Choose a Business Modeler IDE installation type

Several types of Business Modeler IDE installation are possible. All BMIDE installation types can be used to create, import, and modify a template project, and can generate a template package which can be deployed using TEM or Deployment Center.

An important difference among the installation types is whether and how the BMIDE connects to a Teamcenter site. A Teamcenter site connection is necessary for some tasks:

Perform data exchanges, such as:

- Synchronize the data model in a BMIDE template project with the Teamcenter server database.
- Live update non-schema data, such as lists of values (LOVs), from the BMIDE to a production server without shutting down the production server.
- Live deploy a template to a test Teamcenter server.
- Incorporate live update changes made to the production environment into a BMIDE standard template project.

Create certain data model elements, such as:

- Business object display rule
- Dynamic list of values
- Business context rule
- Item revision definition configuration (IRDC)
- System stamp configuration
- Subtype of AppInterface, and many others

Use the following general procedure for choosing a Business Modeler IDE installation type.

1. Ensure that the machine meets prerequisites for a BMIDE.

   > Caution:
   >
   > Do not install BMIDE on a production environment corporate server. Doing so could have unintended consequences, especially during Teamcenter upgrade.

2. Choose the BMIDE installation type that you want to perform.

© 2025 Siemens

Installation type	Teamcenter connection type	Advantage	Limitation
2-tier	Two-tier environment via TCCS.	Allows live deployments even while a web tier is inactive or down for maintenance.	Requires local network access.
4-tier	Four-tier environment via HTTP server.	Allows remote access and live deployments.	Requires an active web tier.
Standalone	None	No requirement for or possibility of unintentional interaction with any Teamcenter site.	Cannot perform actions that require connection to a Teamcenter site.

You may alternatively choose to add BMIDE functionality into your existing Eclipse environment. This consists of manually patching your Eclipse environment with BMIDE jar files. Doing so offers the advantage of allowing you to work on BMIDE templates within your existing custom Eclipse environment. However, adding BMIDE functionality into your existing Eclipse environment does not offer BMIDE functionality to perform actions that require connection to a Teamcenter site.

Install the Business Modeler IDE using TEM

1. Ensure that the proper version of JRE is installed and the **JRE_HOME** environment variable (32-bit system) or the **JRE64_HOME** environment variable (64-bit system) is set.

2. Start Teamcenter Environment Manager (TEM). For example, from the Teamcenter software kit, run **TEM.bat** (Windows) or **TEM.sh** (Linux).

3. Proceed to the **Solutions** panel. In the **Solutions** panel, select **Business Modeler IDE**, and then click **Next**.

   > Caution:
   >
   > Do not install the Business Modeler IDE on a production environment corporate server. Doing so could have unintended consequences, especially during Teamcenter upgrade.

4. Perform the following steps in the **Features** panel:

   a. Under **Base Install**, select one of the following:

   - **Business Modeler IDE 2-tier**

     Connects to a Teamcenter site in a two-tier environment via TCCS.

   - **Business Modeler IDE 4-tier**

© 2025 Siemens

Connects to a Teamcenter site in a four-tier environment via HTTP server.

- **Business Modeler IDE Standalone**

  Does not connect to a Teamcenter site.

When you select one of these options, a server connection profile is added in the Business Modeler IDE.

b. (Optional) Select **Extensions→Mechatronics Process Management→EDA for Business Modeler IDE**.

This installs the EDA Derived Data configuration tool into the Business Modeler IDE. This tool is used to configure Teamcenter EDA, an application that integrates Teamcenter with electronic CAD (E-CAD) design applications, such as Cadence and Siemens EDA.

If you install this option, you must ensure that the **Extensions→Mechatronics Process Management→EDA Server Support** option is also installed to the server.

In addition, later in the installation process when you select templates to install to the Business Modeler IDE, you must select the **EDA Server Support** template (**edaserver_template.xml**).

c. In the **Installation Directory** box, enter the location where you want to install the Business Modeler IDE. The Business Modeler IDE files are installed to a **bmide** subdirectory.

d. Click **Next**.

5. In the **Java Development Kit** dialog box, click the browse button to locate the JDK installed on your system. The kit is used for creating services. Click **Next**.

6. Depending on whether you selected Business Modeler IDE two-tier or four-tier installation, perform the following steps:

- If you selected the **Business Modeler IDE 2-tier** option, perform the following steps in the **2-tier server settings** panel:

  a. In the **Connection Port** box, type the server port number. The default is **1572**.

  b. Click the **Edit** button to the right of the **2-tier Servers** box to change the server connection profile settings, or click the **Add** button to add another server to connect to.

  c. Click the **Advanced** button.

     A. Click the arrow in the **Activation Mode** box to select the mode to use when connecting to the server. The default is **NORMAL**.

© 2025 Siemens

B. Click the ellipsis (**...**) button to the right of the **Configuration Directory** box to select the folder where you want this configuration saved. The default is *TC_ROOT***tccs**.

C. Click **OK**.

d. Click **Next**.

- If you selected the **Business Modeler IDE 4-tier** option, perform the following steps in the **4-tier server configurations** panel.

  a. Leave the **Compress (gzip) the responses from the Web application servers** check box selected if you want faster connection performance from the server.

  b. Click the **Add** button to the right of the **4-tier Servers** table if you want to add another server to connect to.

  c. Click **Next**.

- If you have previously installed Teamcenter client communication system (TCCS) on your system, and you also selected the **Business Modeler IDE 4-tier** option, the **TcCS Settings** panel appears. This panel is used to configure TCCS for use with the Business Modeler IDE. TCCS is used when you need secure Teamcenter communications through a firewall using a forward proxy.

  If you want to use TCCS, you must install it first. To install TCCS, copy the *installation-source***additional_applications\tccs_install.zip** file to a local directory, unzip its contents, and then run the **tccsinst.exe** installation wizard.

  To change the TCCS setup after installation, run the *tccs-installation-location***tccs_Teamcenter Communication Service_installation\Change Teamcenter Communication Service Installation** file.

  - If you do not want to use TCCS, ensure that the **Use TcCS Environments for 4-tier clients** check box is cleared and click **Next**.

    If this check box is cleared, the **4-tier server configurations** panel is displayed after you are finished with the current panel.

  - If you want to use TCCS, perform the following steps:

    a. Select **Do not use proxy** if you do not want to use a forward or reverse proxy.

    b. Select **Use web browser settings** to automatically use proxy settings already configured in a web browser.

    c. Select **Detect setting from network** to automatically use proxy settings from the network.

© 2025 Siemens

d. Select **Retrieve settings from URL** and type a valid proxy URL to use a proxy autoconfiguration file.

e. Select **Configure settings manually** to type valid host and port values for proxy servers.

f. Select the **Use TcCS Environments for 4-tier clients** check box if you want to use TCCS, or clear it if you do not. (This check box is automatically selected if TCCS is installed.)

g. If the **Use TcCS Environments for 4-tier clients** check box is selected, you can use the **Client Filter Text** box to specify a filter text on the available TCCS environments to avoid displaying undesired environments in the rich client logon window. This box is optional and can hold any string.

h. Click **Next**.

7. Perform the following steps in the **Business Modeler IDE Client** panel:

   a. Click the **Add** button to the right of the table to select the templates to install. *Templates* contain the data model for Teamcenter solutions. The **Teamcenter Foundation** template is installed by default. The Foundation template contains the data model used for core Teamcenter functions. All customer templates must extend the Foundation template.

      Select the same templates that were installed on the server so that you can see the same data model definitions in the Business Modeler IDE that were installed on the server.

      To find the templates installed on the server, look in the *TC_DATA***model** directory on the server.

      If you installed the **EDA** option to the Business Modeler IDE, select the **EDA Server Support** template (**edaserver_template.xml**).

   b. If you have any templates of your own to install or a template from a third-party, click the **Browse** button and browse to the directory where the templates are located.

   c. Click **Next**.

8. Complete the remaining panels to finish the installation in Teamcenter Environment Manager. When the installation is complete, exit Teamcenter Environment Manager.

9. Verify the installed files in the *install-location***bmide** directory.

   The following data model files are placed into the *install-location***bmide\templates** folder:

   - **icons***template-name*_icons.zip**

     Contains the icons used by that template.

© 2025 Siemens

- **lang***template-name***_template_***language_locale***.xml**

  Contains the text that is displayed in the Business Modeler IDE user interface for all languages.

- *template-name***_dependency.xml**

  Lists the other templates that this template is built on top of, for example, the Foundation template.

- *template-name***_template.xml**

  Contains the data model for this template, including business objects, classes, properties, attributes, lists of values (LOVs), and so on.

- **master.xml**

  Lists the template XML files included in the data model, for example, the **foundation_template.xml** file.

10. Allocate memory so that Business Modeler IDE has enough memory to run.

Install the Business Modeler IDE using Deployment Center - connected

Before you install the Business Modeler IDE, make sure you log on using an account with administrative privileges and that you have access to the Teamcenter software kit.

This procedure assumes you have an existing Teamcenter environment. Make sure all the required Teamcenter software kits have been added to your software repository in Deployment Center

1. Log on to Deployment Center and select your Teamcenter environment.

2. Proceed to the **Components** tab.

   If the **Selected Components** list does not include **Business Modeler IDE** component type you want to install, add the component:

   a. Click **Add component to your environment** ⊕ to display the **Available Components** panel.

   b. Select the **Business Modeler IDE** component type you want to install (2 Tier or 4 Tier), and then click **Update Selected Components**.

3. In the **Selected Components** list, select **Business Modeler IDE** *installation type*.

4. Enter the required values to configure the Business Modeler IDE:

© 2025 Siemens

Value	Description
Enable Mass Client Deploy? (2-Tier and 4-Tier)	Generates a deploy script that can be run on multiple machines.
Machine Name	Identifies the target deployment machine. The name is used in naming the deploy script. • **Single box** If your environment is a single box environment, the **Machine Name**, **OS**, and **Teamcenter Installation Path** values are inherited from the first component you configured in your environment. Changing these values will change them for other components in your environment. • **Distributed** If your environment is a distributed environment, type the **Machine Name**, **OS**, and **Teamcenter Installation Path** for the machine on which you install the Business Modeler IDE.
OS	Identifies the operating system on the target machine.
Teamcenter Installation Path	Path on the target machine for the installation. See notes in the description above for **Machine Name**.
Java Development Kit Path	Path on the target machine to the Java development kit.
Templates and Clients	Select the applications and clients whose templates you want to copy to the target machine for use in customization. The Teamcenter foundation template is included by default.
Connection Port and **Connection Name** (2-tier)	Specifies the port number and name to use for the 2-tier connection between the Business Modeler IDE and the Teamcenter server.
Compress(gzip) Web Application server response (4-tier)	Compresses traffic between the Business Modeler IDE and the Web Application server.
Character Encoding type (2-tier)	Specifies the character encoding type used by the database server on the target machine.

By default, some parameters are not displayed because they generally do need to be changed from the default values. They can be displayed by clicking **Show all parameters** .

5. Click **Save Component Settings**.

6. Complete configuration of any remaining components.

© 2025 Siemens

7. When all components are fully configured, go to the **Deploy** tab. Click **Generate Install Scripts** to generate deployment scripts to update affected machines. When script generation is complete, note any special instructions in the **Deploy Instructions** panel.

8. Locate deployment scripts, copy each script to its target machine, and run each script on its target machine.

   For more information about running deployment scripts, see *Deployment Center — Usage*.

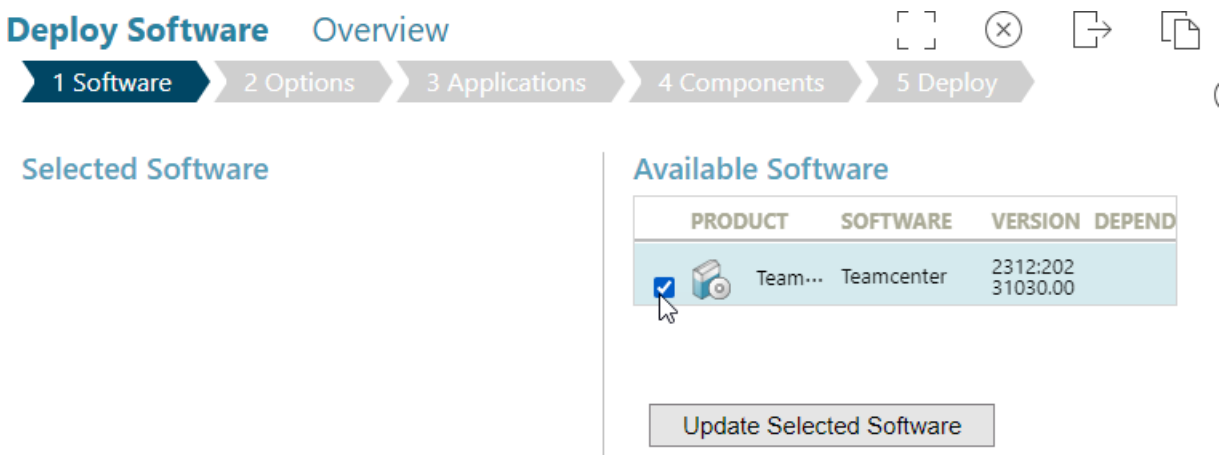
Install the Business Modeler IDE using Deployment Center - standalone

Before you install the Business Modeler IDE, make sure you log on using an account with administrative privileges and that you have access to the Teamcenter software kit.

1. Log on to Deployment Center and create a new environment.



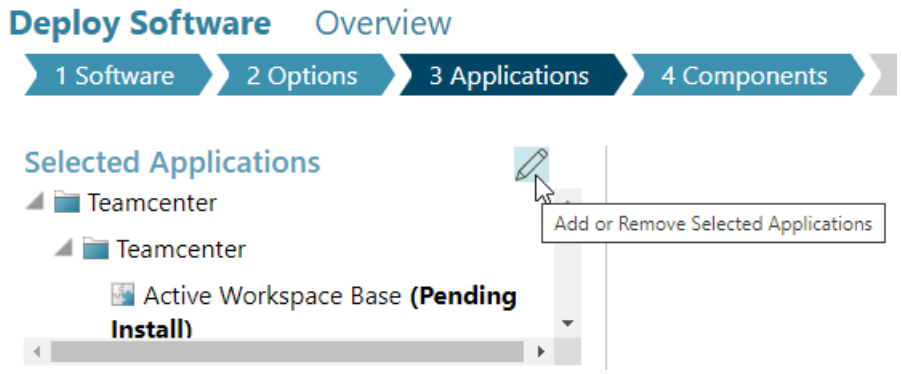
2. In the **Software** tab, add the Teamcenter software.



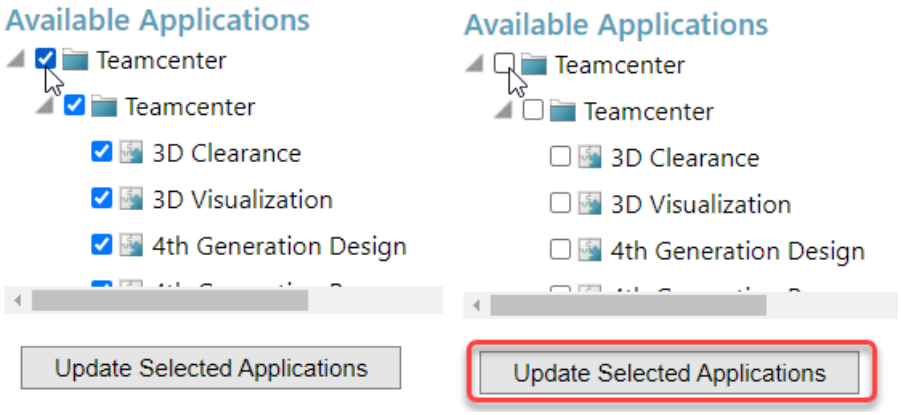
3. In the **Options** tab, accept the defaults (**Java EE** architecture, **Single Box** environment, **Local** architecture).

4. In the **Applications** tab, click **Add or Remove Selected Applications**.

© 2025 Siemens

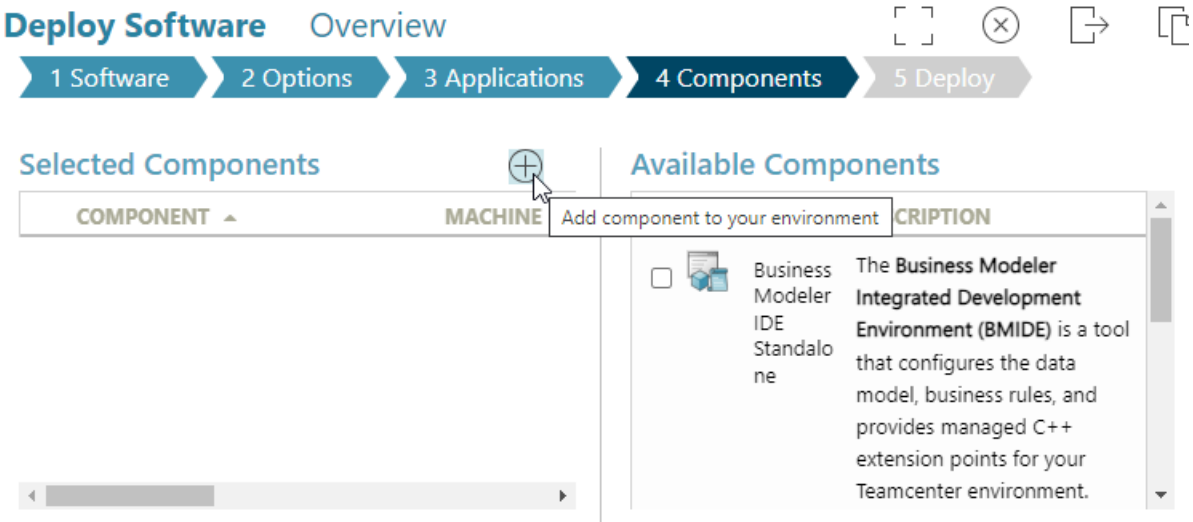


5. In the **Available Applications** list, select and then deselect the **Teamcenter** application group (this performs a select all/deselect all action), then click **Update Selected Applications**.



This clears the **Selected Applications** list.

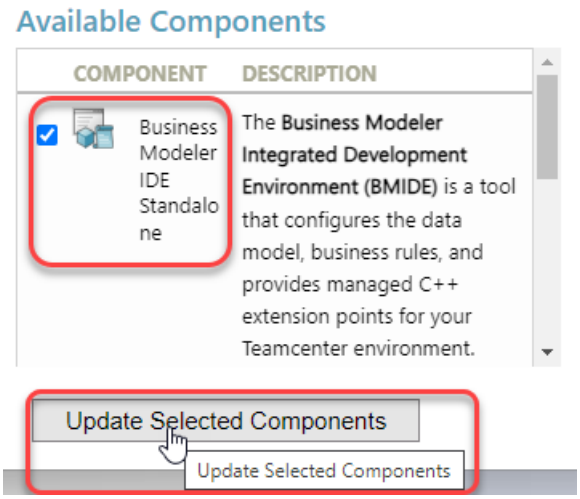
6. In the **Components** tab, click **Add component to your environment** ⊕.



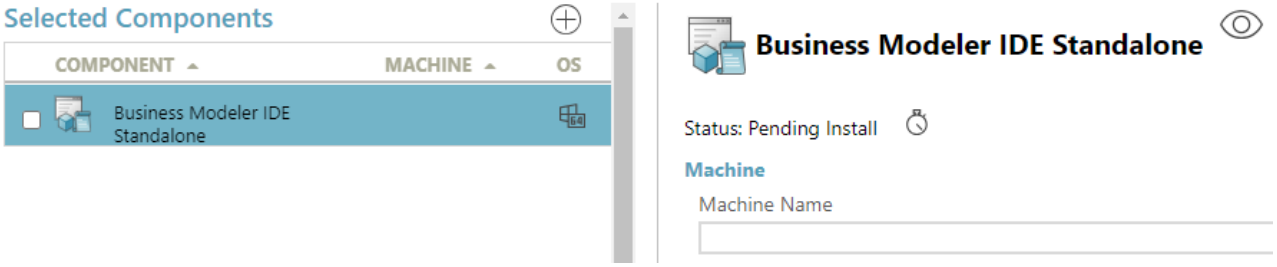
© 2025 Siemens

Business Modeler IDE Standalone is the only available component.

7. Select it and click **Update Selected Components**.



8. In the **Selected Components** panel, click **Business Modeler IDE Standalone** to view the parameters.

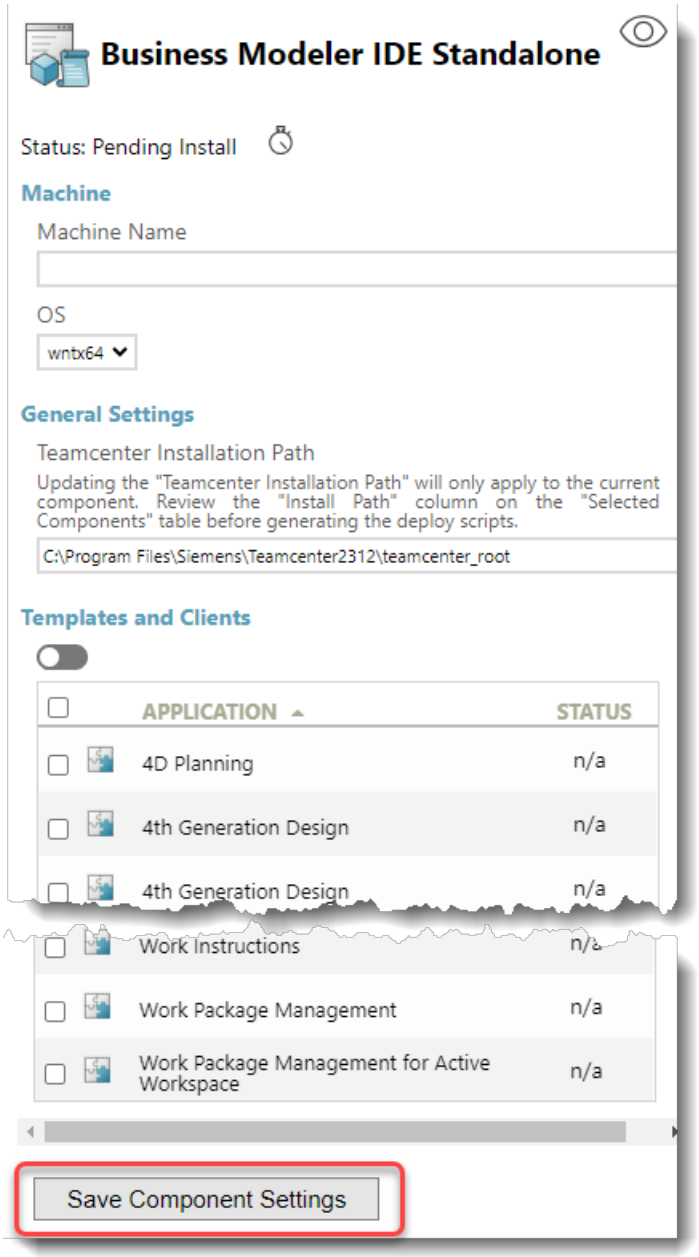


9. Enter parameters for the standalone Business Modeler IDE and then click **Save Component Settings**.

Value	Description
Machine Name	Identifies the target deployment machine. The name is used in naming the deploy script.
OS	Identifies the operating system on the target machine.
Teamcenter Installation Path	Path on the target machine for the installation.
Java Development Kit Path	Path on the target machine to the Java development kit.

© 2025 Siemens

Value	Description
	By default, this parameter is not displayed. It can be displayed by clicking **Show all parameters** .
Templates and Clients	Select the applications and clients whose templates you want to copy to the target machine for use in customization. The Teamcenter foundation template is included by default.



10. In the **Deploy** tab, click **Generate Install Scripts**.

11. Follow the **Deploy Instructions** to deploy the standalone BMIDE.

© 2025 Siemens

For more information about running deployment scripts, see *Deployment Center — Usage*.

Add the Business Modeler IDE to an existing Eclipse SDK environment

If you already have an existing Eclipse SDK environment with the version of Eclipse that is certified for your Teamcenter platform, and Business Modeler IDE plugins have never been installed into the environment, then you can install the Business Modeler IDE plugins into your existing Eclipse environment.

> Caution:
>
> If your Eclipse environment contains Business Modeler IDE plugins installed from an earlier version of Business Modeler IDE, then installing a later version of Business Modeler IDE plugins into the same environment results in version incompatibilities and is not supported.

1. Ensure that your Eclipse SDK environment uses the Eclipse version that is certified for your Teamcenter platform.

   For information about system hardware and software requirements, see the Software Certifications Matrix (**Tc2412PlatformMatrix-*date*.xlsx**) in the **Support White Papers Certifications** page on Support Center.

   To check your Eclipse version, start Eclipse and select **Help**>**About Eclipse SDK**.

2. In the Teamcenter software kit for your Teamcenter platform, go to the following directory:

   additional_applications\bmide_plugins (Windows) or **additional_applications/bmide_plugins** (Linux)

   In that directory, find the file **bmide_plugins.zip**.

   This archive contains the Business Modeler IDE plug-ins within an internal **eclipse\plugins** directory.

3. Extract the contents of the **eclipse\plugins** directory within **bmide_plugins.zip** to your *ECLIPSE_HOME***eclipse\plugins** directory.

4. In the Teamcenter software kit for your Teamcenter platform, go to the following directory:

   bmide\compressed_files

   In that directory, find the file **bmide.zip**.

5. Extract the **bmide.zip** content to some temporary local directory. For example **C:\bmide** (Windows) or **/bmide** (Linux).

© 2025 Siemens

6. From the **plugins** directory within this local directory, copy the following directories and their contents to your *ECLIPSE_HOME***eclipse\plugins** directory.

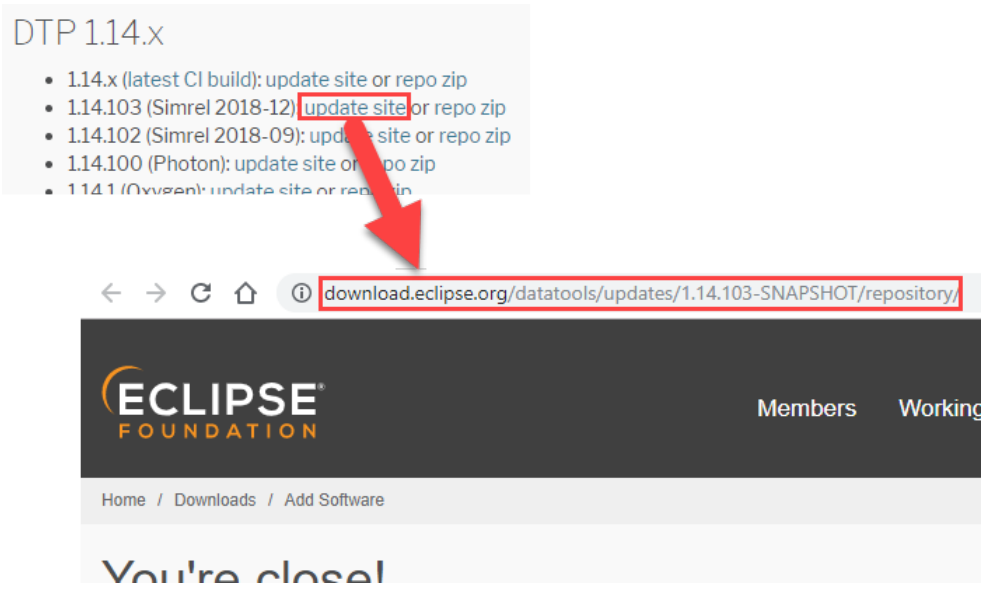
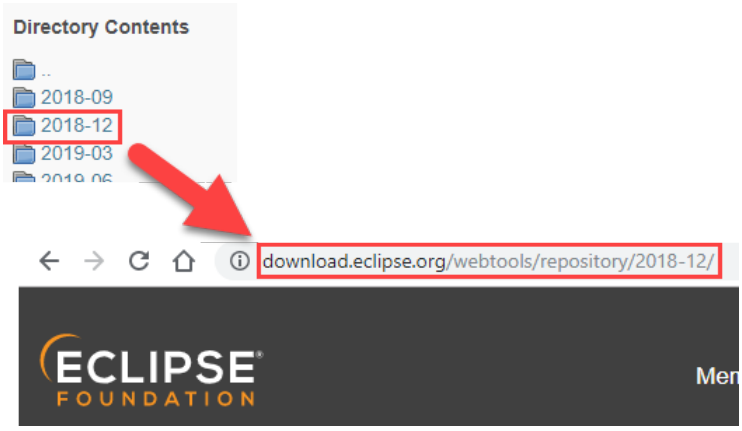
   - antlr
   - commons_lang
   - commons_xmlschema
   - httpclient_*version*
   - org.apache.poi.39

7. Create a list of software repository site URLs for the following plugins. Use the Eclipse site to identify the proper URLs. The examples shown are for Eclipse SDK version 2018-12 (4.1.10.0).

   You will use this list in step 9.

For this plugin	Do this
CDT	Browse to **https://www.eclipse.org/cdt/downloads.php** and find the URL for the CDT software repository for your Eclipse version. The URL looks similar to this: **https://download.eclipse.org/tools/cdt/releases/9.6** Record the URL in your list of plugin software repository sites. 
DTP	Browse to **https://www.eclipse.org/datatools/downloads.php** and find the DTP row for your Eclipse version. Click the **update site** link. The URL of the page that opens looks similar to this: **https://download.eclipse.org/datatools/updates/1.14.103-SNAPSHOT/repository/** Record the URL in your list of plugin software repository sites.

© 2025 Siemens

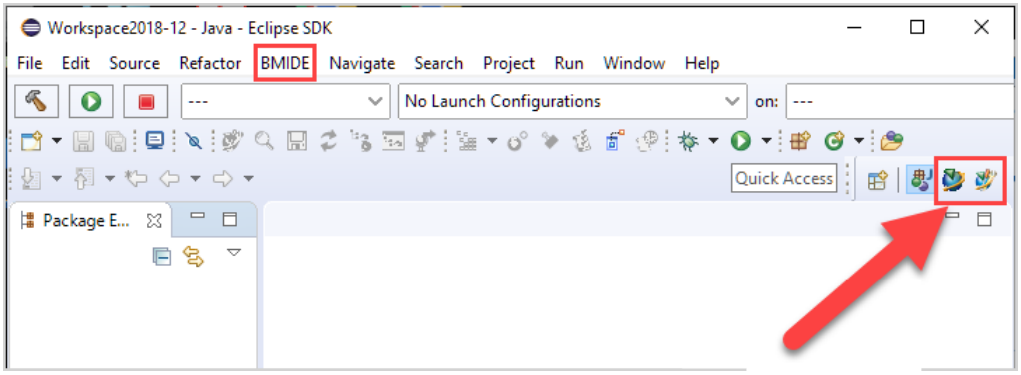
For this plugin	Do this
	
EMF	Add **http://download.eclipse.org/modeling/emf/updates/releases/** to your list of plugin software repository sites. For details about the requirement for Eclipse version and software repository URL for **EMF**, browse to **https://www.eclipse.org/modeling/emf/**
GEF	Add **http://download.eclipse.org/tools/gef/updates/releases** to your list of plugin software repository sites. For details about the requirement for Eclipse version and software repository URL for **GEF**, browse to **https://projects.eclipse.org/projects/tools.gef**
WTP	Browse to **https://download.eclipse.org/webtools/repository/** and then find the directory for your Eclipse version. Click the directory link. The URL for the resulting page looks similar to this: **https://download.eclipse.org/webtools/repository/2018-12/** Record the URL in your list of plugin software repository sites. 

© 2025 Siemens

8. Launch Eclipse.

9. From the top menu bar, choose **Help**>**Install New Software**. Use the Eclipse software installation feature to add the CDT, DTP, GEF, EMF and WTP plugin software update sites and install all of the plugins. Refer to the list of plugin software update URLs you created in step 7.

After all the plugins are installed and you have restarted Eclipse, a **BMIDE** item appears on the top menu bar. Command buttons to open the BMIDE **Advanced** and **Standard** perspectives appear on the toolbar.



Allocate memory to the Business Modeler IDE

Allocate memory to the Business Modeler IDE so that it has enough to launch and run.

If you perform live updates, you must have a minimum of 2 GB of RAM on the system running the Business Modeler IDE to allow for other processes.

You can allocate memory in the following ways:

- **BusinessModelerIDE.ini** file

  To increase the memory allocated to the Business Modeler IDE, open the *install-location***bmide\client\BusinessModelerIDE.ini** file and change the **-Xmx1024M** value to a higher number to allocate maximum Java heap size. For example, if you have 2 GB available to dedicate for this purpose, set the value to **-Xmx2048M**. Do this only if your machine has the available memory.

  The **Xms** value in this file sets the initial Java heap size, and the **Xmx** value sets the maximum Java heap size.

- **BMIDE_SCRIPT_ARGS** environment variable

  To allocate the memory required by scripts during installation, update, or load of templates with large data models, create a **BMIDE_SCRIPT_ARGS** environment variable. Set the **BMIDE_SCRIPT_ARGS** variable to **-Xmx1024M** to allocate 1 GB of RAM to the Business Modeler IDE scripts. If your system has more memory that you can allocate to the Business Modeler IDE, you can set the value higher.

© 2025 Siemens

If you are running the Business Modeler IDE in an Eclipse environment, run the following command to increase virtual memory to 2 GB:

```
eclipse.exe -vmargs -Xmx2048M
```

Caution:

Java standards require that no more than 25 percent of total RAM be allocated to virtual memory. If the amount allocated to the Business Modeler IDE is higher than 25 percent of total RAM, then memory disk swapping occurs, with possible performance degradation.

If you set the **Xmx** value to a higher value than the RAM your system has, you may get the following error when you launch the Business Modeler IDE:

```
Could not create the Java virtual machine.
```

Set the **Xmx** value to a setting that your system supports, in both the **BMIDE_SCRIPT_ARGS** environment variable and the **BusinessModelerIDE.ini** file.

Uninstall a Business Modeler IDE

To avoid serious errors due to accidental use of incompatible versions, if you install a new version of BMIDE, then uninstall the old version of BMIDE.

1. Start Teamcenter Environment Manager (TEM) for the old installation. For example, on a Windows system, run **tem.bat** from the **install** directory where the old version of the Business Modeler IDE is installed.

2. In the **Maintenance** panel, select **Configuration Manager** and click **Next**.

3. In the **Configuration Maintenance** panel, select **Perform maintenance on an existing configuration** and click **Next**.

4. In the **Configuration** panel, make sure the proper configuration is selected and click **Next**.

5. In the **Feature Maintenance** panel, under **Teamcenter**, select **Add/Remove Features** and click **Next**.

6. In the **Features** panel, clear all the check boxes for **Business Modeler IDE** and click **Next**.

7. Proceed to the confirmation panel and click **Start**.

   The old version of the Business Modeler IDE is uninstalled.

© 2025 Siemens

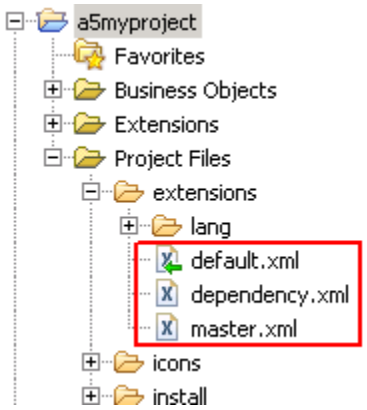
© 2025 Siemens

5. Configure the Business Modeler IDE

Extension files

Introduction to extension files

Your custom data model definitions for a project are saved in XML files within your project folder. When you create a project, three files are created in the **Project Files** > **extensions** folder:



File name	Description
default.xml	Holds extensions that you make to the project. Extension changes are saved to this file when you save the project.
dependency.xml	Specifies the templates that the project extends. For example, the Foundation template.
master.xml	Points to other files that are used to build the consolidated template.

To examine the source code contained in a file, right-click the file and choose **Open With**>**Text Editor**. Do not manually edit the content in extension files.

Add an extension file

If you want to organize the xml code resulting from the custom elements within your BMIDE template project, then you can add new extension files in addition to *default.xml*.

To add a new extension file, in the **BMIDE** view expand the **Project Files** folder, right-click the **extensions** folder and choose **Organize**>**New Extension File**.

© 2025 Siemens

Set the active extension file

New extensions that you add to a project are recorded in the active extension file for the project. The default active extension file is *default.xml* within the **BMIDE** view **[project] Project Files**>**extensions** folder. If your project has more than one extension file, then you can specify which of the extension files receives your new extensions.

1. Right-click the project and choose **Organize**>**Set active extension file**.

2. In the **Extension File Selection** dialog box, select the extension file that you want to make active.

A green arrow symbol appears on the active extension file.

Extension file destination when modifying extension elements

The destination extension file for recording modifications to existing elements is situation dependent as described in the following table.

If the element is defined here	Then modifications are recorded here
Current project	In the element's original extension file. For example, consider the following sequence. 1. Create condition **C1** in **File1** (active extension file). 2. Change the active extension file to **File2**. 3. Modify the description field of condition **C1**. The change is recorded in **File1**, which is the source file of condition **C1**.
Dependent project	• If this is the first modification to the element, then in the active extension file of the current project. For example, consider the following sequence. 1. Create a custom template **customTemplate** which has a dependency on **foundation** template. 2. Modify an existing element (say **LOVMyFoundation**) of **foundation** in the custom template.

© 2025 Siemens

If the element is defined here	Then modifications are recorded here
	The modified value is recorded in the active extension file of **customTemplate** template and not in any extension file of the **foundation** template. • If the element has previously been modified in the current project, then in the extension file of the current project that was the active extension file when the element was first modified in the current project.

Move a custom element

If you create extension files in a project in addition to the **default.xml** file, then you can move a custom element to another extension file for your own organizing convenience. Such moves within a project still keep the elements within the project, and have no practical effect when the project is packaged for distribution.

As you increasingly create your own functionality, you may want to move elements of custom functionality to another project so that the functionality groupings can be distinctly distributed and installed to a Teamcenter environment. To avoid potential collisions, you must track moved elements to ensure that new elements are not created that have the same name as elements that were moved to another project.

Caution:

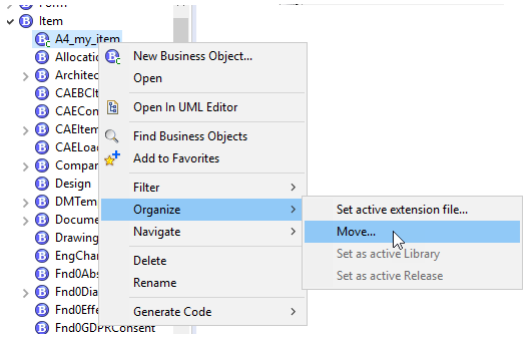
- When you move an element from a source template project to a target template project, the moved element retains the prefix from its source project. There is no mechanism in place to prevent you from subsequently creating a custom element with the same name in the source template; if you do, then you have two different templates containing elements with the same name. This results in a collision if the two templates are both installed to an environment.
- Manually editing the XML files can easily corrupt data, and as a rule should be avoided.

  An exception to this rule is if you are using a source control management (SCM) system and you need to merge changes from two or more users into one file. In this case, you must manually edit the file. After you complete the edit, save the file, and right-click in any view and choose **Reload Data Model**. The Business Modeler IDE reloads all the XML definitions from the source and validates them for correctness. If there are any issues with the XML files or definitions, they are displayed in the **Console** view.

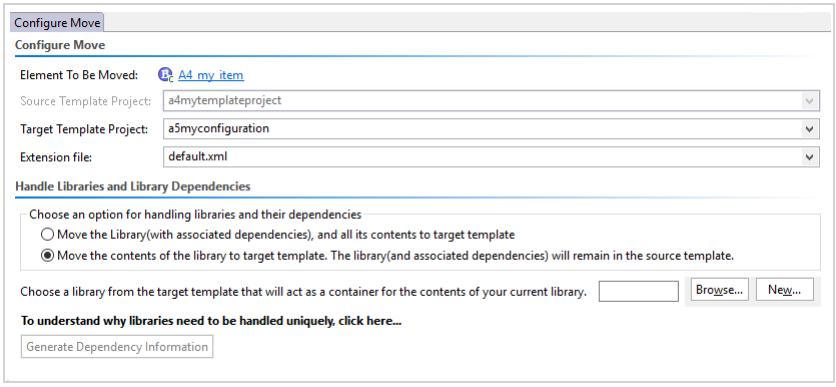
1. If you want to move custom elements from one template project to another, ensure that both projects are imported into the Business Modeler IDE.

2. Right-click the custom element you want to move and choose **Organize→Move**.

© 2025 Siemens



The **Configure Move** tab is displayed.



3. In the **Target Template Project**, select the template project to receive the element.

4. In **Extension file**, select the extension file to receive the element.

5. If you are moving the element to a different template project, then select one of the following options:

   - **Move the library (with associated dependencies) and all its contents to the target template.**

     Moves the selected element and any dependent elements to the target template. In addition, if the selected element stores entities in a library such as operations, the library is also moved to the target template.

   - **Move the contents of the library to the target template. The library (and associated dependencies) will remain in the source template.**

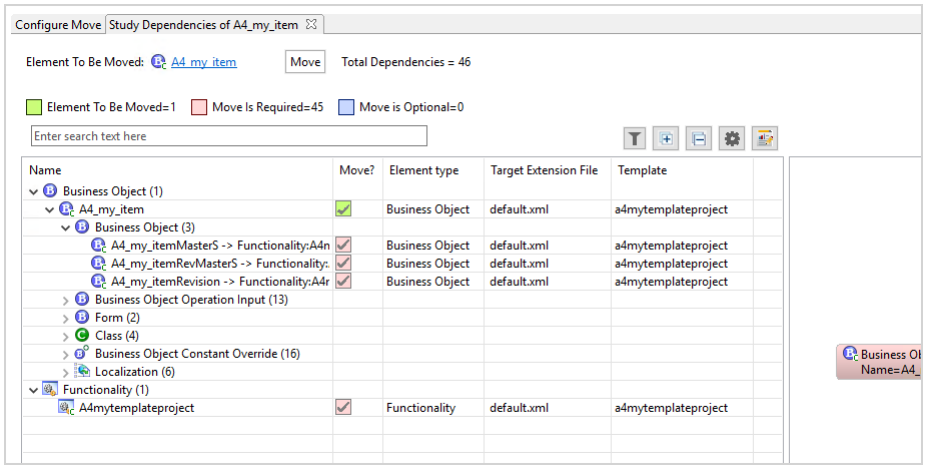
     Moves the selected element and any dependent elements to the target template. In addition, if the selected element stores entities in a library such as operations, the contents of the library are also moved to the target template into a library of your choosing.

     If you select this option, then click **Browse** to select the library in the target template to receive the element, or click **New** to create a new library in the target template.

© 2025 Siemens

6. Click **Generate Dependency Information**.

   The **Study Dependencies** tab is displayed. This lists all the dependent elements that are to be moved. For example, if you select an item business object to be moved, accompanying custom revision and form business objects are also selected.

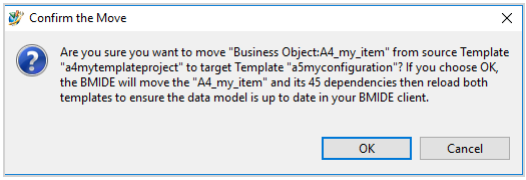


7. Select the check boxes next to each element to indicate they are to be moved. You can click the command buttons at the top right of the table to change the view or generate an HTML report of elements in the table.



8. When you are done studying the dependencies and have made your final selection of elements to move, click Move .

   A confirmation dialog box is displayed.

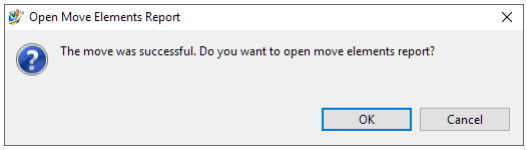


9. Click **OK** to confirm the move.

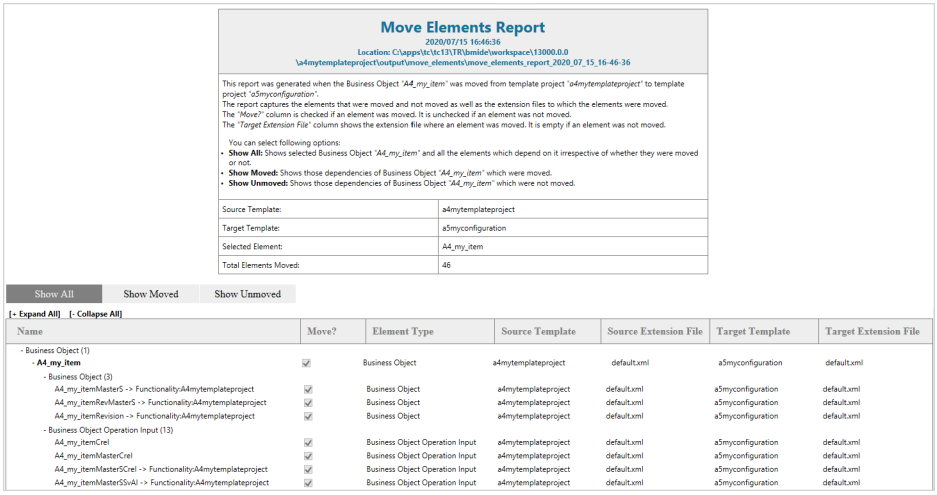
   The element and its dependent elements are moved.

10. When the move is complete, click **OK** in the dialog box to view the move report.

© 2025 Siemens



The report shows the results of the move.



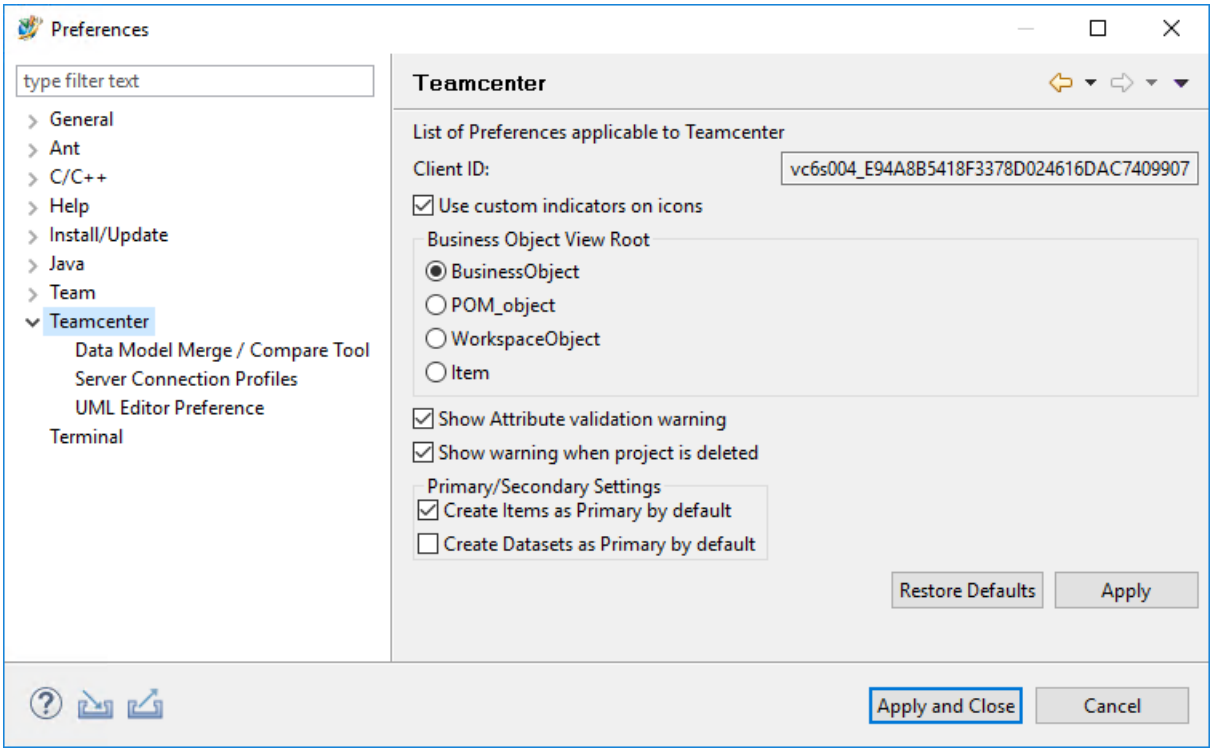
Set Business Modeler IDE preferences

Perform the following steps to change preferences for the Business Modeler IDE client. Setting preferences is optional.

1. Choose **Window**>**Preferences**.

2. In the **Preferences** dialog box, select **Teamcenter**.

© 2025 Siemens



3. You can change the following preferences:

Teamcenter preferences (root page)

Use custom indicators on icons Places a **c** symbol next to custom data model items that you create, such as business objects, classes, LOVs, and the like.

Business Object View Root Select one of the following to set the business object that is the highest (root) object displayed in the **Business Objects** folder:

BusinessObject Displays all business objects, including run-time business objects. (Default)

Run-time objects (children of the **RuntimeBusinessObject** object) behave much like standard business objects. However, run-time business objects are not persisted in the database as business objects but are instantiated at run time.

POM_object Displays the business objects available for creating new business object types.

WorkspaceObject Displays the objects that can be displayed in the end-user workspace.

© 2025 Siemens

Item	Displays the most commonly used business objects such as **Item** and **Document** that are used to represent work objects to be tracked in Teamcenter.
Show attribute validation warning	Shows a warning if attributes added to existing classes may cause discrepancies in the database.
Show warning when project is deleted	Shows a confirmation dialog box when a user attempts to **delete a project**.
Primary/ Secondary Settings	Select the following to define whether business objects are created as **primary business objects**.
Create Items as primary by default	Specifies that **Item** business objects and their children are created as primary business objects.
Create Datasets as primary by default	Specifies that **Dataset** business objects and their children are created as primary business objects.

Data Model Merge / Compare Tool preferences (sub-page)

Sets the live update preferences for the Incorporate Latest Live Update Changes wizard.

Server Connection Profiles preferences (sub-page)

Sets the Teamcenter servers to connect to. You must create a profile so you can deploy your extensions to a server or query a server for data.

UML Editor Preference preference (sub-page)

Click the **UML Editor Background Color** box to select a new color to display behind the items displayed in the UML Editor.

4. To set the active extension file indicator, in the left pane of the **Preferences** dialog box, choose **General**>**Appearance**>**Label Decorations**, and select **Teamcenter Active Extension File Decorator**. The **active extension file** is the file in the project selected to hold data model changes.

5. When done making changes, click **Apply** to commit the changes for that preference. Click **Restore Defaults** to revert changes to the originally shipped settings for the preference.

6. Click **OK** to commit the changes you have made to all preferences.

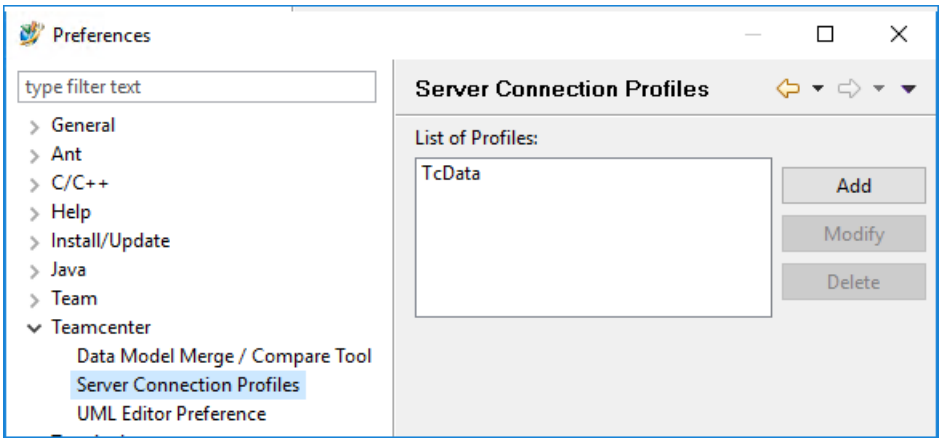
© 2025 Siemens

Add a server connection profile

Some functions of the BMIDE, such as deploying data model changes to a test server, require a connection to a Teamcenter server. You configure and make connections by means of a server connection profile.

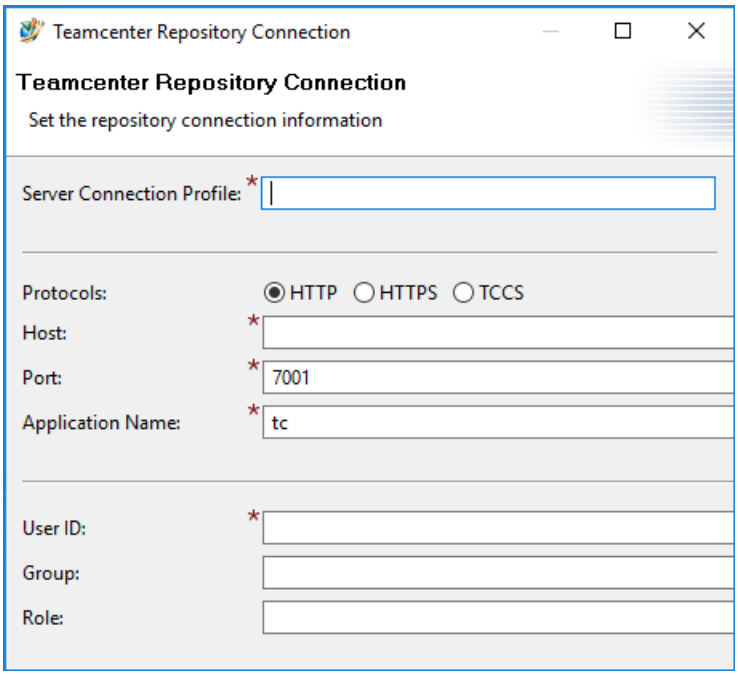
1. Choose **Window**>**Preferences**.

2. In the **Preferences** dialog box, choose **Teamcenter**>**Server Connection Profiles**.



3. Click **Add**.

   The **Teamcenter Repository Connection** wizard runs.



© 2025 Siemens

4. Enter parameter values for the connection.

For this parameter	Do this
Server Connection Profile	Type the name you want to use for the profile. For example, you might type the name of the server.
Protocols	Select the protocol to use to communicate with the server. **HTTP** Sets a Web communication protocol (used in a four-tier Teamcenter environment). If you selected the **Business Modeler IDE 4-tier** option under **Base Install** when you installed the Business Modeler IDE, a server connection profile named **TcWeb1** is automatically set up using the **HTTP** protocol. **HTTPS** Sets a secure Web communication protocol (used in a four-tier Teamcenter environment). **TCCS** Sets a Teamcenter client communication system (TCCS) protocol. This protocol is used when you need Teamcenter communications through a firewall using a forward proxy. If you selected the **Business Modeler IDE 2-tier** option under **Base Install** when you installed the Business Modeler IDE, a server connection profile named **TcData** is automatically set up using the **TCCS** protocol.
Host	Type the host name assigned to the Teamcenter web application server.
Port	Type the port assigned to the Teamcenter web application server. For example, if you chose HTTP protocol and you are using Apache Tomcat as your web application server, you can enter the default port of **8080**.
Application Name	If you selected HTTP or HTTPS as the protocol, enter the name of the application to connect to. For example, if you are using HTTP as the protocol to connect to Teamcenter, type the default application name of **tc**. This resolves the Web address to: *web_app_server***:***port***/tc**
Environment Name	If you selected TCCS as the protocol, select the environment that was set up when TCCS was installed. To install TCCS, copy the *installation-source***\additional_applications\tccs_install.zip** file to a local directory, unzip its contents, and then launch the **tccsinst.exe** installation wizard.

© 2025 Siemens

For this parameter	Do this
	To change the TCCS setup after installation, run the *tccs-installation-location***tccs_Teamcenter Communication Service_installation\Change Teamcenter Communication Service Installation** file.
User ID	Type the ID of the authorized user on the Teamcenter server.
Group (Optional)	Type the group the user is assigned to. For example, type **dba** for the database administration group.
Role (Optional)	Type the role the user is assigned. For example, type **DBA** for the database administrator role.

5. Click **Finish**.

   The server profile is added to the list in the **Preferences** dialog box.

6. Click **OK** to save the preferences.

Testing the connection to the server

Perform the following steps to test the connection to the server:

1. Start the server by launching the Teamcenter rich client.

2. In the Business Modeler IDE, open the **Item** business object and click the **Display Rules** tab.

3. In the **Display Rules** editor, click **Add**.

4. In the **Display Rule** dialog box, next to **Organization**, click **Browse**.

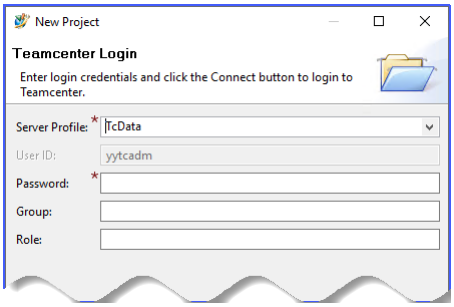
   If this is the first time in this session that the Business Modeler IDE needs to query the server for data, the Teamcenter Repository Connection wizard prompts you to log on to a server to look up the available groups and roles in the organization.

   If the connection is made properly, the **Search Organization** dialog box displays the groups and roles from your server.

Login to a Teamcenter server

Various Business Modeler IDE client operations require connecting to a running Teamcenter server. In such situations, a **Teamcenter Login** dialog box is displayed.

© 2025 Siemens



1. In the **Teamcenter Login** dialog box, specify option values for connecting to the server.

For this option	Do this
Server Profile	Select the server connection profile to use.
Password	Type the server access password for the profile's **User ID**.
Group	(Optional) Type the group name for the profile's **User ID**.
Role	(Optional) Type the role name for the profile's **User ID**.

2. Click **Connect**.

   The Business Modeler IDE client connects to the server.

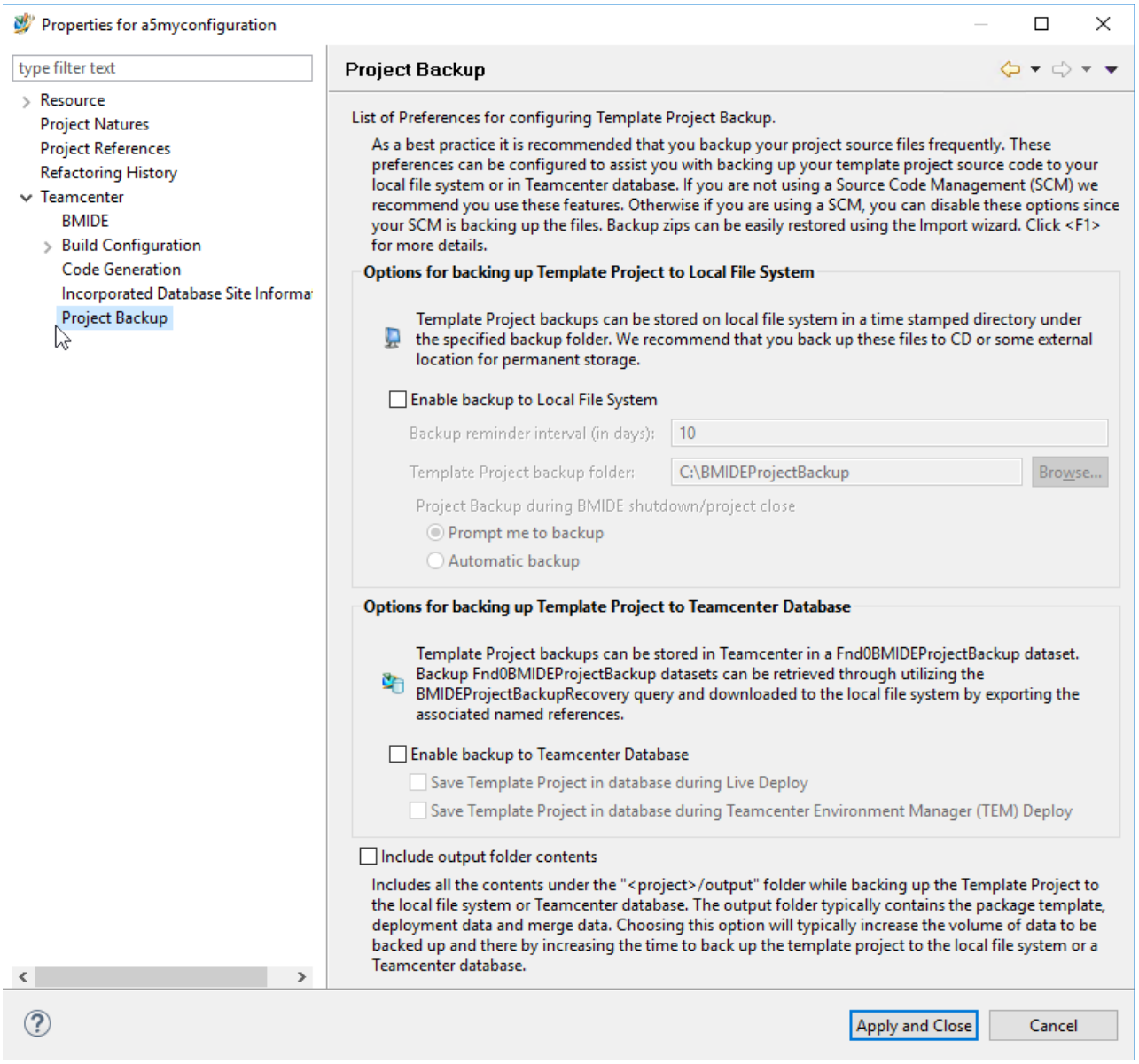
Back up project data

Besides maintenance of project data in a third party Source Control system, which is necessary in a multi-developer environment, Business Modeler IDE itself offers multiple options for regular backup of project data. Existence of a good backup allows you to recover from failures and to **import a project** to a fresh installation of Business Modeler IDE.

1. Use the **Project Backup** dialog box to set project backup. To access this dialog box, right-click the project and choose **Properties**>**Teamcenter**>**Project Backup**.

   By default, projects are backed up to your local computer when you close a project or shut down the Business Modeler IDE (**Enable backup to Local File System** check box), and backed up to the database when you deploy the project to a server (**Enable backup to Teamcenter Database** check box).

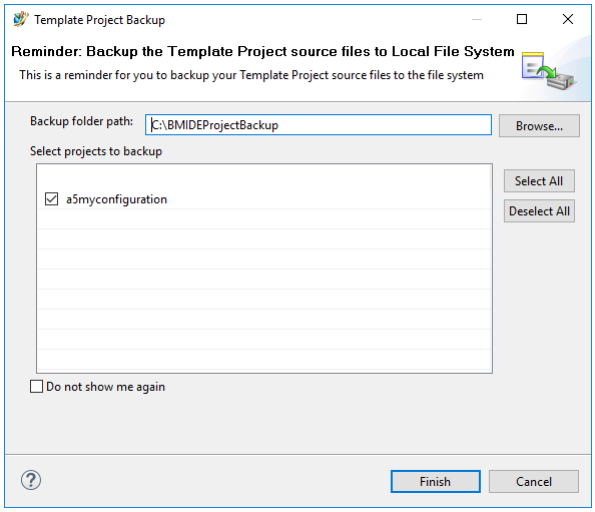
© 2025 Siemens



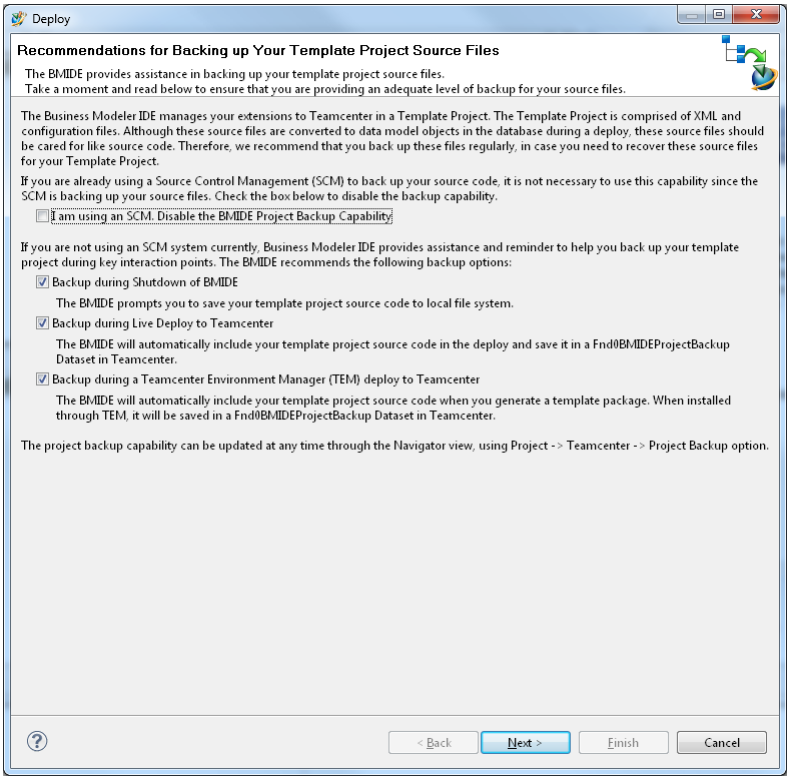
2. Back up your project data.

   - When you close a project or shut down the Business Modeler IDE, execute the backup using the following dialog box.

© 2025 Siemens



- When you first deploy a project, the **Deploy** wizard prompts you to select backup options.



© 2025 Siemens

6. Upgrade the Business Modeler IDE

Business Modeler IDE upgrade process

To install a newer version of the Business Modeler IDE, follow these steps:

1. **Install the new version of the Business Modeler IDE**.

2. **Migrate your old projects to the new data model format**.

   You can also **create new projects**.

3. If your project has any dependent templates, use Teamcenter Environment Manager (TEM) to **add dependent templates back into the project**. In the **Feature Maintenance** panel, select **Add/Update Templates for working within the Business Modeler IDE Client**.

4. If you have server code customizations from previous versions, you must regenerate the code and rebuild your libraries.

5. **Uninstall the previous version of the Business Modeler IDE**.

> Note:
>
> If your upgrade of the Business Modeler IDE is part of a larger process to upgrade Teamcenter, see *Teamcenter Upgrade Using TEM*.

Upgrade a template project to the current data model format

If you have installed a new version of the Business Modeler IDE, you can use a project from the previous version. But first you must upgrade the project to the new data model format. This upgrade is necessary because the XML format used for data model files can change between product releases, and the project must be adjusted to fit the new XML format.

> Caution:
>
> After a template project is upgraded, it cannot be used for installation or upgrade in a previous version of Teamcenter. To find the version the template has been upgraded to, open the **dependency.xml** file in the **extensions** folder of the template project and view the **currentTemplateVersion** value.

You can upgrade a project three ways:

© 2025 Siemens

Welcome window

When you first open the Business Modeler IDE after installing it, the **Welcome** window is displayed. Click the **Upgrade your BMIDE template from a previous Teamcenter release** link in this window to run the import wizard. This imports your template into the new version of the Business Modeler IDE.

Import wizard

If your template project is not already in the workspace, use the Import wizard to import it into the new version of the Business Modeler IDE.

1. Choose **File**>**Import**.

2. In the **Import** dialog box, choose **Business Modeler IDE**>**Import a Business Modeler IDE Template Project**.

   While importing the project, the Business Modeler IDE automatically upgrades the project to the new data model format.

Re-run Template Project Upgrade wizard

If your template project is already in the workspace, upgrade it to the new version of the Business Modeler IDE:

1. On the menu bar, choose **BMIDE**>**Upgrade Tools**>**Re-run Template Project Upgrade Wizard**.

   The wizard runs.

2. In the **Template Project Upgrade** dialog box, click the arrow in the **Project** box to select the project to upgrade.

3. Click **Finish**.

   The project is upgraded to the new data model format. The **Console** view displays success or failure messages for the upgrade.

   After upgrade, open the **Project Files** folder and check for any error or warning messages in the log in the **output\upgrade** folder.

Upgrade the custom template when you upgrade to the latest version of Teamcenter

1. Import the older project into the latest version of the Business Modeler IDE. This updates the data model to the latest data model version.

2. If your project contains custom code, you must treat it as though it were new code. Perform the following steps:

© 2025 Siemens

a. Clean up the **output** and **source** folders.

b. Generate Code for the project.

c. Perform these steps in stages to aid in troubleshooting:

A. Insert your old source code into the new files.

B. Build the project.

3. Package the template in the Business Modeler IDE.

4. Install the packaged template to the upgraded server.

Refactor postactions on create operations

The create operation creates objects in memory, and the save operation commits the created objects to the database. The create operation of a business object should be followed by a save operation to commit the objects created in memory to the database, for example:

```
ITEM_create_item()
ITEM_save_item( itemTag ) or AOM_save_with_extensions ( itemTag )

ITEM_create_rev()
ITEM_save_rev( revTag ) or AOM_save_with_extensions ( revTag )
```

This create and save pattern requires that any postactions attached to the create message (operation) should not assume the created objects are already saved into the database, and any create operation overrides should not make that assumption either. The create operation override refers to the override of the create operations (**finalizeCreateInput**, **validateCreateInput**, **setPropertiesFromCreateInput**, and **createPost**) on a custom business object.

When a pre-Teamcenter 9.1 custom template is upgraded, the Teamcenter migration service recognizes all the existing postactions that are attached to any of the following create messages and adds them into the attached value list of the **Fnd0MigratedPostActions** global constant:

AE_create_dataset
ITEM_create
ITEM_create_rev
FULLTEXT_create
GRM_create

You can open the **Global Constants Editor** to view the attached value list on the **Fnd0MigratedPostActions** global constant. The format for an attached value is as follows:

business-object-name||*extension-name*||*source-message-name*||
target-message-name

© 2025 Siemens

For example:

```
Item||autoAssignToProject||ITEM_create||IMAN_save
CAEAnalysisRevision||createDefaultDatasets||ITEM_create||IMAN_save
Dataset||setOrgOnCreation||AE_create_dataset||AE_save_dataset
FullText||setOrgOnCreation||FULLTEXT_create||AE_save_dataset
Thumbnail||CreateTNRelationForIR||GRM_create||IMAN_save
```

All the create postactions in the attached value list are automatically dispatched (or executed) as postactions of the save message. In doing so, the create and save pattern has no impact on the existing create postactions, which continue to work without mandatory refactoring. It is mandatory that both the new create postactions and the create operation overrides do not make any database query against the newly created objects.

You can perform the following procedure to refactor an existing create postaction. (The refactoring of the create postactions is optional):

1. Upgrade a pre-Teamcenter 9.1 project by choosing **File**>**Import**>**Business Modeler IDE**>**Import a Business Modeler IDE Template Project**.

   When the project is upgraded, a postaction migration service script is run to identify the postactions attached to the create messages described previously. You can find this service listed in the upgrade log. To see the upgrade log, open the **Project Files** folder and look in the **\output\upgrade** folder.

2. Open the **Global Constants Editor** to view the create postactions added to the **Fnd0MigratedPostActions** global constant:

   Examine the codes for each create postaction and perform one of the following actions:

   - The create postaction does not make any database query against the newly created objects, and no refactoring is needed.

     Use the **Global Constants Editor** to modify the **Fnd0MigratedPostActions** global constant attachment by deleting the attached value corresponding to the postaction. Once removed from this global constant attachment, the postaction is then executed as a postaction of the create operation as before.

   - The create postaction makes a database query against the newly created objects and can be refactored.

     a. Modify the postaction code so that no database query is made against the newly created objects.

     b. Use the **Global Constants Editor** to modify the **Fnd0MigratedPostActions** global constant attachment by deleting the attached value corresponding to the postaction. Once removed

© 2025 Siemens

from this global constant attachment, the postaction is then executed as postaction of the create operation as before.

3. Run the **Create Operation Override Report** by choosing **BMIDE→Reports**. (This report appears in the list of available reports only after upgrading a template project.)

   The report includes overrides of create operations on the following business objects:

   Dataset
   Form
   Item
   ItemRevision

   For each create override operation, examine the codes and ensure no database query is made against the newly created objects.

© 2025 Siemens

© 2025 Siemens

7. Learning about the Business Modeler IDE

Business Modeler IDE interface

Business Modeler IDE interface overview

The Business Modeler IDE is built on the Eclipse platform. Elements of the user interface include perspectives, views, and editors.

A *perspective* is an arrangement of views and editors.

A *view* is a tabbed window within the UI that provides a view of data.

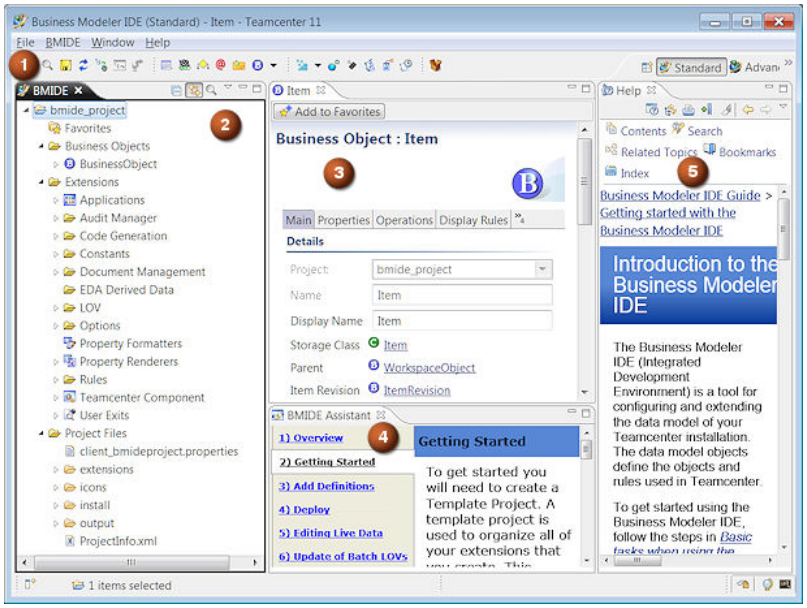
An *editor* is a window that allows you to edit source files.

You can rearrange the user interface by dragging and dropping views and editors.

Two perspectives are provided: **Standard** and **Advanced**. To open one of these perspectives, choose **Window→Open Perspective**.

Standard perspective

The Standard perspective provides the **BMIDE** view, which is a consolidated location for favorites, data model elements, and project files.

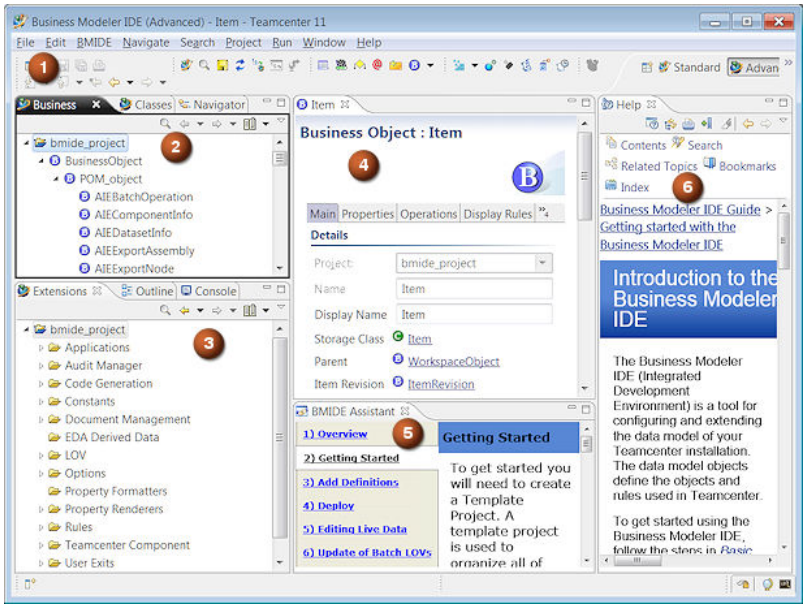


© 2025 Siemens

1	Toolbar	Contains buttons for the most commonly used actions.
2	**BMIDE** view	Provides a single view for favorites, data model elements, and project files.
3	Editor	Allows users to edit data model elements.
4	**BMIDE Assistant** view	Helps new users get started using the Business Modeler IDE.
5	**Help** view	Provides online help for the Business Modeler IDE. You can launch this view by pressing the F1 key or clicking the ? button in the lower left corner of any dialog box.

Advanced perspective

The **Advanced perspective** contains separate views for business objects, classes, project files, and data model extensions.



1	Toolbar	Contains buttons for the most commonly used actions.
2	**Business Objects** view	Displays business objects, the fundamental objects that model business data.
3	**Extensions** view	Displays extensions to the data model.
4	Editor	Allows users to edit data model elements.

© 2025 Siemens

5	**BMIDE Assistant** view	Helps new users get started using the Business Modeler IDE.
6	**Help** view	Provides online help for the Business Modeler IDE. You can launch this view by pressing the F1 key or clicking the button in the lower left corner of any dialog box.

Business Modeler IDE menu commands

The following is a list of menu commands from the Business Modeler IDE user interface. Commands are found in the **BMIDE** menu and on the shortcut menu when you right-click in the user interface.

	Menu command	Description
	Add Business Object	Creates a business object, the fundamental object that models business data.
	Add Condition	Creates a conditional rule.
	Add Id Context	Creates an ID contexts for use with alias or alternate identifiers.
	Add LOV	Creates a list of values.
	Add Naming Rules	Creates a rule to define how an object is named.
	Add Status	Creates a workflow status.
	Add to Favorites	Adds the selected data model element to the favorites list.
	Back	Returns to the previous business object that was selected in the view. (This menu item is available only in the **Advanced** perspective.) You can also use the menu next to the button to choose one of the previous classes.
	**Bookmarks**	Provides a list of bookmarked business objects. (This menu item is available only in the **Advanced** perspective.) Pull down the menu and choose a bookmark to select a business object in the view. The list of bookmarks is preset with some of the common business objects. You can also add business

© 2025 Siemens

	Menu command	Description
		objects to the list using the **Add Bookmark** menu command on the shortcut menu.
	Delete	Deletes the selected data model elements. You can only delete custom objects.
	Deploy Template	Sends extensions to a server.
	Editors	Displays the following menu commands: • Global Constants Editor Edits the values of global constants. • GRM Rules Editor Edits Generic Relationship Management (GRM) rules. • Verification Rules Editor Specifies when **Teamcenter Component** objects can be used in Teamcenter. • Event Type Mappings Editor Displays characteristics of the selected event mapping.
	Filter	Displays the following menu commands when you right-click an object. • **Hide all COTS** *object* **elements** Hides all commercial-off-the-shelf (COTS) objects and displays only the custom data model elements. • **Hide this folder and all** *objects* **below** Hides the selected folder and its contents. • Customize Hidden Groups

© 2025 Siemens

	Menu command	Description
		Allows you to select the data model elements to display in the user interface.
	**Find** *object*	Searches for a data model element. This menu command appears when you right-click an object or folder.
	**Forward**	Displays the next business object that was selected in the view. (This menu item is available only in the **Advanced** perspective.) You can also use the menu next to the button to choose one of the next business objects.
	Generate Code→C++ Classes	Generates C++ source code for the selected business object.
	**Hide COTS**	Hides the COTS (commercial-off-the-shelf) data model elements and displays only the custom data model elements you have created.
	Navigate	Displays the following menu commands: • **Expand Selection**  Expands all the child objects below the selected business object. • **Collapse Selection**  Collapses all the child objects below the selected business object. • **Go to→Parent** Takes you to the parent business object.
	New *object*	Runs a wizard to add a new object. This menu command appears when you right-click an object or folder. You can also add objects using the **New Model Element** button  .

© 2025 Siemens

	Menu command	Description
	New Model Element	Adds custom data model elements.
	Open	Opens the selected object.
	Open Event Type Mappings Editor	Displays an editor to connect an event to a business object type.
	Open Global Constants Editor	Allows you to modify existing global constants.
	Open GRM Rules Editor	Allows you to add, modify, or remove Generic Relationship Management (GRM) rules. A GRM rule applies constraints on the relationship between two business objects.
	Open in UML Editor	Displays the business object in the UML editor. Also creates a **.tmd** file for the UML view of the business object and places this file in the **Project Files\output** folder.
	Open Verification Rules Editor	Displays available verification rules that specify when **Teamcenter Component** objects can be used in Teamcenter.
	Organize	Displays the following menu commands when you right-click a data model element: • Set active extension file Sets the file in which to place model changes. The extension files reside in the **Project Files\extensions** folder of the project. • Move Moves the selected custom model element to an extension file in which model changes are placed. The extension file can be in the current project or another project. This menu command is enabled only when a custom item is selected (custom items appear with a **c** symbol). The extension file resides in the **Project Files\extensions** folder of the project. • **Set as active Library**

© 2025 Siemens

	Menu command	Description
		Sets the selected library as the one in which to save library objects. This menu command is enabled only when a library is selected in the **Extensions\Code Generation\Libraries** folder. • **Set as active Release** Sets the selected release as the one in which to associate custom data model elements. This menu command is enabled only when a release level is selected in the **Extensions\Code Generation\Releases** folder.
	**Organize Extensions**	Displays the following menu commands: • **Set active extension file** Sets the file in which to place model changes. The extension files reside in the **Project Files\extensions** folder of the project. • **Add new extension file** Adds a file in which to place model changes. • **Move Model Elements to Extensions File** Moves the selected custom model elements to a file in which model changes are placed. This menu command appears only when a custom item is selected (custom items appear with a **c** symbol). The extension file resides in the **Project Files\extensions** folder of the project. • **Add localization files** Adds language support files. The files reside in the **Project Files\extensions\lang** folder. • **Import a localization file** Imports language support files from an external source. The files are imported to the **Project Files\extensions\lang** folder.

© 2025 Siemens

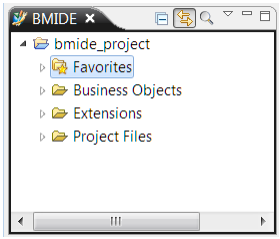
	Menu command	Description
	Generate Software Package	Packages the custom data model objects for installation to a production server.
	**Reload Data Model**	Reloads the data model for the selected project.
	Rename	Applies a new name to the business object.
	Reports	Runs reports on the data model in the selected project.
	Save Data Model	Saves the custom data model objects in the project.
	Search Conditions	Allows you to look for conditions in the system. Conditions are conditional statements that resolve to true or false. You can also see all the conditions in the **Extensions** view in the **Rules→Conditions** folder.
	Tools	Displays the following menu commands: • Merge Samples Provides samples to learn how to use the live update merge tool • Push Template to Reference Directory Copies a template to a reference directory so that you can build another project on top of it.
	Upgrade Tools	Displays the following menu commands: • Re-run Template Project Upgrade Wizard Upgrades the project template to the newest version of the Business Modeler IDE. • **Property Name and Relation Name Migration Wizard** Migrates the text for property and relation names from a previous project.

© 2025 Siemens

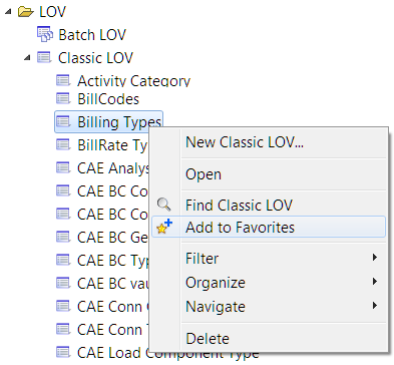
Menu command	Description
	• **Default Location Creation Wizard** Creates a default set of localization files for an upgraded project.

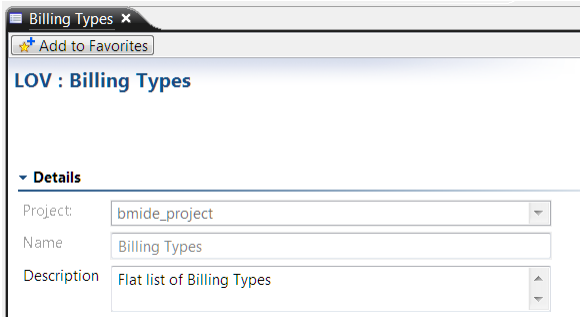
Set favorites in the Business Modeler IDE

The **Favorites** folder holds data model elements for quick access.



To add elements to the **Favorites** folder, right-click an element and choose **Add to Favorites** from the context menu, or click the **Add to Favorites** button on an editor.





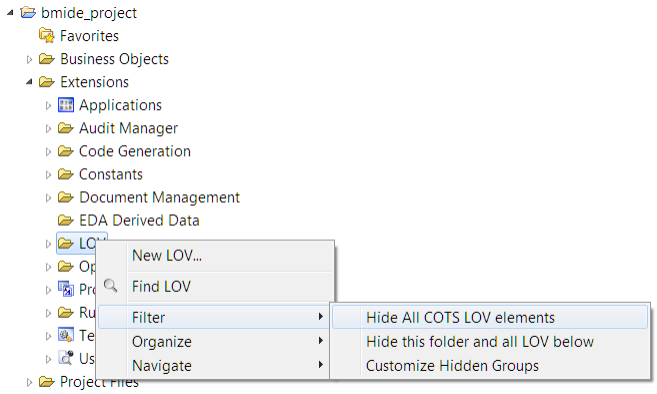
© 2025 Siemens

Filter visible elements in the Business Modeler IDE

Filtering hides the selected element category or only hides COTS (commercial-off-the-shelf) elements. This reduces screen clutter.

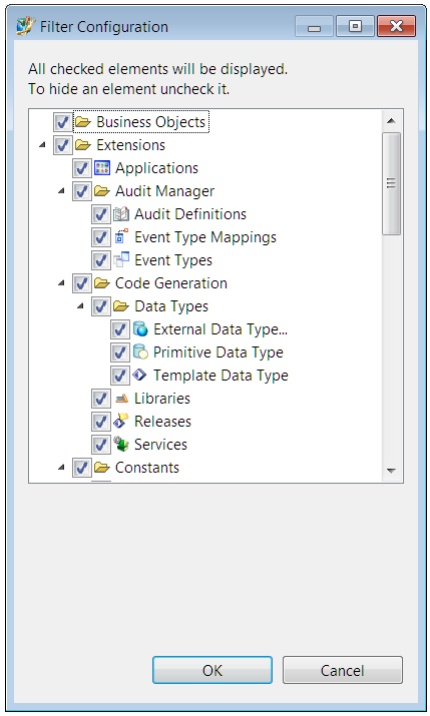
- Hide elements

  Right-click a folder and choose **Filter**>**Hide All COTS** *element-name* **elements** or **Hide this folder and all** *element-name* **below**.



- Customize hidden groups

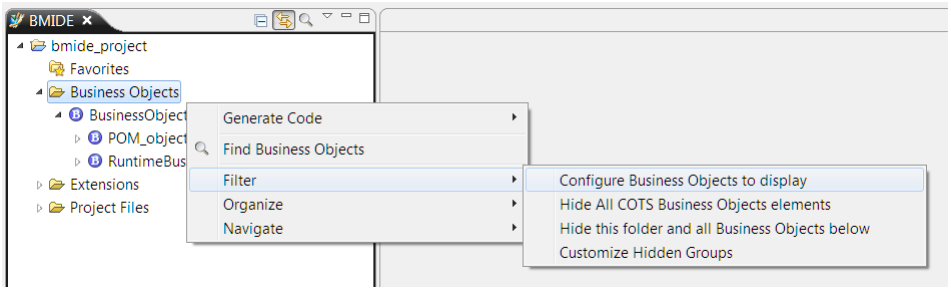
  Choose **Filter**>**Customize Hidden Groups** to quickly hide entire categories of element types.



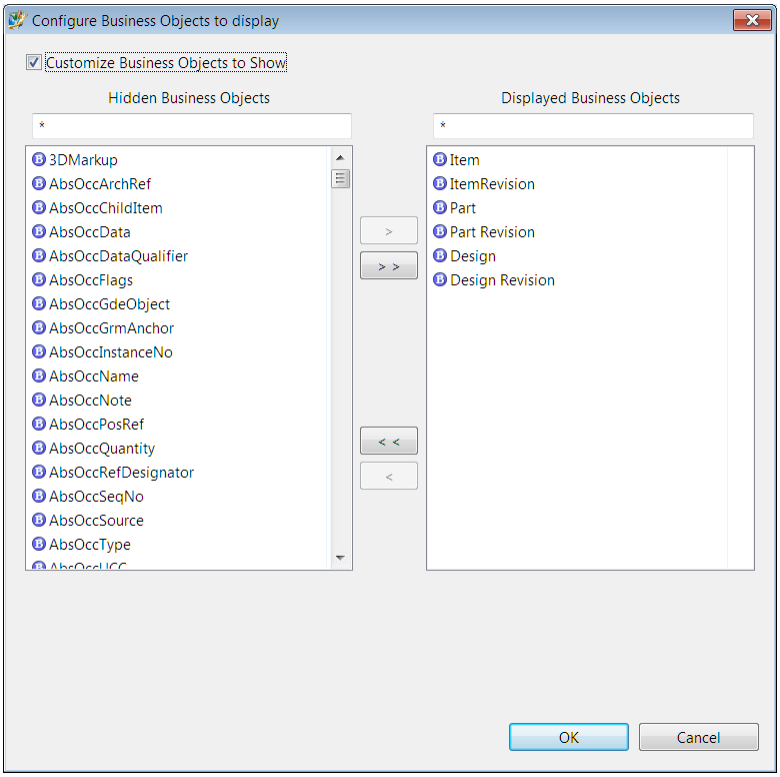
© 2025 Siemens

- Configure business objects to display

  To filter business objects, right-click the **Business Objects** folder and choose **Filter**>**Configure Business objects to display**.



Select the **Customize Business Objects to Show** check box to display only the selected business object types.



Business Modeler IDE workshops

What are the BMIDE workshops?

The Business Modeler IDE workshops are a series of short procedures that guide you through creating and configuring a new item type. This is a common configuration task. The workshops show you how to:

© 2025 Siemens

- Create a business object to represent a part.
- Add pieces to the business object to make it fully functional, such as

  - properties
  - naming rules
  - lists of values
  - relationship rules
  - deep copy rules

The workshops build on one another. When you complete the sequence, you will have a representative set of custom functionality for a custom item.

Prerequisites for the BMIDE workshops

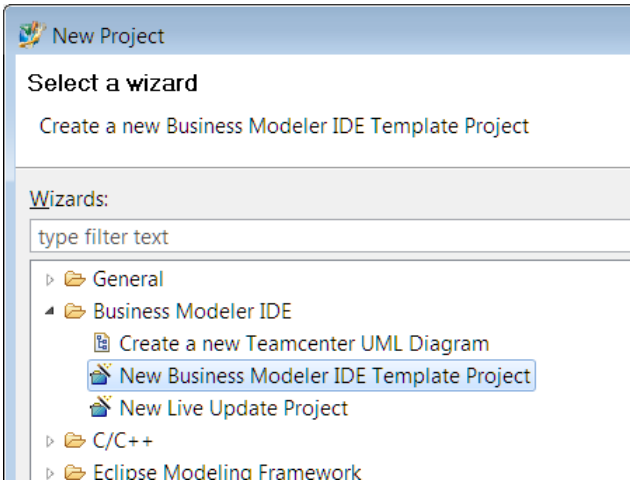
In order to perform the BMIDE workshops, the following software should be installed:

- Business Modeler IDE
- A test Teamcenter server to which you can deploy your custom data model
- A rich client to connect to the test server

Workshop 1: Create a template project

Perform this workshop to **create a template project** to hold your customizations.

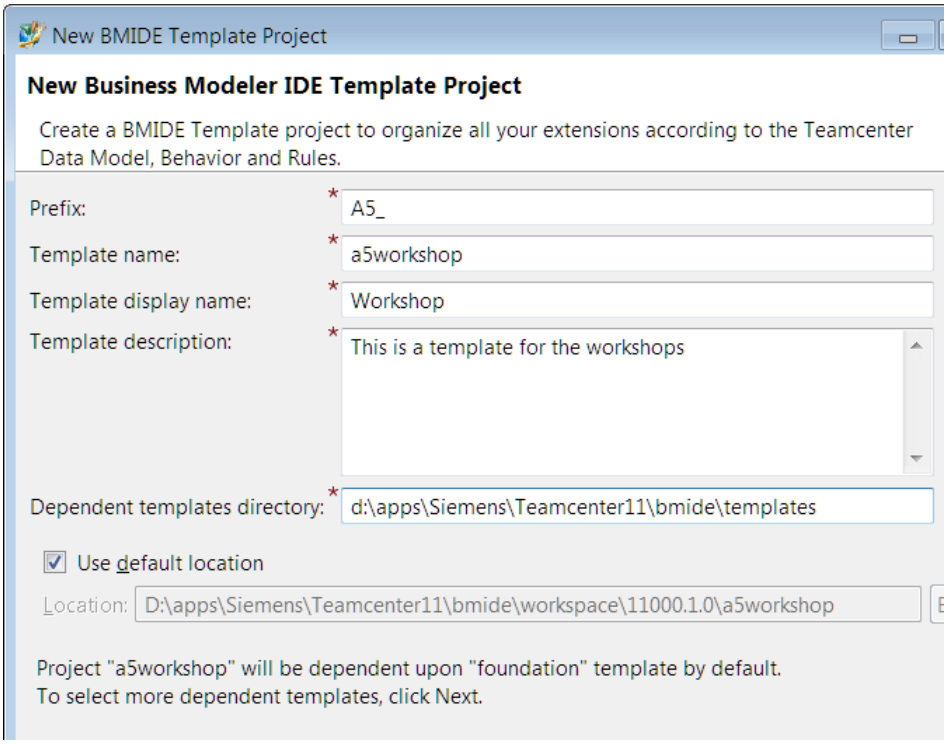
1. Open the BMIDE and choose **File→New→Project→Business Modeler IDE→New Business Modeler IDE Template Project**, and then click **Next**.



2. In the **New Business Modeler IDE Template Project** dialog box, enter the following project parameter values.

© 2025 Siemens

For this parameter	Do this
Prefix	Type **A5_**. This prefix is placed on every data model item you create to identify it as belonging to this project.
Template name	Type **a5workshop**.
Template display name	Type **Workshop**.
Template description	Type a brief description of the template.
Dependent templates directory	Specify the directory where the data model template XML files are stored. Normally, the template files are in the Teamcenter install location *\TR\bmide\templates*. Caution: Do not point to the templates in the data location **\TD\model** . The templates in **\TD\model** are used only for database updates, and the BMIDE must never use the templates in that folder.



© 2025 Siemens

3. Click **Finish**.

Workshop 2: Explore the user interface

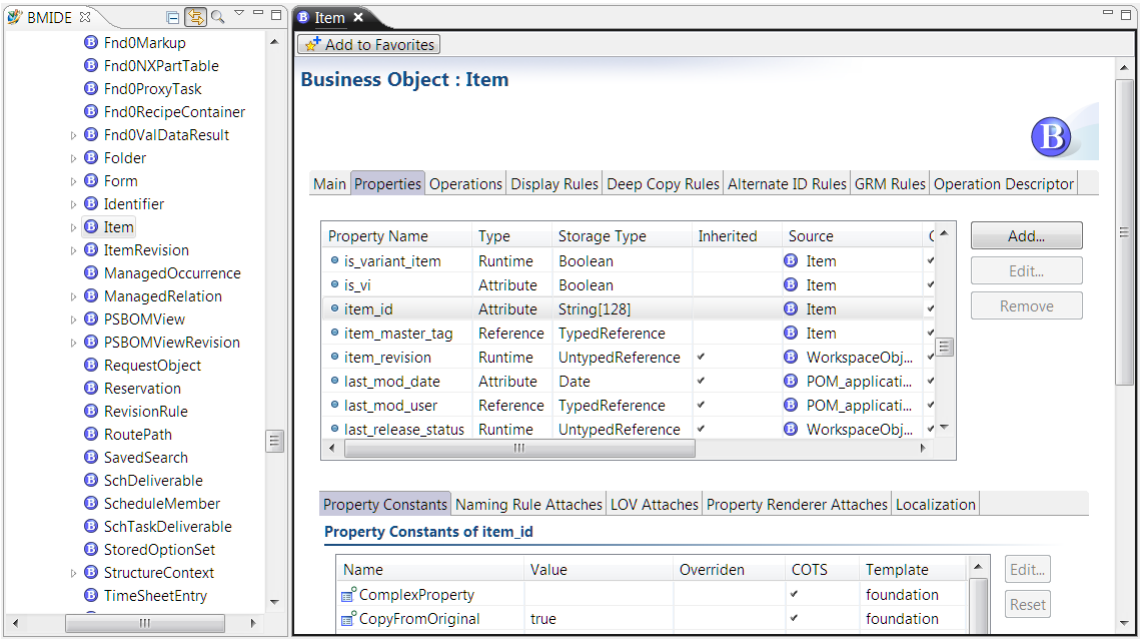
Perform this workshop to become familiar with the **Business Modeler IDE** user interface.

1. In the **BMIDE** view of the **Standard** perspective, expand the **workshop** project.

2. Expand the **Business Objects** folder and notice how the types (business objects) are arrayed in a hierarchical tree. Child types inherit their characteristics from their parents. Expand the **Extensions** folder and notice all the other types of data model objects you can create.

3. Click the **Find** button on the **BMIDE** view toolbar and search for **Item**.

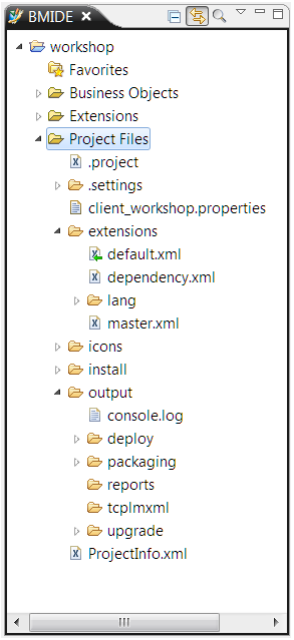
4. Double-click the **Item** business object and click the **Properties** tab to view properties on the business object, such as the **item_id** property. This is where you can add new properties to business objects.



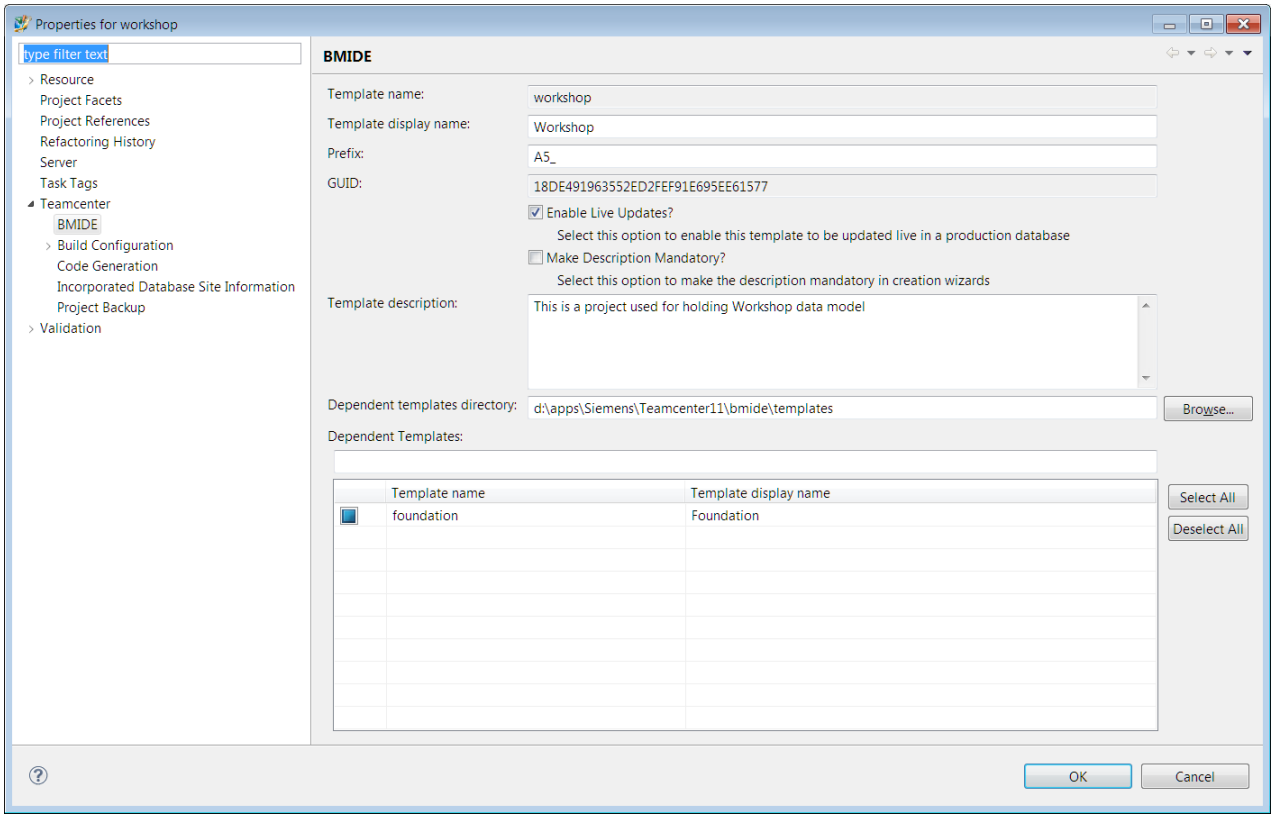
5. Close the **Business Objects** folder and open the **Project Files** folder. The **extensions\default.xml** file contains the custom data model you create. The **output** folder contains files generated when you deploy your project to a server or package your project into a template.

   These files are maintained in the project workspace, which is located at *installation-location***bmide\workspace*****version***workshop**.

© 2025 Siemens

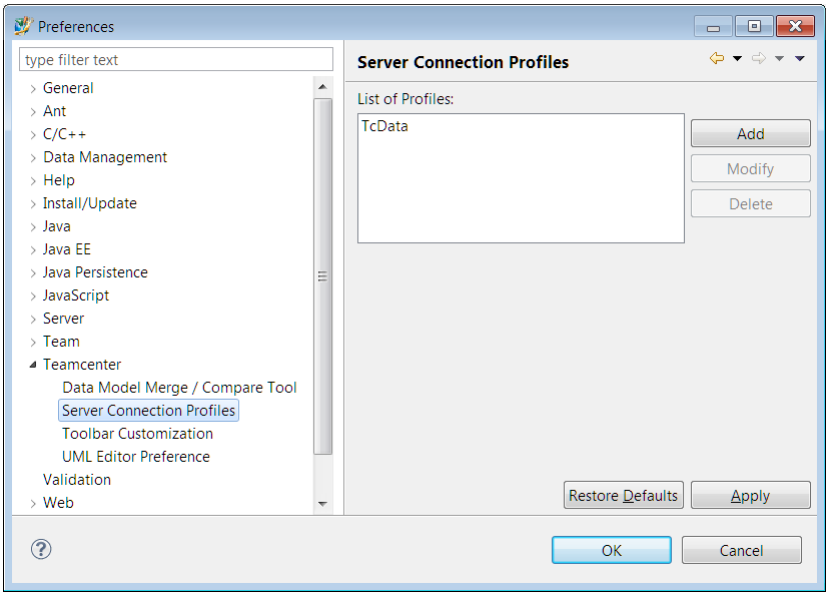


6. To see the project properties, right-click the **workshop** project and choose **Properties**. You can change the project settings here.



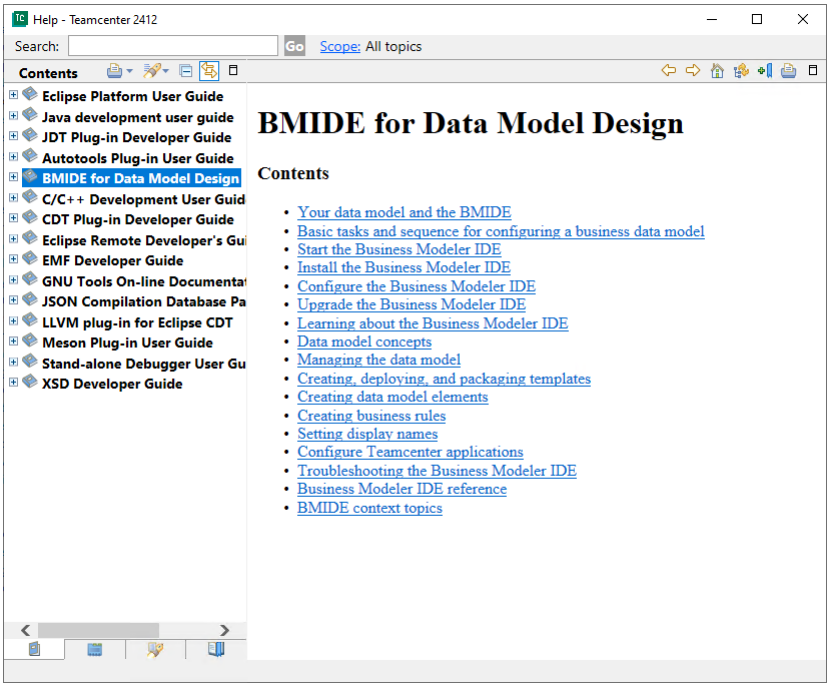
© 2025 Siemens

7. To verify that you have a server connection profile to connect to your test server, choose **Window→Preferences→Teamcenter→Server Connection Profiles**. This profile was set up automatically when you installed the Business Modeler IDE.



8. You can access online help by pressing the F1 key or clicking the question mark button in the lower left corner of a dialog box.

   To access the online manual, choose **Help→Help Contents**, and in the **Help** window, choose **BMIDE for Data Model Design** in the left pane.



© 2025 Siemens

Workshop 3: Create a new item type

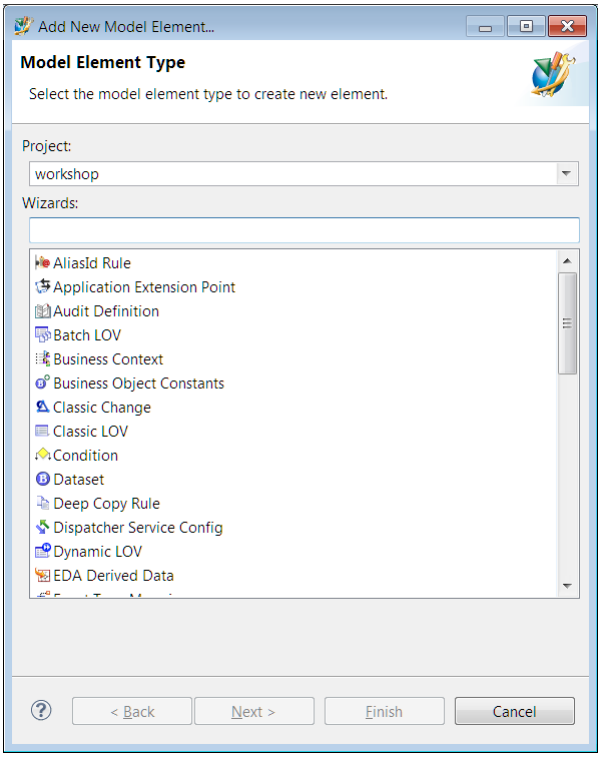
Perform this workshop to **create custom item business objects** to represent new types of objects. Business objects are the fundamental objects used to model data. You create business objects to represent product parts, documents, change processes, and so on.

This workshop shows you how to create a child of the **Item** business object. **Item** is the most common business object under which children are created, and it is used to represent product parts.

1. Create a new item to represent a part.

   a. On the toolbar, choose **BMIDE**>**New Model Element**.

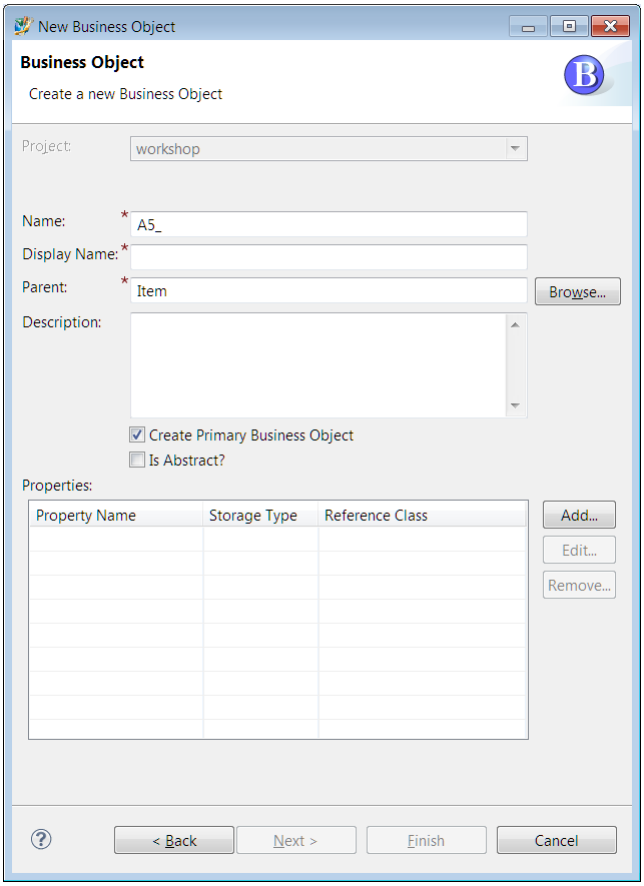
      The **Model Element Type** wizard is displayed.



   b. Type **Item** in the **Wizards** box and click **Next**.

      The **Business Object** wizard is displayed.

© 2025 Siemens



c. Name the item **A5_WorkshopPart** and specify the display name as **Workshop Part**.

> Note:
>
> You could add custom properties in this dialog box, but instead you add them later on the **Properties** tab of the new business object.

d. Click **Finish**.

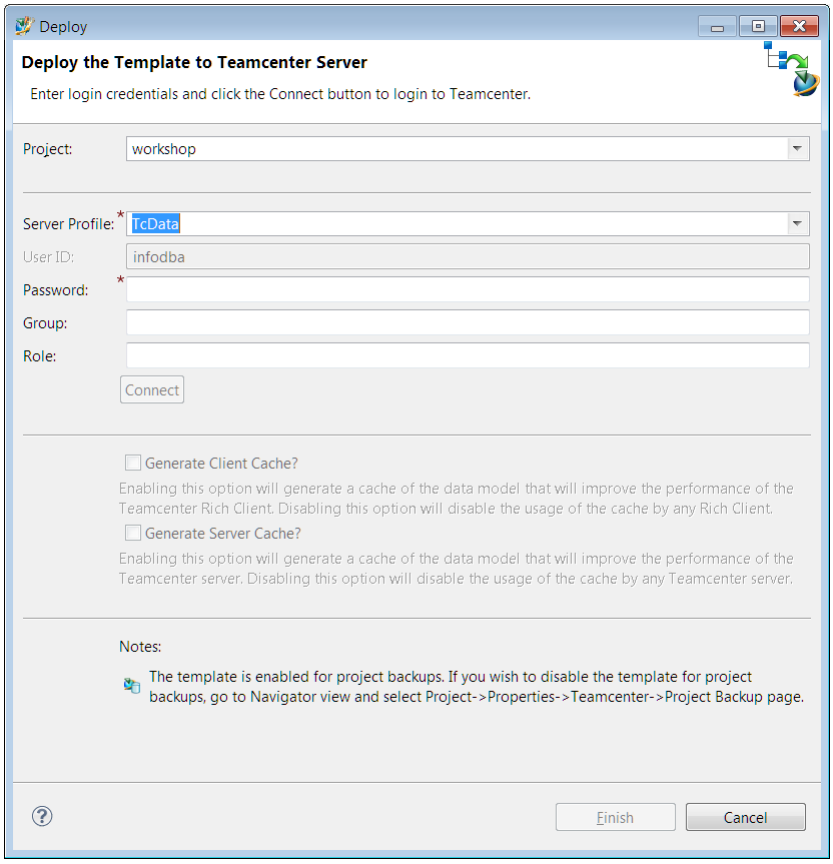
2. Deploy the changes to the rich client.

a. On the menu bar, choose **BMIDE**>**Save Data Model** to save your changes.

b. Ensure that the Teamcenter test server is running.

c. On the menu bar, choose **BMIDE**>**Deploy Template**. Type the password and click the **Connect** button to establish a connection with the server.

© 2025 Siemens



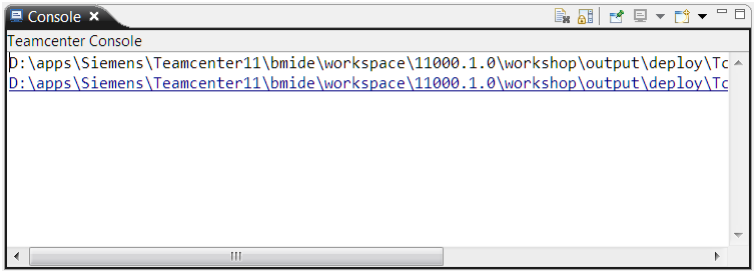
d. Select the **Generate Server Cache?** check box to generate shared server cache that contains the new data model.

> Note:
>
> If you do not select the **Generate Server Cache?** check box when you deploy schema changes (for example, changed business objects and properties), when you log on to the test server, you may see the following message:
>
> `The schema file is out of date. Please regenerate.`

e. Click **Finish**.

f. When deployment is done, check the status in the **Console** view.

© 2025 Siemens



g. To ensure your changes appear in the rich client user interface, exit the rich client and restart it to clear cache.

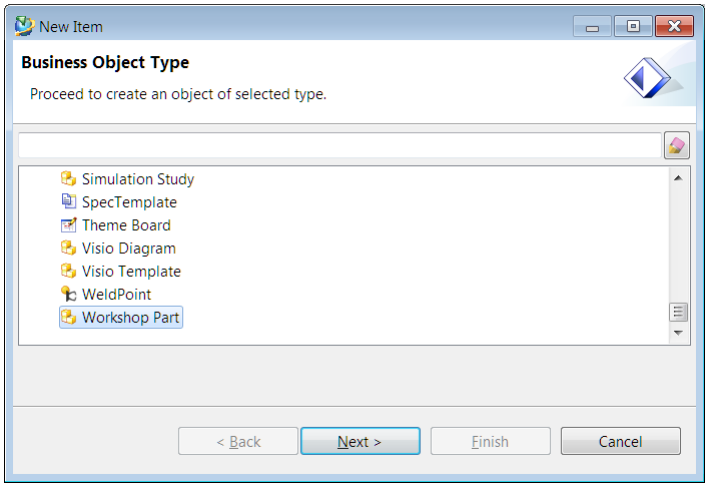
When you restart, you may want to launch the rich client using the *install-location***portal\portal.bat -clean** option to ensure the cache is cleared.

> Note:
>
> If your changes still do not appear in the user interface, delete the **Teamcenter** subdirectory in the user's home directory on the client. This directory is automatically created again when the user starts the rich client.
>
> On a Windows client, it is typically **Users***user-name***\Teamcenter**. On a Linux client, it is typically **$HOME/Teamcenter/**.

3. Verify creation of the new item.

a. Run the rich client.

b. In My Teamcenter, choose **File**>**New**>**Item** and select **Workshop Part** from the list.



c. Click **Next** to create an instance of the item.

© 2025 Siemens

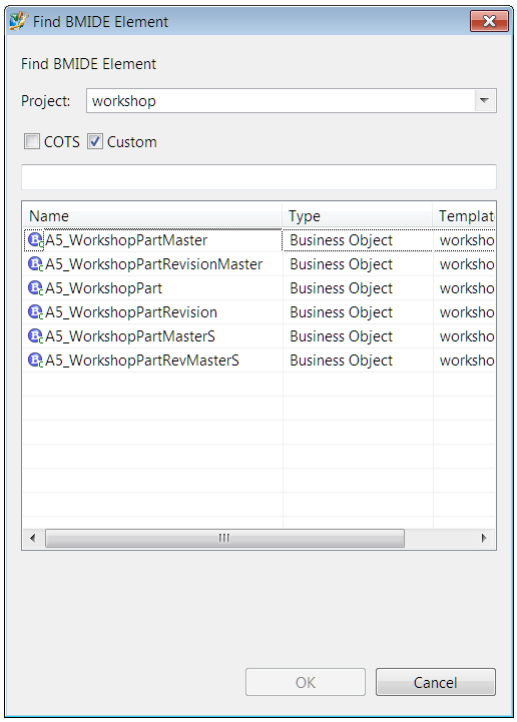
Note:

You can **create your own custom icon** to assign to the new business object.

Workshop 4: Create custom properties

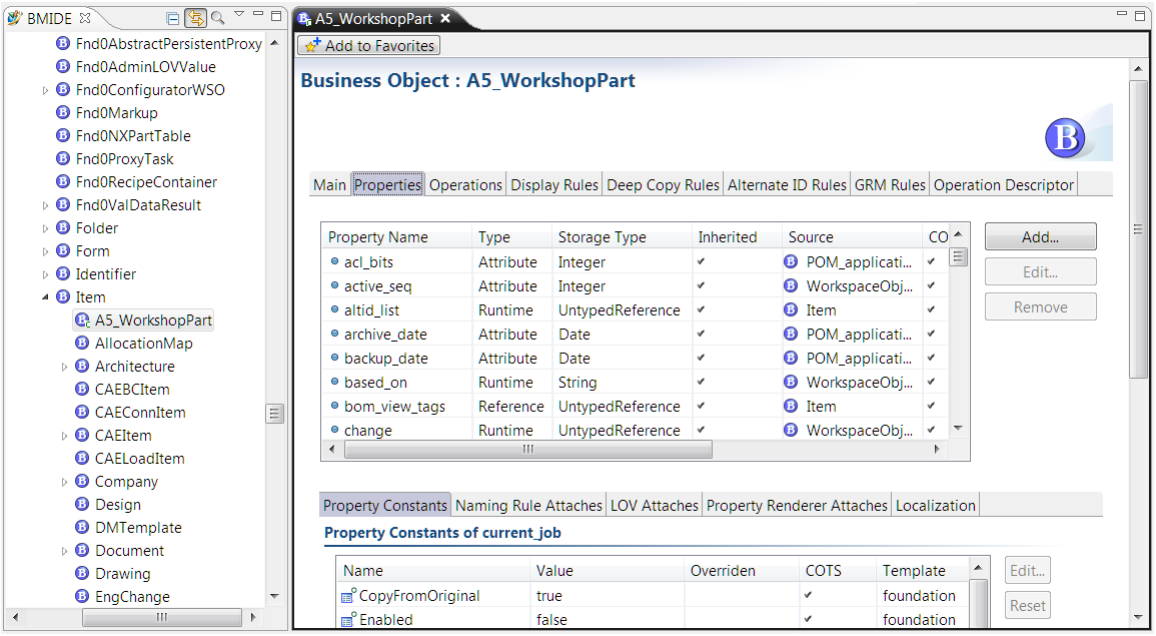
Perform this workshop to **create new properties** to contain information for your custom workshop part business object. (Properties contain object information such as name, number, description, and so on.)

1. When you create an item business object, revision and form business objects are also created. To see them, click the **Find** button and clear the **COTS** check box in the **Find BMIDE Element** dialog box.



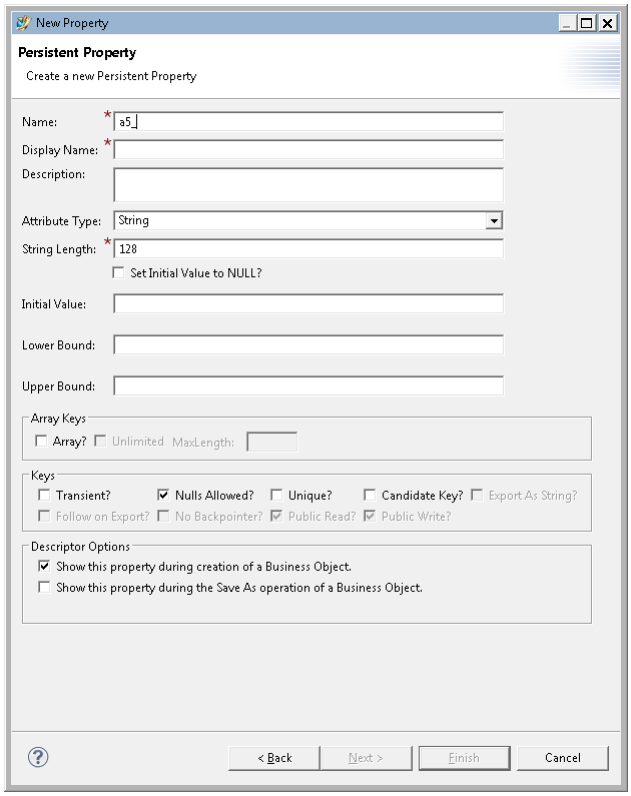
2. Select the **A5_WorkshopPart** business object. When it opens, click the **Properties** tab.

© 2025 Siemens



3. To the right of the **Properties** tab, click the **Add** button, select the **Persistent** property type, and click **Next**.

   The **Persistent Property** dialog box is displayed.



© 2025 Siemens

4. Add the following persistent properties.

Name	Display name	Attribute type
a5_supplier	Supplier	LongString
a5_safety_code	Safety Code	Integer

When you create the properties, ensure that the **Show this property during creation of a Business Object** ☑ check box is selected.

5. Enable the new properties for use in the client user interface using the **Enabled** properties constant.

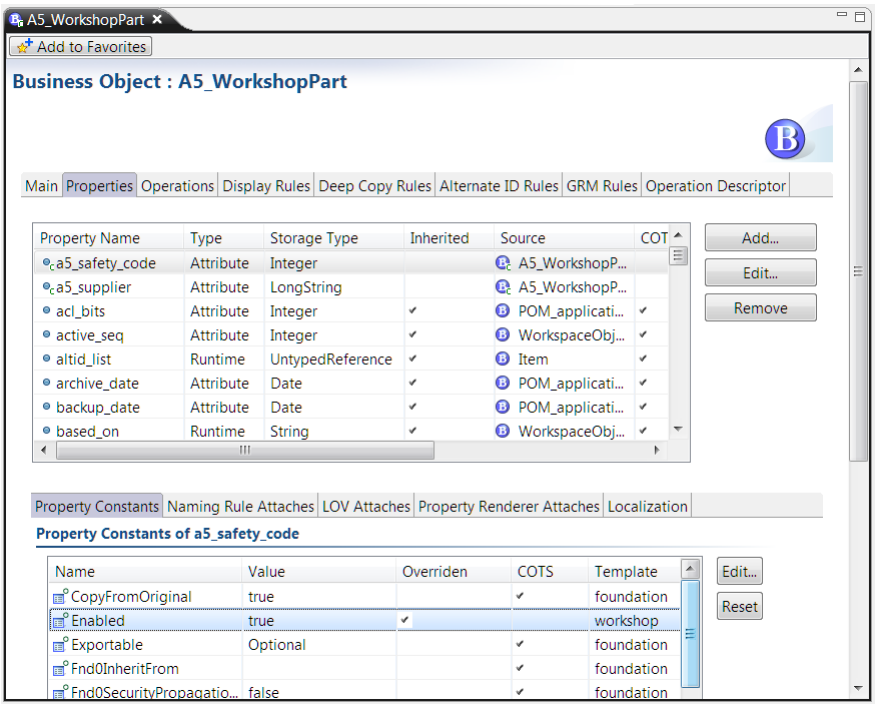
   a. Select the new property in the properties table.

   b. In the **Property Constants** table, select the **Enabled** constant.

   c. Verify that its value is set to **true**.

      If it is not, click the **Edit** button to the right of the **Property Constants** table and set it to **true**.

   d. Repeat these steps for all the custom properties.



6. Select the **A5_WorkshopPartRevision** business object. Open it and click the **Properties** tab.

© 2025 Siemens

7. To the right of the **Properties** tab, click the **Add** button, and select the **Persistent** property type, and click **Next**.

8. Add the following persistent properties.

Name	Display name	Attribute type
a5_weight	Weight	Double
a5_material_code	Material Code	String
a5_material_description	Material Description	LongString

When you create the properties, ensure that the **Show this property during creation of a Business Object** check box is selected.

9. Enable the new properties for use in the client user interface using the **Enabled** properties constant.

10. Deploy the changes to the rich client.

    a. On the menu bar, choose **BMIDE**>**Save Data Model** to save your changes.

    b. Ensure that the test server is running.

    c. On the menu bar, choose **BMIDE**>**Deploy Template**. Type the password, click the **Connect** button, and when a connection is established, select the **Generate Server Cache?** check box and click **Finish**.

    d. When deployment is done, check the status in the **Console** view.

    e. To ensure your changes appear in the rich client user interface, exit the rich client and restart it to clear cache.

    When you restart, you may want to launch the rich client using the *install-location***portal****portal.bat -clean** option to ensure the cache is cleared.

    > Note:
    >
    > If your changes still do not appear in the user interface, delete the **Teamcenter** subdirectory in the user's home directory on the client. This directory is automatically created again when the user starts the rich client.
    >
    > On a Windows client, it is typically **Users***user-name***Teamcenter**. On a Linux client, it is typically **$HOME/Teamcenter/**.

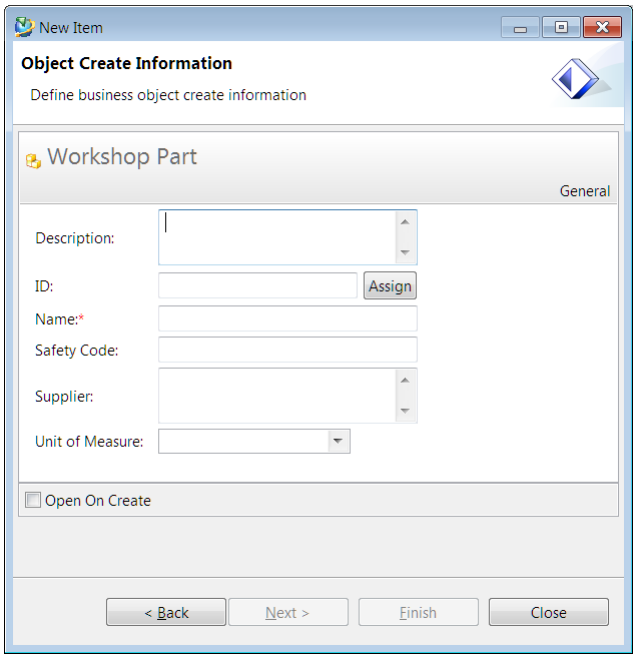
11. Verify the new properties in the user interface.

© 2025 Siemens

a. Run the rich client.

b. In My Teamcenter, choose **File→New→Item** and select **Workshop Part** from the list.

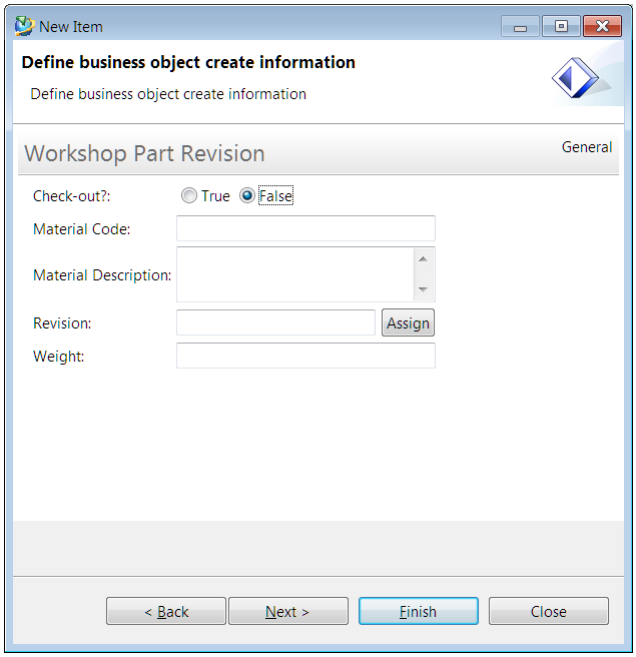
The new custom item properties appear in the item creation dialog box. This is a result of adding the properties using the **Show this property during creation of a Business Object** check box in the **New Property** dialog box. (Selecting this check box also places the properties on the **CreateInput** operation on the **Operation Descriptor** tab in the Business Modeler IDE.)



c. Click **Next** to create the master form for the workshop part.

d. Click **Next** to see the new custom item revision properties in the item revision creation dialog box.

© 2025 Siemens



The properties do not automatically appear elsewhere in the user interface unless you **make some additional changes**.

Workshop 5: Display custom properties in the client user interface

Object properties (such as name, number, description, and so on) are displayed in many places in the user interface. Perform this workshop to display the custom properties for the workshop part in the **Summary** tab and the **Viewer** tab using XML style sheets. XML style sheets allow you to change display in the rich client interface.

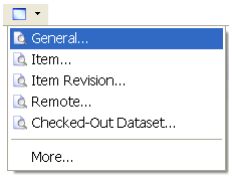
> Note:
>
> Although this is not a Business Modeler IDE exercise, it is important to know how to display your custom properties in the client so that you can take full advantage of the custom properties you create in the Business Modeler IDE. XML style sheets are only one method you can use to display custom properties. Others include creating your own Java forms and exposing the custom properties using JavaBeans or abstract rendering.

1. Find style sheets in the rich client.

   a. Click the **Open Search View** button on the menu bar  .

   b. Click the arrow on the **Select a Search** button and choose **General**.

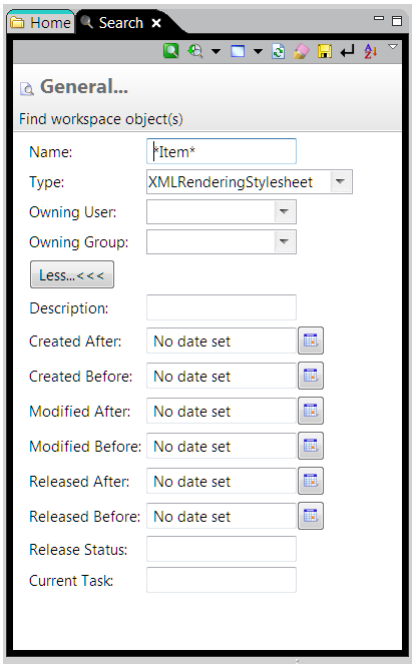
© 2025 Siemens



c. Clear the **Owning User** and **Owning Group** boxes.

d. In the **Type** box, type **XMLRenderingStylesheet**.

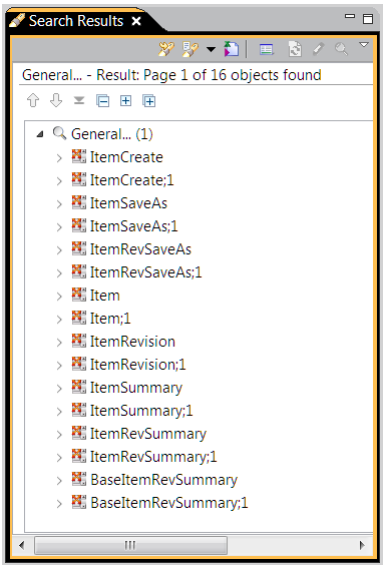
To find all style sheets for items or item revisions, type ***Item*** in the **Name** box. (You want to find item and item revision style sheets because the workshop part is based on the item business object, and you want to use the same types of style sheets.)



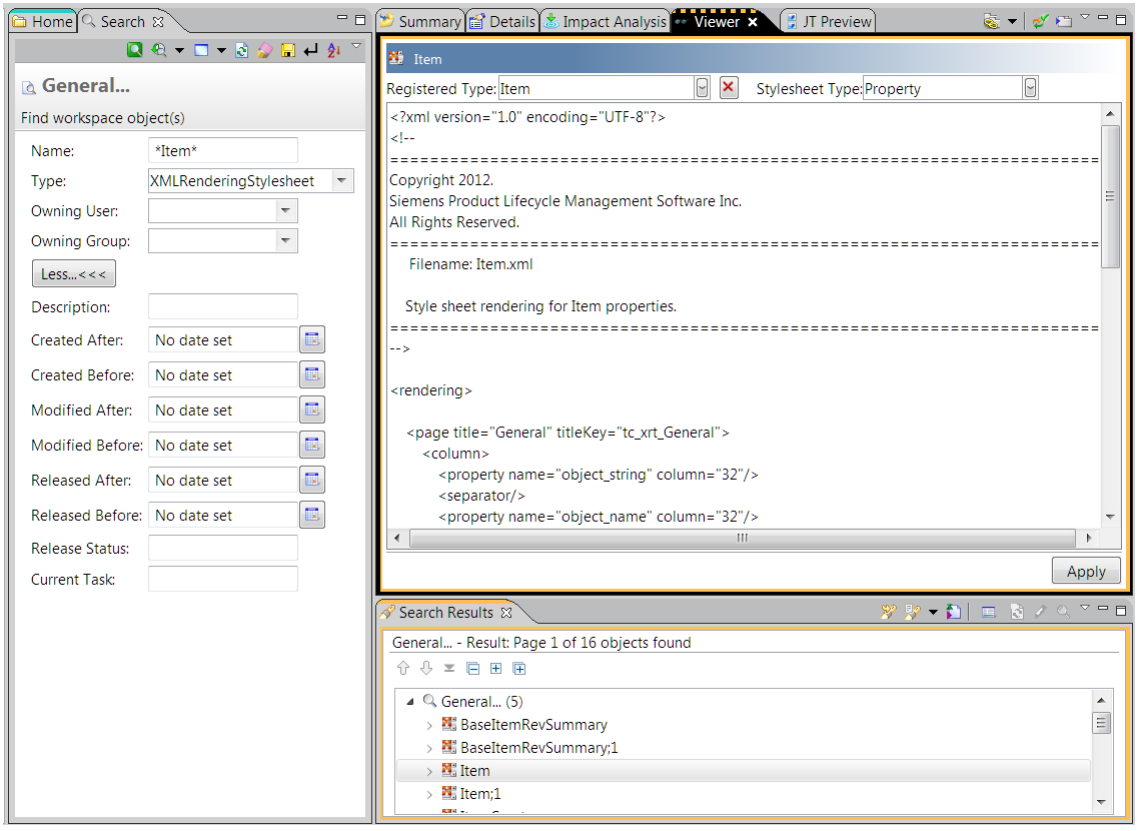
e. Press the Enter key or click the **Execute the Search** button .

The results are displayed in the **Search Results** view.

© 2025 Siemens



f. In the **Search Results** tab, select the style sheet you want to view. Click the **Viewer** tab to see the style sheet.



2. Save style sheets for use with the custom business objects.

a. In the **Search Results** view, select a style sheet.

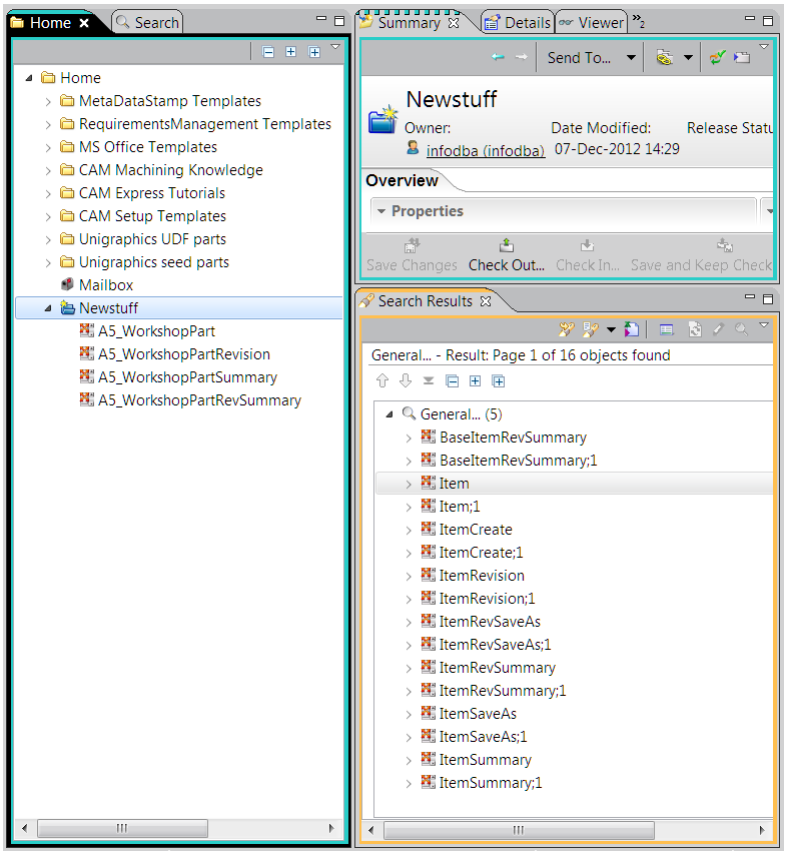
© 2025 Siemens

b. Choose **File**>**Save As** to rename the style sheet for use with your custom business object.

Save the following style sheets with new names.

Original XML style sheet	Save as
Item	A5_WorkshopPart
ItemRevision	A5_WorkshopPartRevision
ItemSummary	A5_WorkshopPartSummary
BaseItemRevSummary	A5_WorkshopPartRevSummary

The new style sheet datasets are saved in your **Newstuff** folder in the **Home** view.



c. Change the named reference for the file by selecting each style sheet dataset and choosing **View**>**Named References**. In the **Name** column in the **Named References** dialog box, change the old name of the file to the new save as file name.

3. Register the new style sheets for use with the custom business objects.

a. Select a custom style sheet, such as **A5_WorkshopPart**, and click the **Viewer** tab.

© 2025 Siemens

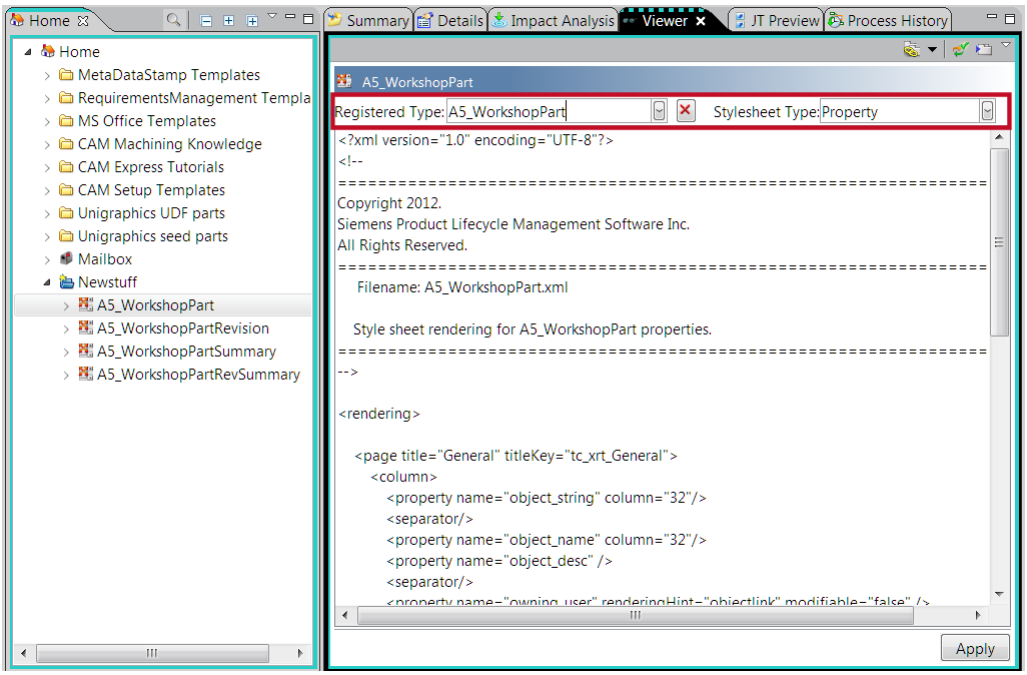
b. In the **Viewer** tab, click the arrow in the **Registered Type** box to select the business object type you want to register it to, and click the arrow in the **Stylesheet Type** box to select the place in the user interface where the style sheet is used.

c. Change the file comments at the top of the file as needed. For example, revise the file name and the description of the file.

d. Click the **Apply** button in the lower right corner of the view.

Assign the registered type and style sheet type as follows.

Style sheet name	Registered type	Style sheet type
A5_WorkshopPart	**Workshop Part (A5_WorkshopPart)**	**Property**
A5_WorkshopPartRevision	**Workshop Part Revision (A5_WorkshopPartRevision)**	**Property**
A5_WorkshopPartRevSummary	**Workshop Part Revision (A5_WorkshopPartRevision)**	**Summary**
A5_WorkshopPartSummary	**Workshop Part (A5_WorkshopPart)**	**Summary**



4. Edit the style sheets to include the properties you want displayed in the layout.

a. Select a custom style sheet, such as **A5_WorkshopPart**, and click the **Viewer** tab.

b. Add the custom properties.

© 2025 Siemens

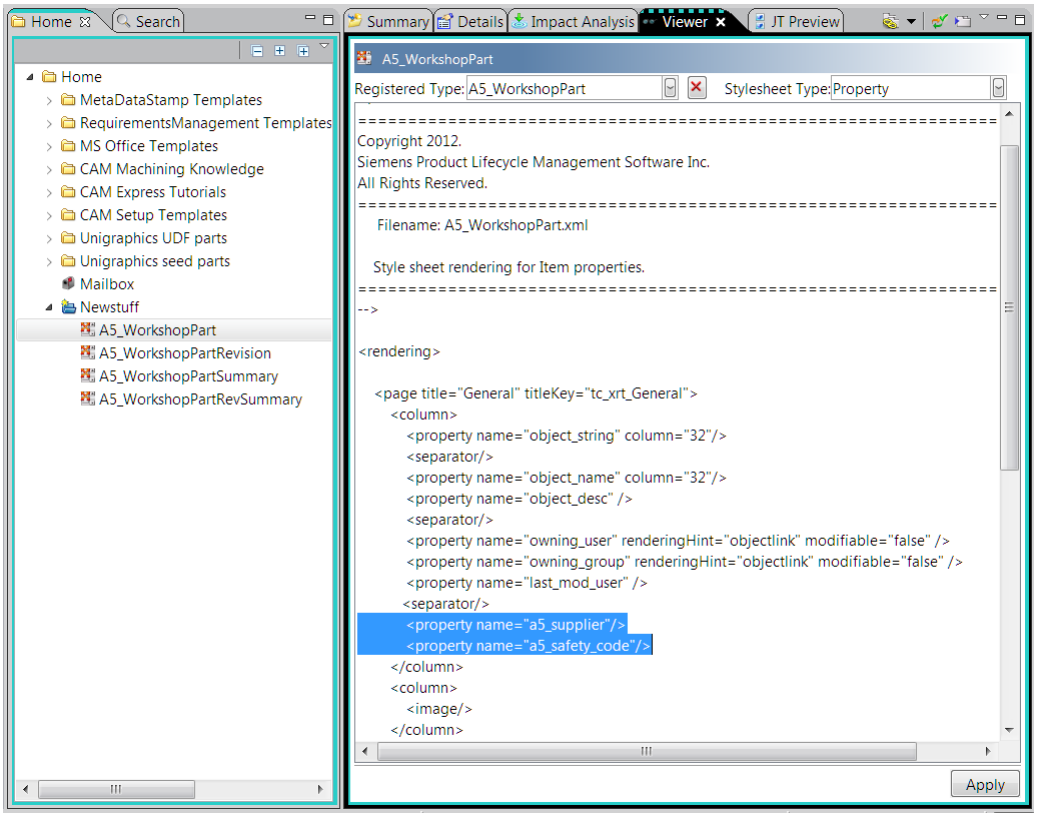
- Place the following custom item properties in the workshop part style sheets (**A5_WorkshopPart** and **A5_WorkshopPartSummary**):

```
<property name="a5_supplier"/>
<property name="a5_safety_code"/>
```

- Place the following custom item revision properties in the workshop part revision style sheets (**A5_WorkshopPartRevision** and **A5_WorkshopPartRevSummary**):

```
<property name="a5_weight"/>
<property name="a5_material_code"/>
<property name="a5_material_description"/>
```

c. After making changes, click the **Apply** button in the lower right corner of the view.



5. Verify the new properties in the user interface.

a. To ensure your changes appear in the rich client user interface, exit the rich client and restart it to clear cache.

When you restart, you may want to launch the rich client using the *install-location***portal\portal.bat -clean** option to ensure the cache is cleared.

© 2025 Siemens

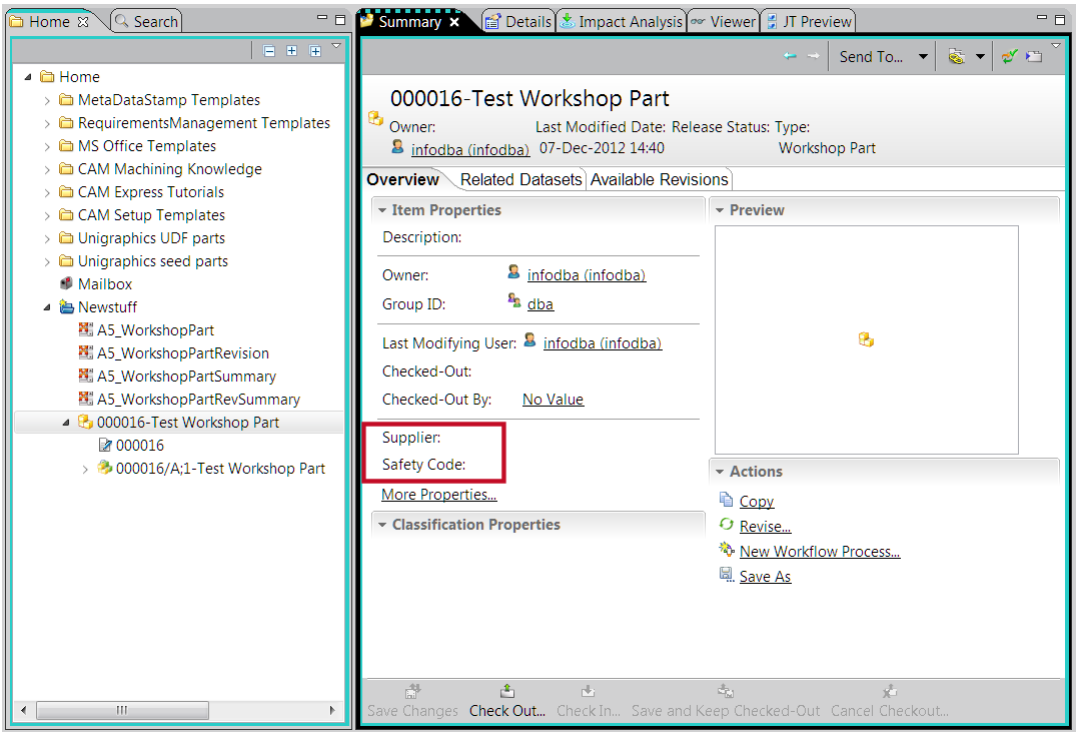
Note:

If your changes still do not appear in the user interface, delete the **Teamcenter** subdirectory in the user's home directory on the client. This directory is automatically created again when the user starts the rich client.

On a Windows client, it is typically **Users***user-name***Teamcenter**. On a Linux client, it is typically **$HOME/Teamcenter/**.

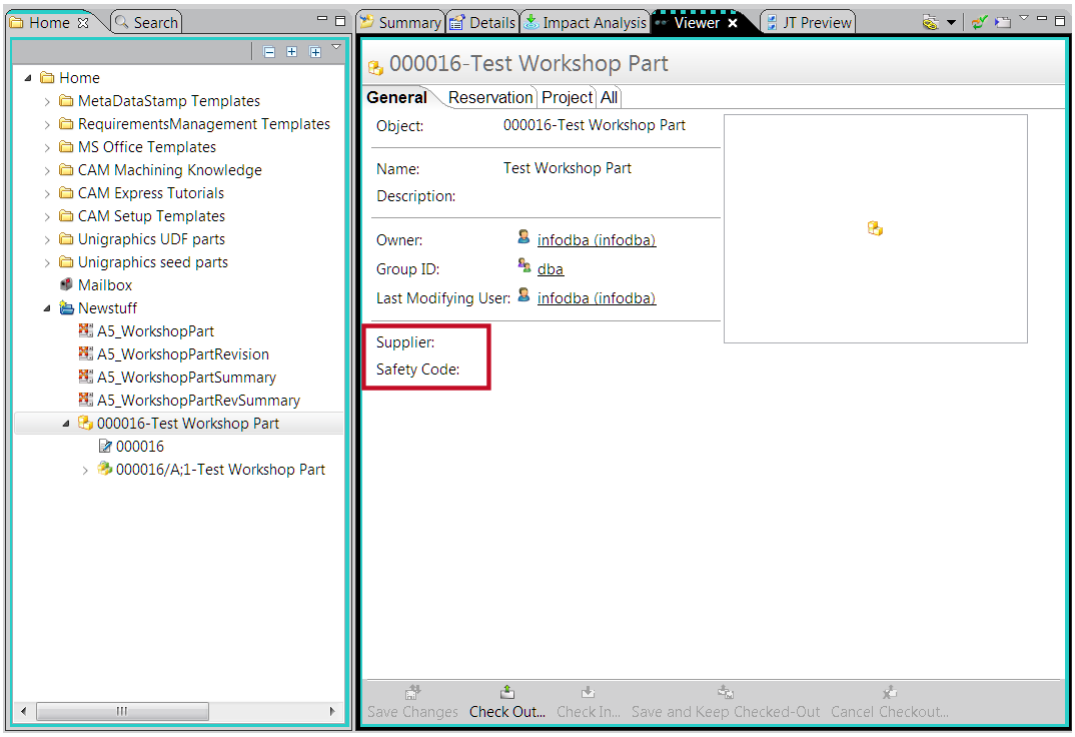
b. In My Teamcenter, select a workshop part item and click the **Summary** tab.

The new properties appear in the user interface.

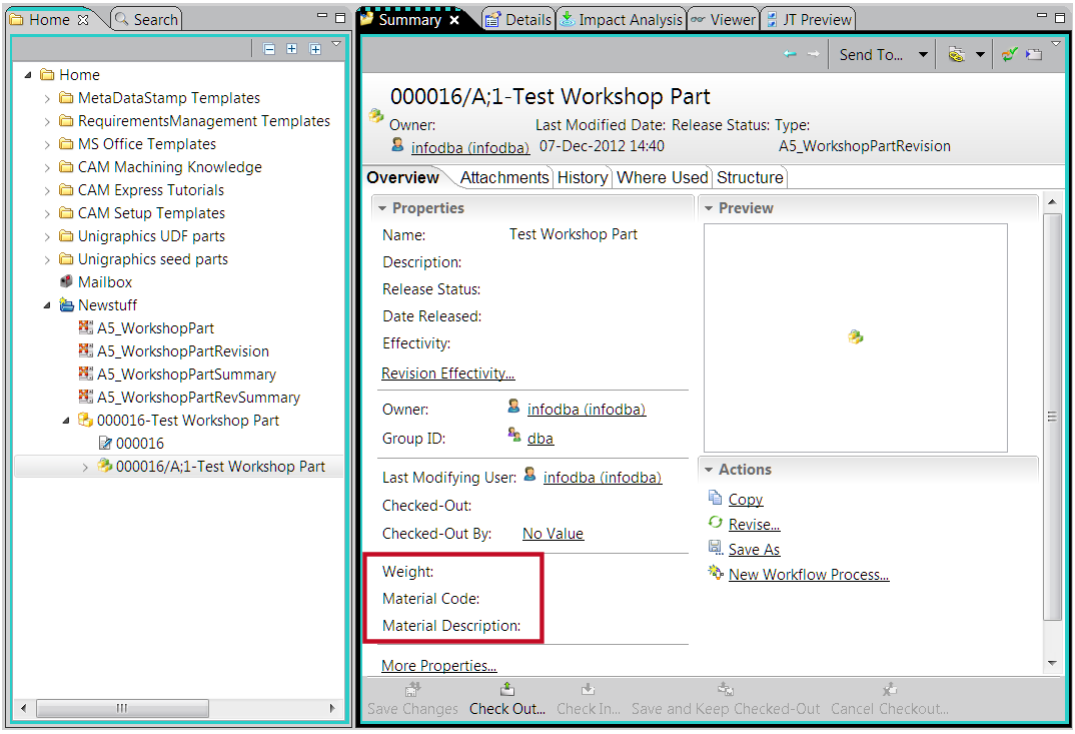


c. With the workshop part item still selected, click the **Viewer** tab.

© 2025 Siemens

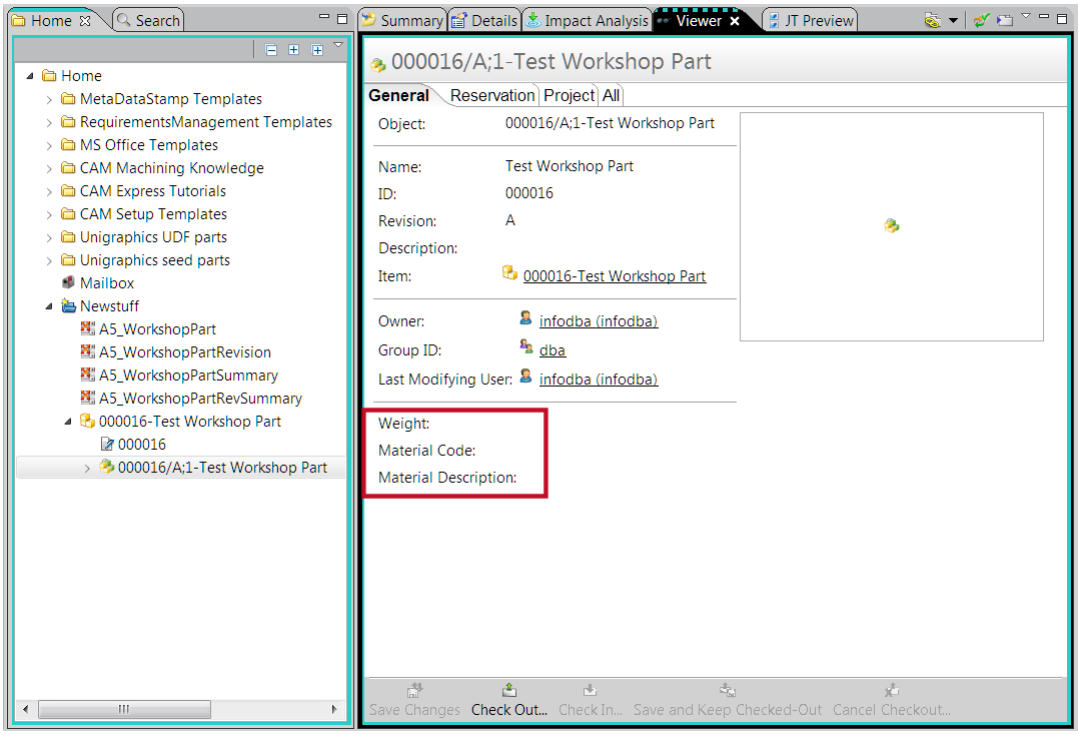


d. Select a workshop part item revision and click the **Summary** tab.



e. With the workshop part item revision still selected, click the **Viewer** tab.

© 2025 Siemens



Workshop 6: Create lists of values

Perform this workshop to **create lists of values (LOVs)** that you attach to the safety code and supplier properties on the workshop part. End users click a menu on these properties to select from a list of choices.

1. Create the safety codes LOV and attach it to the safety code property.

    a. Create the safety codes LOV.

        A. In the **Extensions** folder, right-click the **LOV** folder and choose **New LOV**.

        B. In the **New LOV** wizard, ensure **Classic** is selected and click **Next**.

        C. Perform the following steps in the **Classic LOV** dialog box:

            i. In the **Name** box, type **A5_SafetyCodes**.

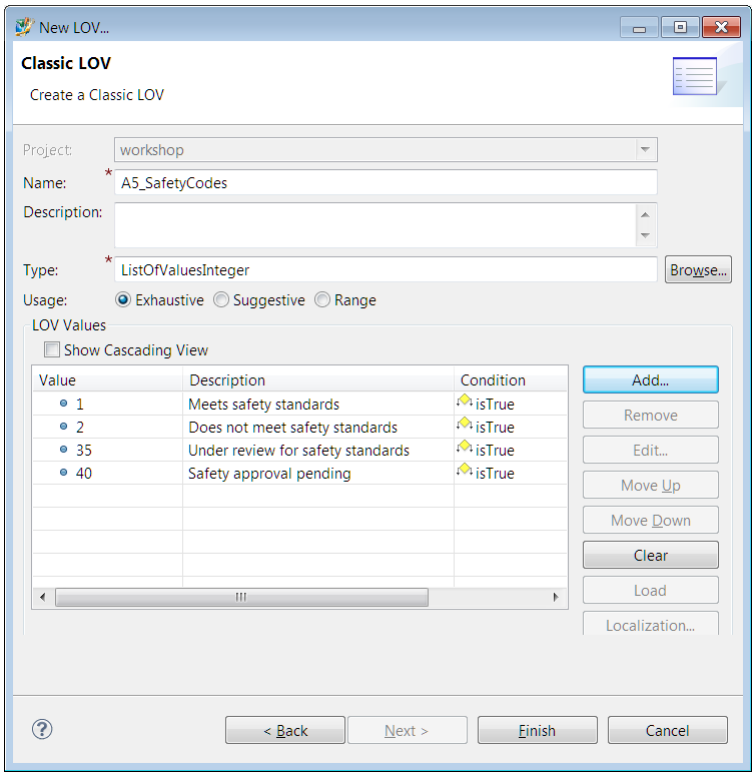
            ii. In the **Type** box, select **ListOfValuesInteger**.

            iii. Click the **Add** button to the right of the **LOV Values** table and add the following LOVs.

© 2025 Siemens

Value	Description
1	Meets safety standards
2	Does not meet safety standards
35	Under review for safety standards
40	Safety approval pending

iv. Click **Finish**.



b. Attach the safety codes LOV to the safety code property.

A. Open the **A5_WorkshopPart** business object and click the **Properties** tab.

B. Select the **a5_safety_code** property on the properties table.

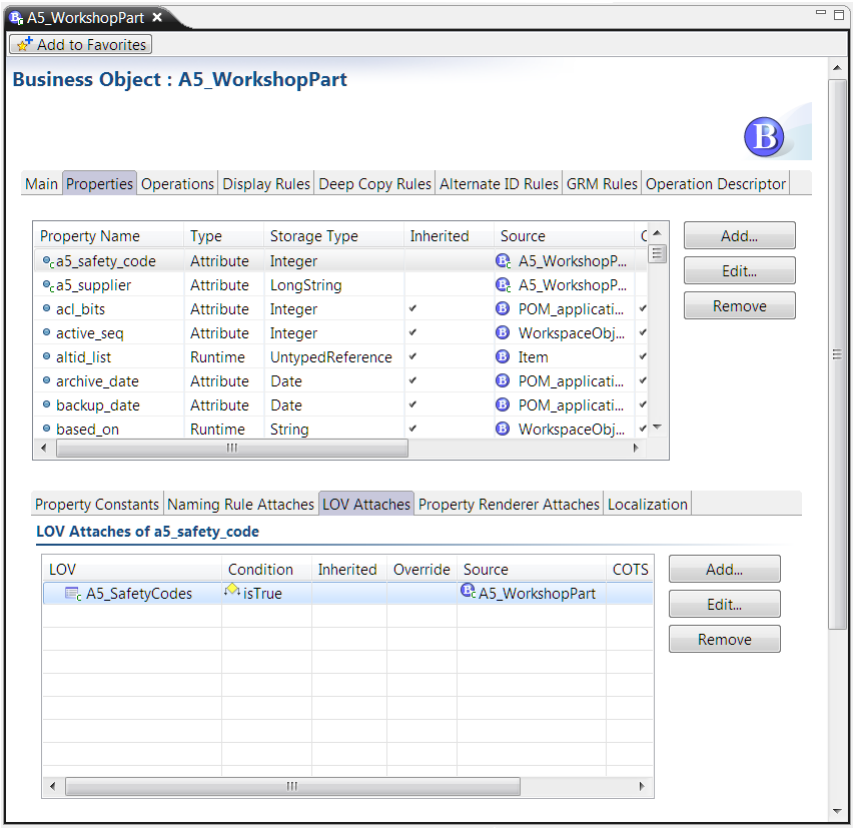
C. Click the **LOV Attaches** tab.

D. Click the **Add** button to the right of the **LOV Attaches** table.

E. In the **LOV** box, select the **A5_SafetyCodes** LOV.

F. Click **Finish**.

© 2025 Siemens



2. Create the supplier cascading LOV and attach it to the supplier property. A cascading LOV presents the user with sublists.

   a. Create the supplier LOV.

      A. In the **Extensions** folder, right-click the **LOV** folder and choose **New LOV**.

      B. In the New LOV wizard, ensure **Classic** is selected and click **Next**.

      C. Perform the following steps in the **Classic LOV** dialog box:

         i. In the **Name** box, type **A5_WestCoastSuppliers**.

         ii. In the **Type** box, select **ListOfValuesString**.

         iii. Click the **Add** button to the right of the **LOV Values** table and add the following LOVs.

Value	Value display name
Supplier1	**Supplier 1**
Supplier2	**Supplier 2**

© 2025 Siemens

iv. Click **Finish**.

D. Create an **A5_EastCoastSuppliers** string LOV with the following values.

Value	Value display name
Supplier3	**Supplier 3**
Supplier4	**Supplier 4**

E. Create an **A5_Suppliers** string LOV to hold the other LOVs using the following values.

Value
East
West

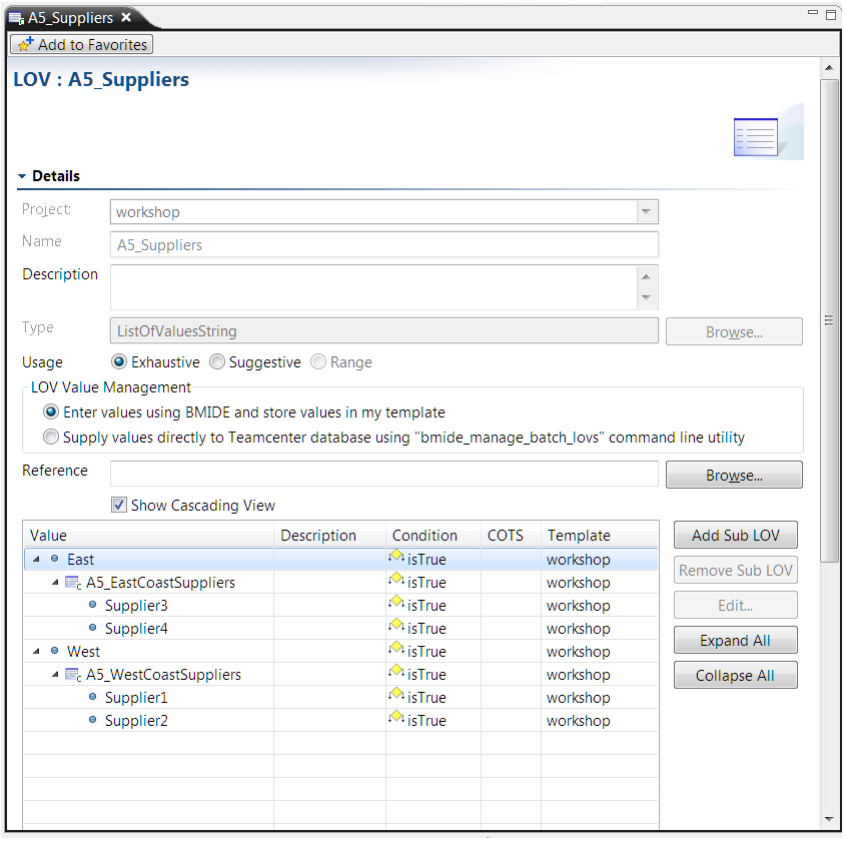
F. Add the child LOVs to the parent LOV.

i. Open the **A5_Suppliers** LOV and select the **Show Cascading View** check box.

ii. Select the **East** value, click the **Add Sub LOV** button, and add the **A5_EastCoastSuppliers** LOV.

iii. Select the **West** value, click the **Add Sub LOV** button, and add the **A5_WestCoastSuppliers** LOV.

© 2025 Siemens



b. Attach the suppliers LOV to the supplier property.

A. Open the **A5_WorkshopPart** business object and click the **Properties** tab.

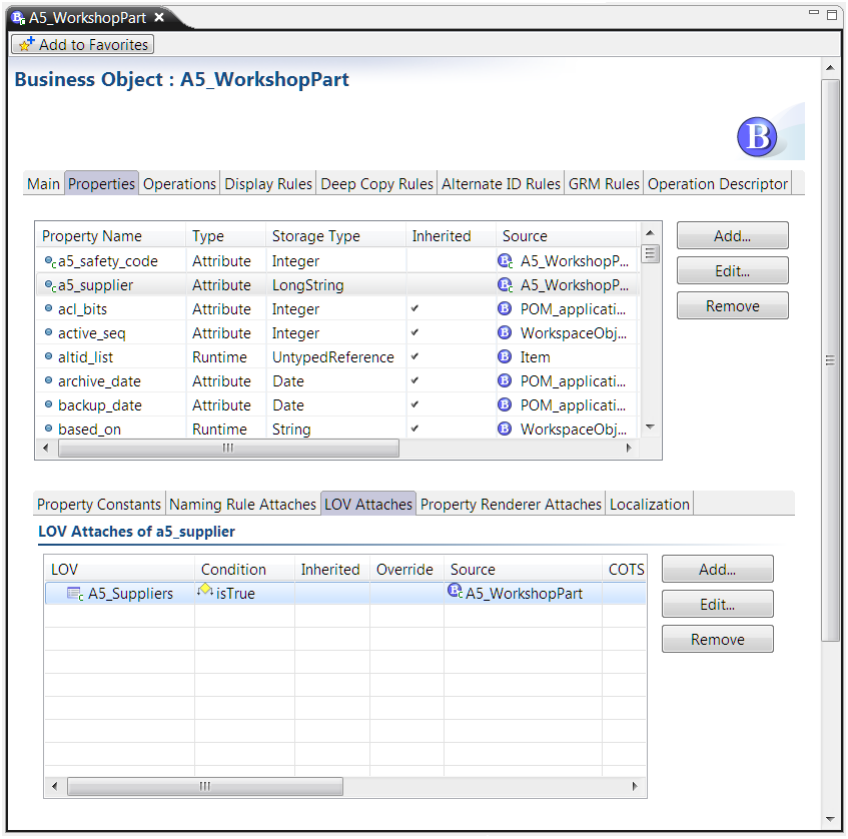
B. Select the **a5_supplier** property in the properties table.

C. On the **LOV Attaches** tab, click the **Add** button to the right of the **LOV Attaches** table.

D. In the **LOV** box, select the **A5_Suppliers** LOV.

E. Click **Finish**.

© 2025 Siemens



3. Deploy the changes to the rich client.

   a. On the menu bar, choose **BMIDE**>**Save Data Model** to save your changes.

   b. Ensure that the test server is running.

   c. On the menu bar, choose **BMIDE**>**Deploy Template**. Type the password, click the **Connect** button, and when a connection is established, select the **Generate Server Cache?** check box and click **Finish**.

   d. When deployment is done, check the status in the **Console** view.

   e. To ensure your changes appear in the rich client user interface, exit the rich client and restart it to clear cache.

      When you restart, you may want to launch the rich client using the *install-location***portal\portal.bat -clean** option to ensure the cache is cleared.

© 2025 Siemens

> Note:
>
> If your changes still do not appear in the user interface, delete the **Teamcenter** subdirectory in the user's home directory on the client. This directory is automatically created again when the user starts the rich client.
>
> On a Windows client, it is typically **Users***user-name***Teamcenter**. On a Linux client, it is typically **$HOME/Teamcenter/**.

4. Verify the LOVs.

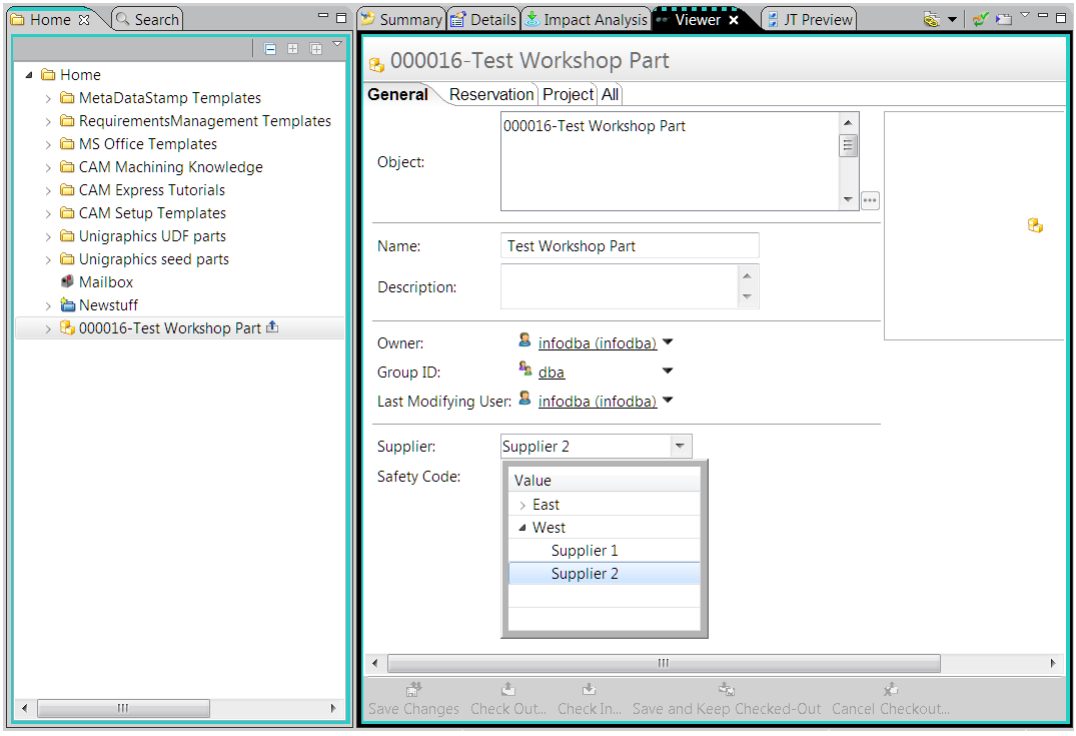
   a. Run the rich client.

   b. In My Teamcenter, choose **File→New→Item**, select **Workshop Part** from the list, and create an instance of the part.

   c. Select the workshop part item and click the **Viewer** tab.

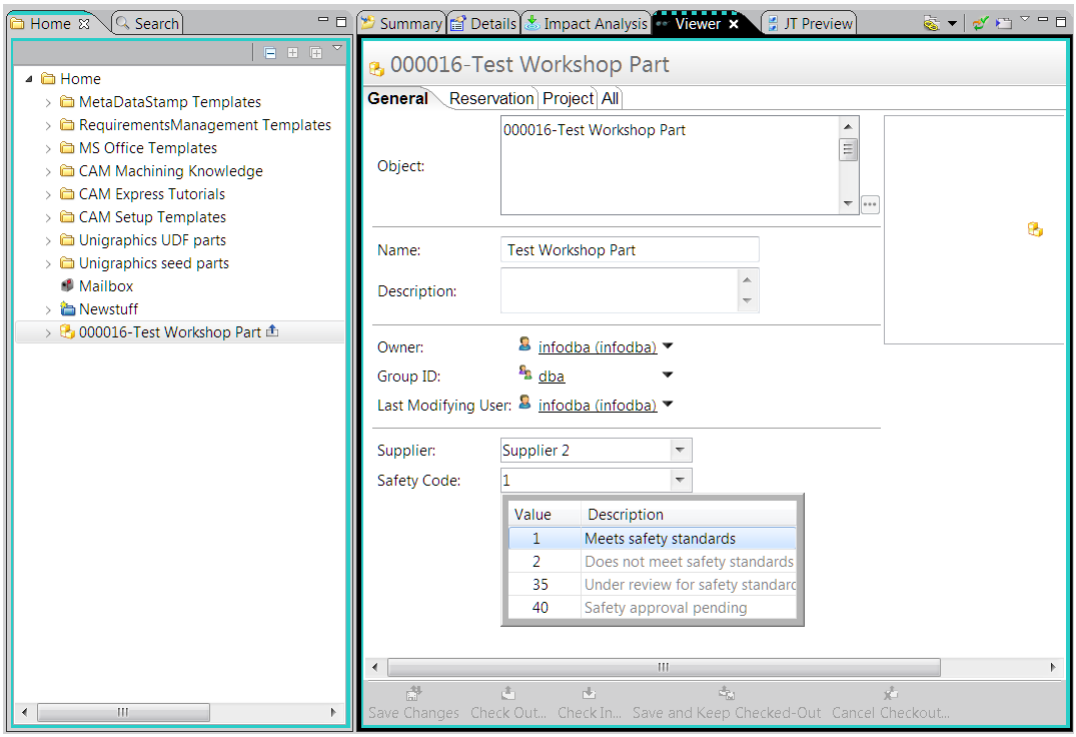
   d. Click the **Check-Out** button at the bottom of the view and check out the item.

   e. Click the arrow in the **Suppliers** box and select a supplier from the cascading list.



   f. Click the arrow in the **Safety Code** box and select a code from the list.

© 2025 Siemens



g. Click the **Save and Check-In** button to save your changes to the workshop part.

Workshop 7: Add a naming rule

Perform this workshop to create a rule that defines how IDs are automatically assigned to new workshop parts. You create a naming rule and attach it to the **item_id** property on the **A5_WorkshopPart** business object. In addition, you turn on number autogeneration so that Teamcenter generates sequences for the property whenever a user clicks the **Assign** button.

1. Create the naming rule.

   a. In the **Extensions** folder, select **Rules**, right-click **Naming Rules**, and choose **New Naming Rules**.

   b. In the **Name** box in **Naming Rule** dialog box, type **A5_WorkshopPartNamingRule**.

   c. Click the **Add** button in the **Naming Rule** dialog box.

   d. Perform the following steps in the **Pattern** dialog box:

      A. In the **Pattern** box, type **"Part-"AA"-"nnnn**.

      B. Select the **Generate Counters** check box.

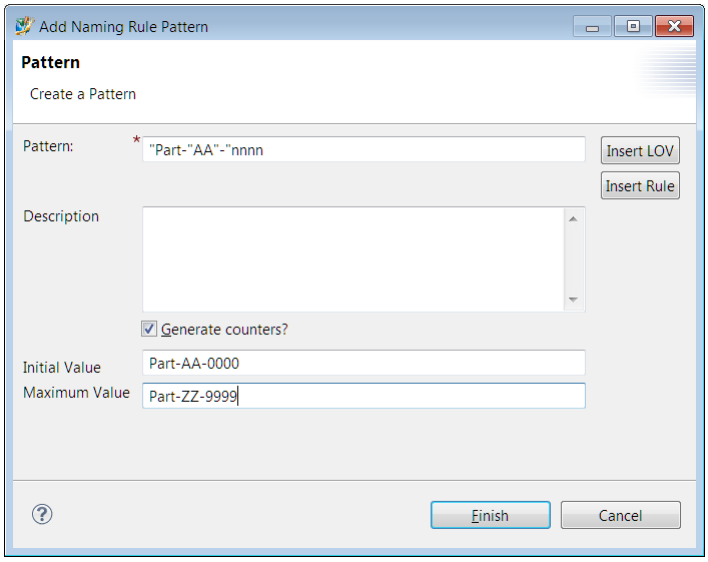
      C. In the **Initial Value** box, type **Part-AA-0000**.

© 2025 Siemens

D. In the **Maximum Value** box, type **Part-ZZ-9999**.

E. Click **Finish** in the **Pattern** dialog box.

e. Click **Finish** in the **Naming Rule** dialog box.



2. Attach the naming rule to the **item_id** property on the business object.

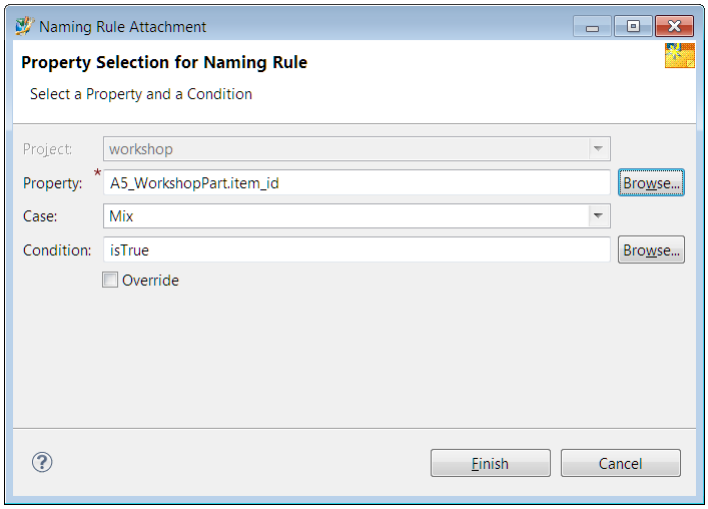
a. From the **Naming Rules** folder, open the **A5_WorkshopPartNamingRule** naming rule.

b. Click the **Attach** button to the right of the **Naming Rules Attachments** table.

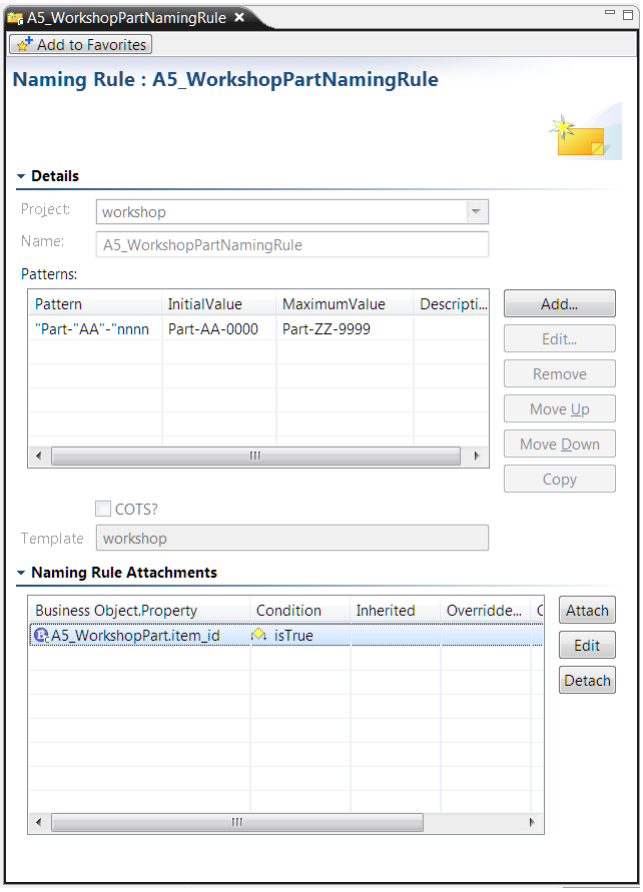
c. Click the **Browse** button to the right of the **Property** box and select the **A5_WorkshopPart** business object and the **item_id** property.

d. Click **Finish**.

© 2025 Siemens



After the naming rule is attached, the naming rule appears.



3. Deploy the changes to the rich client.

   a. Choose **BMIDE**>**Save Data Model** to save your changes.

© 2025 Siemens

b. Ensure that the test server is running.

c. On the menu bar, choose **BMIDE**>**Deploy Template**. Type the password, click the **Connect** button, and when a connection is established, select the **Generate Server Cache?** check box and click **Finish**.

d. When deployment is done, check the status in the **Console** view.

e. To ensure your changes appear in the rich client user interface, exit the rich client and restart it to clear cache.

When you restart, you may want to launch the rich client using the *install-location***portal\portal.bat -clean** option to ensure the cache is cleared.

> Note:
>
> If your changes still do not appear in the user interface, delete the **Teamcenter** subdirectory in the user's home directory on the client. This directory is automatically created again when the user starts the rich client.
>
> On a Windows client, it is typically **Users***user-name***\Teamcenter**. On a Linux client, it is typically **$HOME/Teamcenter/**.

4. Verify the naming rule.

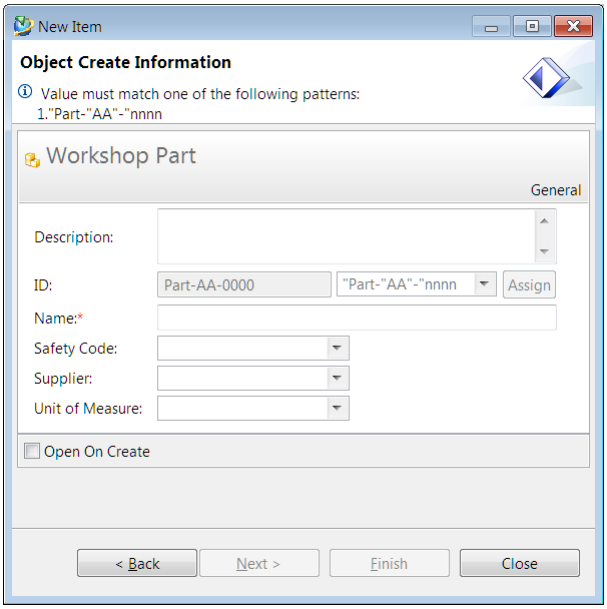
a. Run the rich client.

b. In My Teamcenter, choose **File→New→ Item**, select **Workshop Part** from the list, and click **Next**.

c. Click the **Assign** button.

You see the new naming rule in effect on the item revision creation dialog box.

© 2025 Siemens



Workshop 8: Create a form

Perform this workshop to **create a form** to hold object properties. Although there is a master form for each item business object, and a master revision form for each item revision business object, you can create additional forms to hold properties.

1. Create a new form.

   a. Click the **Find** button on the **BMIDE** view toolbar and search for the **Form** business object.

   b. Right-click the **Form** business object in the **BMIDE** view and choose **New Business Object**.

   c. Name the form **A5_WorkshopMfgForm** and give it the display name **Workshop Mfg Form**.

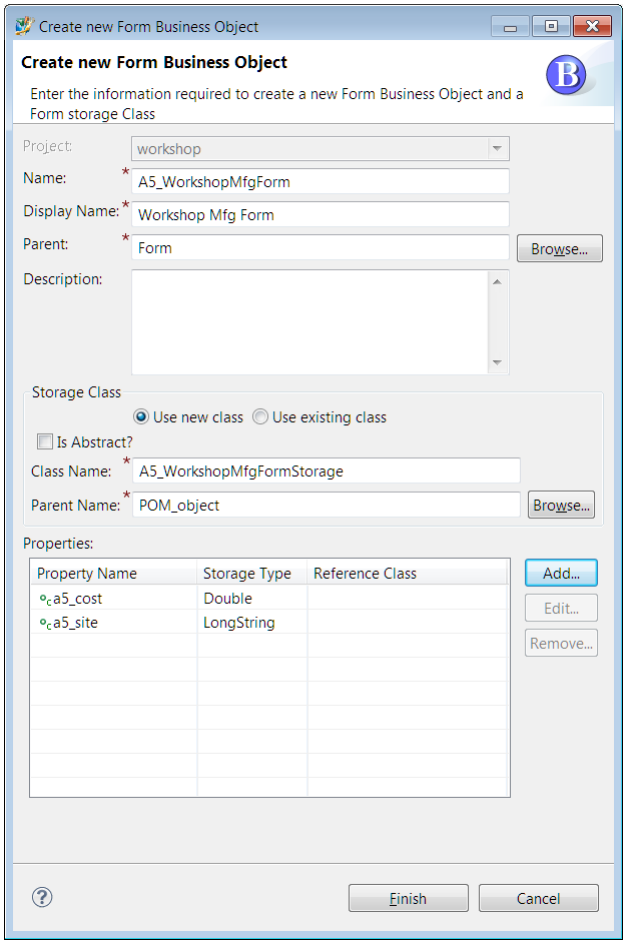
   d. Click the **Add** button to the right of the **Properties** table an add the following properties.

Name	Display name	Attribute type
a5_cost	Cost	Double
a5_site	Site	LongString

> Note:
>
> If you want to attach an LOV to these properties on a form business object, attach the LOV to the same property on the form's parent business object. The parent serves as the storage class for the properties on the children. Unless you do this, the LOV is not attached to the property and does not display in the end user interface.

© 2025 Siemens

e. Click **Finish**.



2. Enable the new properties on the form for use in the client user interface using the **Enabled** properties constant.

   a. In the **BMIDE** view, open the **A5_WorkshopMfgForm** business object and click the **Properties** tab.

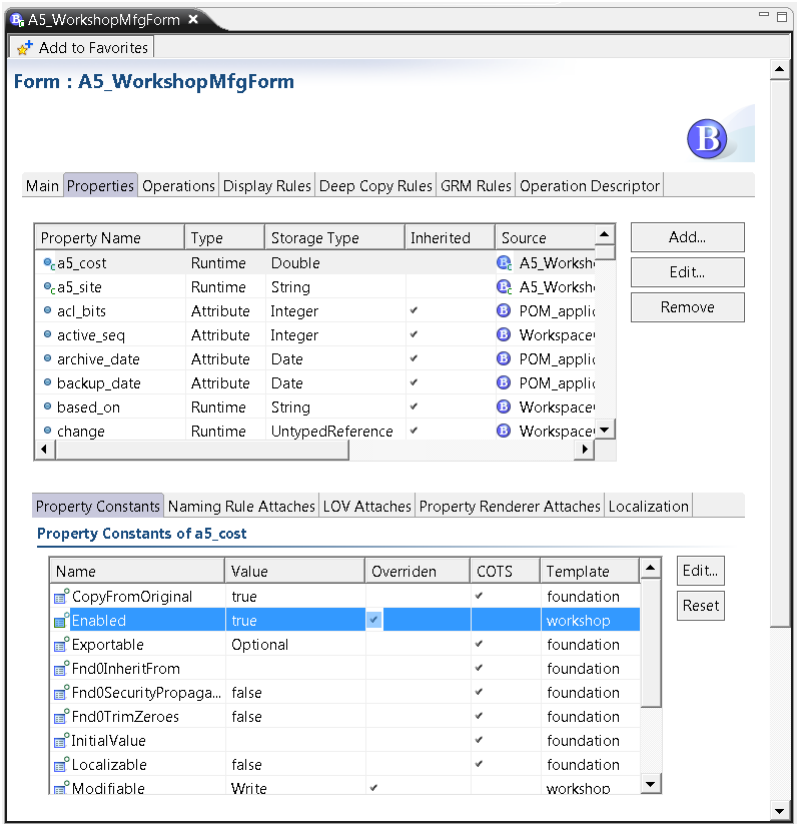
   b. Select the new **a5_cost** property in the properties table.

   c. On the **Property Constants** tab, select the **Enabled** constant.

   d. Ensure that the **Enabled** constant is set to **true**. If it is not, click the **Edit** button to the right of the **Property Constants** table and select the **Value** check box to set it to **true**.

   e. Repeat these steps for the new **a5_site** property.

© 2025 Siemens



3. Deploy the changes to the rich client.

   a. On the menu bar, choose **BMIDE**>**Save Data Model** to save your changes.

   b. Ensure that the test server is running.

   c. On the menu bar, choose **BMIDE**>**Deploy Template**. Type the password, click the **Connect** button, and when a connection is established, select the **Generate Server Cache?** check box and click **Finish**.

   d. When deployment is done, check the status in the **Console** view.

   e. To ensure your changes appear in the rich client user interface, exit the rich client and restart it to clear cache.

      When you restart, you may want to launch the rich client using the *install-location***portal\portal.bat -clean** option to ensure the cache is cleared.

© 2025 Siemens

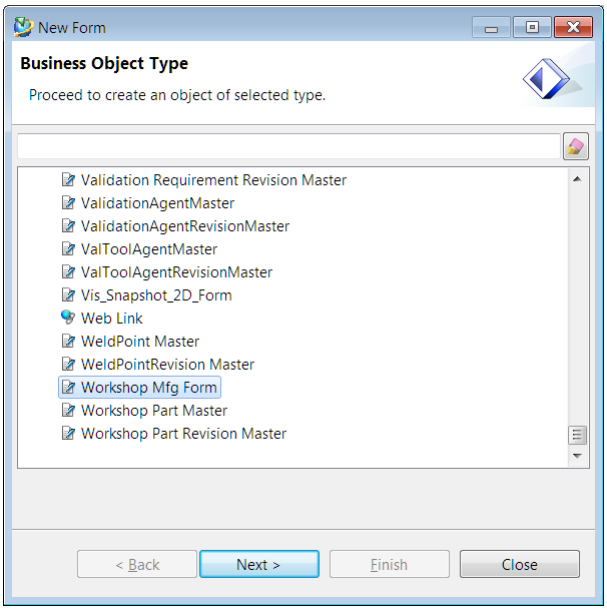
> Note:
>
> If your changes still do not appear in the user interface, delete the **Teamcenter** subdirectory in the user's home directory on the client. This directory is automatically created again when the user starts the rich client.
>
> On a Windows client, it is typically **Users***user-name***Teamcenter**. On a Linux client, it is typically **$HOME/Teamcenter/**.

4. Create an instance of the new form in the user interface.

   a. Run the rich client.

   b. In My Teamcenter, select a workshop part revision that you already created and choose **File→New→Form**.

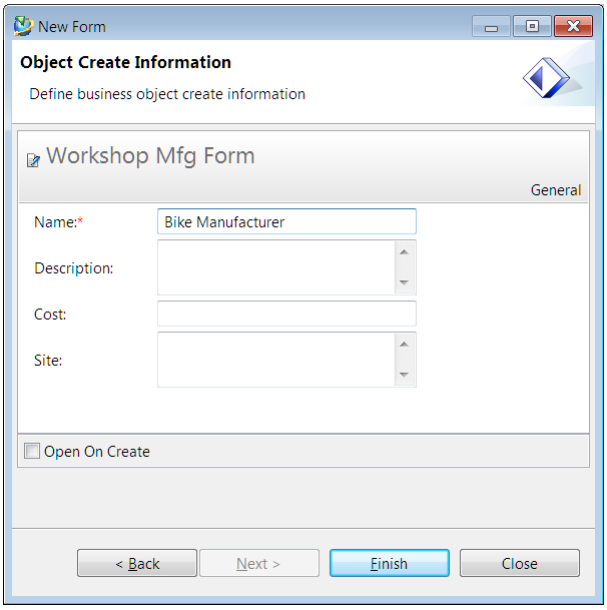
   c. In the New Form wizard, select **Workshop Mfg Form** from the list and click **Next**.



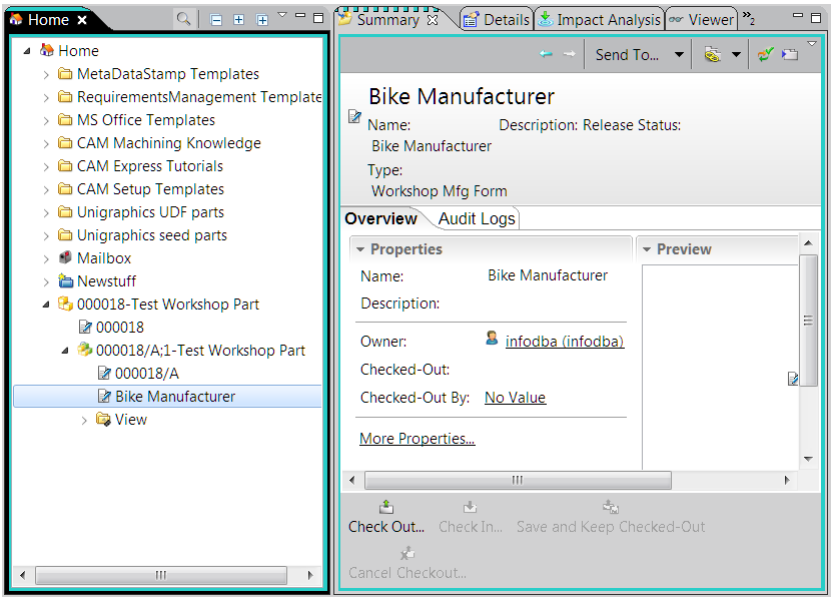
   d. Give the form a name.

      Note how the new properties are displayed on the form. Click **Finish**.

© 2025 Siemens



The new form is added under the workshop part revision.



Workshop 9: Add a relationship rule

Perform this workshop to create a **Generic Relationship Management (GRM) rule** to limit the number of **A5_WorkshopMfgForm** forms that can be related to a workshop part revision.

GRM rules apply constraints on the relationship between two business objects. When you create a GRM rule, you select the primary and secondary business objects for the relationship, the relationship they have to one another, and the constraints to be applied.

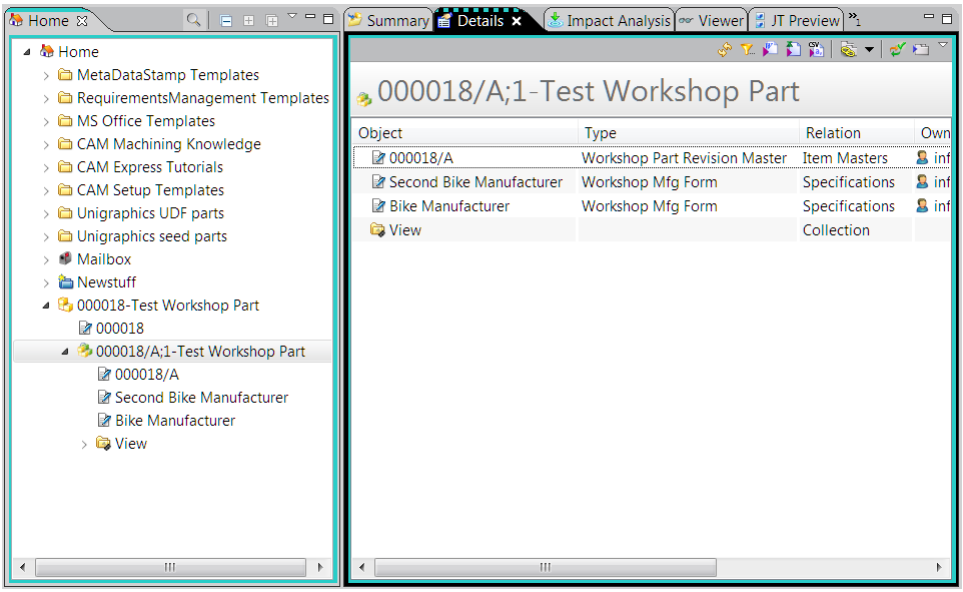
© 2025 Siemens

1. Temporarily add a new form to the workshop part revision.

   a. Currently, there is no limit on the number of **Workshop Mfg Form** (**A5_WorkshopMfgForm**) forms that can be added to a workshop part revision.

      To illustrate this, in the rich client, select the workshop part revision that already has a **Workshop Mfg Form** form on it, choose **File→New→Form**, and create another **Workshop Mfg Form** form.

      The new form is added under the workshop part revision. Notice how the forms are related to the revision with the **Specifications** relation.



   b. Delete the new form.

      This was just to show that there are no restrictions in place yet for the number of this type of form you can relate to the workshop part revision. In the following steps, you create a rule to allow only one of this form type to be related to the revision.

2. Create the GRM rule.

   a. Click the **Find** button on the **BMIDE** view toolbar and search for the **A5_WorkshopPartRevision** business object.

   b. Open the **A5_WorkshopPartRevision** business object and click the **GRM Rules** tab.

   c. Click the **Add** button to the right of the table.

   d. Perform the following steps in the **GRM Rule** dialog box:

© 2025 Siemens

A. Click the **Browse** button to the right of the **Primary Object** box and select the **A5_WorkshopPartRevision** business object.

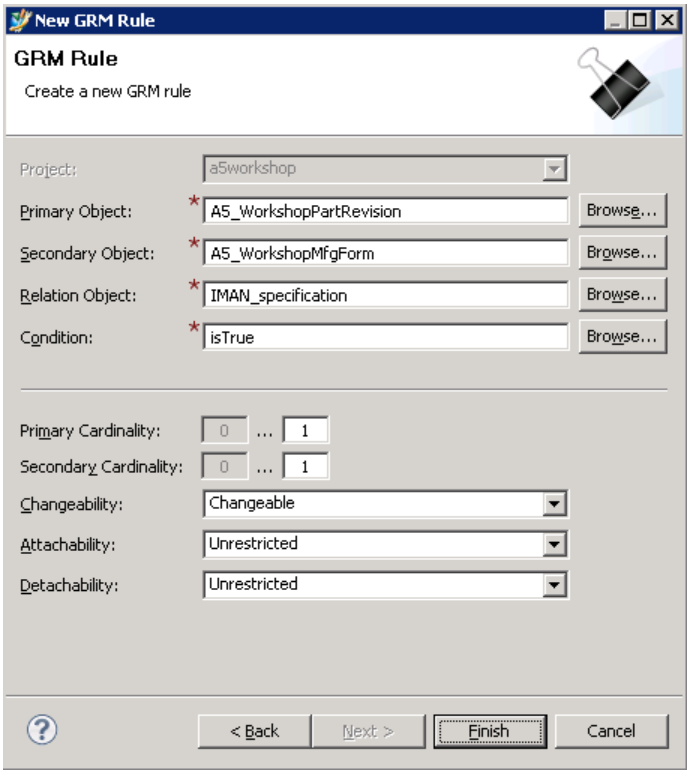
B. Click the **Browse** button to the right of the **Secondary Object** box and select the **A5_WorkshopMfgForm** business object.

C. Click the **Browse** button to the right of the **Relation Object** box and select the **IMAN_Specification** business object.

D. Click the **Browse** button to the right of the **Condition** box and select the **isTrue** condition.

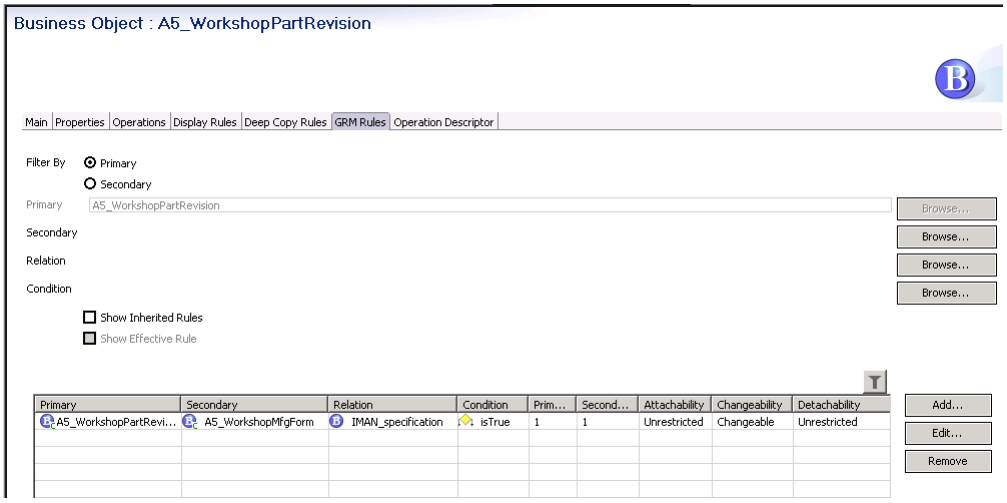
E. In the **Primary Cardinality** and **Secondary Cardinality** boxes, change the ***** symbol to **1**. (The ***** means unlimited. **-1** also means unlimited.)

F. Click **Finish**.



The new rule appears in the **GRM Rules** tab.

© 2025 Siemens



3. Deploy the changes to the rich client.

   a. On the menu bar, choose **BMIDE**>**Save Data Model** to save your changes.

   b. Ensure that the test server is running.

   c. On the menu bar, choose **BMIDE**>**Deploy Template**. Type the password, click the **Connect** button, and when a connection is established, select the **Generate Server Cache?** check box and click **Finish**.

   d. When deployment is done, check the status in the **Console** view.

   e. To ensure your changes appear in the rich client user interface, exit the rich client and restart it to clear cache.

      When you restart, you may want to launch the rich client using the *install-location***portal\portal.bat -clean** option to ensure the cache is cleared.

      > Note:
      >
      > If your changes still do not appear in the user interface, delete the **Teamcenter** subdirectory in the user's home directory on the client. This directory is automatically created again when the user starts the rich client.
      >
      > On a Windows client, it is typically **Users***user-name***Teamcenter**. On a Linux client, it is typically **$HOME/Teamcenter/**.

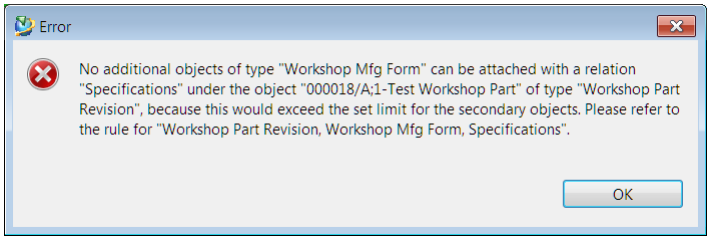
4. Verify the new GRM rule.

   a. Run the rich client.

© 2025 Siemens

b. In My Teamcenter, select the workshop part revision that already has a **Workshop Mfg Form** form on it, and choose **File→New→Form**.

The following error message is displayed. This shows that the GRM rule restriction is working.



c. Click **OK**.

5. Edit the rule to change the primary and secondary cardinality from **1** to ***** and redeploy the project.

You must reset the cardinality so you can add objects to the revision in the next workshop.

Workshop 10: Add deep copy rules

Perform this workshop to create new deep copy rules to propagate related objects to your workshop part revision. When you revise a workshop part revision, the rules state that **Workshop Mfg Form** (**A5_WorkshopMfgForm**) forms are copied and related to the new revision, **MSWord** datasets are referenced, and **MSExcel** datasets are not copied at all.

1. Revise a workshop part revision.

To see the existing deep copy rules on the workshop part revision, select an existing workshop part revision and choose **File→Revise**. Note how objects related to the form are copied into the revision B object.

2. Create deep copy rules on the workshop part revision.

a. Click the **Find** button on the **BMIDE** view toolbar and search for the **A5_WorkshopPartRevision** business object.

b. Open the **A5_WorkshopPartRevision** business object and click the **Deep Copy Rules** tab.

c. Click the **Add** button to the right of the table.

d. Create the following rule to govern how **A5_WorkshopMfgForm** forms are copied to the new revision:

A. In the **Operation Type** box, select the **Revise** operation.

© 2025 Siemens

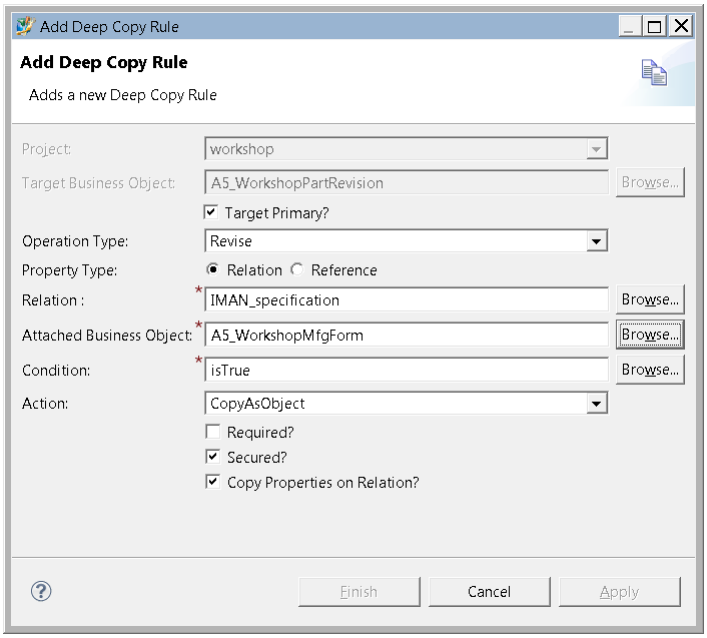
B. For **Property Type**, select **Relation**.

C. In the **Relation** box, select the **IMAN_specification** business object.

D. In the **Attached Business Object** box, select the **A5_WorkshopMfgForm** business object.

E. In the **Action** box, select the **CopyAsObject** action.

F. Click **Apply**.



e. Create the following rule to govern how **MSWord** datasets are copied to the new revision:

A. In the **Operation Type** box, select the **Revise** operation.

B. For **Property Type** select **Relation**.

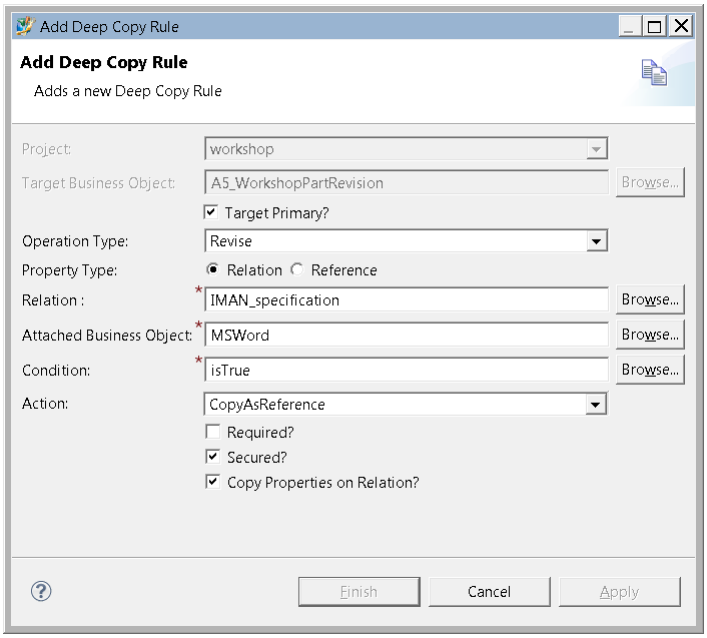
C. In the **Relation** box, select the **IMAN_specification** business object.

D. In the **Attached Business Object** box, select the **MSWord** business object.

E. In the **Action** box, select the **CopyAsReference** action.

F. Click **Apply**.

© 2025 Siemens



f. Create the following rule to govern how **MSExcel** datasets are *not* copied to the new revision:

A. In the **Operation Type** box, select the **Revise** operation.

B. For **Property Type** select **Relation**.

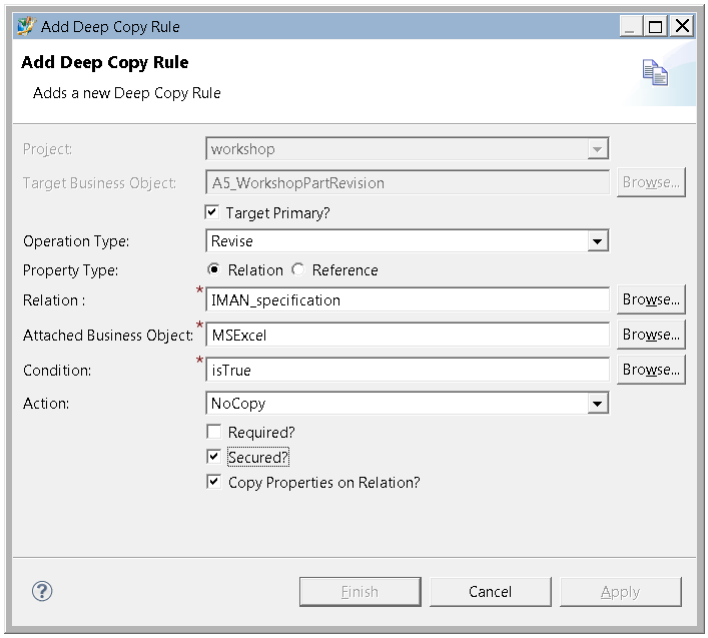
C. In the **Relation** box, select the **IMAN_specification** business object.

D. In the **Attached Business Object** box, select the **MSExcel** business object.

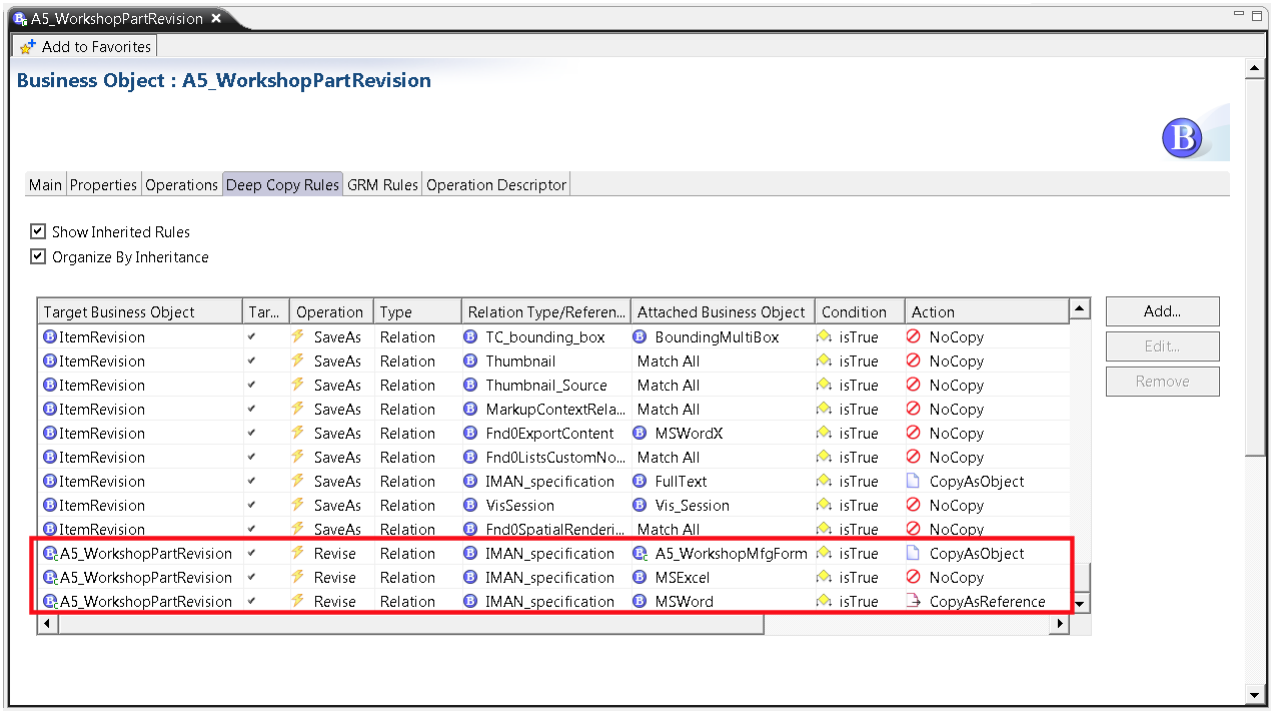
E. In the **Action** box, select the **NoCopy** action.

F. Click **Finish**.

© 2025 Siemens



The rules appear on the **Deep Copy Rules** tab.



3. Deploy the changes to the rich client.

   a. On the menu bar, choose **BMIDE→Save Data Model** to save your changes.

   b. Ensure that the test server is running.

© 2025 Siemens

c. On the menu bar, choose **BMIDE→Deploy Template**. Type the password, click the **Connect** button, and when a connection is established, select the **Generate Server Cache?** check box and click **Finish**.

d. When deployment is done, check the status in the **Console** view.

e. To ensure your changes appear in the rich client user interface, exit the rich client and restart it to clear cache.

When you restart, you may want to launch the rich client using the *install-location***portal\portal.bat -clean** option to ensure the cache is cleared.

> Note:
>
> If your changes still do not appear in the user interface, delete the **Teamcenter** subdirectory in the user's home directory on the client. This directory is automatically created again when the user starts the rich client.
>
> On a Windows client, it is typically **Users***user-name***Teamcenter**. On a Linux client, it is typically **$HOME/Teamcenter/**.

4. Verify the new deep copy rules.

a. Run the rich client.

b. In My Teamcenter, create some data for this test.

A. Create new workshop part. (Choose **File→New→Item** and select **Workshop Part**.)

B. Select the workshop part revision.

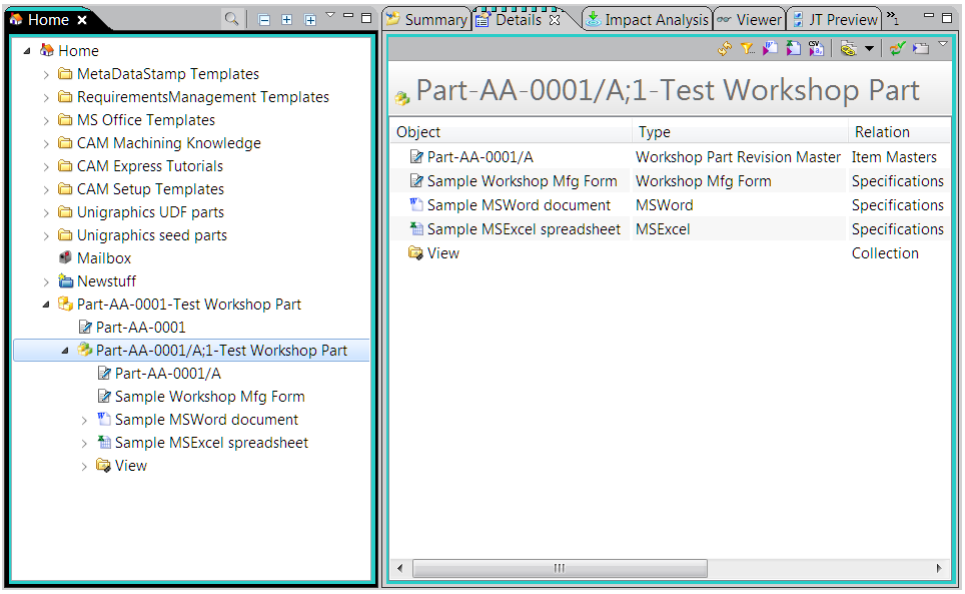
C. Create an **A5_WorkshopMfgForm** form. (Choose **File→New→Form** and select the **A5_WorkshopMfgForm** form.)

D. Select the workshop part revision and create a Microsoft Word dataset. (Choose **File→New→Dataset** and select the **MSWord** dataset.)

E. Select the workshop part revision and create a Microsoft Excel dataset. (Choose **File**>**New**> **Dataset** and select the **MSExcel** dataset.)

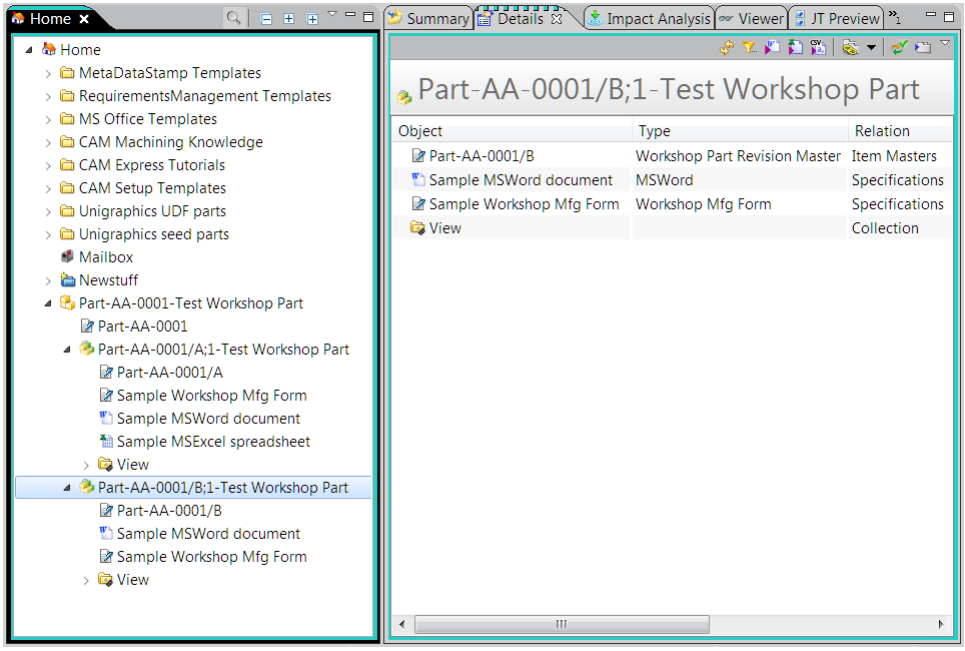
Notice how all the objects are related to the workshop part revision with the **Specifications** relation.

© 2025 Siemens



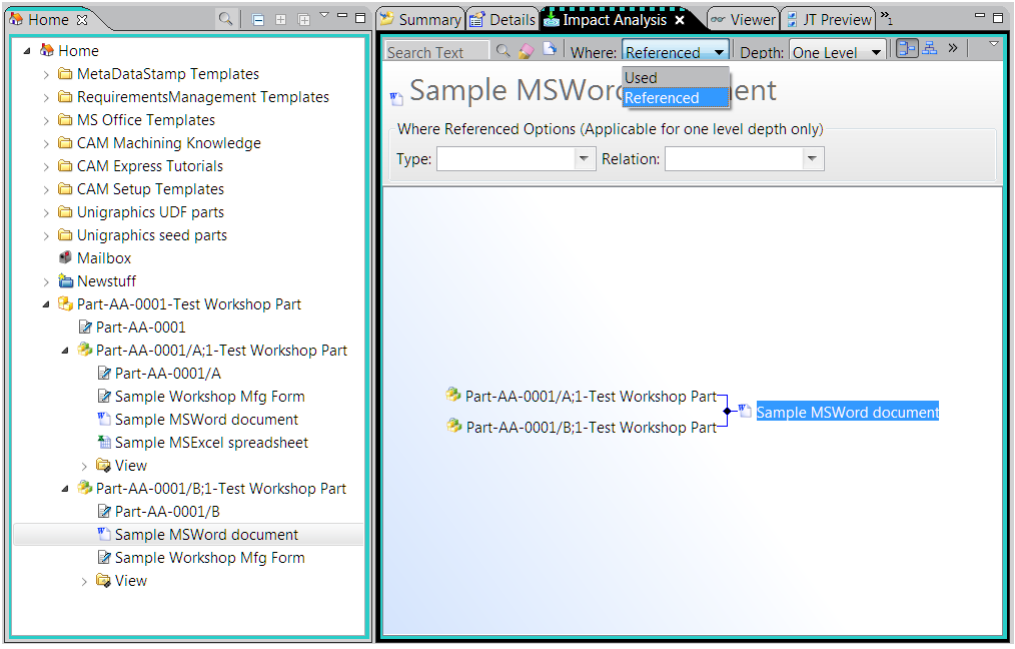
c. To see your new deep copy rules at work, select the workshop part revision and choose **File**>**Revise**.

Notice how the form and the Word dataset were copied forward to the revision B, but that the Excel dataset was not.



d. As you recall, you set the Word dataset to be copied as a reference document. View the Word dataset in the **Impact Analysis** view to verify it is referenced by both revisions. Click the **Impact Analysis** view and in the **Where** box, select **Referenced**.

© 2025 Siemens



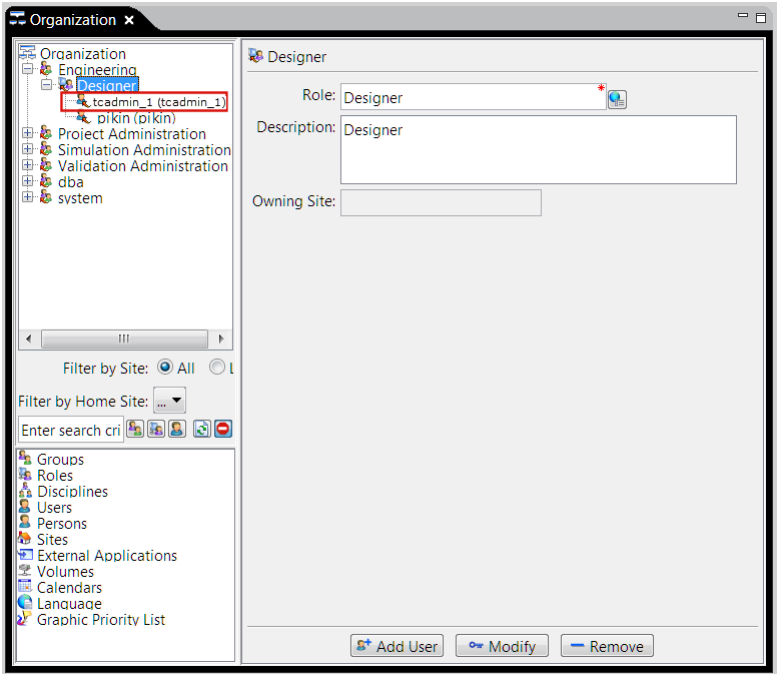
Workshop 11: Add a business object display rule

Perform this workshop to **create a business object display rule** to hide your workshop part from members of a specific group or role. Business object display rules determine the members of the organization who cannot view a business object type in menus in the Teamcenter user interface. In this workshop, you hide the **Workshop Part** from members of the **Designer** role.

1. Log on to the rich client as a member of a group with **DBA Privileges** enabled. For this example, assume that **tc_admin_1** is such a user.

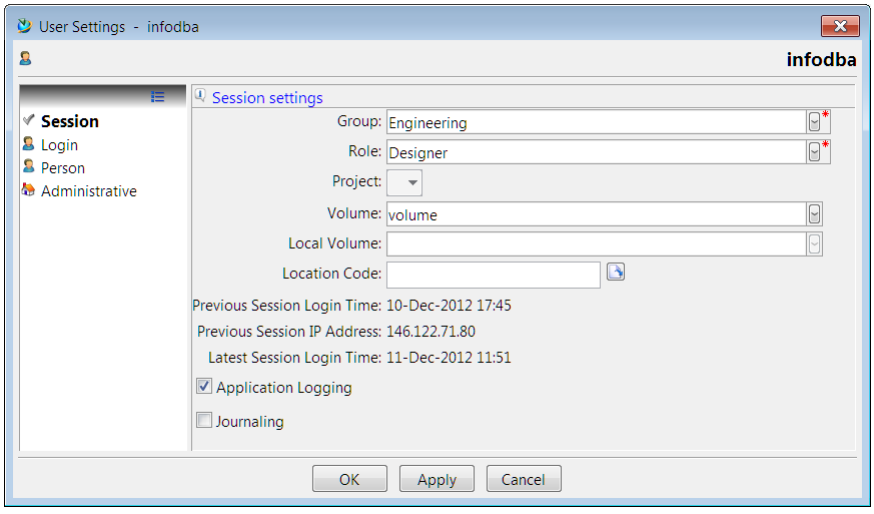
2. In the Organization application, click the **Add User** button to add the **tc_admin_1** user to the **Designer** role under the **Engineering** group.

© 2025 Siemens



3. To verify the **Designer** role behavior, update your session user by choosing **Edit** on the menu bar and selecting **User Setting**.

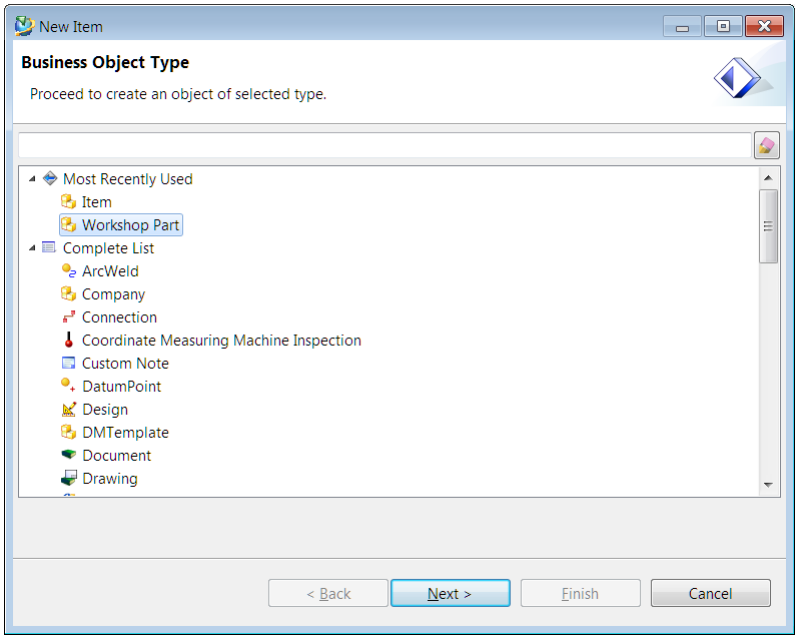
4. In the **User Settings** dialog box, change the group to **Engineering** and the role to **Designer**. Click **OK**.



5. To validate that the **Designer** role members can view the **Workshop Part** item in the list of available items, in My Teamcenter, choose **File→New→Item**.

   The **Workshop Part** item is visible in the list.

© 2025 Siemens



6. Return the user settings to their previous values. Choose **Edit** on the menu bar, click **User Setting**, and in the **User Settings** dialog box, change the group back to **dba** and the role to **DBA**. Click **OK**.

7. Log off the rich client.

8. Open the Business Modeler IDE. Now you can create a display rule to hide the **Workshop Part** item from members of the **Designer** role.

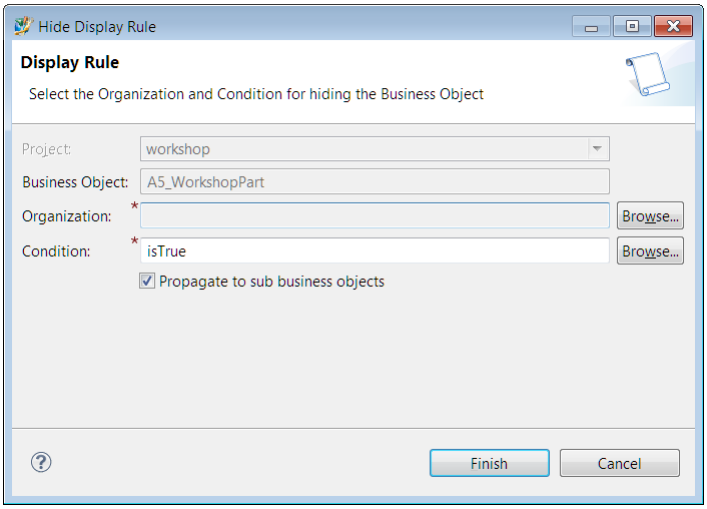
9. Open the **A5_WorkshopPart** business object.

10. Click the **Display Rules** tab.

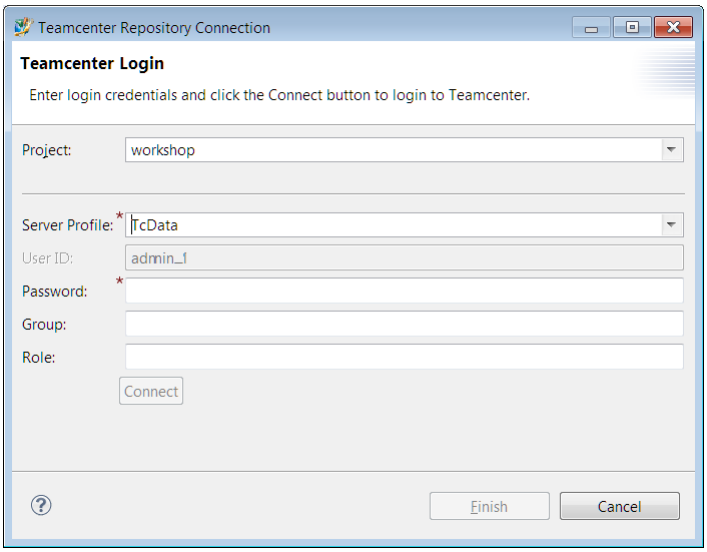
11. Click the **Add** button to the right of the **Hide Business Object Rules** table.

12. Click the **Browse** button to the right of the **Organization** box.

© 2025 Siemens

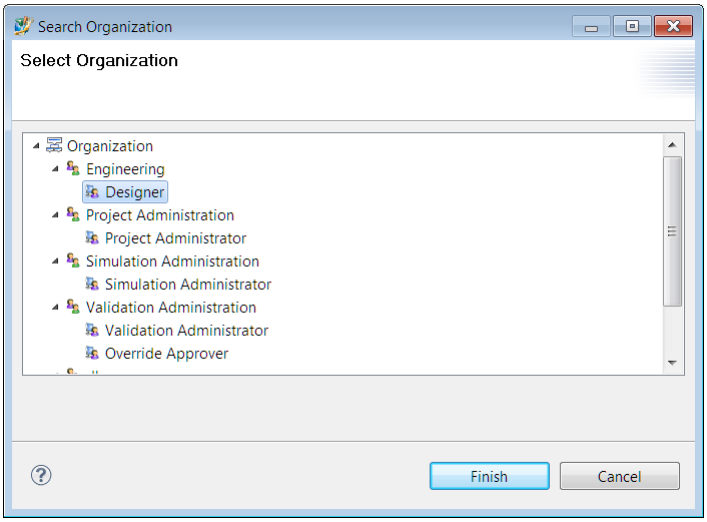


13. Log on to Teamcenter with an admin account.



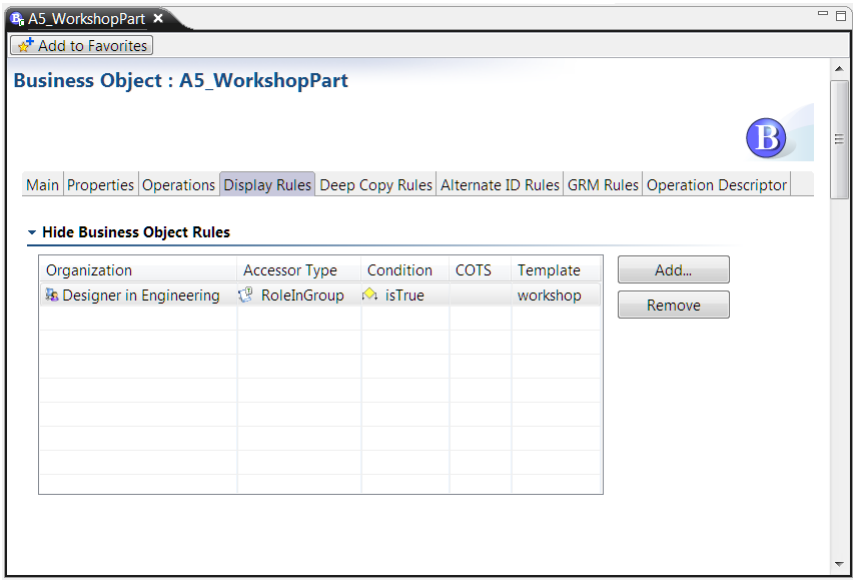
14. In the **Select Organization** dialog box, select the **Designer** role under the **Engineering** group and click **Finish**.

© 2025 Siemens



15. Click **Finish** in the **Display Rule** dialog box.

    The new display rule appears in the **Hide Business Object Rules** table.



16. Deploy the changes to the rich client.

    a. On the menu bar, choose **BMIDE→Save Data Model** to save your changes.

    b. Ensure that the test server is running.

    c. On the menu bar, choose **BMIDE→Deploy Template**. Type the password, click the **Connect** button, and when a connection is established, select the **Generate Server Cache?** check box and click **Finish**.

© 2025 Siemens

d. When deployment is done, check the status in the **Console** view.

e. To ensure your changes appear in the rich client user interface, exit the rich client and restart it to clear cache.

When you restart, you may want to launch the rich client using the *install-location***portal\portal.bat -clean** option to ensure the cache is cleared.

17. Log on to the rich client as **tc_admin_1** with **DBA** role.

18. In My Teamcenter, choose **File→New→Item**.

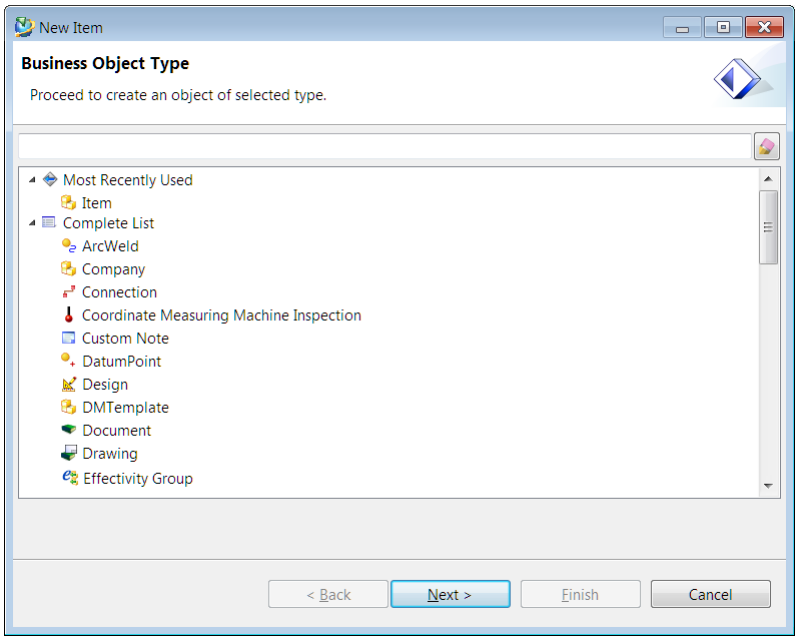
The **Workshop Part** item is visible in the list. This is as expected, because members of the **DBA** role are allowed to see the **Workshop Part** item.

19. To verify that the **Workshop Part** is hidden from those assigned the **Designer** role, update your session user by choosing **Edit** on the menu bar and selecting **User Setting**.

In the **User Settings** dialog box, change the group to **Engineering** and the role to **Designer**. Click **OK**.

20. In My Teamcenter, choose **File→New→Item** and scroll to the bottom of the list of items.

The **Workshop Part** item is not on the list because it is hidden from members of the **Design** role per the display rule.



© 2025 Siemens

Workshop 12: Add a predefined extension rule

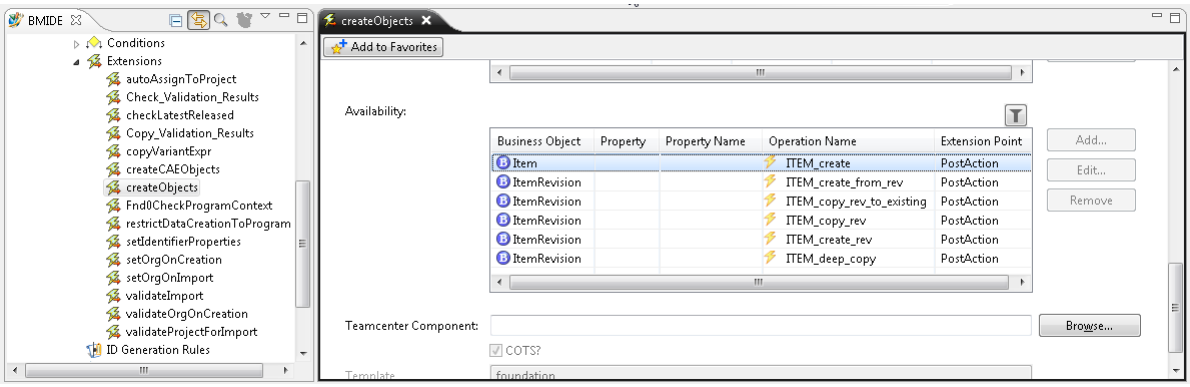
Perform this workshop to **add a predefined extension rule** (**createObjects**) to the **A5_WorkshopPart** business object. Then when you create a new instance of the **A5_WorkshopPart** business object, a new **MSWord** dataset is automatically created and related to the **A5_WorkshopPart** instance with the reference relation.

1. In the **Extensions** folder, choose **Rules→Extensions** and open the **createObjects** extension. This is the extension you are going to add as a postaction to the **A5_WorkshopPart** business object.

   Whenever you plan to use an extension, you must look at its characteristics in the **Availability** table to ensure that the extension is available for use on the business object you want to use it with. Note that the **createObjects** extension is available for use on **Item** and **ItemRevision** business objects (and their children) as a postaction. This confirms that this extension is available for use on the **A5_WorkshopPart** business object because the **A5_WorkshopPart** business object is a child of the **Item** business object.

   Also note that there are two parameters on this extension, the **objectType** parameter that asks for the type of object to be created, and the **relationType** parameter that asks for the relationship that the new object is to have.

   Finally, note the **ITEM_create** operation; this is the operation you use to call the **createObjects** extension.



2. Find the **A5_WorkshopPart** business object and open it.

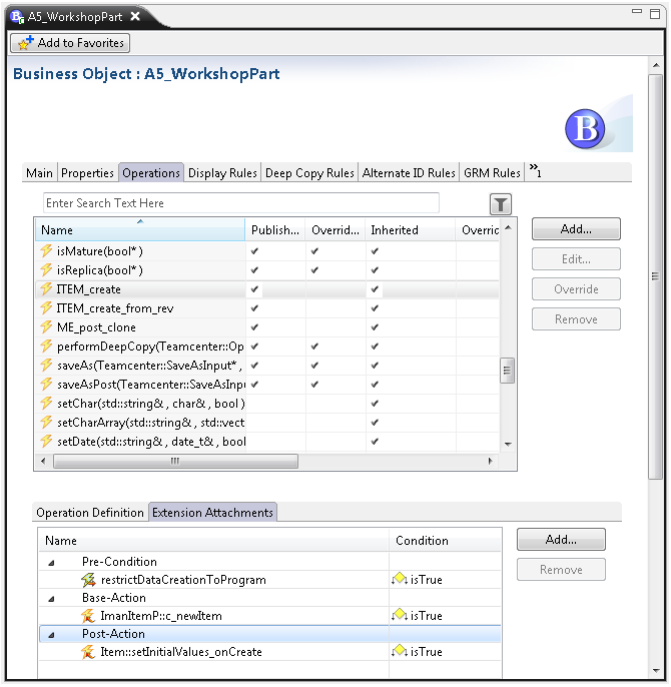
3. Click the **Operations** tab.

4. Select the **ITEM_create** operation.

5. Click the **Extension Attachments** tab and select **Post-Action**.

   You are going to add the **createObjects** extension as a postaction on the **ITEM_create** operation. This means that when a new **A5_WorkshopPart** business object instance is created and saved, the **createObjects** extension is called to create a dataset and attach it.

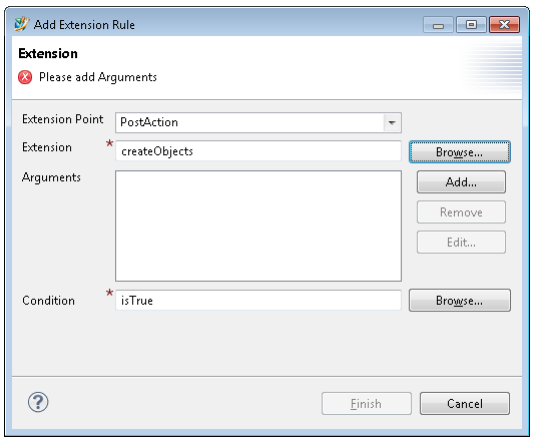
© 2025 Siemens



6. With **Post-Action** selected, click the **Add** button to the right of the extension attachments table.

7. Perform the following steps in the **Add Extension Rule** dialog box:

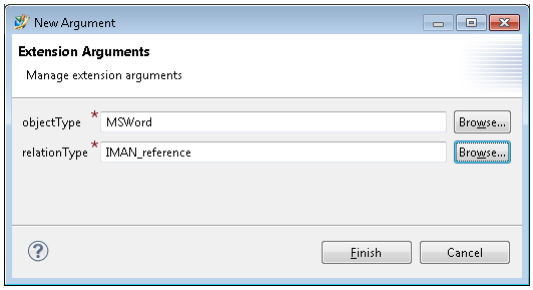
   a. Click the **Browse** button to the right of the **Extension** box and select the **createObjects** extension.



> Tip:
>
> The **createObjects** extension appears on the list of available extensions only if it is made available to the business object type. Therefore, before attempting to add an extension to an operation on a business object, always check the **Availability** table on the extension to ensure that the business object is included.

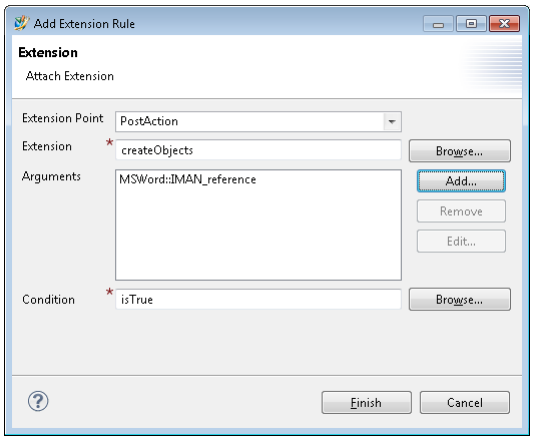
© 2025 Siemens

b. Click the **Add** button to the right of the **Arguments** box. In the **objectType** box, select **MSWord**, and in the **relationType** box, select **IMAN_reference**.



8. Click **Finish**.

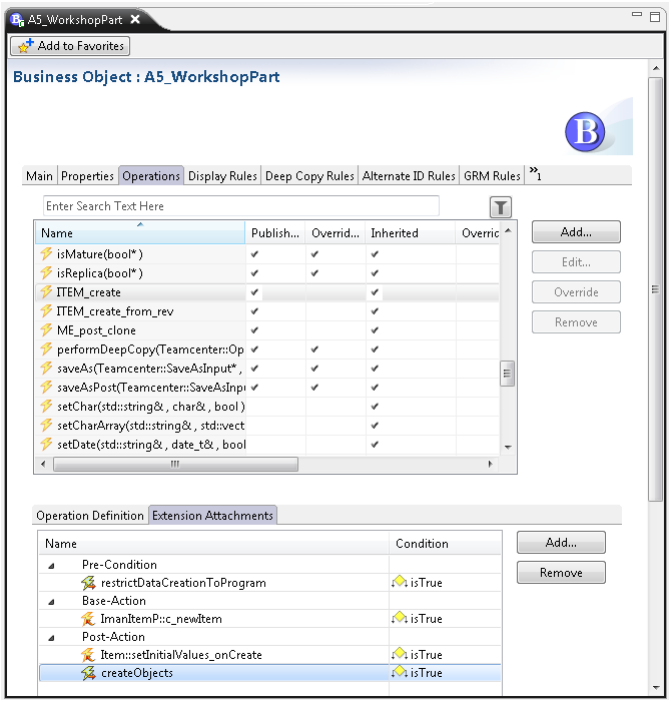
The extension appears.



9. Click **Finish**.

The **createObjects** extension appears as a postaction on the **ITEM_create** operation.

© 2025 Siemens



10. Deploy the changes to the rich client.

    a. On the menu bar, choose **BMIDE→Save Data Model** to save your changes.

    b. Ensure that the test server is running.

    c. On the menu bar, choose **BMIDE→Deploy Template**. Type the password, click the **Connect** button, and when a connection is established, select the **Generate Server Cache?** check box and click **Finish**.

    d. When deployment is done, check the status in the **Console** view.

    e. To ensure your changes appear in the rich client user interface, exit the rich client and restart it to clear cache.

    When you restart, you may want to launch the rich client using the *install-location***portal\portal.bat -clean** option to ensure the cache is cleared.

    > Note:
    >
    > If your changes still do not appear in the user interface, delete the **Teamcenter** subdirectory in the user's home directory on the client. This directory is automatically created again when the user starts the rich client.
    >
    > On a Windows client, it is typically the **Users***user-name***Teamcenter** directory. On a Linux client, it is typically the **$HOME/Teamcenter/** directory.

© 2025 Siemens

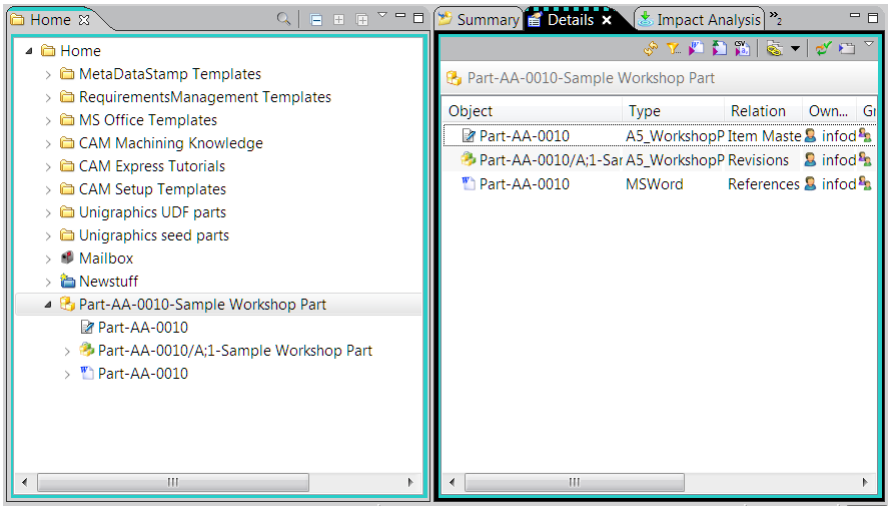
11. Verify the behavior of the extension rule.

    a. Run the rich client.

    b. In My Teamcenter, choose **File→New→Item** and select **Workshop Part** from the list.

    c. Create an instance of the item.

    An **MSWord** dataset is created and attached to the **Workshop Part** instance with the reference relation.



Workshop 13: Add a compound property

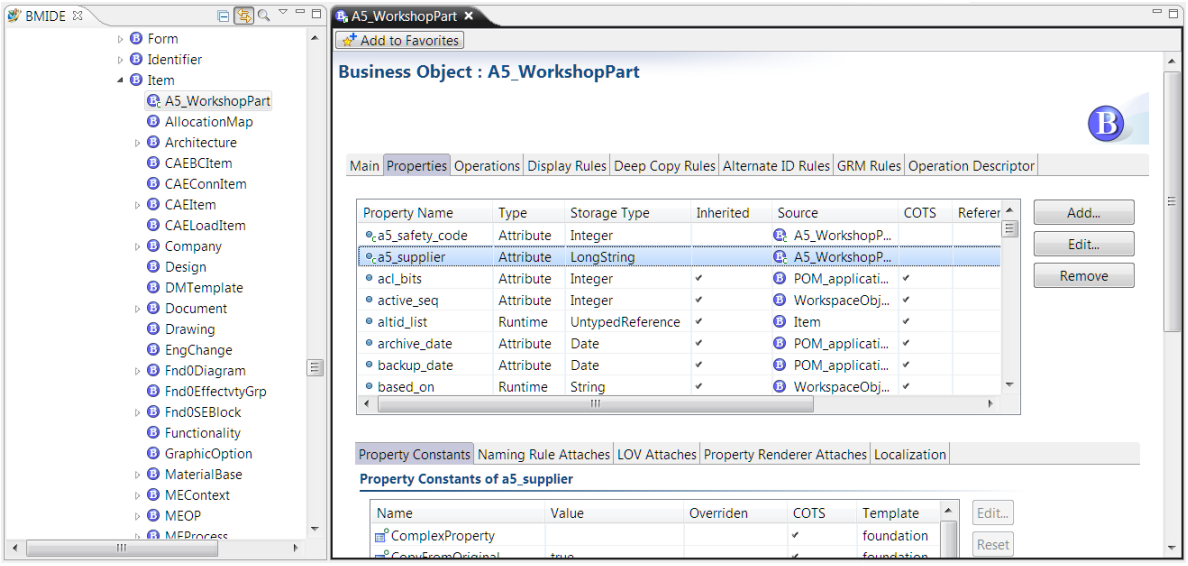
Perform this workshop to **create a compound property** that displays a property from one business object on another business object. In this workshop, you display the **a5_supplier** property from the **A5_WorkshopPart** business object on the **A5_WorkshopPartRevision** business object.

1. Open the **A5_WorkshopPart** business object and click the **Properties** tab.

   In an earlier workshop, you added the **a5_safety_code** and **a5_supplier** properties to this business object to display the **Safety Code** and **Supplier** properties on instances of workshop parts.

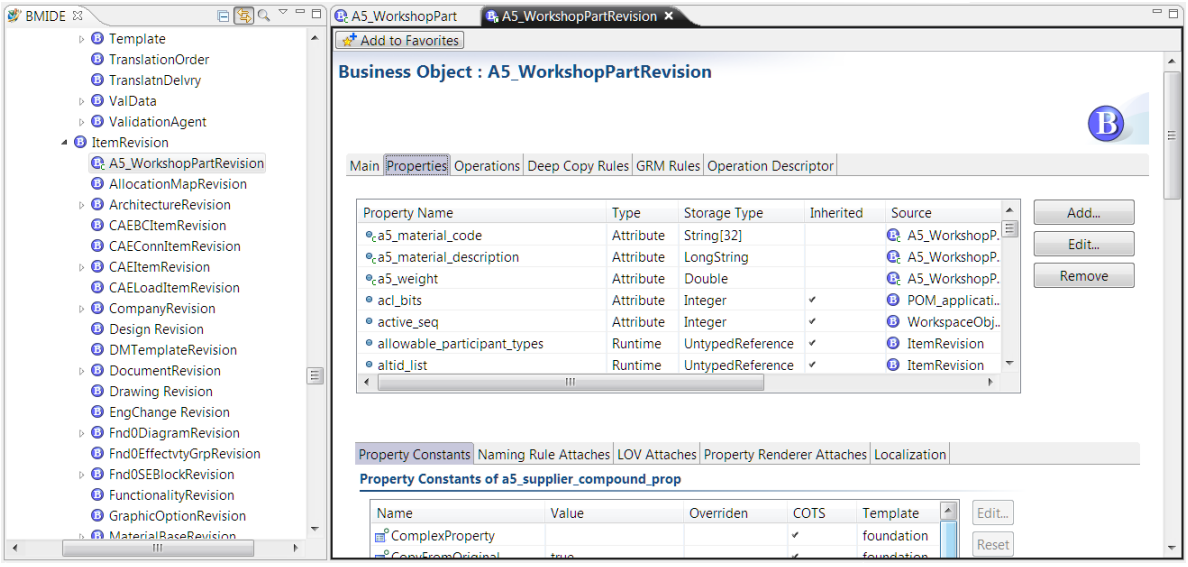
   However, you also want the **a5_supplier** property to display on the workshop part revisions. The solution is to create a compound property rule to display the property on the **A5_WorkshopPartRevision** business object.

© 2025 Siemens



2. Open the **A5_WorkshopPartRevision** business object and click the **Properties** tab.

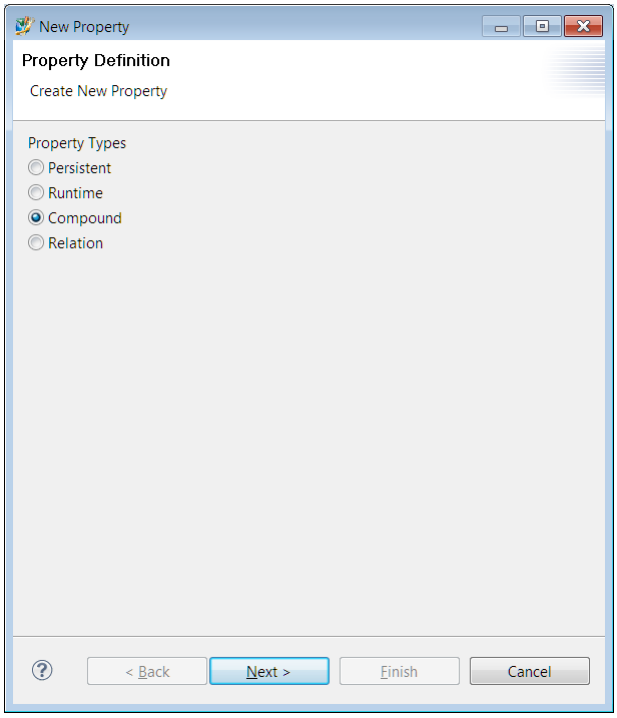
   Note how the business object displays the custom **a5_material_code**, **a5_material_description**, and **a5_weight** properties, but not the **a5_supplier** property. You can create a compound property to add the **a5_supplier** property on this business object.



3. Click the **Add** button to the right of the **Properties** table.

4. In the **Property Definition** dialog box, select **Compound** and click **Next**.

© 2025 Siemens



5. Perform the following on the **Compound Property Page**:

   a. In the **Name** box, type **a5_supplier_compound_prop**.

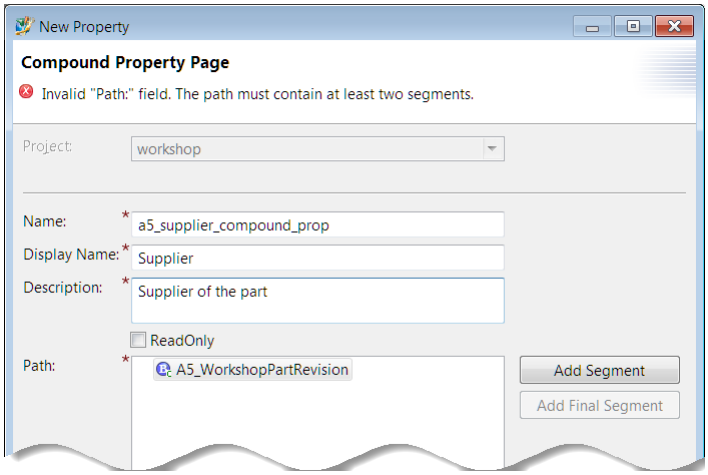
   > Tip:
   >
   > Label the property as a compound property so you can tell at a glance that it originates from another property.

   b. In the **Display Name** box, type **Supplier**.

   > Tip:
   >
   > Enter the display name so it is identical to the display name on the originating property (in this case, the display name of the **a5_supplier** property on the **A5_WorkshopPart** business object). If you enter a different display name, the end user would be confused to see different display names for the same property when it is displayed on different business object instances.

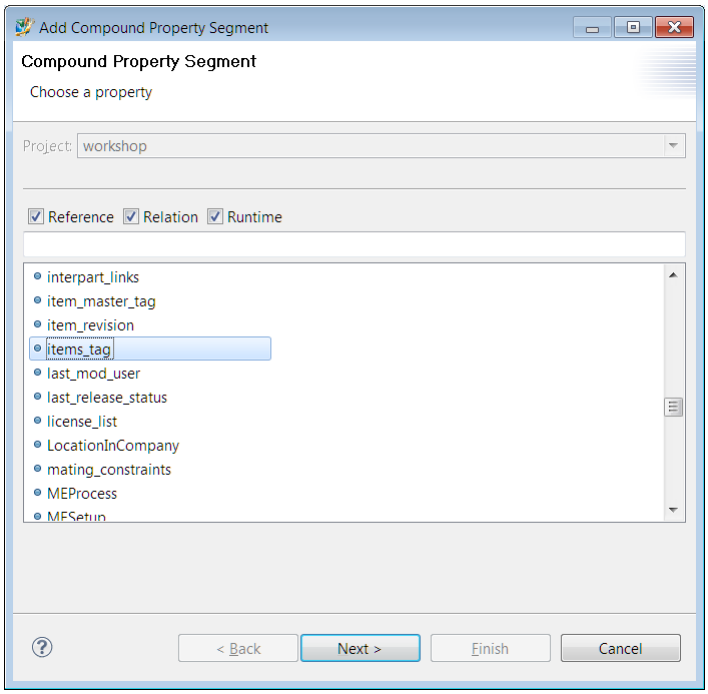
   c. In the **Description** box, type a description for the compound property.

© 2025 Siemens



d. Click the **Add Segment** button.

The **Compound Property Segment** dialog box is displayed.

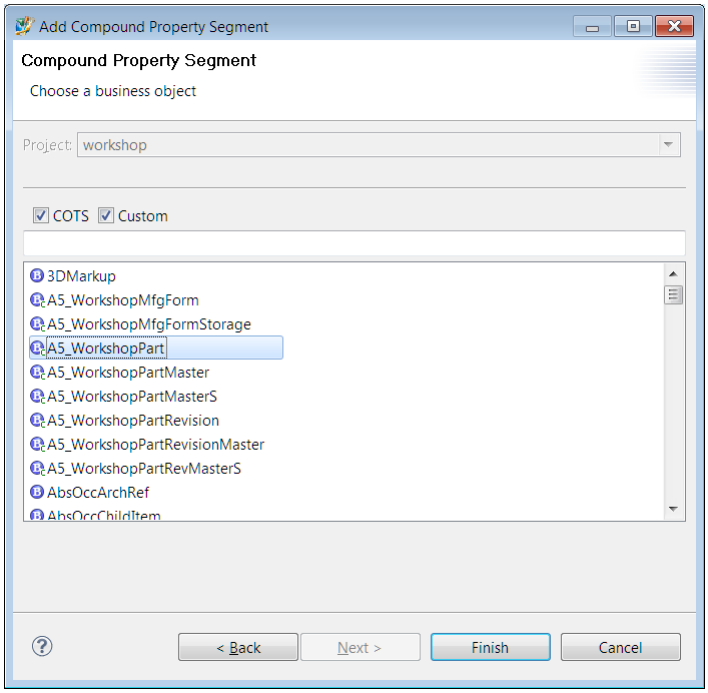


e. Select **items_tag** as the property on the target business object to hold the compound property.

f. Click **Next**.

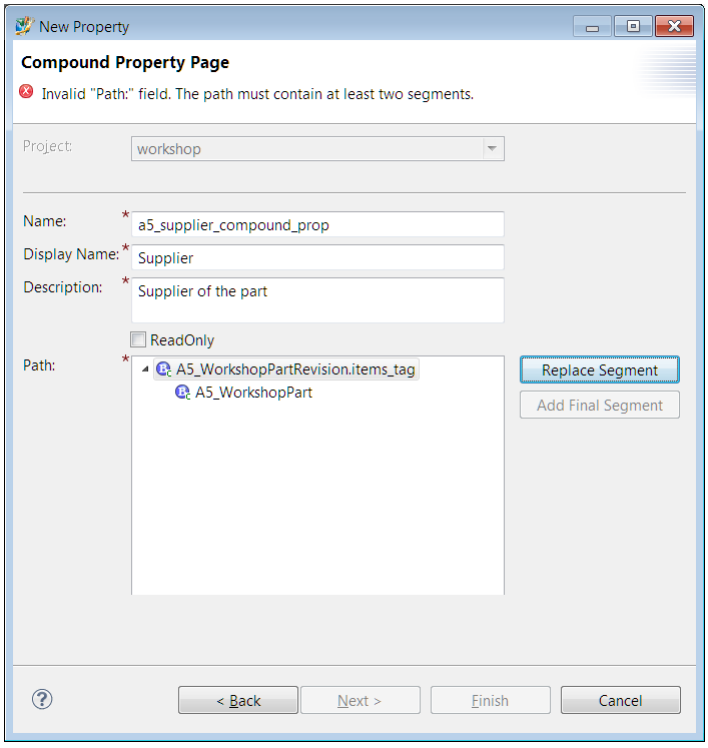
g. Select the **A5_WorkshopPart** business object as the source of the property.

© 2025 Siemens



h. Click **Finish**.

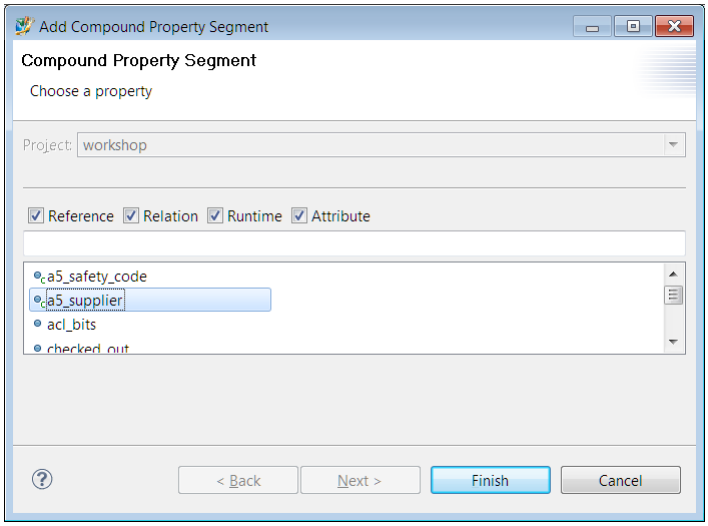
Notice how **A5_WorkshopPartRevision.items_tag** is displayed as the first segment. The first segment shows the target business object and the property on the target business object used to hold the compound property.



© 2025 Siemens

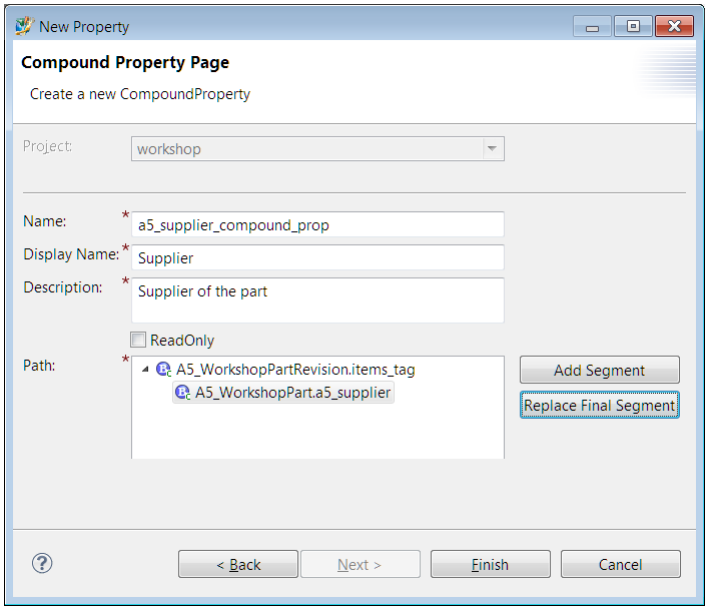
i. Select the **A5_WorkshopPart** business object and click the **Add Final Segment** button.

j. Select the **a5_supplier** property.



k. Click **Finish**.

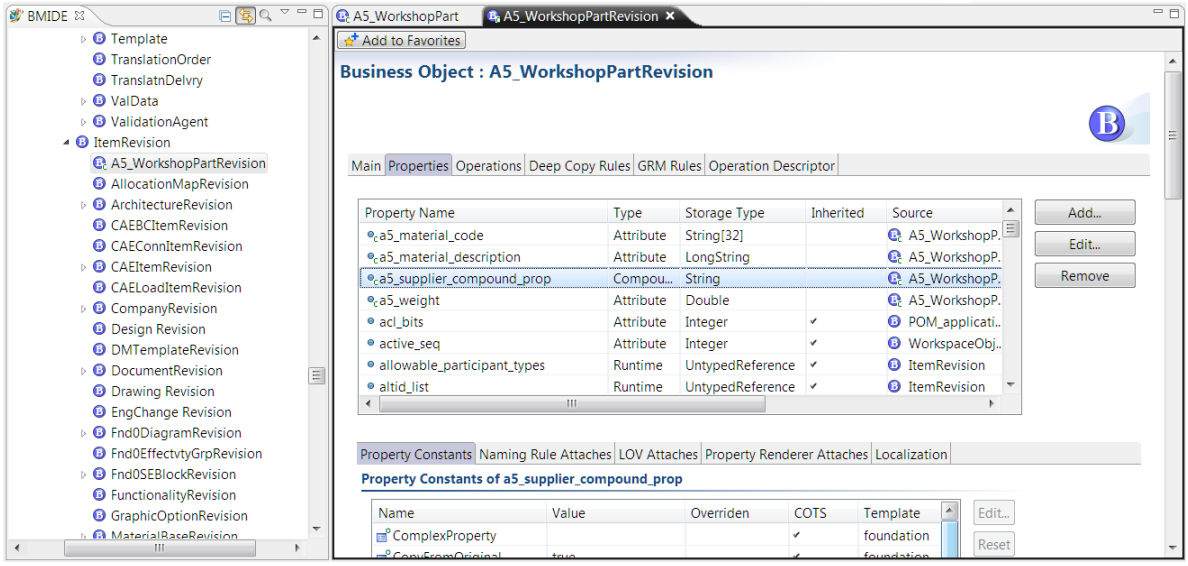
The completed compound property is displayed. Notice how **A5_WorkshopPart.a5_supplier** is displayed as the final segment. The final segment shows the source business object and the originating property for the compound property.



l. Click **Finish**.

The new compound property is displayed on the **Properties** tab.

© 2025 Siemens



6. Deploy the changes to the rich client.

   a. On the menu bar, choose **BMIDE**>**Save Data Model** to save your changes.

   b. Ensure that the test server is running.

   c. On the menu bar, choose **BMIDE**>**Deploy Template**. Type the password, click the **Connect** button, and when a connection is established, select the **Generate Server Cache?** check box and click **Finish**.

   d. When deployment is done, check the status in the **Console** view.

   e. To ensure your changes appear in the rich client user interface, exit the rich client and restart it to clear cache.

      When you restart, you may want to launch the rich client using the *install-location***portal\portal.bat -clean** option to ensure the cache is cleared.

      > Note:
      >
      > If your changes still do not appear in the user interface, delete the **Teamcenter** subdirectory in the user's home directory on the client. This directory is automatically created again when the user starts the rich client.
      >
      > On a Windows client, it is typically **Users***user-name***Teamcenter**. On a Linux client, it is typically **$HOME/Teamcenter/**.

7. Verify the behavior of the compound property.

   a. Run the rich client.

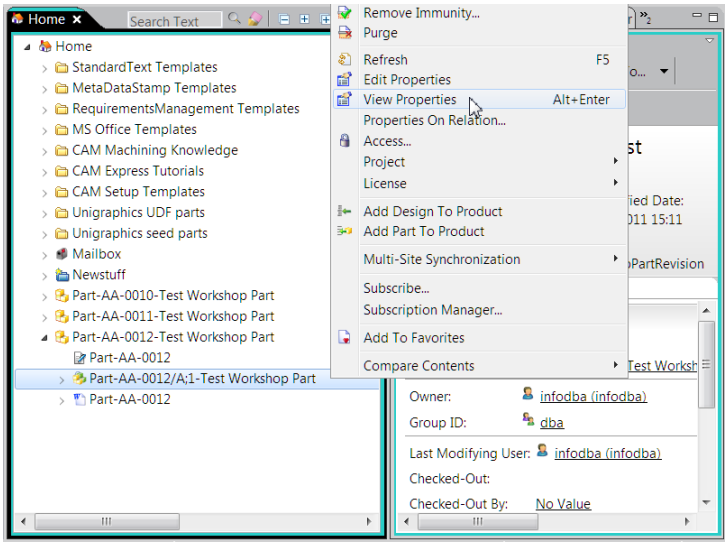
© 2025 Siemens

b. In My Teamcenter, choose **File→New→Item** and select **Workshop Part** from the list.

c. Create an instance of the item.

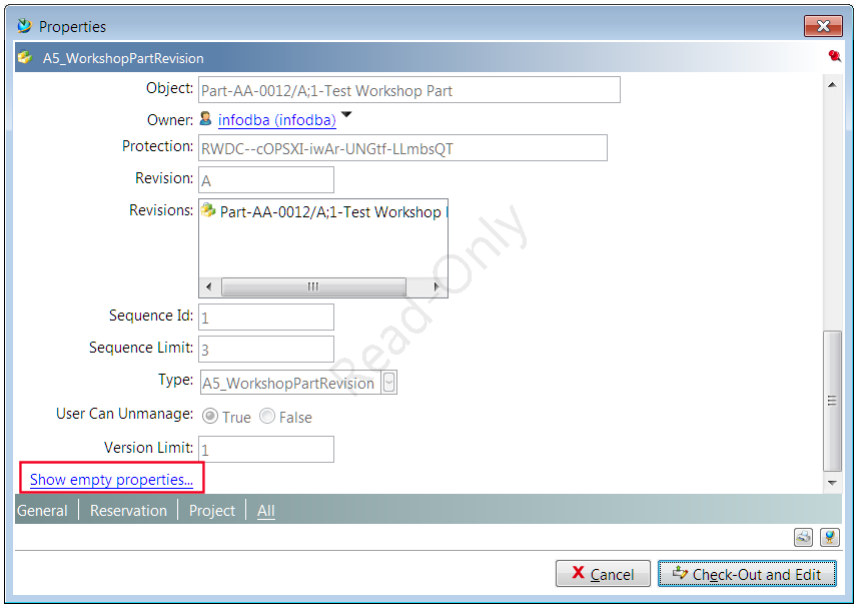
d. Open the item revision and view the property:

A. Right-click the workshop part revision and select **View Properties**.



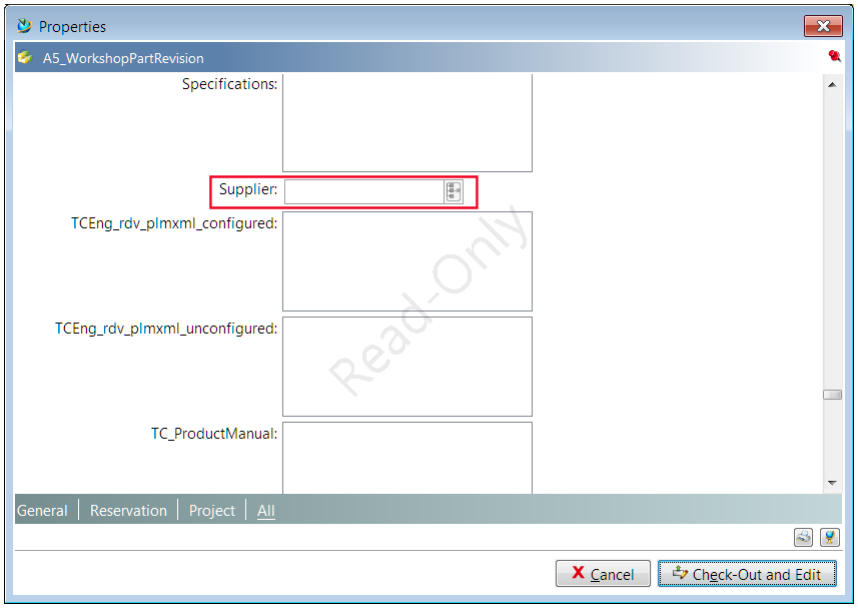
B. Click the **All** link in the **Properties** dialog box.

C. Scroll to the bottom of the **Properties** dialog box and click **Show empty properties**.



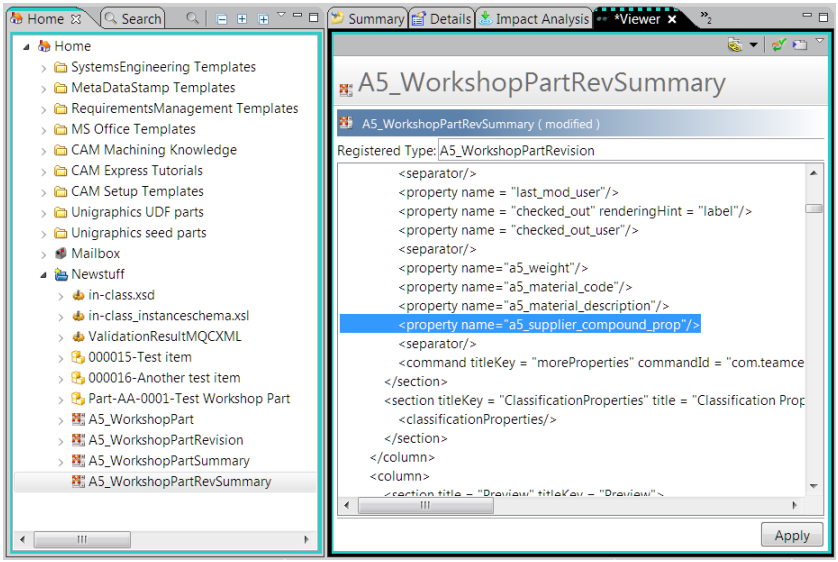
D. Scroll down to find the **Supplier** property.

© 2025 Siemens



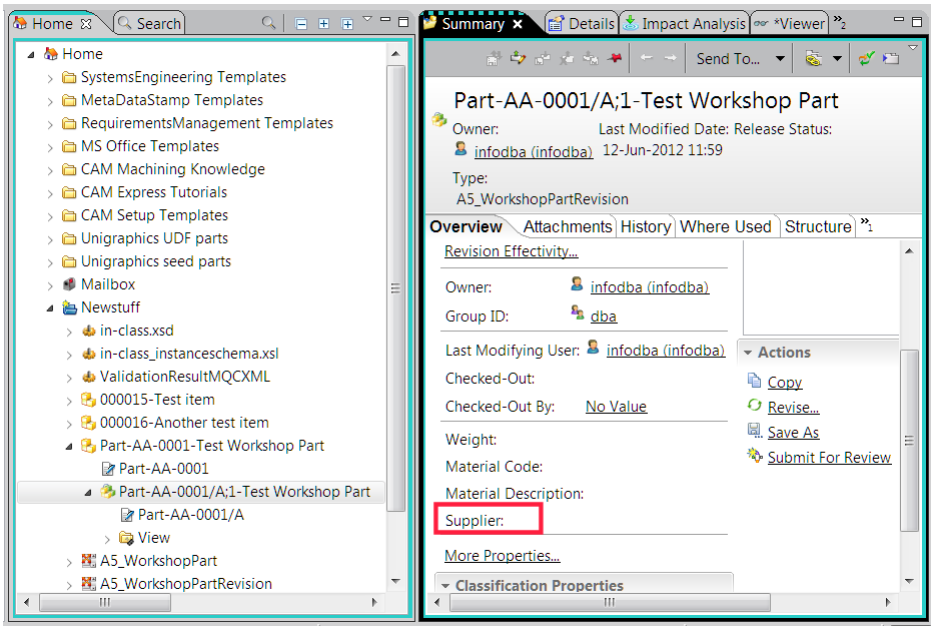
e. If you want this property to be visible in the **Summary** view on the workshop part revision, you can add the new **a5_supplier_compound_prop** compound property to the **A5_WorkshopPartRevSummary** XML rendering style sheet.

For example, open the **A5_WorkshopPartRevSummary.xml** file and add the **<property name="a5_supplier_compound_prop"/>** code as follows.



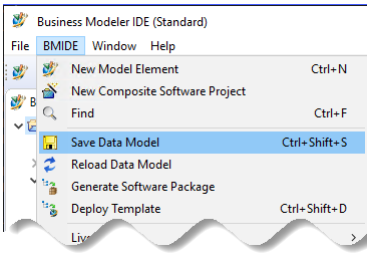
The compound property displays on the workshop part revision **Summary** view.

© 2025 Siemens



Workshop 14: Save your data model changes

1. After you finish performing extension tasks in a BMIDE, choose **BMIDE**>**Save Data Model**, or on the main toolbar click **Save Data Model** .



Note:

If the BMIDE uses a **source control management (SCM) system** but the project is not hooked up to the SCM plug-in, then when you use the **Save Data Model** option, a dialog box appears and asks if the files are to be made writable.

If you choose	Then
Yes	The files are made writable outside of the control of the SCM system. You must later synchronize the modified files with the SCM system.
No	The save operation is canceled. This is the recommended option.

© 2025 Siemens

If you choose	Then
	You should then either hook up the project to the SCM plug-in, or check out the files manually, and then run the save operation again.

For information about how to install an SCM plug-in and hook it up to a Business Modeler IDE templates project, refer to the documentation provided by the SCM plug-in.

2. (Recommended) Right-click your project and select **Reload Data Model**. Check the **Console** view for data model errors.

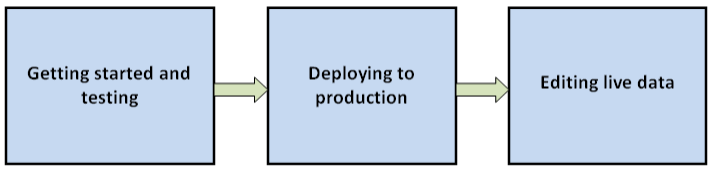
3. (Optional) To see your extension work in the extension file, open the **Project Files** folder, expand the **extensions** folder, and right-click the extensions file (for example, **default.xml**) and choose **Open With**>**Text Editor**. The file opens in an editor view, where you can examine the extension source code.

After saving changes, you can deploy the changes to the test server.

Business Modeler IDE process

Business Modeler IDE process overview

The Business Modeler IDE is a tool that shows you the business objects, properties and business rules that control the system. You can use the Business Modeler IDE to extend Teamcenter with your own business objects, properties, and business behavior. When you extend Teamcenter using the Business Modeler IDE, you should follow best practices for storing extensions into a template, testing the template in a test Teamcenter database, and then packaging and deploying the validated template to your production Teamcenter site. Later you may want to provide updates to a live running production site. All of these best practices have established processes for you to follow. The Business Modeler IDE provides tools to assist you through these tasks so that these steps are easy to manage.



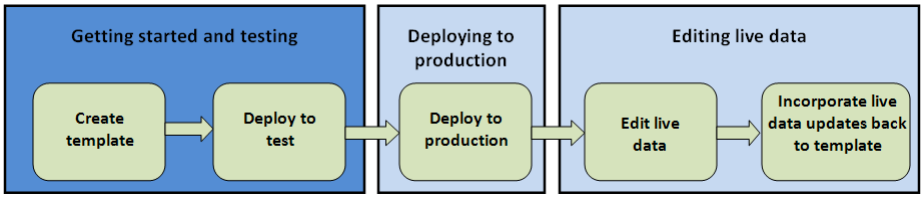
Business Modeler IDE process flow

Develop and test extensions

The Business Modeler IDE is a tool for extending Teamcenter with new business objects, properties, and business behavior. As you create extensions, the Business Modeler IDE stores them in a template. The template organizes your extensions as a single unit that encapsulates your individual extensions.

© 2025 Siemens

You can deploy this template as a single unit to any Teamcenter database server for testing or production usage. The deployment process ensures that all of your extensions are added to the default Teamcenter configuration.



First, create a new template project so that you can store your extensions in it for easy deployment to database sites. Then, perform a live deploy to push your template to a test Teamcenter server where you can validate it in a safe environment.

Typically, during this phase of development, you add new business objects and properties in addition to business behavior. Business objects and properties are considered schema changes and require a production Teamcenter server to be shut down and all users logged off to make the update. However, you can update a test server with a live deploy as long as you are the only user logged on to the system.

Follow this process:

- To extend Teamcenter when you have not yet created a template.
- To continue to add more extensions to your existing template.



Developing extensions and testing

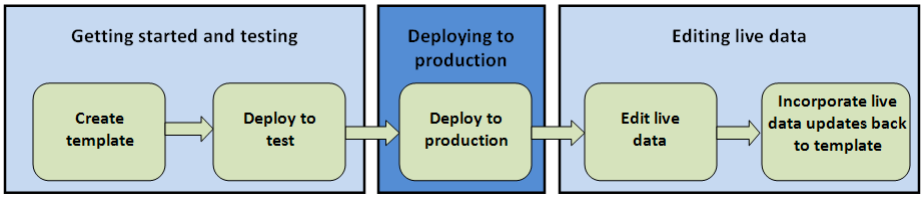
1. **Create a Business Modeler IDE template project** if you have not already done so.
2. Use the Business Modeler IDE to create new extensions to the data model and behavior.
3. **Perform a live deploy** to push your template to a test environment and validate your extensions.

To continue adding new extensions repeat steps 2 and 3.

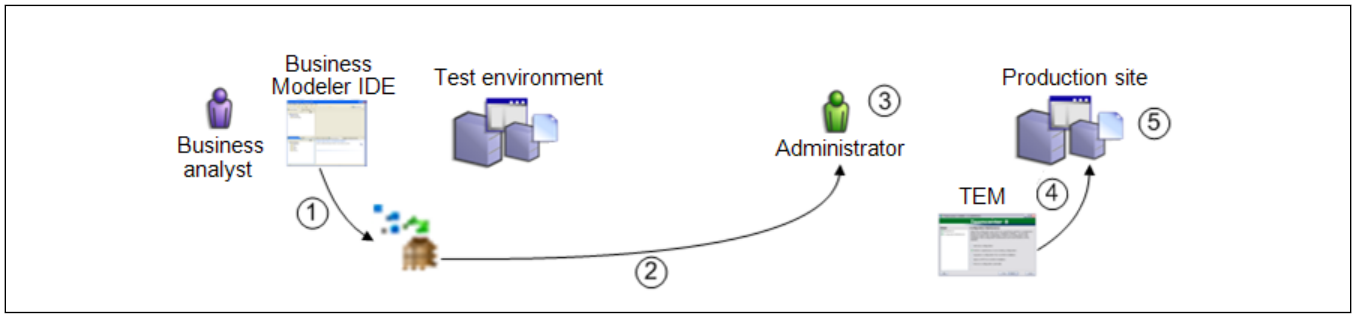
© 2025 Siemens

Deploy a template to a production site

After you validate your template in the test environment, you can update your production environment. Use the Business Modeler IDE to create a template package, shut down the production server, and use Teamcenter Environment Manager (TEM) to install the template package.



Follow this process to update your production site with your template.



Deploying a template to a production site

1. After you validate your extensions in a test environment, use the Business Modeler IDE to **build a template package**.

2. Send the template package to the production site administrator.

3. Log off all users and shut down the production Teamcenter site.

4. The production site administrator uses TEM to **install the template package** into the production site.

5. The production site administrator starts up the server at the production site and allows users to log on.

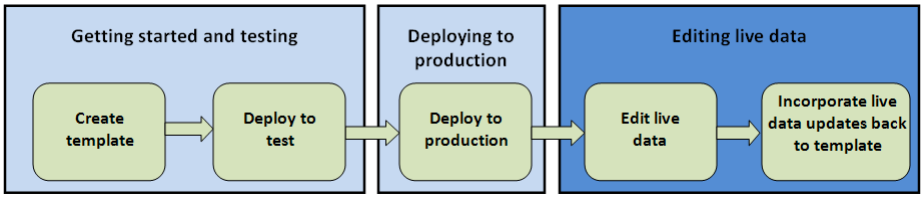
If you add more extensions later and have validated the template in a test environment, you can use TEM to update your template at the Teamcenter site by using the **Teamcenter Foundation→Update Database** menu commands in the **Feature Maintenance** panel.

© 2025 Siemens

Edit extensions in a live production site

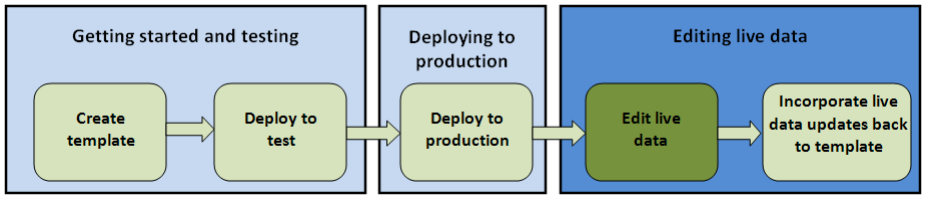
After you deploy your extensions to a production server, you may want to update some of your Business Modeler IDE extensions. For example, you may want to add a new value to a list of values (LOV). This type of update is called a *live update* because you are updating the data on a live system. Typically, you make this change directly to the extensions that are already deployed to the production database so you do not have to shut down the Teamcenter server. Therefore, only non-schema type changes, such as LOVs, statuses, business rules, and so on, can be made at this time.

Follow this process to edit your extensions deployed to a live production environment.



The Business Modeler IDE provides the live update project that points directly to the Teamcenter production database and allows you to make updates to your existing extensions in your deployed template. These updates are made to the extensions stored in the Teamcenter database; they do not affect the development environment template. Do not make the updates to your development environment template at this point. The development environment may contain ongoing work as you prepare for your next system downtime. This ongoing work may contain schema changes intended for the test environment, and it is difficult to separate them from the LOVs and status changes intended for the production site.

Control Business Modeler IDE elements that can be updated live

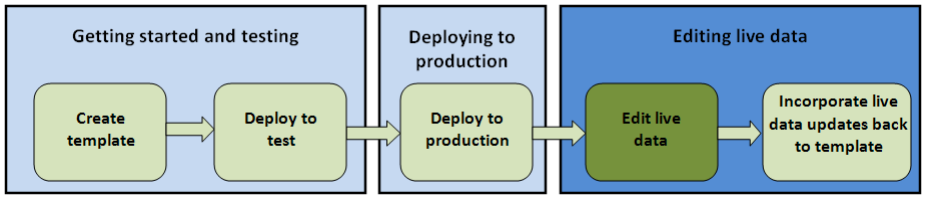


The Business Modeler IDE normally allows you to configure many different types of elements: business objects, properties, business rules, and so on. However, this list is limited when performing live updates. For example, you cannot update schema live because it forces you to log off all users to make the update. All elements in the Business Modeler IDE that are not schema can be updated live and are enabled by default. To disable specific elements or enable only a specific set of elements, you can configure the **Live Update** preference in the rich client.

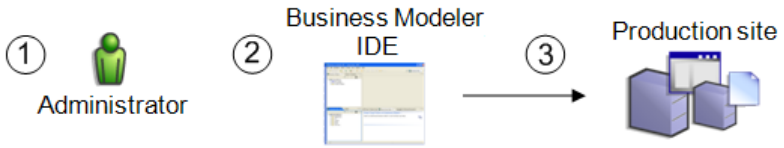
You should **configure the Live Update preference** before you allow administrators to use the Business Modeler IDE to make live updates.

© 2025 Siemens

Edit live data



Once the production site is running, you can edit the live data at any time. First, you must configure your Business Modeler IDE to point to the production site, and then you can edit the data.



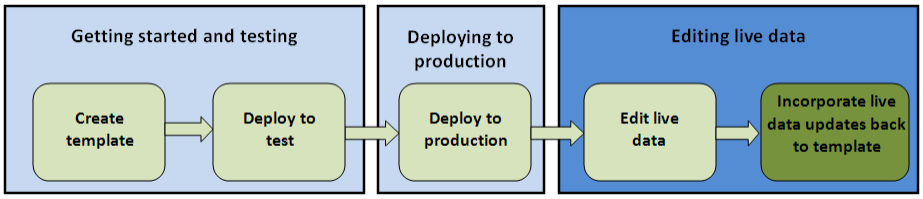
1. Configure your Business Modeler IDE to point to a production site and **create a live update project** so that it can download the template data.

2. Use the Business Modeler IDE to perform any necessary changes to the business behavior.

3. After you finish making changes, click the **Deploy Template** button on the toolbar to **push these changes to the production site**.

To continue editing extensions, repeat steps 2 and 3.

Incorporate live data updates from the production site



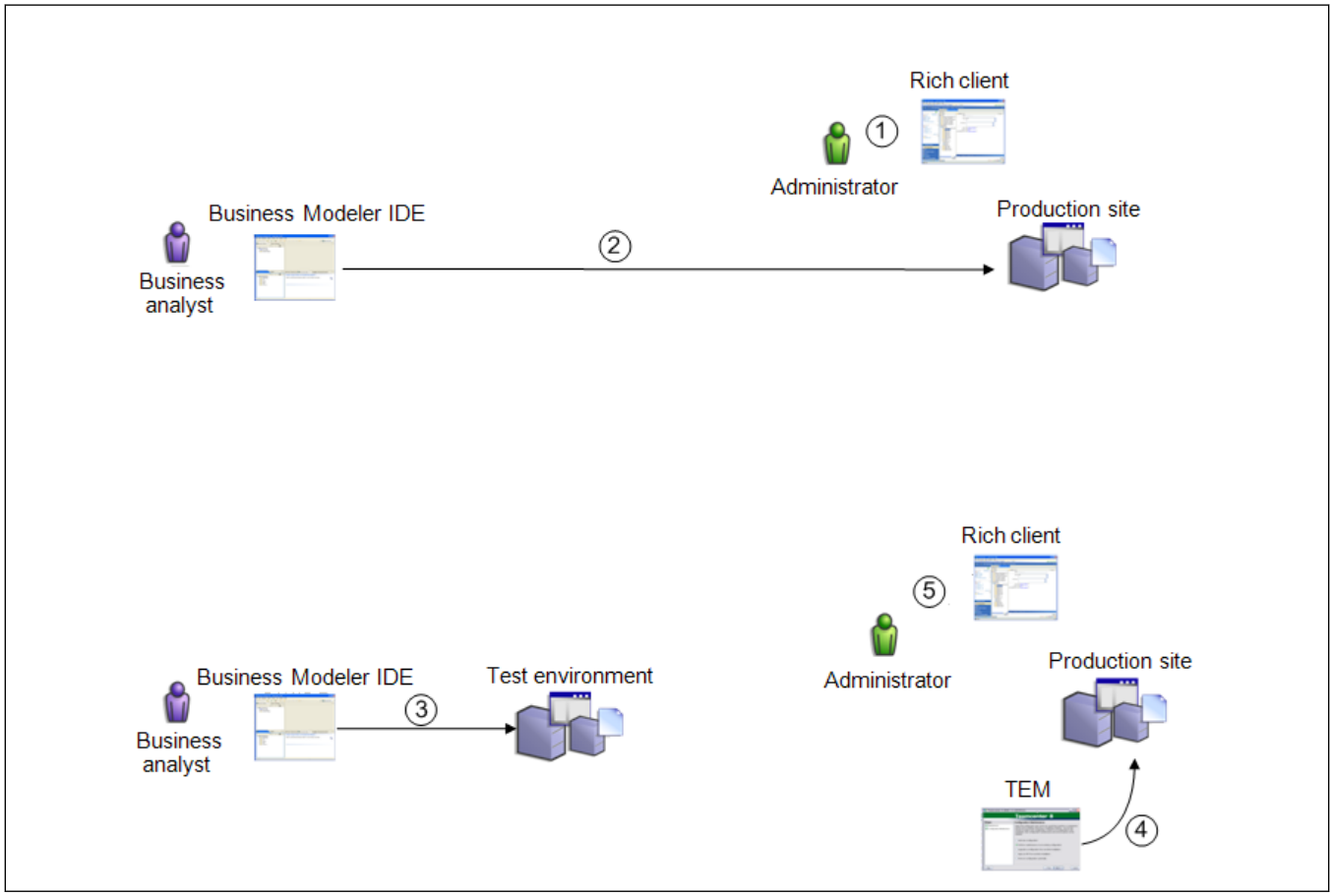
Eventually, you must incorporate all the live data updates performed at the production site back into the development environment. This event occurs as you prepare for your next system downtime.

If you do not perform this step, and you deploy your development environment template (which does not include the latest live data updates) to the production site, the live data updates already present in the production site are removed. Because this is undesirable, the system blocks the deployment. To ensure that the deployment is not blocked, you must perform this step.

To incorporate the latest live data, you can run a wizard from the development environment Business Modeler IDE to log on to the production Teamcenter server and retrieve the latest extensions. Or, if

© 2025 Siemens

your production site is offline from your Business Modeler IDE, you can ask an administrator to run a command line to extract the latest updates from the site and send them to you for incorporation. In either case, once the extensions are incorporated, you can continue with testing, packaging, and deploying to your production site.



Incorporating live data updates from the production site

1. Before you extract the latest live updates, it is a good practice to first block anyone from making any more live updates to ensure you get the latest. You can do this by clearing the **Allow Live Updates** check box in the **Live Update** preference dialog box in the rich client at the production site.

2. From your development environment Business Modeler IDE, launch the **Incorporate Latest Live Update Changes** wizard. Log on to the production site to pull the latest updates to your Business Modeler IDE. The Business Modeler IDE presents the changes to you in a graphical merge tool. Use this tool to determine which elements you want to keep or discard. Click **Finish** to save the changes to your local Business Modeler IDE template.

© 2025 Siemens

> Note:
>
> If your production site is not accessible to your Business Modeler IDE client over the network, you can ask the administrator of the production site to **extract the latest live data updates using the package_live_updates command line utility** and send it to you for incorporation.

3. At this point, you have integrated the latest live updates with your ongoing development work. Deploy this template to your test environment and validate that it works as expected.

4. To update your production site, **deploy the template to a production site**.

5. Before allowing users to edit live data again, reset the **Allow Live Updates** check box on the **Live Update** preference.

Development environments

Single production database environment

The first scenario is the simplest. It consists of a single production database and the Business Modeler IDE client. In this environment, a single user uses the Business Modeler IDE 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 to Teamcenter, the Business Modeler IDE user can log on Teamcenter to establish the connection and then **live update** the template from the Business Modeler IDE.

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 Business Modeler IDE to build the template feature, and then use Teamcenter Environment Manager (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.

Test and production database environment

This scenario is very similar to the first scenario except that it adds a test environment for the Business Modeler IDE user. In this scenario, the Business Modeler IDE user creates a template for extending the data model. The Business Modeler IDE user **live updates** the template to a test database, where the extensions can be tested.

After the extensions are tested to a satisfactory level, the user community is notified of a system downtime. When the system is shut down, **use the package wizard** in the Business Modeler IDE to build

© 2025 Siemens

the template feature, and then use Teamcenter Environment Manager (TEM) to **install or update the feature** 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 running while reiterating the development and test process.

User testing environment

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 Business Modeler IDE user creates a template for extending the data model. The Business Modeler IDE user **live updates** the template to a test database, where the extensions can be tested.

After the extensions are tested to a satisfactory level, the company is ready to roll out these changes. Before rolling the template out, you can set up a safe environment for training or for user acceptance testing. **Use the package wizard** to build the template feature, and then use Teamcenter Environment Manager (TEM) in the user testing environment to **install or 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, notify the user community of a system downtime. 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.

Multiple developer environment

This scenario is similar to the previous scenarios with the added requirement that there are two or more users of the Business Modeler IDE 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 Business Modeler IDE client. Examples of SCM systems include: CVS, Subversion, ClearCase, Perforce, and Visual Source Safe. By default, the Business Modeler IDE is equipped to connect to a CVS repository. All other integrations must be added separately to the Business Modeler IDE. 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

© 2025 Siemens

license fees and more robust functionality. To use an SCM with the Business Modeler IDE, the SCM must have a plug-in that integrates Eclipse with the SCM repository.

In this scenario, one Business Modeler IDE 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 checking all files and directories. Next, all other Business Modeler IDE users should synchronize their SCM repository to get the project files and directories downloaded to their machine and then import the project into the Business Modeler IDE using the **import wizard**.

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 reduces the amount of file merges. To create your own extension files, open the **Project Files** folder, right-click the **extensions** folder, and choose **Organize→New extension file**.

Each developer should have his own test environment for unit testing. Following the process established in the previous scenarios, each developer should create extensions, live update 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. 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. A project administrator or one of the developers should synchronize the Business Modeler IDE with the SCM to download the latest source files from the main repository. This gathers up all of the latest XML definitions from each developer to this Business Modeler IDE client. Note that after each synchronization with the repository, in the Business Modeler IDE, right-click the project and choose **Reload Data Model** to reload the data model and validate it for correctness. Use the **package wizard** to build the template feature, and then use Teamcenter Environment Manager (TEM) to **install the template package** into the integration environment.

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.

This scenario has the same benefits of the previous scenarios with added scalability to add more Business Modeler IDE 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.

> Note:
>
> Connecting the Business Modeler IDE to an SCM provides an extra benefit of backing up the project files in the SCM. You should back up the SCM on a regular basis.

© 2025 Siemens

SCM system

Using a source control management (SCM) system to manage files

The Business Modeler IDE manages the master XML files that contain your extension definitions. Although these XML definitions are converted to data model objects in the database during a deploy, these files are not stored in the database and should be cared for like source code. Back up these files regularly.

If more than one person will be working on the data model of a single Business Modeler IDE template, then you are required to use a source control management (SCM) system to integrate all developers' Business Modeler IDE clients with the central SCM repository. The SCM plug-in can be used to maintain version control of the source files, and to ensure protection of the data in the source files. In addition, with an SCM plug-in, multiple developers can work on the same project concurrently.

Even if only one person will be using the Business Modeler IDE to manage the data model, Siemens Digital Industries Software strongly suggests you use a SCM system plug-in with Eclipse to manage the project and source files.

You can use any SCM system that provides plug-ins to Eclipse, such as ClearCase, CVS, Subversion, or Perforce. Each developer connects to the repository to share and synchronize definitions.

Selecting a source control management (SCM) system

You can use source control management (SCM) systems with the Business Modeler IDE to manage your source files. To learn about SCMs, 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.

Any SCM that has a plug-in compatible with Eclipse can be integrated with the Business Modeler IDE. While there are many SCMs available that integrate with Eclipse, Siemens Digital Industries Software has found the following to work well:

- CVS (Concurrent Versions System)

  CVS is a free shareware application. By default, the Business Modeler IDE already has the CVS plug-in built into it; this saves you time from having to download it. If you have a CVS repository setup, you can connect to it from the Business Modeler IDE in the matter of a few minutes.

- Subversion

  Represents the next generation of CVS. Like CVS, Subversion is a free shareware application. Both Subversion and CVS offer adequate tools for checking in and out source files and synchronizing with a central repository.

- Perforce

© 2025 Siemens

Manages a central database and versioning repository. Perforce is a commercial SCM product.

- ClearCase

  Like Perforce, ClearCase is a commercial product. These commercial products offer a richer set of tools and functionality for managing source files.

Siemens Digital Industries Software product development has utilized all four of these SCM products with Eclipse and was able to work well with a **multideveloper 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 consider a commercial system.

Getting started with a source control management system (SCM)

Once you select a source control management (SCM) system, you should install the SCM server on a centrally located machine to the developers. The SCM server is the central repository that contains the source code. Each person using the Business Modeler IDE must also have the appropriate SCM plug-in installed into the *BMIDE_ROOT* so that the Business Modeler IDE can communicate with the SCM repository. Follow any directions on how to set this up to connect with the repository. After setup, Siemens Digital Industries Software recommends the following process to start using the Business Modeler IDE:

1. Open the **Advanced** perspective if it is not already active. Choose **Window→Open Perspective→Other→Advanced**.

2. One developer should create a Business Modeler IDE template project in Business Modeler IDE. This template is the template that all other developers will share and add definitions to. You do not want to have a separate template for each developer, because that is working contrary to the use of the SCM. The SCM is used to keep definitions separate and merge them into a single template.

3. After this developer creates the template project, you should create many source files. In the **Navigator** view, right-click the **extensions** folder and choose **Organize→Add new extension file**. The best way for multiple developers to add definitions to one template is to have each developer work in a separate file. This reduces the number of file merges that need to occur. Organize and name new files 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, they can add all of these definitions to their file and not interfere with other developers adding definitions to another file. Therefore this first developer should work with the project manager to discuss what kinds of new functionality are 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.

4. Once all files and folders are arranged, this first developer should check in the entire template project into the SCM. For systems like ClearCase and Perforce, 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

© 2025 Siemens

documentation on how to fill in the fields in this wizard. After completing this step, the template project is connected to the SCM. Next synchronize all files and folders to the repository. This pushes the files and folders to the central repository where others can start downloading them.

5. Each developer should launch their Business Modeler IDE and download the Business Modeler IDE template project to their client. For ClearCase and Perforce, you may have to launch a separate administrator tool to synchronize or download the latest Business Modeler IDE template files from the repository. This places the template folder and all of the files in a directory on your machine. Next use the **File→Import→Business Modeler IDE→Import a Business Modeler IDE Template Project** wizard to import the template folder from your local machine into your Business Modeler IDE.

   For CVS or Subversion users, you can use the **CVS Repositories** view to locate and download the template folder to your Business Modeler IDE. After downloading, right-click the project and choose **Reload Data Model** to load the template project into the Business Modeler IDE.

6. Once each developer has the template project in the Business Modeler IDE, he should set the active source file to the file that is assigned to him. The test environment should not be shared with other developers and this developer should be the only person deploying to it.

7. Developers can start to use the Business Modeler IDE to add definitions. After each developer finishes with these definitions, he should test them first by deploying to a local test environment.

8. After testing in the individual environment is complete, each developer should submit or synchronize his latest code back to the repository. This step uploads all changes created by the developer and should also download any new files that the repository has.

9. To test all developer changes together in one environment, an administrator can wait until all developers have submitted their changes. Then the administrator should synchronize or download the latest code. This imports all changes from all developers into their Business Modeler IDE clients. **Use the Package Wizard** to build a deployment package and then **use Teamcenter Environment Manager (TEM) to deploy this package** into a Teamcenter database that is being used as an integration environment.

   If integration testing fails, fix the issues in an individual test environment and resubmit to the repository. The administrator can synchronize again, to import 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 user testing environment.

10. Finally after all user testing passes, the deployment package can be deployed to the production environment through TEM.

© 2025 Siemens

Business Modeler IDE help

Accessing help in the Business Modeler IDE

You can access extensive help from within the Business Modeler IDE, and you can browse, search, bookmark, and print help topics.

To access this help	Do this
For the active view	Press F1 or choose **Help→Dynamic Help**.
For an active dialog box	In the lower left corner of the dialog box, click  .
The *BMIDE for Data Model Design* manual	Choose **Help→Help Contents**, and in the Help window left pane, choose *BMIDE for Data Model Design*.
The BMIDE Assistant	Choose **Window→Show View→BMIDE Assistant**.
Details about Help functionality in the Business Modeler IDE application, including how to bookmark help pages and refine search results	Choose **Help→Search**, and in **Search expression** type **help** and then click **Go**.
Getting Started wizards	Choose **Help→Welcome**.

Configure help on Linux systems

When you first attempt to open online help on a Linux system by choosing **Help**>**Help Contents**, you may see the following message:

```
Could not open a Web browser because
there are none configured.
Check the Web browser preferences.
```

To correct the problem, perform the following steps:

1. Choose **Window**>**Preferences**.
2. Choose **General**>**Web Browser** in the left pane of the **Preferences** dialog box.
3. Click **New** to provide the path to a Web browser, or **Search** to locate a Web browser.
4. Click **OK**.

© 2025 Siemens

Add a topic to the Business Modeler IDE online help

If your company has specific Business Modeler IDE procedures that you want to document and make available to users, you can add new topics to the Business Modeler IDE online help. You should also consider **creating your own online help book** that contains only your company's help topics, which can be much easier to maintain and carry forward from release to release.

To retain your new topics release-to-release, make sure you add them to the next release of the help by repeating the following process.

1. Extract the online help from the JAR file.

    a. Browse to the following directory:

        install-location`\bmide\client\plugins`

    b. Create a new subdirectory:

        `com.teamcenter.bmide.helpguide`

    c. Extract the following JAR file into the new directory:

        `com.teamcenter.bmide.helpguide_`*version*`.jar`

    d. Rename the JAR file suffix so that it is no longer being read by the Business Modeler IDE. For example, add **.bak** to the end of the file. From this point onward, the Business Modeler IDE online help is read from the directory you created.

2. Add the new help topic.

    a. Create the new help topic using an HTML editor.

    b. Copy the new topic to the **guide** folder in the **com.teamcenter.bmide.helpguide** directory.

        If your topic uses any images, copy them into the **guide\graphics** subdirectory.

    c. Add the new topic to the table of contents in the **toc.xml** file located in the **com.teamcenter.bmide.helpguide** directory.

        Use an XML editor to add the topic to the **toc.xml**, and follow the conventions in the rest of the file. Take care to properly close entries with **/>**, and if there are subtopics, close the master topic with the **</topic>** tag. Verify the structural validity of the file using an XML parser.

3. Verify that the new topic displays in the online help.

    a. Run the Business Modeler IDE.

© 2025 Siemens

b. Choose **Help→Help Contents**, and in the Help window choose the *BMIDE for Data Model Design* in the left pane.

c. Look for the new topic in the table of contents.

You can also remove topics by following similar procedures. Just remove topics from the **toc.xml** file and the **guide** subdirectory.

Create a new online help book in the Business Modeler IDE

Using a basic Eclipse development environment, you can create your own online help book that contains only your company's help topics, which can be added to the local or served help in the Business Modeler IDE.

The following procedures describe the basics of how to:

- Create a new help book
- **Test the new help book in Business Modeler IDE**
- **Add content to the help book**

Create a new help book

1. Ensure that your Eclipse environment uses the Eclipse version that is certified for your platform.

   For information about system hardware and software requirements, see the Hardware and Software Certifications knowledge base article on Support Center.

   If you need to install the certified version of the Eclipse SDK, then use the following procedure.

   a. Find the certified version of Eclipse for your platform on the Hardware and Software Certifications knowledge base article on Support Center.

   b. Download the SDK for the certified version of Eclipse from the Eclipse software archive.

   c. Extract the Eclipse SDK to a directory.

      By default, the files are extracted to an **eclipse** subdirectory.

2. Run Eclipse. For example, on a Windows system, run the **eclipse\eclipse.exe** file.

3. Create a new help plug-in project.

   a. Choose **File** > **New** > **Plug-in Project**.

© 2025 Siemens

b. In the **Plug-in Project** dialog box, type a name in the **Project name** box, for example, **myhelp**. Click **Next**.

c. In the **Content** dialog box, ensure that in **Options** the option **Generate an activator, a Java class that controls the plug-ins's life cycle** is selected, and then click **Next**.

d. In the **Templates** dialog box, select the plug-in with **sample help content** and click **Next**.

e. In the **Sample Help Table of Contents** dialog box, in the **Label for table of Contents** box type a name for your book. Select the **Primary** check box. Click **Finish**.

f. If you are prompted to open the associated perspective, then choose to do so.

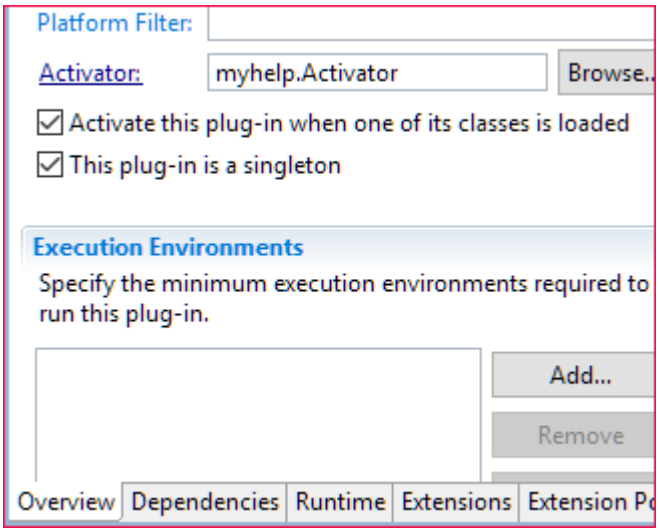
In the **Project Explorer** view for your new plugin project, Eclipse displays the project files, and in the work area, the **Overview** editor.

g. In the **Overview** editor, at the bottom of the **General Information** area, ensure the following:

- The **Activator** box contains the project name you specified in 3.b above.
- These two options are selected:

Activate this plug-in when one of its classes is loaded

This plug-in is a singleton



Test the new help book in Business Modeler IDE

1. To test the temporary help, export it to the Business Modeler IDE.

© 2025 Siemens

a. In the bottom center of the project window, to the right of the **Overview** and **Dependencies** tabs, click the **Runtime** tab.

b. In the **Runtime** editor, to the right of the **Exported Packages** pane, click **Add**, select your help project, and click **OK**.

c. Choose **File** > **Export**, and in the Export wizard, choose **Plug-in Development** > **Deployable plug-ins and fragments** and click **Next**.

d. In the **Deployable plug-ins and fragments** dialog box, select your help project, and to the right of the **Directory** box click **Browse** to browse to the location where the Business Modeler IDE client is installed, for example:

install-location`\bmide\client`

e. Click **Finish** and in the **Save Resources** dialog box, select all the resources and then click **OK**.

The help plug-in is saved as a JAR file in the **plugins** directory:

install-location`\bmide\client\plugins`

2. Verify that the book appears in the Business Modeler IDE online help.

a. Run the Business Modeler IDE.

b. Choose **Help** > **Help Contents**.

Your new book appears in the left pane of the help window.

Add content to the help book

1. In the **html** folder of the help project, add more HTML files, and add the files to the **toc.xml** file (the table of contents).

For detailed information about how to create online help content using Eclipse, see the *User Assistance Support* topics in the *Platform Plug-in Developer's Guide* > *Programmer's Guide* at the following URL:

http://help.eclipse.org

2. Test the help as you work.

a. In the **Package Explorer** view, right-click the help project and choose **Run As** > **Eclipse Application**.

A run-time version of Eclipse opens.

© 2025 Siemens

b. Choose **Help→Help Contents**.

Your new book appears in the **Contents** pane of the help window.

c. Close the help window and the run-time version of Eclipse.

3. To update the book in the Business Modeler IDE, re-export it by choosing **File** > **Export** > **Plug-in Development** > **Deployable plug-ins and fragments**.

Serve your custom help to Business Modeler IDE users

You can serve your Eclipse-based help from an Eclipse Platform information center server, in which case your Business Modeler IDE users can access the help via a web browser or from within their local Business Modeler IDE. By serving the help, content that you create when you **Add a topic to the Business Modeler IDE online help** or **Create a new online help book in the Business Modeler IDE** becomes instantly available.

The procedures below describe how to:

- Set up the Eclipse Platform information center server.
- **Configure client machines to access the served help from within BMIDE**.

Set up the Eclipse Platform information center server

1. Download the Eclipse **Platform Runtime Binary** for the Eclipse version that is certified for your platform.

   For information about versions of operating systems, third-party software, and Teamcenter software that are certified for your platform, see the Hardware and Software Certifications knowledge base article on Support Center.

2. Unzip the binary to a directory. This directory becomes your Eclipse Information Center site.

3. Copy your Business Modeler IDE help plug-in into the *infocenter-installation-location***eclipse\plugins** directory.

   You may want to verify that the Eclipse platform installation is installed successfully. You can launch Eclipse platform from the *infocenter-installation-location***eclipse** directory using a command similar to the following:

   ```
   eclipse -command start -eclipsehome infocenter-installation-location\eclipse -host host-name
   -port port-number -noexec
   ```

4. To start up the information center server, issue a start command similar to the following:

© 2025 Siemens

```
java -classpath infocenter-
installation-location\eclipse\plugins\org.eclipse.help.base_version.jar
org.eclipse.help.standalone.Infocenter -command start -eclipsehome infocenter-installation-
location\eclipse -port port-number
```

5. To verify that the help is running over the Web, access the help in your Web browser using an address similar to the following:

   ```
   http://host-name:port-number/help/index.jsp
   ```

6. To shut down the information center server, open another prompt and issue a shutdown command similar to the following:

   ```
   java -classpath infocenter-
   installation-location\eclipse\plugins\org.eclipse.help.base_version.jar
   org.eclipse.help.standalone.Infocenter -command shutdown -eclipsehome infocenter-
   installation-location\eclipse -port port-number
   ```

Configure client machines to access the served help from within BMIDE

So that your custom served help adds to or supersedes the locally installed Business Modeler IDE help, have each of the Business Modeler IDE users who want to use your served help perform the following steps on their machine.

1. In the Business Modeler IDE, choose **Window→Preferences**.

2. In the **Preferences** dialog box left pane, choose **Help→Content**.

3. In the **Content** pane, do the following:

   a. Select **Include remote help and give it priority**.

      With this selection,

      - The served help is added to the help content.

      - If the file name (not including any version number) of served help content matches the file name of content in the local **plugins** directory,

        install-location\bmide\client\plugins

        then the served help is used instead of the local help.

   b. Click **New**.

   c. In the **Add new information center** dialog box, enter appropriate values:

© 2025 Siemens

Name A name to identify the Information Center. The name you enter does not affect connectivity.

URL The address for the information center, similar to the following:

`http://`*host-name*`:`*port-number*`/help/index.jsp`

(as verified in **Step 5** of *Set up the Eclipse Platform information center server* above).

d. Click **Test Connection** to verify that the URL is correct and that the server is running.

e. Click **OK** to close the **Add new information center** dialog box.

f. Click **OK** to apply the change and close the **Preferences** dialog box.

© 2025 Siemens

8. Data model concepts

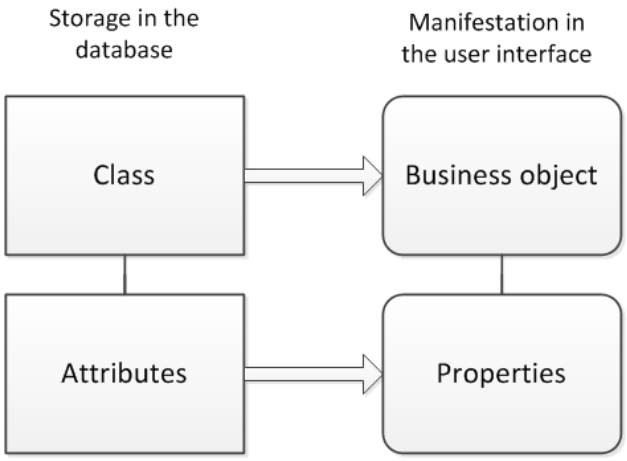
Teamcenter data model overview

The Teamcenter *data model* is a collection of abstract objects structured and organized to represent:

- Part designs and design documents, and the relationships between them
- Business processes that are applied to part designs and design documents

You use the Business Modeler IDE to configure and extend the Teamcenter data model to better correspond to your business.

The following diagram and table describe basic data model terms.



Class	A class represents an object table in the database. Each row in the database table is an instance of the class.
Attributes	Attributes are the persistent characteristics of the class. Attributes of a class define the columns of the class table in the database.
Business object	A business object represents a type of business data that you handle in a Teamcenter client. Business objects let you define the values to store in the database and set how objects behave, such as their lists of values, naming rules, display names, and so on. Many business objects directly correspond to a storage class that has the same name, and their properties are stored as attributes on that storage class. These are known as *primary business objects*. *Secondary business objects* are child business objects that store their information on their parent business object's storage class and do not have a storage class of their own. Business objects can be either *persistent* or *run-time* business objects. The persistent, or Persistent Object Model (POM), business objects are stored in the database in their

© 2025 Siemens

	associated storage class. The run-time business objects are not stored in the database but are calculated at run time.
Properties	Properties are business object characteristics. There are four property types:
Persistent	Holds permanent values. Persistent properties are stored in their business object's storage class as attributes. In other words, attributes of a storage class are expressed as persistent properties on the business object that uses that storage class.
Run-time	Displays values calculated at run time. When the value of a run-time property is requested, the system calculates it. Example: You want to create a run-time property to represent crankshaft torque, so you input the **lever_force**, **lever_length**, and **lever_angle** properties. When the value of torque needs to be displayed, it is calculated.
Compound	Retrieves the value of a property specified on another business object and uses it for the compound property.
Relation	Defines relationships between objects. Example: A document that specifies the part design is attached to a part by means of a specification relation property.

POM schema

At the root of the class and persistent business object structures is the **POM_object**. *POM* stands for:

- *Persistent*: survives from session to session
- *Object*: an entity being managed (a part, file, user, and so on)
- *Manager*: a method of managing the objects

POM defines an architecture (schema) for managing Teamcenter data in a database and in a running Teamcenter session. Business objects represent the different kinds of objects we want to manage, such as items, datasets, forms, folders, and so on.

The POM object model is distinct from the Teamcenter object model. POM is the *persistent* or physical object model, different from the *logical* object model that Teamcenter presents to users. POM is a

© 2025 Siemens

layer between Teamcenter and the database. As such, POM performs many tasks, such as managing concurrent access to persistent data to avoid multiple users overwriting each other's changes.

POM manages data in tables within a relational database, such as Oracle. Basic things to keep in mind about POM are:

- Each POM class is represented in the database by a table.
- Each instance of a POM class is represented by a row in the class table.
- Each class attribute is represented by a column in the class table.
- An object that is derived from multiple classes requires a *join* across each of the class tables from which it is derived.

Classes are the persistent representations of the POM schema and can be **viewed in the Classes view of the Advanced perspective**. You can **subclass from a class**, and all the attributes of the parent class are inherited by the subclass.

Business objects (types) are abstract implementations of POM classes. Each POM class is mapped to a primary business object whose name is the same as the POM class name. Primary business objects are derived from classes such as **Dataset**, **Folder**, **Form**, **Item**, **ItemRevision**, and so on. Sub-business objects are under these primary business objects.

The following terms are used to describe classes, business objects, and properties.

Class

A class is the definition of an object implemented in the Teamcenter data model. A class has a set of attributes, some of which are inherited from parent classes and some of which are defined directly for the class.

Attribute

An attribute is a persistent piece of information that characterizes all objects in the same class.

Business object

A business object is the specialization of a Teamcenter class based on properties and behavior.

Primary business object

A primary business object corresponds to each POM class; that is, the primary business object name is the same as the POM class name. Teamcenter automatically creates the primary business object when a new POM class is instantiated.

Sub-business object

Sub-business objects are all business objects that belong to the same POM class. Sub-business objects inherit all the properties and behavior of the primary business object. Sub-business objects model the actual objects that end-users create.

© 2025 Siemens

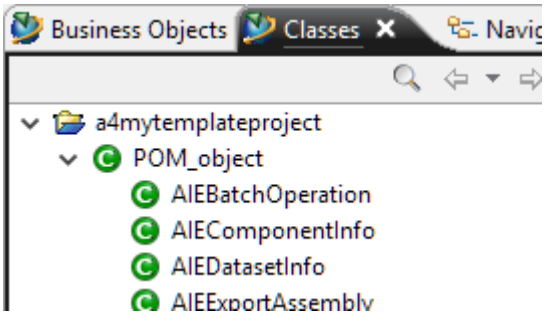
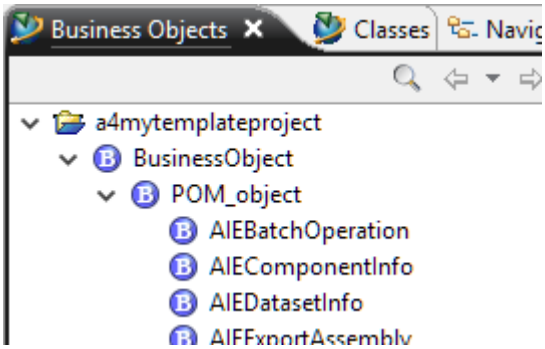
Sub-business objects are also known as secondary business objects.

Property

A property is a piece of information that characterizes all objects of the same type. At a minimum, all sub-business objects have properties that correspond to the attributes for the associated class.

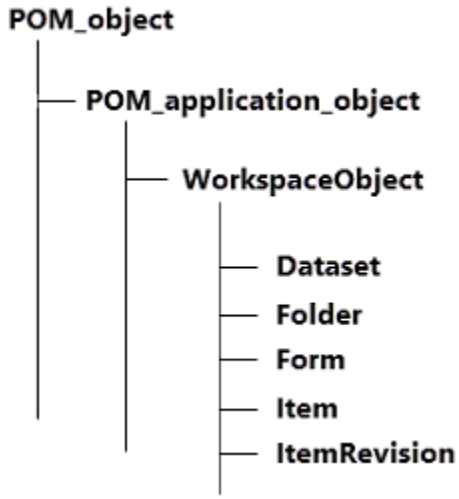
Viewing POM schema

The Teamcenter persistent object manager (POM) defines the data model architecture, or schema, using classes and business objects.

Element	Description	Appearance in BMIDE Advanced perspective
Classes	The persistent representations of the schema. Each class is mapped to a *primary business object* whose name is the same as the class name. In the Business Modeler IDE, classes can be seen in the **Advanced** perspective **Classes** view.	 **Classes view (POM schema)**
Business objects	The logical representations of the classes. Also known as *types*. In the Business Modeler IDE, business objects can be seen in • the **Advanced** perspective **Business Objects** view, • and in the **Standard** perspective **BMIDE** view.	 **Business object view (logical representation of the schema)**

Business objects represent the different kinds of objects you want to manage, such as items, datasets, forms, folders, and so on. The following figure shows these objects in an abbreviated view of the schema.

© 2025 Siemens



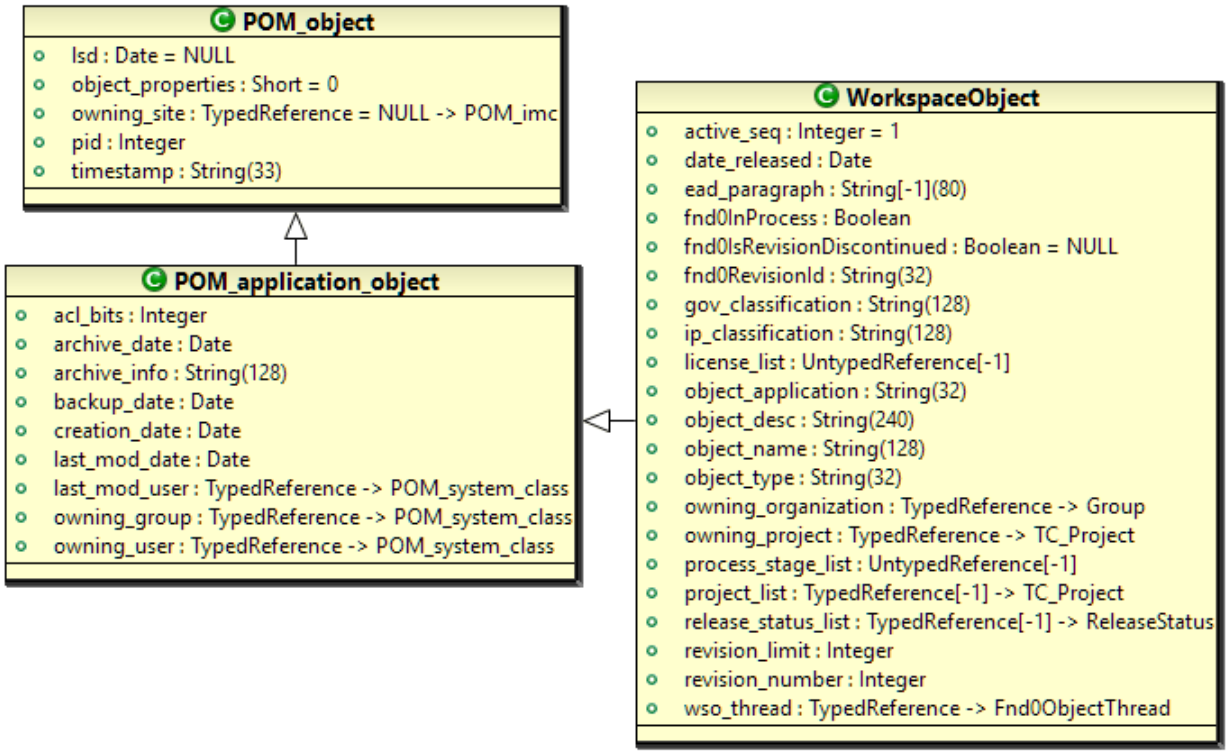
Abbreviated view of the schema showing object types

To see the information on a class or business object, in the Business Modeler IDE right-click the object and choose **Open in UML Editor**.

To see the inheritance relationships the object has to other objects, in the UML Editor right-click the object and choose **Show→Inheritance to Root**.

> Example:
>
> The following image is taken from the UML Editor and shows the inheritance of the **WorkspaceObject** class.

© 2025 Siemens



Inheritance of the WorkspaceObject class

All POM objects derive from the root class **POM_object**, and thus every POM object inherits the **owning_site**, **pid**, and **timestamp** attributes from the **POM_object** class.

Application objects that derive from the **POM_application_object** class inherit its additional attributes, such as **creation_date**, **last_mod_date**, **last_mod_user**, **owning_group**, and **owning_user**.

A popular class to subclass from is the **WorkspaceObject** class. It adds numerous attributes, including **object_desc**, **object_name**, and **object_type**, to name a few.

> Tip:
>
> For more information about how to work with the UML Editor, see *Open a class or business object in the UML editor*.

Schema versus non-schema objects

Schema refers to classes and attributes managed by the Business Modeler IDE template. *Non-schema* refers to all elements managed by the Business Modeler IDE template except for classes and attributes. Examples of non-schema objects are business objects, properties, status, LOVs, rules, and so on.

© 2025 Siemens

Class structure and attribute inheritance

You must understand class structure and attribute inheritance to effectively perform these rich client tasks:

- Create closure rules in PLM XML/TC XML Export Import Administration

  Closure rules control the scope of the data translation on both input and output. They specify how the data structure is traversed. Closure rules use classes or business objects (types) to define the data structure.

  In the PLM XML/TC XML Export Import Administration application, select the **ClosureRule** node in the lower-left pane of the window. Click the **Add clause** (**+**) button to the right of the clause table. To select the first kind of object to use in the data structure, click in the **Primary Object Class Type** cell and select **Class** or **Type** (business object), and type the name of the class or business object in the **Primary Object** cell. For the second object in the structure, use the **Secondary Object Class Type** cell and the **Secondary Object** cells.

- Create queries in Query Builder

  Queries are searches for objects in the database. When you build queries, you specify the kind of objects you are looking for (the class) and characteristics of the objects (attributes).

  In the Query Builder application, click the **Search Class** button to select the class. You are presented a class tree similar to the **Classes** view in the **Advanced** perspective of the Business Modeler IDE. When you select a class, its attributes are shown in the **Attribute Selection** pane. Double-click attributes to add them to the query. Click the **Display Settings** button and choose **Real Names** to see the attributes with their internal name (not the name displayed in the rich client UI).

- Create property sets in Report Builder

  Property sets are collections of Teamcenter data, such as class attributes or business object (type) properties. You can create a property set that contains only the information you want to use in a report.

  In the Report Builder application, choose **File→Create Property Set**. Click the **?** button to the right of the **Search** box to look for a class or business object (type). When you select a class you are presented with a class tree similar to the **Classes** view in the **Advanced** perspective of the Business Modeler IDE. Double-click attributes to add them to the property set. Click the **Display Settings** button and choose **Real Names** to see the attributes with their internal name (not the name displayed in the rich client UI).

- Create cubes in Teamcenter Reporting and Analytics

  Reporting and Analytics is a stand-alone reporting application. When it is installed and deployed in a Teamcenter environment, it integrates with Report Builder and displays reports in the **TcRA Reports** folder. If you want to create a report that has a specific set of data, you must first use Mapper to

© 2025 Siemens

create a cube that contains that data. A *cube* defines the data displayed in a Reporting and Analytics report. For example, if a report has columns that list the item ID, description, and creation date, these columns are provided by the cube.

In the Reporting and Analytics Mapper application, choose **Cube→Define** to create a cube. In the **Search** tab in the **Search classes** box, type the name of the class you want to use for the cube. In the **Objects and Steps** tab, choose the attributes on the class that you want to use in the cube.

You also must understand business object structure and property inheritance to perform these Business Modeler IDE tasks:

- **Create Generic Relationship Management (GRM) rules**

  Generic Relationship Management (GRM) rule applies constraints on the relationship between two business objects. When you create a GRM rule, you select the primary and secondary business objects for the relationship, the relationship they have to one another, and the constraints to be applied.

  On the toolbar in the Business Modeler IDE, click the **Open GRM Rules Editor** button  . Click the **Add** button in the **GRM Rules** editor. Click the **Browse** button to the right of the **Primary Object**, **Secondary Object**, and **Relation Object** boxes to choose the business objects and their relationship to one another.

- **Create compound properties**

  A compound property is a property on a business object that can be displayed as a property of an object (the display object) although it is defined and resides on a different object (the source object). A compound property uses relation and reference properties to traverse from the source to the destination object.

  In the Business Modeler IDE, open a business object and click the **Properties** tab in the resulting view. Click the **Add** button to the right of the properties table, select **Compound**, and click **Next**. Click the **Add Segment** button to choose the business objects and properties to use for the rule.

© 2025 Siemens

9. Managing the data model

Find objects in the Business Modeler IDE

To find objects, click the **Find** button at the top of a view, or the **Browse** button in a dialog box. In the **Find** dialog box, you can choose to search for the custom objects you have created or the standard COTS (commercial off-the-shelf) objects that ship with Teamcenter.

Use a wildcard ***** to find all instances of the object, or before and after a search string to find all instances of the string.

Open objects in the Business Modeler IDE

1. Right-click the object.
2. Choose **Open**.

   Details of the object appear in a new view. You can modify the object by entering new values.

Delete objects in the Business Modeler IDE

You can only delete the custom objects you have created. You cannot delete COTS (commercial-off-the-shelf) objects, because they are standard objects provided by Teamcenter.

If the custom object you want to delete has children, you must delete the children first before you can delete the parent. Also, when you delete an object that is referenced by other rules or objects in the system, the Business Modeler IDE tells you which references to clean up first before you can delete the selected object.

> Caution:
>
> You must use the rich client to delete all instances you created on your test server before you delete the object in the Business Modeler IDE. If you delete a custom object before removing instances of that object, errors can appear in the deployment log the next time you deploy. This is because the instance is still in the database, although the object definition has been removed.

1. Right-click the custom object
2. Choose **Delete**.

   The object is removed from the Business Modeler IDE.

© 2025 Siemens

Note:

- If you attempt to delete some objects, the Deprecate Object wizard runs. This is because some objects cannot be deleted outright until a certain number of releases have passed, and instead can only be **deprecated**.
- When you delete a custom primary business object, the associated class is deleted. And when you delete a custom class, the associated business object is deleted. However, if you delete a custom secondary business object, the associated storage class is not deleted.

Modifying objects in the Business Modeler IDE

Adding properties to certain basic COTS business objects, for example `POM_object`, `User`, and `BusinessRule`, is not allowed. In such cases, the **Add** button on the object's properties tab is disabled.

If you attempt to modify an object and the **Modify object** dialog box appears, then for some reason the original modification is not allowed.

For example, the **Modify object** dialog box may appear when you are working with conditions. The condition values that can be changed depend on whether the condition is COTS, whether it is marked as secure, and whether it is referred to by any business rule:

If the condition is	and it is	then
COTS	secure	Nothing can be changed.
COTS	not secure	Only the expression can be changed.
custom	not referred to by any business rule	You can select new input parameters to change the signature, or type new expressions in the **Expression** box.
custom	referred to by any business rule (for example, naming rule attachments, LOV attachments, and so on)	You cannot modify the signature.

Deprecating objects in the Business Modeler IDE

Deprecation means that an object is going to be deleted in a future product release. When an object is marked as deprecated, it cannot be deleted until a certain number of releases have passed since the deprecation.

To deprecate an object, delete it; a deprecation dialog box appears asking if you want to deprecate the object. You can only deprecate an object if the deprecation policy is set for the project, the object is custom, and the object was created in an earlier version.

© 2025 Siemens

Your company has its own deprecation policy that states the number of releases that must pass after an object's deprecation before the object can be deleted. To change the number of releases, as well as whether a deprecation policy is set for the project, you must change the project properties. Right-click the project, choose **Properties**, and choose **Teamcenter→Code Generation** in the left pane. Select the **Enable Deprecation Policy** check box to allow for removal of obsolete objects from the project, and click the arrow in the **Number of Allowed Releases before Deletion** box to select how many releases before objects can be deleted from the project.

You can deprecate business object operations and property operations. To deprecate an operation, on the **Operations** tab, select an operation that is from an earlier release, and choose the **Deprecate** button.

You can also un-deprecate objects that have previously been un-deprecated. For example, to un-deprecate an operation, select a deprecated operation on the **Operations** tab and click the **Undeprecate** button.

Naming objects in the Business Modeler IDE

The Business Modeler IDE performs a validation at the time you create an object to ensure its name is unique. You are not allowed to create duplicate objects with the same name. This includes objects names that differ only in their case (uppercase or lowercase).

When you create new data model objects, Siemens Digital Industries Software requires that you add a prefix to the name to designate the objects as belonging to your organization. The prefix is set with the **Prefix** box when you create a template project. Thereafter, whenever you create new objects in that project, the same prefix is added to the name of all objects. For example, if the prefix is set as **A4_** when the project is created, when you create a new **Item** class, you could name it **A4_Item**. This prevents naming collisions in future versions if you name an object the same as a Teamcenter object in a future release. Siemens Digital Industries Software requires that you apply this consistent naming convention to all your new data model.

Use following conventions when choosing a prefix:

- Ensure the prefix contains two to four characters. Underscores are allowed.
- Place an uppercase letter in the first character position.
- Place a digit in character position two, three, or four.
- You can use digits 4-9. Do not use digits 0, 1, 2, or 3 because 0 and 1 are reserved for Siemens Digital Industries Software templates, and 2 and 3 are reserved for third-party template development.

Following are examples of acceptable prefixes:

A4
B5cd
F7_

© 2025 Siemens

X999

You can **rename a business object** by right-clicking it in the **Business Objects** folder and choosing **Rename**.

Although every effort has been made to ensure the aforementioned letters and numerals are available for prefix naming use, some Siemens Digital Industries Software released templates may already have used prefixes you want to create. For example, the Industry Catalyst template uses the **I*nn*9** prefix. If you have any concerns about the uniqueness of your naming prefix, check with your Siemens Digital Industries Software representative.

For each class there is a database table with the same name. Each attribute on the class equates to a column in a database table. Database tables must be uniquely named so that data can be properly mapped from the application to the database. Database systems such as Oracle convert all table names to uppercase. This means that the Teamcenter class spelled **ItemRevision** is mapped to a table spelled **ITEMREVISION** in the database. Because all table names are converted to uppercase, in Teamcenter you cannot define two class names that differ only in case, because both would result in the same table space name. Because every class has a business object, business objects must also follow the same convention for uniqueness.

Business object (type) names and class names entered in the Business Modeler IDE while performing data model extensions must be USASCII7 characters only. This prevents any template installation, upgrade, or deployment issues.

For any existing business object (type) names or existing class names that do not follow the USASCII7 format, you can use the **change_type_name** utility to rename the type name to a valid USASCII7 name.

Reloading the data model

You can reload the data model for a project by right-clicking a project and choosing **Reload Data Model**. This reloads all data model elements from the source files.

Reload the data model if the dependent templates have been changed during use of the Business Modeler IDE, or if you are using a source control management (SCM) system and source files have been updated.

Import and export the data model

Import a Business Modeler IDE template package

A template package contains extensions to the data model that can be distributed for installation to a production environment. You can import a **packaged template** into the Business Modeler IDE to create a project. This is an effective method for recovering a project that may have become corrupted.

© 2025 Siemens

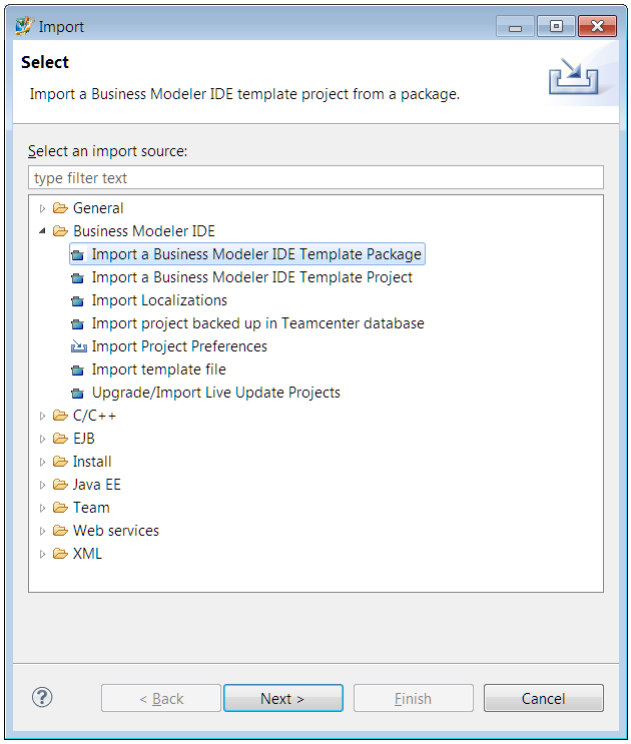
Note:

You cannot use this procedure to import a composite software package that was generated using **Generate Software Package**. Instead, use the **File→Import→General→Existing Projects into Workspace** sequence to import the composite software project.

1. Choose **File→Import**.

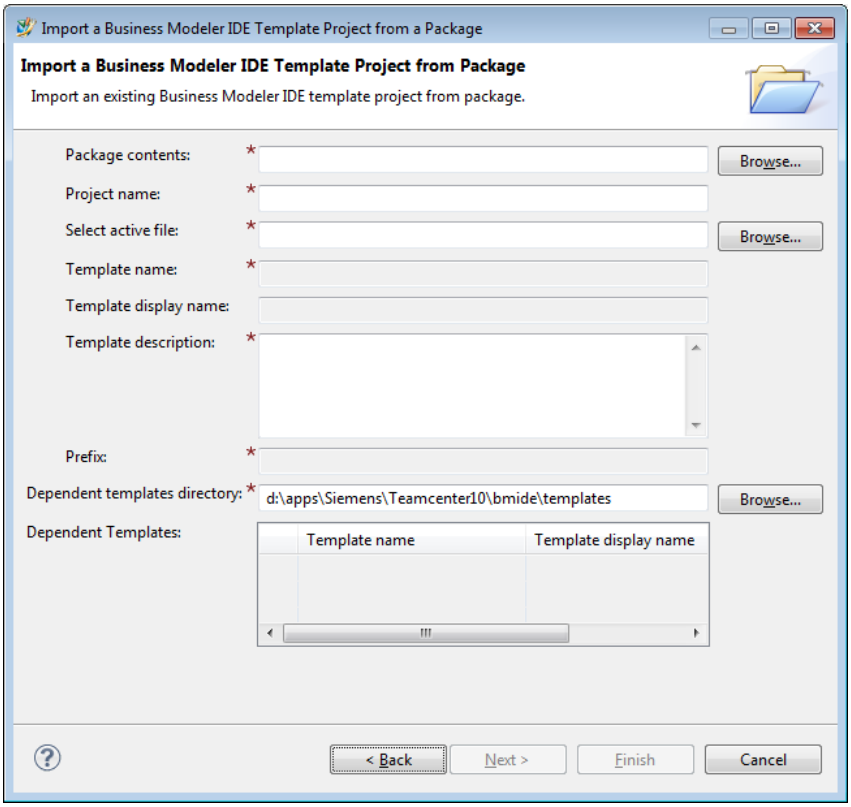
   The Import wizard runs.

2. In the **Select** dialog box, choose **Business Modeler IDE→Import a Business Modeler IDE Template Package**.



3. Click **Next**.

© 2025 Siemens



4. Perform the following steps in the **Import Business Modeler IDE Template Package** dialog box:

   a. Click the **Browse** button to the right of the **Package contents** box to choose the folder that contains the package. This can be a directory on a mapped drive or a directory on your computer.

   b. The name of the package appears in the **Project name** box. Change the name if you want to create a new project based on the contents of the package.

   c. Click the **Browse** button to the right of the **Select active file** box to select the extension file to receive the imported template package data.

   d. Type a description of the template in the **Template description** box.

   e. Click the **Browse** button to the right of the **Dependent templates directory** box to choose the directory where the dependent template XML files are stored. The templates directory is created when you install the Business Modeler IDE.

   f. Click **Finish**.

      If a package from a previous version is being imported, a dialog asks if you want to migrate the package to the current data model format.

      The wizard imports the package and displays it in views.

© 2025 Siemens

5. Create extensions using the new project as you would a project you created yourself.

Import a Business Modeler IDE template project

A project stores all extensions made against the data model in XML files. You can import a project from another Business Modeler IDE installation. Importing also migrates the template to the latest version of the data model.

Note:

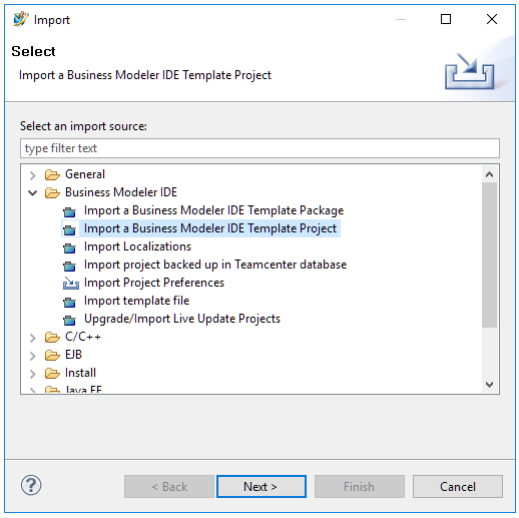
- Two or more Business Modeler IDE installations cannot point to the same project unless they are both using a source control management (SCM) system. Import the project only after you have created a view in the SCM that contains the project to be imported.
- You cannot use the following procedure to import a Composite Software Project. Instead, use the **File→Import** command with the **General→Existing Projects into Workspace** option.

1. Map a drive to the project directory on another computer, or copy the project directory to your own computer.

2. Choose **File→Import**.

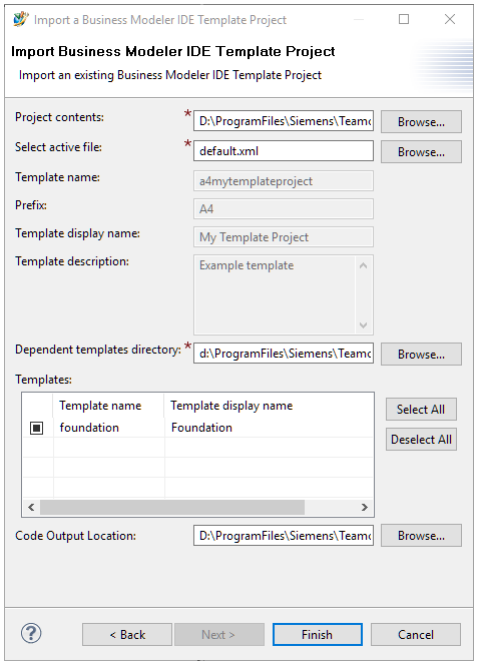
   The Import wizard runs.

3. In the **Select** dialog box, choose **Business Modeler IDE→Import a Business Modeler IDE Template Project**.



4. Click **Next**.

© 2025 Siemens



5. In the **Import Business Modeler IDE Template Project** dialog box, enter parameter values for the project.

Parameter	Description
Project contents	The folder that contains the project. This can be a directory on a mapped drive, or can be a directory that you have already copied to your computer.
Select active file	The extension file to make active after the project is imported. During normal extension work, you **set the active file** to hold the extensions.
Dependent templates directory	The directory where the dependent template XML files are stored. The templates directory is created when you install the Business Modeler IDE.
Templates	The templates to load for the project.
Code Output Location	The output directory for stub files generated when doing codeful customization.

6. Click **Finish**.

If a project from previous release is being imported, than a dialog asks whether you want to migrate the project to the current data model format.

The wizard imports the project and displays it in views.

© 2025 Siemens

7. Create extensions using the new project just as you would a project you created yourself.

> Note:
>
> The project files remain in the original location. A link is created from your workspace to the project folder. Any new extension files created against this project are added to the original location.

Import a backed-up project

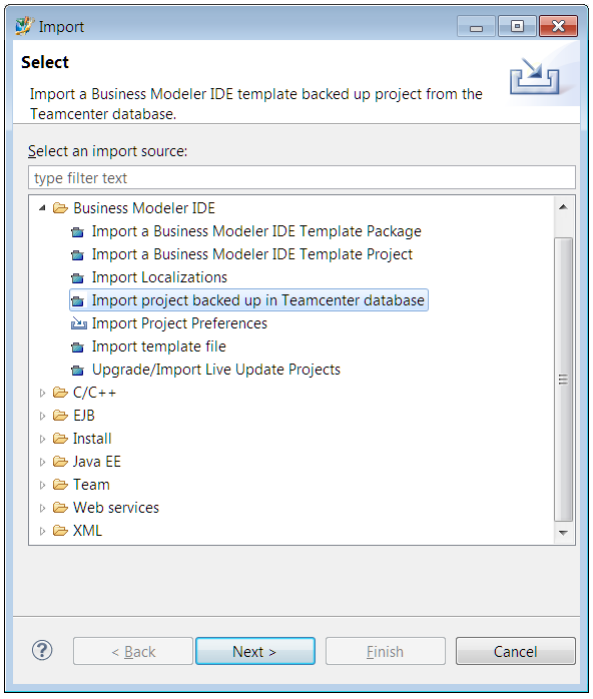
You should regularly **back up your project** in the database to ensure against data loss. To back up your project, right-click the project and choose **Properties→Teamcenter→Project Backup**.

To restore the project later, you can import it.

1. Choose **File→Import**.

   The Import wizard runs.

2. In the **Select** dialog box, choose **Business Modeler IDE→Import a project backed up in Teamcenter database**.

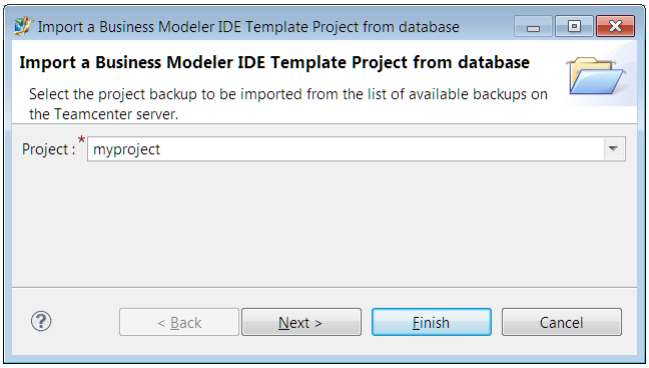


3. Click **Next**.

4. In the **Teamcenter Login** dialog box, log on to the database server and click **Next**.

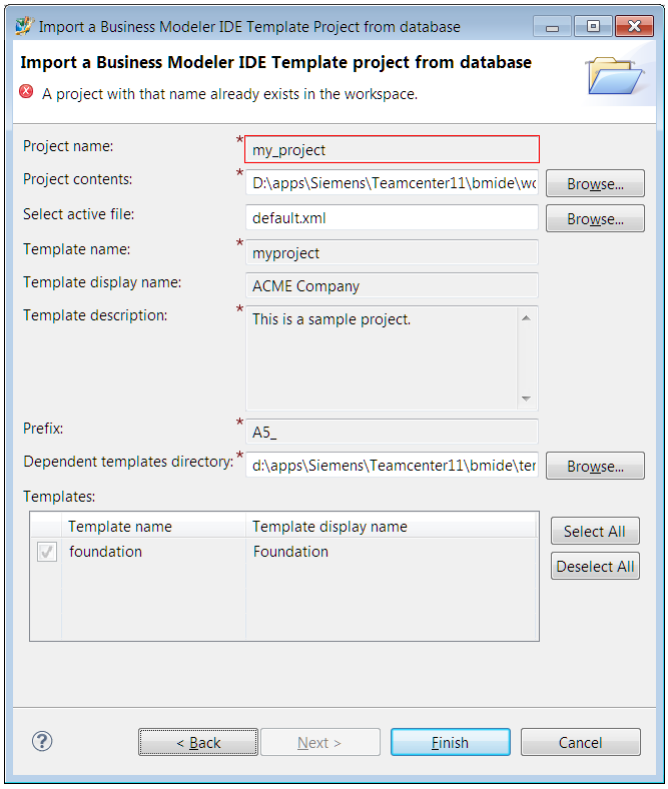
© 2025 Siemens

The following dialog box is displayed.



5. Click the arrow in the **Project** box to choose the backed-up project to import. Click **Next**.

   The following dialog box is displayed.



6. Perform the following steps in the **Import a Business Modeler IDE Template project from database** dialog box:

   a. Click the **Browse** button to the right of the **Project contents** box to choose the workspace location to import to.

© 2025 Siemens

b. Click the arrow in the **Select active file** box to choose the extension file to make active after the project is imported.

During normal extension work, you **set the active file** to hold the extensions.

c. Click the **Browse** button to the right of the **Dependent templates directory** box to choose the directory where the dependent template XML files are stored. The templates directory is created when you install the Business Modeler IDE.

d. Click **Finish**.

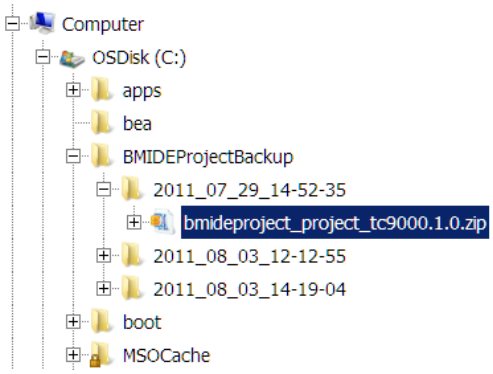
The wizard imports the project and displays it in views.

Alternative methods to restore a project

If the preceding procedure for importing the project backup does not work, you can try the following alternative methods:

- Local backup

  The backed-up projects are saved on your hard disk (by default in the **C:\BMIDEProjectBackup** directory).

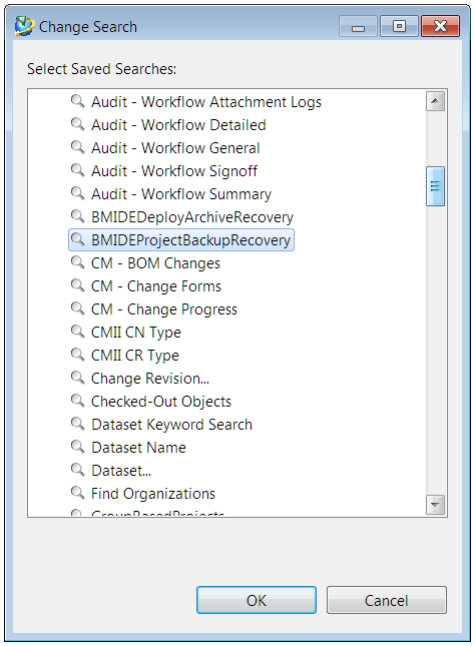


  Unzip the retrieved project backup ZIP file and **import the project into the Business Modeler IDE**.

- Database backup

  1. To locate backed-up project ZIP files in the database, in the rich client use the **BMIDEProjectBackupRecovery** saved query.

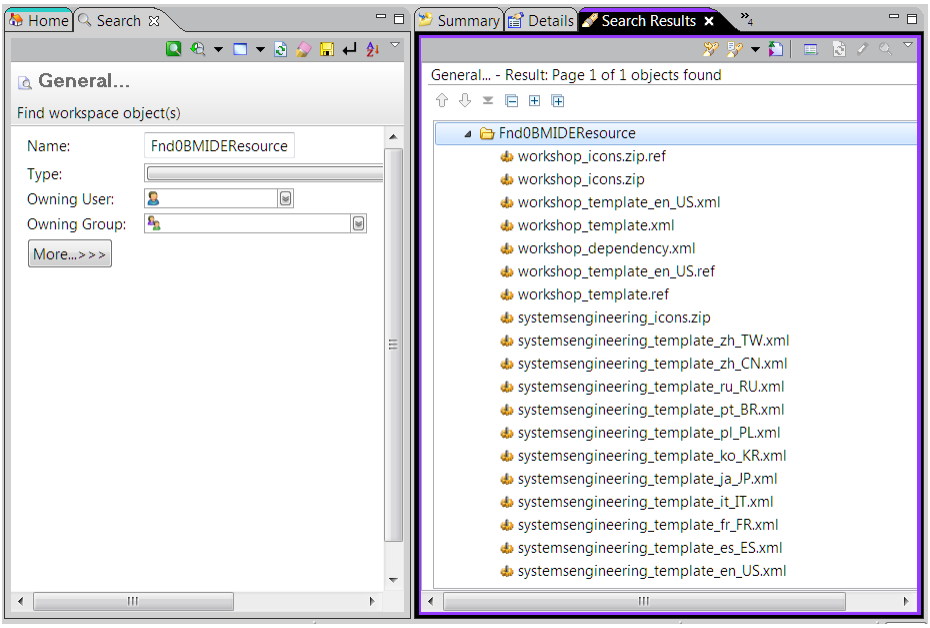
© 2025 Siemens



2. In the search results, right-click the project archive file, choose **Named References**, select the file, and click **Download**.

- Resource dataset files

  Additional Business Modeler IDE resource dataset files are stored in the **Fnd0BMIDEResource** folder. You can also use these files to restore your Business Modeler IDE environment if needed.



- **manage_model_files** utility

© 2025 Siemens

Use the **manage_model_files** utility to retrieve projects backed up in the database.

Import project preferences

You can import the following characteristics from another project to your project:

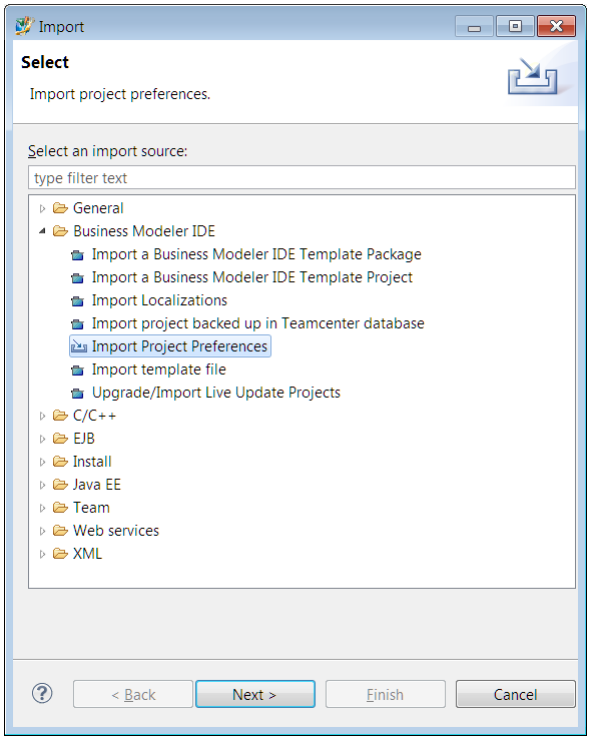
- The contents of the **Favorites** folder
- Filter settings that hide COTS elements or folders

Before importing these preferences, you must first **export** them from the source project using the **Export project preferences** wizard. Then you can import the resulting file.

1. Choose **File→Import**.

   The Import wizard runs.

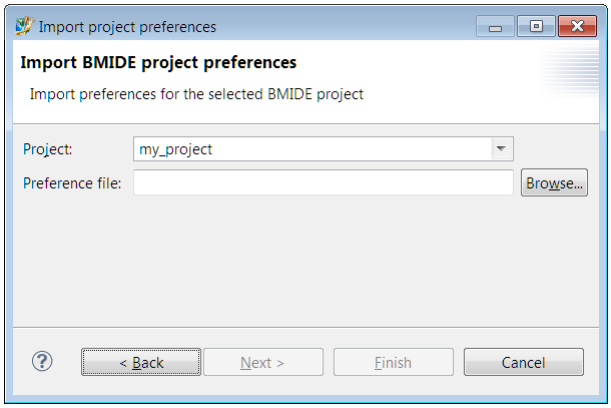
2. In the **Select** dialog box, choose **Business Modeler IDE→Import project preferences**.



3. Click **Next**.

   The following dialog box is displayed.

© 2025 Siemens



4. Click the **Browse** button to the right of the **Preference file** box.

5. Browse to the location of the preference file and select it, for example, **project_preferences.xml**.

6. Click **Finish**.

   The file is imported and its preferences are added to your project.

Import a template file

You can import the contents of a template file into a Business Modeler IDE project. The data model elements in the template file are written to an **extension file**.

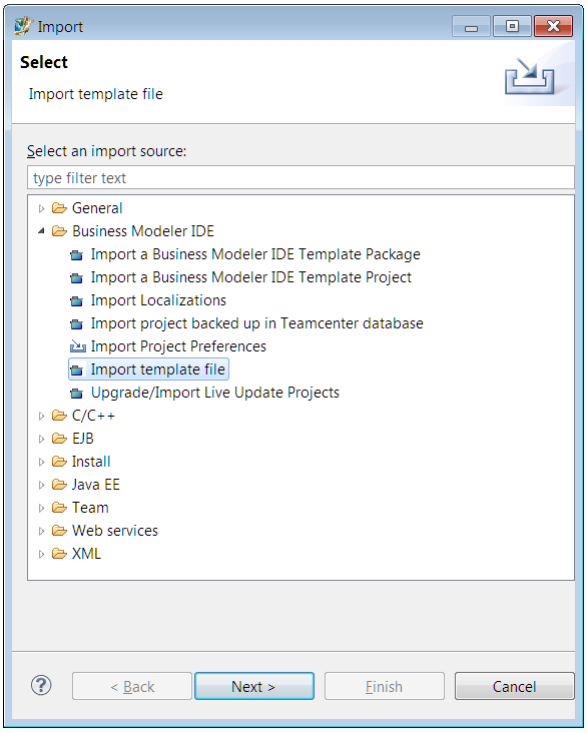
This is useful when you want to import a template file to your project. You can also import an extension file from one project template that you want to share with another project template.

1. Choose **File→Import**.

   The Import wizard runs.

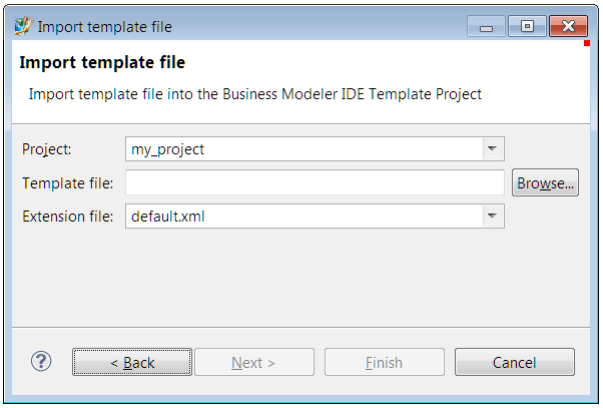
2. In the **Select** dialog box, choose **Business Modeler IDE→Import template file**.

© 2025 Siemens



3. Click **Next**.

   The **Import template file** dialog box is displayed.



4. Perform the following steps in the **Import template file** dialog box:

   a. In the **Project** box, select the project into which you want to import the template file.

   b. Click the **Browse** button to the right of the **Template file** box to choose the XML model file to be imported, for example, a template XML file or an extension file.

© 2025 Siemens

> Note:
>
> This wizard does not migrate the file to the latest data model format. You must use a template file that was created using the latest release XML format, or that has already been migrated to this format.

c. Click the arrow in the **Extension file** box to choose the extension file in your project into which the model elements are to be placed.

During normal extension work, you set the active file to hold extensions.

d. Click **Finish**.

The data model is imported into the extension file.

5. To verify the model is imported, browse for new data model objects in the Business Modeler IDE views.

To see the data model in the extension file, access the **Project Files** folder, expand the **extensions** folder, and double-click the extension file to open it in an editor view.

> Caution:
>
> Never manually edit the XML files; this may corrupt data.

Import a live update project

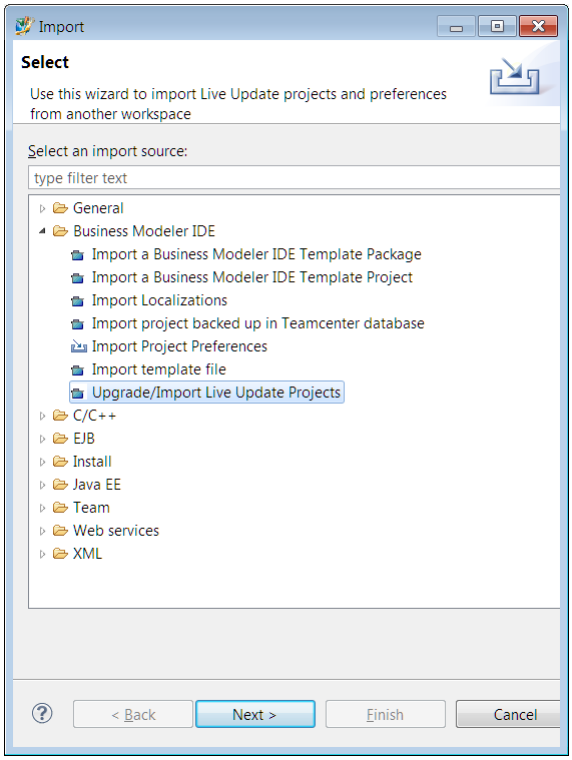
A **live update** data project holds only data to be deployed to a running production server. Create one live update project for each custom template enabled to receive live updates on the server.

You can import an existing live update project into your workspace, or upgrade a live update project from a previous Teamcenter version.

1. Choose **File→Import**.

2. In the **Select** dialog box, choose **Business Modeler IDE→Upgrade/Import Live Update Project**. Click **Next**.

© 2025 Siemens



3. Click the arrow on the **Installation** box to select the Business Modeler IDE installation from which you want to import live update projects.

> Note:
>
> The paths to the installations are stored in a **.bmide** file in the *USER_HOME* directory, for example, **C:\users***user-name*.

4. Click **Finish**.

Export project preferences

For your convenience in applying the same view preferences to multiple projects, you can export the following characteristics of your project to a file, and subsequently **import the project preferences file** to another project.

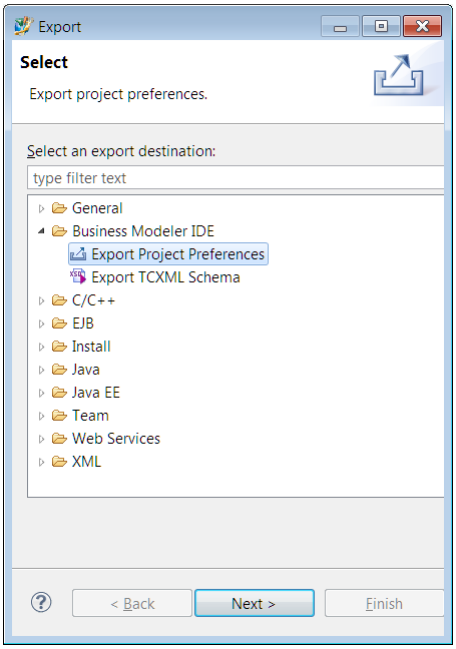
- The content of the **Favorites** folder
- Filter settings that hide COTS elements or folders

Procedure

1. Choose **File**>**Export**.

© 2025 Siemens

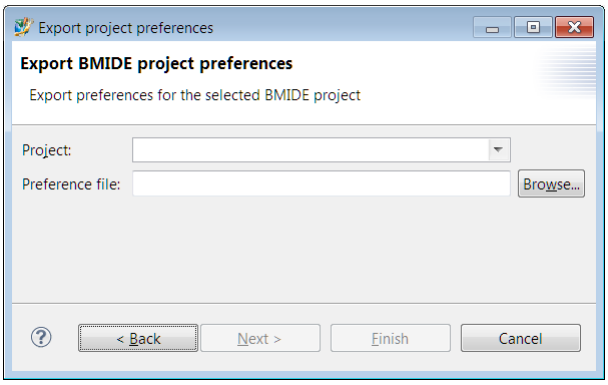
2. In the **Select** dialog box, choose **Business Modeler IDE>Export project preferences**.



3. Click **Next**.

   In the **Select** dialog box, choose **Business Modeler IDE→Export project preferences**.

   The **Export BMIDE project preferences** dialog box is displayed.



4. To the right of the **Preference file** box, click **Browse**.

5. Browse to the location where you want to save the exported file, and in the **File name** box, type the name you want the file to have, for example, **project_preferences.xml**. Include the **.xml** extension on the file name.

   Click **Save**.

   The path and file name are displayed in the **Preference file** box.

© 2025 Siemens

6. Click **Finish**.

The export file is created in the location you specified.

Export a TC XML schema file

You can generate a TC XML schema file for your project from the Business Modeler IDE. Schema files contain data model structure in a predefined file format (**.xsd**). The TC XML schema file is used when you map data for transferring from one PLM system to another (for example, Teamcenter Enterprise to Teamcenter).

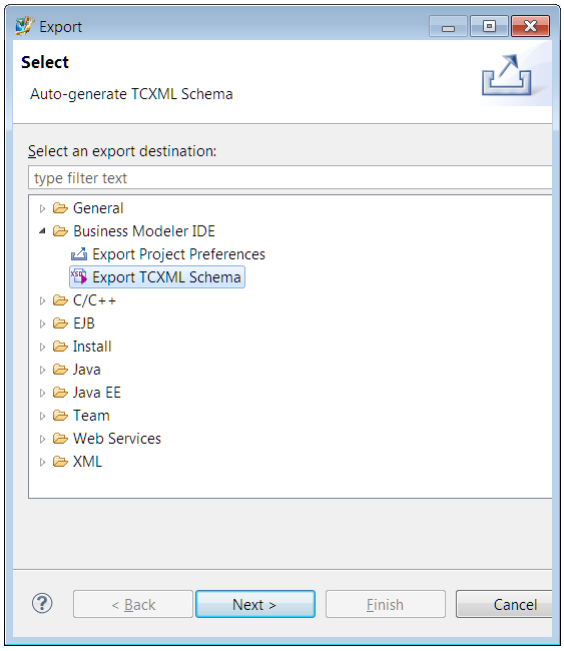
The TC XML schema is autogenerated at installation and upgrade, and the resulting **TCXML.xsd** file is placed in the *TC_DATA* directory on the Teamcenter server.

> Note:
>
> This topic describes how to generate a Teamcenter schema file. To generate an Teamcenter Enterprise schema file, use the Administration Editor. After generating the schema file, verify that the structure of the resulting file is correct. For example, if some input items are missing, they are replaced with **change_me**. Look for instances of **change_me** in the schema file and change them as appropriate. If you do not, you may encounter errors when you use this file and attempt to create a project or a factor using the Mapping Designer application.
>
> Use the **Dry Run for Export to Remote Site** command in the Teamcenter Enterprise classic client to generate a sample XML export file of the exportable objects based on the Teamcenter Enterprise schema. This file can be used as a starting point for mapping Teamcenter Enterprise objects to Teamcenter and vice versa.

1. Choose **File→Export**.

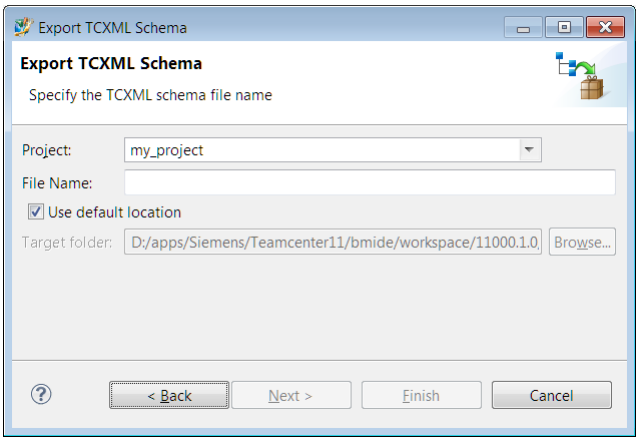
2. In the **Select** dialog box, choose **Business Modeler IDE→Export TCXML Schema**.

© 2025 Siemens



3. Click **Next**.

   The **Export TCXML Schema** dialog box is displayed.



4. Perform the following steps in the **Export TCXML Schema** dialog box:

   a. Click the arrow in the **Project** box to select the project for which you want to generate the schema file.

   b. In the **File Name** box, type the name you want the schema file to have. (Include the **.xsd** extension on the file name.)

   c. Leave the **Use default location** check box selected if you want the file to be placed in your workspace in the **output\tcplmxml** folder under your project.

© 2025 Siemens

If you want to change the folder where the file is created, clear the **Use default location** box, and click the **Browse** button to the right of the **Target folder** box to choose the folder where the file is to saved.

d. Click **Finish**.

The file is saved in the target folder.

By default, the TC XML file is saved to the **output\tcplmxml** folder under your project.

> Note:
>
> You can also locate the file in your workspace on your system. To find the *workspace* location, in the **Advanced** perspective, choose **File→Switch Workspace**. For example, on a Windows system, it is saved by default to:
>
> *install-location*`\bmide\workspace\`
> *version*`\`*project*`\output\tcplmxml`

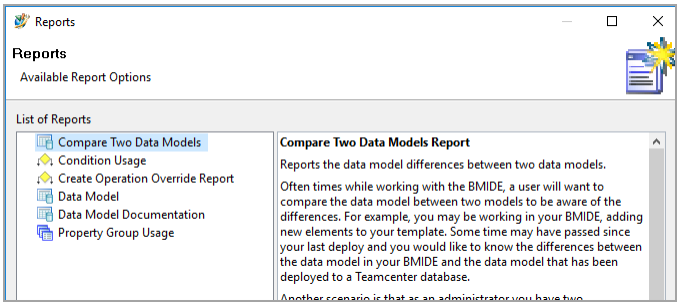
Now you can use the schema file when mapping data model between Teamcenter Enterprise and Teamcenter.

Data model reports

Run data model reports

Create reports by choosing **BMIDE**>**Reports**.

The **Reports** dialog box is displayed.



The output of the reports is in HTML format. To view HTML reports in the Business Modeler IDE, you must have a web browser installed.

You can create the following kinds of reports:

- **Compare Two Data Models**

© 2025 Siemens

This report provides the data model differences between two data models. Often when working with the Business Modeler IDE, you want to compare the data model between two models.

You can also generate this report using the **bmide_generate_compare_report** utility.

- **Condition Usage**

  This report provides the details of a condition and all model elements that use the condition.

  You can also generate this report using the **bmide_generate_condition_report** utility.

- **Create Operation Override Report**

  This report appears only after you **upgrade a project** to the latest data model.

  This report provides the details of custom business objects where the create operations are overridden.

- **Data Model**

  This report provides the details of a given category of model elements.

  You can also generate this report using the **bmide_generate_datamodel_report** utility.

- **Data Model Documentation**

  This report generates a multiple HTML-paged report of all data model elements for all templates in the data model.

  You can also generate this report using the **bmide_generate_datamodel_doc_report** utility.

  You can also obtain a report of the COTS data model in the *Teamcenter Data Model Report* provided in the Teamcenter documentation distribution image. Expand the *DataModelReport.zip* file to a local directory and open *help\en_US\custom\DataModelReport\index.html*.

- **Property Group Usage**

  This report generates a list of properties assigned to property groups used in **propagation rules**. Propagation rules allow you to propagate data from one business object type to another.

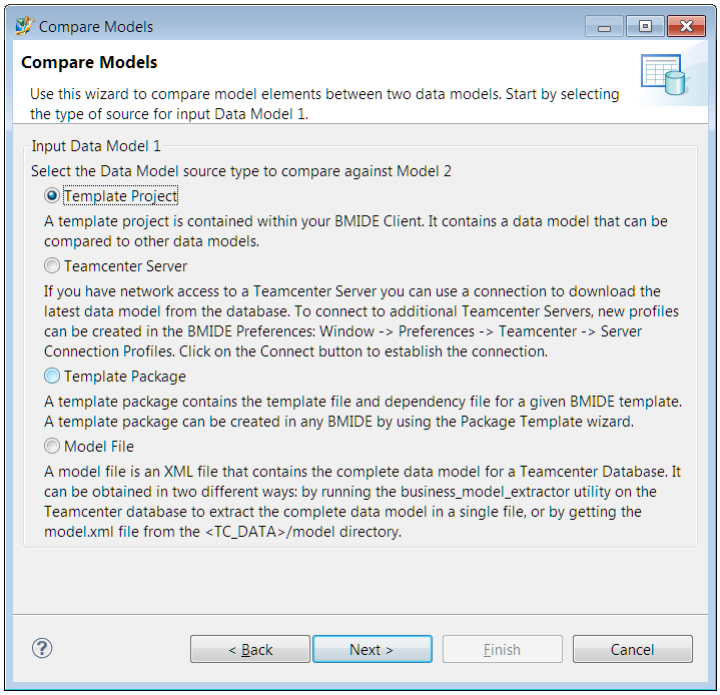
Run a report comparing the data model from two different sources

1. Choose **BMIDE→Reports**.

2. Select **Compare Two Data Models** and click **Next**.

© 2025 Siemens

The **Compare Models** dialog box is displayed.



3. In the **Input Data Model 1** box, select the source of the first data model to compare and click **Next**:

   - **Template Project**
   - **Teamcenter Server**
   - **Template Package**
   - **Model File**

4. Locate the first data model to compare. The dialog box that appears next is dependent upon the input source you selected:

   - **Template Project**

     Select the project and templates to include in the report.

   - **Teamcenter Server**

     Type the user name and password for **server access** and click **Connect**.

   - **Template Package**

     Browse for a **template package**.

© 2025 Siemens

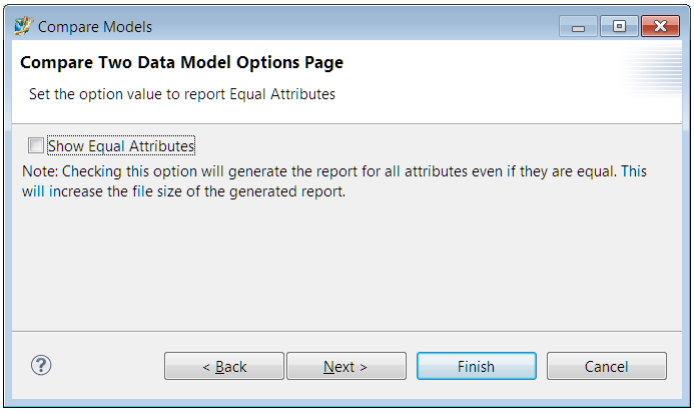
- **Model File**

  Select the *TC_DATA***model\model.xml** model file.

5. In the **Input Data Model 2** box, select the source of the second data model to compare and click **Next**:

6. As you had done earlier for the first data model, locate the second data model and click **Next**.

   The **Compare Two Data Model Options Page** dialog box is displayed.

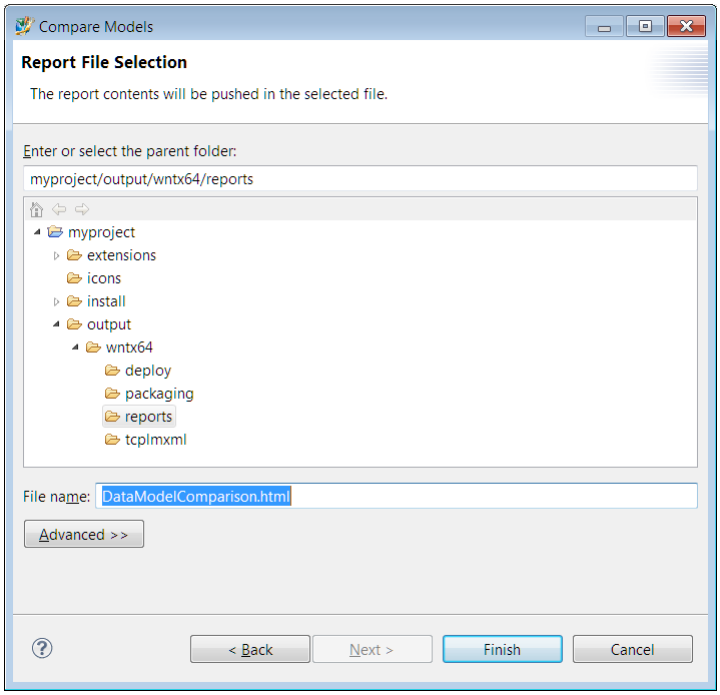


7. On the **Compare Two Data Model Options Page**, leave the **Show Equal Attributes** check box clear if you want the report to show only differences between the data models, or select it if you want the report to show all the elements that are the same between the two data models.

   Click **Next**.

   The **Report File Selection** dialog box is displayed.

© 2025 Siemens



8. In the **Report File Selection** dialog box, select the *project***output****reports** folder as the location where to write the report, and in the **File Name** box, type a name for the report.

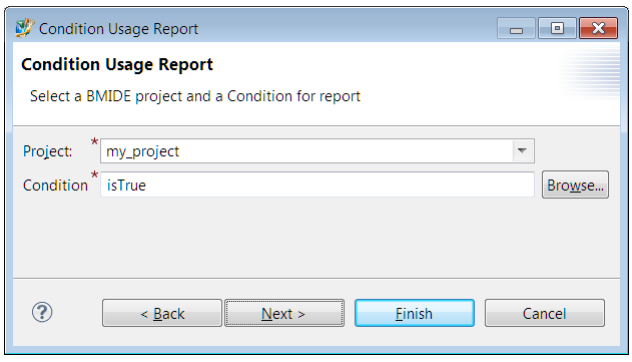
9. Click **Finish**.

Run a condition usage report

1. Choose **BMIDE→Reports**.

2. Select **Condition Usage** and click **Next**.

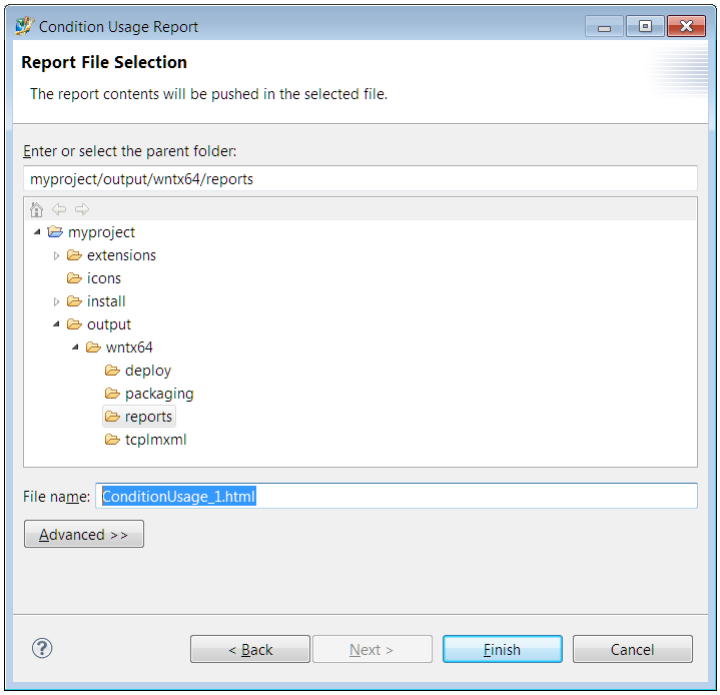
   The **Condition Usage Report** dialog box is displayed.



3. In the **Condition Usage Report** dialog box, select the condition to report, and the project it resides in. Click **Next**.

© 2025 Siemens

The **Report File Selection** dialog box is displayed.



4. In the **Report File Selection** dialog box, select the *project***output****reports** folder as the location where to write the report, and in the **File Name** box, type a name for the report.

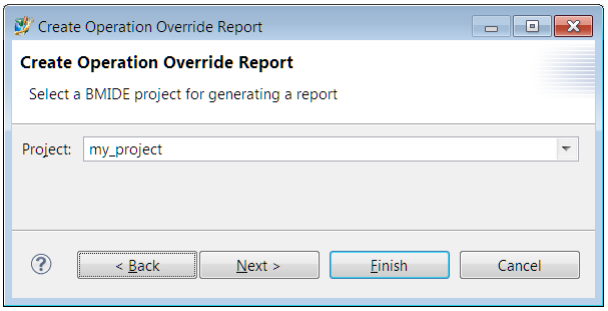
5. Click **Finish**.

Run a report on create operation overrides

1. Choose **BMIDE→Reports**.

2. Select **Create Operation Override Report** and click **Next**.

   The Create Operation Override Report dialog box is displayed.



3. In the **Project** box, select the project on which you want to run the report and click **Next**.

© 2025 Siemens

4. In the **Report File Selection** dialog box, select the *project***output****reports** folder as the location where to write the report, and in the **File Name** box, type a name for the report.

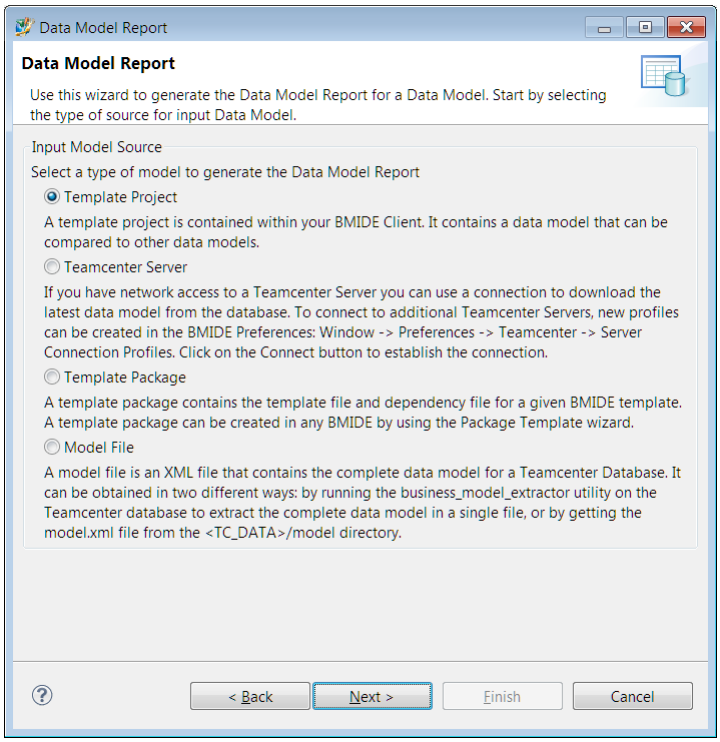
5. Click **Finish**.

Run a report of individual types of data model

1. Choose **BMIDE→Reports**.

2. Select **Data Model** and click **Next**.

   The **Data Model Report** dialog box is displayed.



3. In the **Input Model Source** box, select the source of the data model and click **Next**:

   - **Template Project**
   - **Teamcenter Server**
   - **Template Package**
   - **Model File**

4. The dialog box that appears next is dependent upon the input source you selected:

© 2025 Siemens

- **Template Project**

  Select the project and templates to include in the report.

- **Teamcenter Server**

  Type the user name and password for **server access** and click **Connect**.

- **Template Package**

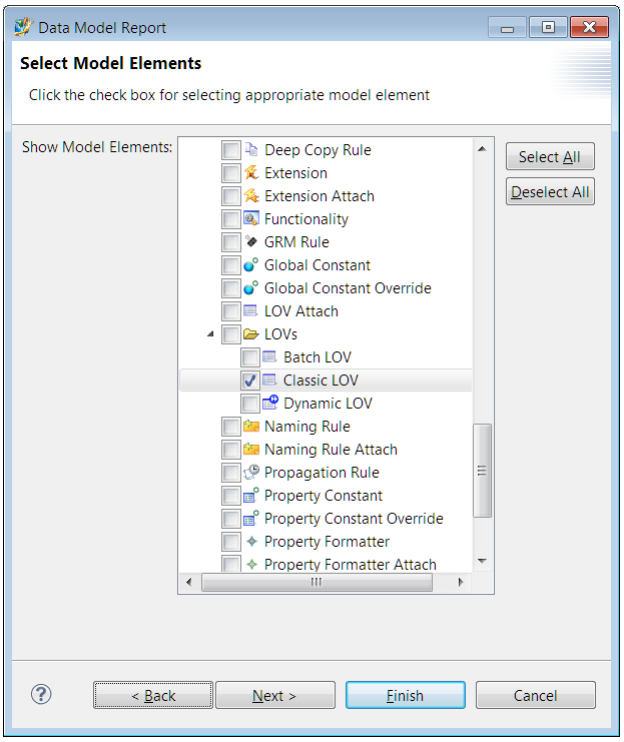
  Browse for a **template package**.

- **Model File**

  Select the *TC_DATA***model\model.xml** model file.

5. Click **Next**.

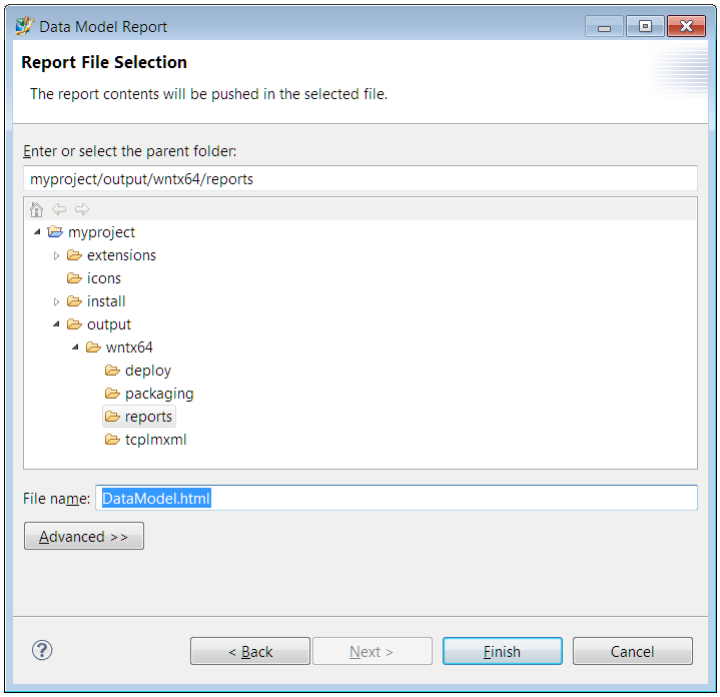
   The **Select Model Elements** dialog box is displayed.



6. In the **Select Model Elements** dialog box, select the model elements you want to appear in the report and click **Next**.

   The **Report File Selection** dialog box is displayed.

© 2025 Siemens



7. In the **Report File Selection** dialog box, select the *project***output****reports** folder as the location where to write the report, and in the **File Name** box, type a name for the report.

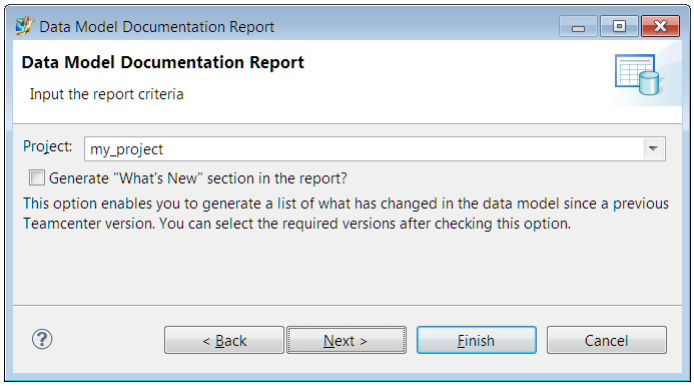
8. Click **Finish**.

Run a report of all data model

1. Choose **BMIDE→Reports**.

2. Select **Data Model Documentation** and click **Next**.

   The **Data Model Documentation Report** dialog box is displayed.



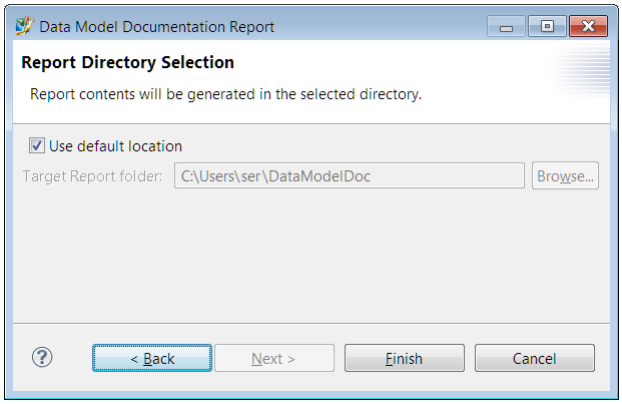
3. In the **Project** box, select the project on which you want to run the report.

© 2025 Siemens

4. Select the **Generate "What's New" section in the report?** check box to include a section showing the new data model added since previous versions of Teamcenter. Select the previous versions.

5. Click **Next**.

   The **Report Directory Selection** dialog box is displayed.



6. Clear the **Use default location** box to specify where to save the report.

7. Click **Finish**.

   The report is generated at the specified location.

Run a property group usage report

A property group usage report lists the:

- **Propagation rules** that use the selected property group.
- Properties assigned to the selected property group.

> Note:
>
> To assign a property group to a property, on the source business object type select the property and then select the **Fnd0PropagationGroup** property constant.

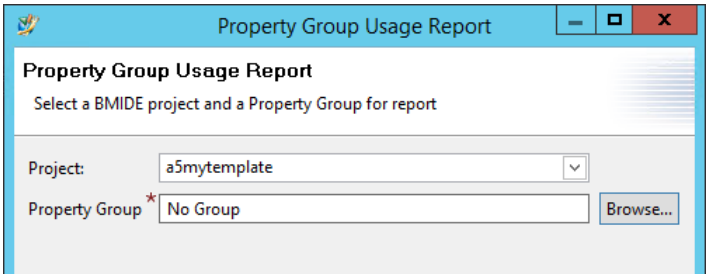
Procedure

1. Choose **BMIDE**>**Reports**.

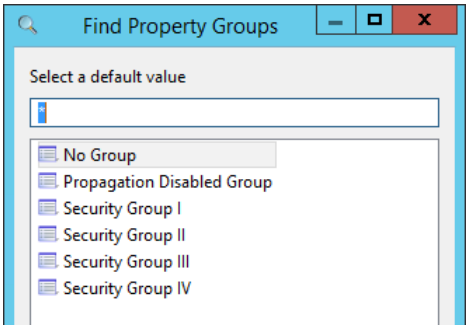
2. Select **Property Group Usage** and click **Next**.

   The **Property Group Usage** dialog box is displayed.

© 2025 Siemens



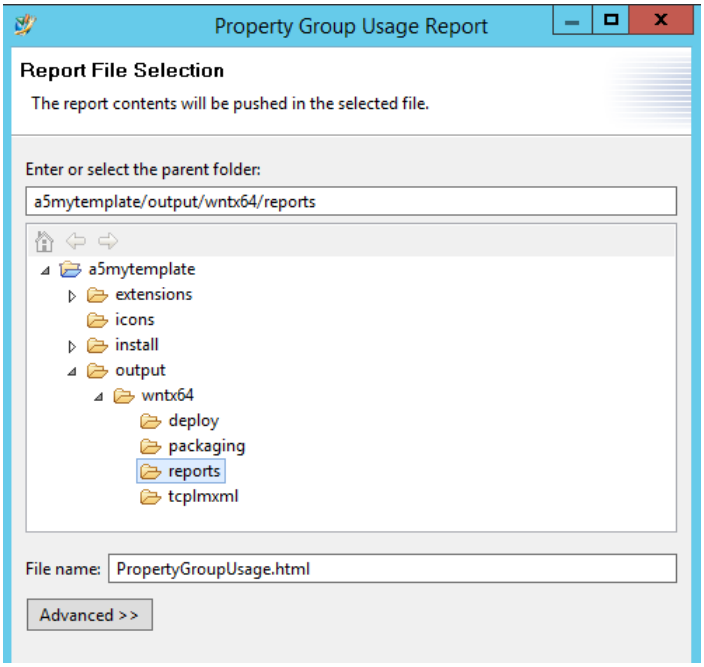
3. Click **Browse** and select the property group on which to run the report:



The list of property groups is defined in the **Fnd0PropertyGroupNames** list of values (LOV).

4. Click **Next**.

The **Report File Selection** dialog box is displayed.



© 2025 Siemens

5. In the **Report File Selection** dialog box, select the *project***output****reports** folder as the location for the report, and in the **File Name** box, type a name for the report.

6. Click **Finish**.

   The report is generated at the specified location and is displayed in a new view.

Graphically represent the data model in the UML editor

Introduction to the UML editor

UML (Unified Modeling Language) is a commonly used method to graphically represent data models. The **UML editor** allows you to graphically view and change the data model for business objects and classes. You can do much of your data model extension work directly from the UML editor.

> Note:
>
> UML files only store the names and relative positions of the data model. The definitions of data model created with the UML editor are not stored in the UML files. They are stored in the project's template XML files.

Open a class or business object in the UML editor

1. Browse to the business object or class you want to work with. To search for a business object or class, you can click the **Find** button at the top of the view.

   Item is the most common business object or class with which you work.

2. Right-click the business object or class and choose **Open In UML Editor**.

   The business object or class appears in the UML editor as a box. You can also drag other business objects into the UML editor.

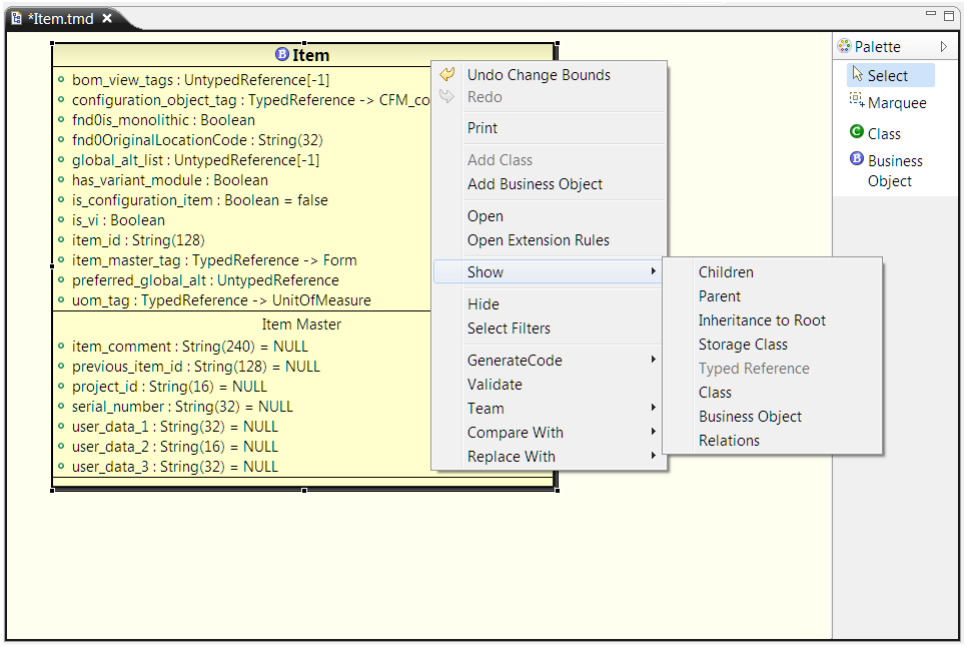
3. To learn the basics of the UML editor, perform the following steps:

   a. Work with the shortcut menu.

      Right-click the name of the business object or class that appears at the top of the box.

      A shortcut menu appears.

© 2025 Siemens



Try the following tasks:

- To learn how to display hierarchy, choose **Show** and choose **Children**, **Parent**, or **Inheritance to Root**. If you are viewing a business object, also choose **Show→Storage Class** to see the class where the business object data is stored. To filter out what is displayed, choose **Select Filters**.
- To create a new business object or class, choose **Add Business Object** or **Add Class**.
- To undo actions, choose **Undo**.
- To see operations on a business object, choose **Open Extension Rules**.
- To see the relationships between business objects, drag business objects into the UML editor, press the Ctrl or Shift key and click multiple business objects to select them, and right-click and choose **Show→Relations**.

b. Work with the palette.

Click the arrow at the top of the **Palette** bar docked on the right of the UML editor. The palette expands to display a series of buttons. Try the following tasks:

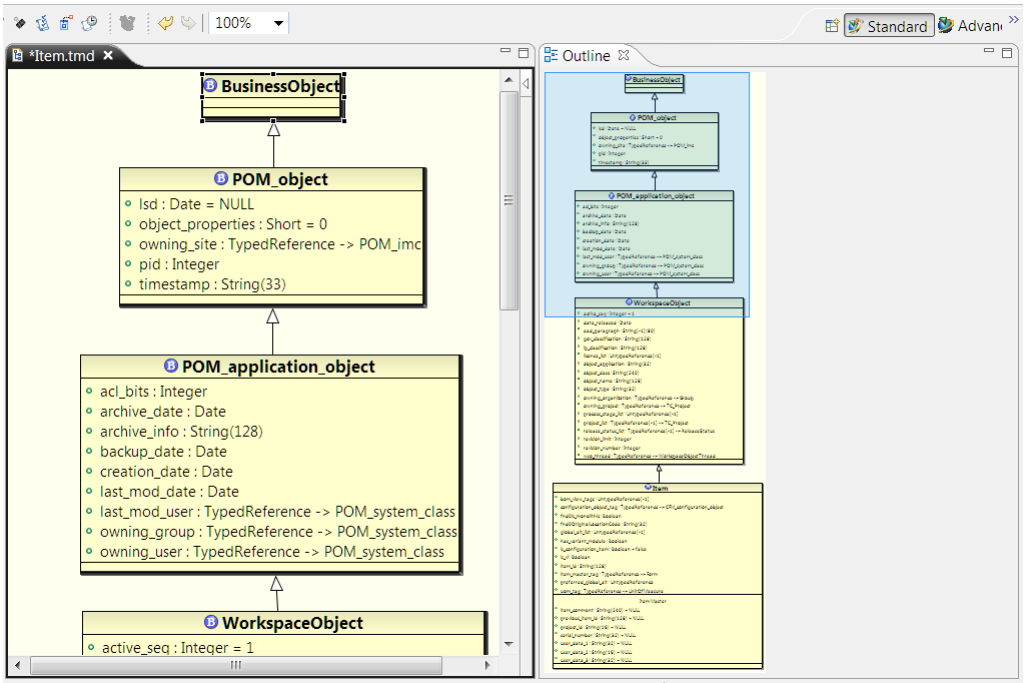
- To select individual business objects or classes, click the **Select** button.
- To select a group of business objects or classes, click the **Marquee** button and drag a square around a grouping you want to select.
- To create a new business object or class, drag **Class** or **Business Object** into the UML editor.

© 2025 Siemens

c. Work with the **Outline** view.

The **Outline** view displays a thumbnail of the UML editor with a gray box indicating what is currently displayed.

- Drag the gray box across the **Outline** view to change what is displayed in the UML editor.
- Click in the UML editor and click the zoom control in the toolbar at the top of the window to change the view from 100 percent to a different size.



d. View the UML file.

The data is saved in a file with a **.tmd** suffix as displayed on the tab at the top of the UML editor. To open this **.tmd** file later and view it in the UML editor, access the **Project Files** folder and double-click the **.tmd** file in the **UML Diagrams** folder. Saving the diagram does not save the data model; you must choose **BMIDE→Save Data Model**.

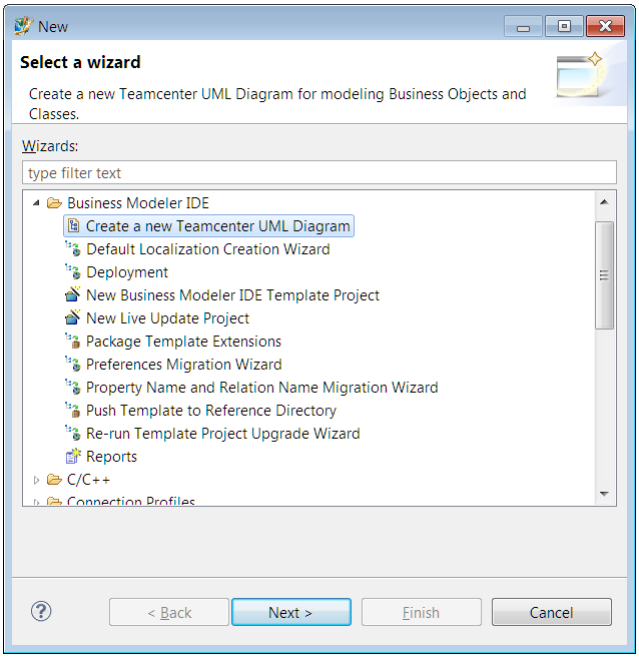
4. To save the changes to the data model, choose **BMIDE→Save Data Model**, or click the **Save Data Model** button on the main toolbar.

5. After you make changes to the data model using the UML editor, you can deploy your changes to the test server. Choose **BMIDE→Deploy Template** on the menu bar, or select the project and click the **Deploy Template** button on the main toolbar.

Create a new UML diagram

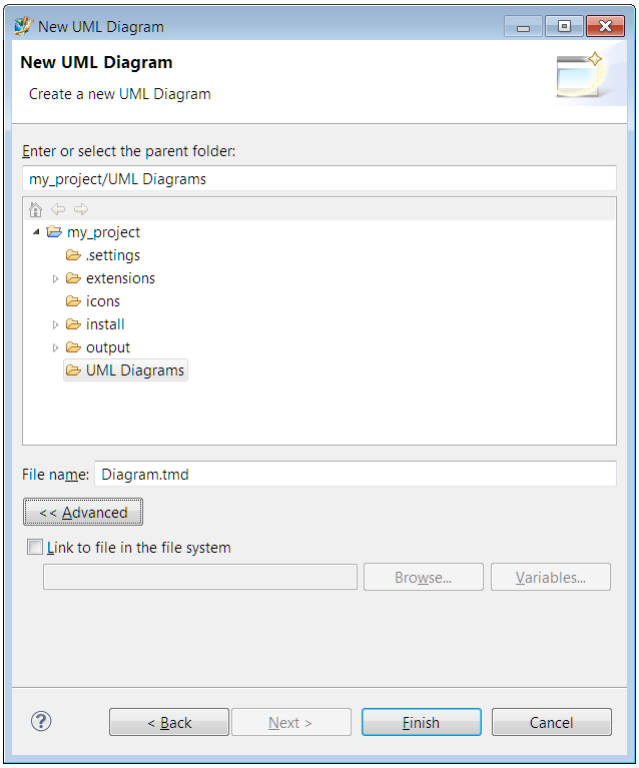
1. Choose **File→New→Other→Business Modeler IDE→Create a new Teamcenter UML diagram**

© 2025 Siemens



2. Click **Next**.

   The New UML Diagram wizard runs.



3. Perform the following steps in the **New UML Diagram** dialog box:

© 2025 Siemens

a. Select the folder where you want to save the diagram, for example, **UML Diagrams**.

b. In the **File name** box, type the name you want to give to the diagram. The file has a **.tmd** file suffix.

c. Click **Finish**.

The UML editor appears.

4. To work in the UML editor, drag and drop business objects or classes into the new diagram. Right-click in the view and choose menu commands.

You can also create business objects or classes by dragging the **Class** or **Business Object** icons from the UML editor palette into the editor.

5. To save the UML diagram, click the **Save** button on the main toolbar. The diagram is saved in a file with a **.tmd** suffix as displayed on the tab at the top of the UML editor.

To open this **.tmd** file later and view it in the UML editor, access the **Project Files** folder and double-click the **.tmd** file in the **UML Diagrams** folder.

> Note:
>
> Saving the diagram does not save the data model. To save the data model, choose **BMIDE→Save Data Model**, or click the **Save Data Model** button on the main toolbar.

© 2025 Siemens

10. Creating, deploying, and packaging templates

Introduction to templates

A *template* is an XML file that contains the data model for an application, also known as a solution. For example, the **foundation_template.xml** file contains the data model for the Foundation solution, the base Teamcenter application. When you use the Business Modeler IDE to create a data model, the new data model is rolled up into a template.

To quickly deploy your template to a Teamcenter test server for testing purposes, you can choose **BMIDE**>**Deploy Template**. This is also known as *live update*. You can also use live update to send operational data such as LOVs and rules to a production server.

To package your data model into a template for installation to a Teamcenter production server, you can choose **BMIDE**>**Generate Software Package**.

Templates can exist in three locations:

- *install-location***bmide\templates**

  This folder stores templates that are used for reference only within the Business Modeler IDE. The templates in this location are used when you create a project, and supply the base model for your data model extensions. This folder is of interest only to the Business Modeler IDE and does not affect your database status.

- *TC_DATA***model**

  This folder represents the current status of your database. Templates are placed here when you use Teamcenter Environment Manager (TEM) to install Foundation or new templates, or when you deploy templates from the Business Modeler IDE to a test server. All Business Modeler IDE utilities that update the database look here to find the templates to be applied to the database. This is a crucial folder for any installation or upgrade that makes database changes.

- *workspace-location**version**project***output\packaging**

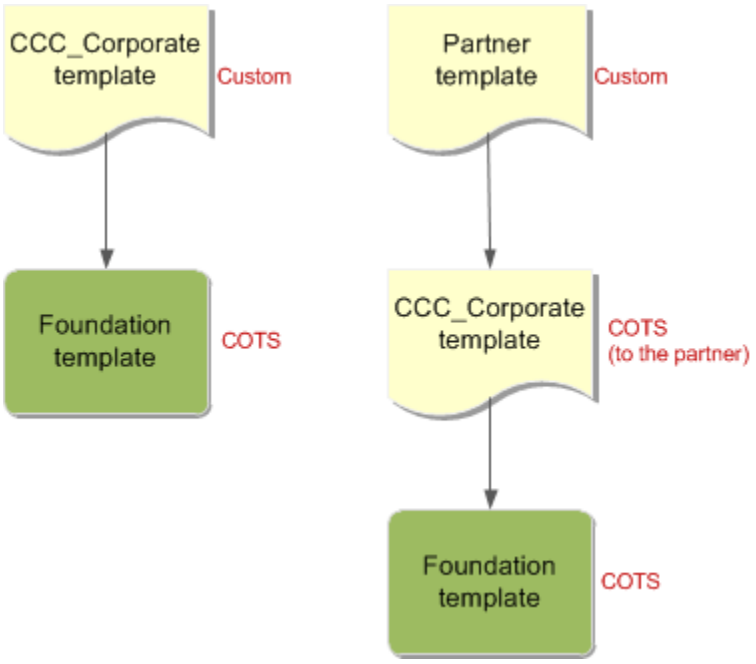
  This folder is the default location where templates are packaged by the Business Modeler IDE. Templates here are a consolidation of all data model extensions you performed in your project. Once generated, you can open the template file and dependency file and verify for correctness. If you also have an installation of Teamcenter to which you want to install your template using the TEM installer, you browse to this location to obtain the template during the installation.

© 2025 Siemens

Understanding custom versus COTS templates

Custom data model objects are those objects you create and store in a custom template. *COTS* (commercial-off-the-shelf) data model objects are the objects that your custom data model objects are dependent on. Therefore, custom templates are dependent on COTS templates. You can change or delete only the custom objects in the data model.

The Foundation template is always a COTS template. When a customer creates the **CCC_DEV** template, it is considered custom. If a customer provides their template to a partner to extend, the **CCC_DEV** template is a COTS template to the partner. The **Partner** template is considered custom to the partner.



Note:

The state of the COTS or custom templates is not stored in the database, but is determined at run time by the Business Modeler IDE.

Template process in the Business Modeler IDE

For ease of use external to Teamcenter, the data model is separated into template files that contain the objects for a Teamcenter application (also known as a solution). For example, the base Teamcenter application is the Foundation solution, and its objects are contained in the **foundation_template.xml** file.

Following is the general process for working with templates in order to extend, configure, or maintain your data model.

© 2025 Siemens

1. **Create a template project** to hold your custom data model. Load the Teamcenter application templates that you want to change.

2. In your template project, make the changes you want.

3. **Deploy your template** to a test server to verify its behavior.

4. **Package your template** for installation to a production server.

5. **Install your template** to a production server.

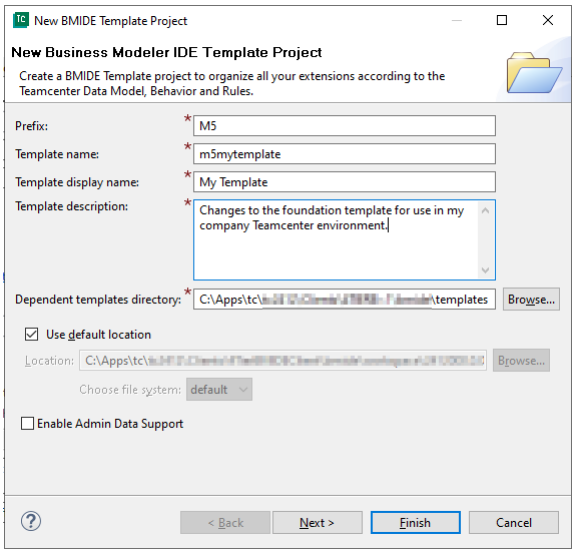
Once your template is installed, you can **Live update** the non-schema data model (such as lists of values) on the production server.

Create a Business Modeler IDE template project

To establish a container for the changes that you want to make to the Teamcenter data model, you create a template project in the Business Modeler IDE.

1. Choose **File**>**New**>**New Business Modeler IDE Template Project**.

2. In the first **Business Modeler IDE Template Project** dialog box pane, enter basic project information.



For this parameter	Do this	Remarks
Prefix	Enter a 2- to 4-character prefix.	The first character must be an uppercase letter. The second character must be a digit from 4 through 9.

© 2025 Siemens

For this parameter	Do this	Remarks
		The third and fourth characters, if entered, must also be digits from 4 through 9. To **ensure unique naming**, the prefix will be affixed to the name of all objects created in the template.
Template name	Enter a lower-case name that begins with the project prefix value.	This names the template file created when this project's **extensions are packaged for distribution**.
Template display name	Enter an easily readable name.	The display name appears for this template in Teamcenter Environment Manager when the template is listed for potential installation to an environment.
Template description	Enter a description of the project.	
Dependent templates directory	Specify the directory where the data model template XML files are stored.	Normally, the template files are in the Teamcenter bmide install location *\TR\bmide\templates*. Caution: Do not point to the templates in the data location *\TD\model*. The templates in *\TD\model* are used only for database updates, and the Business Modeler IDE client must never use the templates in that folder.
Use default location	To create the project directory within your default workspace location, select the check box.	For Linux environments, users must have permission to the workspace directory.
Location	If you want to create the project in a location other than the default, then clear the **Use default location** check box and specify another location.	While the **Use default location** check box is selected, the default workspace location is shown. You may want to specify a location other than the default if, for example, you are using a source control management (SCM) system to manage your XML source files, and the SCM cannot be configured to recognize the default location. The location must follow these rules: • The destination directory name must match the **Template name**. • The directory must not be within another project's directory.

© 2025 Siemens

For this parameter	Do this	Remarks
		• The directory must not be in the parent directory of your workspace.
Enable Admin Data Support	If you want to include admin data in your project, check this check box.	Folders will be created for you to add your admin data files like, queries, rule tree, preferences, style sheets, and so on. Two script files (**install_***template-name***.default** and *template-name***_update.xml**) are created for you to add your import and update utility commands.

3. In the **Dependent Templates** pane, select the check box for each template that you want to extend with this project.

   Templates contain the data model for Teamcenter solutions. The **Foundation** template contains the data model used for core Teamcenter functions. The **Foundation** template is always selected.

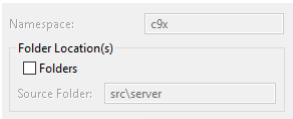
   The dependent templates are not themselves bundled into your project template. In order for your template extensions to work, your project's dependent templates must be installed on the server where you install your custom template.

4. If you want to support localization of the display names of the data model that you create in your template, then in the **Locales Selector** pane, select the languages that you want to be viewable in the Teamcenter end-user interface.

   > Caution:
   >
   > Select only the locales that your database supports. Typically, you cannot mix western locales (English, French, Spanish, and so on) with locales having different character sets (Korean, Japanese, Chinese, Russian).

   > Tip:
   >
   > If necessary, you can change the supported languages later, after the project has been created.

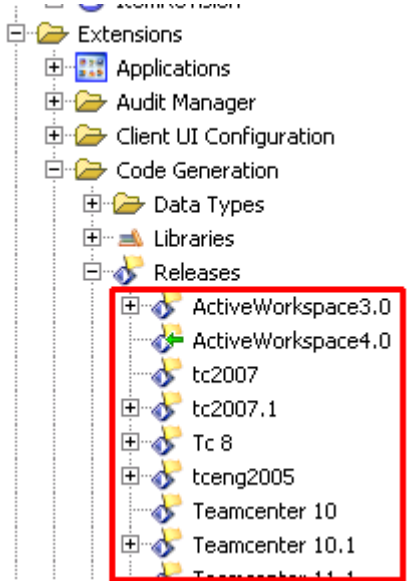
5. If you plan to write code to extend the data model and operations, then set up the location of generated source files in the **Code Generation Information** pane.



© 2025 Siemens

For this parameter	Do this	Remarks
Namespace	Enter a namespace for the code generation.	Namespace defaults to the project name. Namespaces allow for grouping code under a name to prevent name collisions. Collisions result when different libraries may use the same class names for different classes. Each template can specify a namespace associated with the C++ classes and with data types created in the template. For example, the custom classes for the Widget template might specify **Widget** as the namespace for its classes. This allows both the Foundation and Widget templates to have a class called **Item** by using their different namespaces as the differentiator.
Base Path	Specify the workspace location where generated code is placed.	For a Windows system, generated code is saved by default to: *install-location*`\bmide\workspace\` *version**project*`\output\` On Linux systems, users need to have permissions to the workspace directory.
Source Folder	Review the name of the folder that will contain your business logic source code.	The source folder is within the Base Path location. If you want store your source code in a location other than the default, then select the **Folders** check box and enter a new folder.
Enable Deprecation Policy	To support removal of obsolete objects from the project, select **Enable Deprecation Policy**	*Deprecation* means announcing that something is no longer supported and that it will be removed in a future product release.
Number of Allowed Releases before Deletion	If you have selected **Enable Deprecation Policy**, then select an option for the number of releases that must elapse past the release against which code is created before the code can be deleted from the project.	Releases are listed in the **Extensions\Code Generation\Releases** folder for a project.

© 2025 Siemens

For this parameter	Do this	Remarks
		

The **Copyright** box displays the copyright text placed into each generated file.

6. If you plan to write code to extend the data model and operations, then in the **Build Configuration Information** pane, enter information needed for C++ code generation.

For this parameter	Do this	Remarks
Teamcenter Installation	Click **Browse** and select the location where the Teamcenter server is installed.	
Compiler Home	Click **Browse** and select the location where your C++ compiler is located.	For example, if your platform is Windows and you are using Microsoft Visual Studio, browse to *compiler-install-path*\VC\bin. For information about supported C++ compilers, see Hardware and Software Certifications knowledge base article on Support Center.

7. If you plan to write code to extend the data model and operations, then in the **Teamcenter Service Bindings Configuration Information** pane, set the kinds of client binding files to create when you generate services code.

© 2025 Siemens

For this parameter	Do this	Remarks
Teamcenter Services client kit home	Click **Browse** and select the location where the service-oriented architecture files have been extracted from the **soa_client.zip** file.	The **soa_client.zip** file is included with the Teamcenter software distribution image.
Rich Client plugin folder	Click **Browse** and select the location of the **plugins** folder under the Teamcenter rich client installation.	
.NET MSBuild Location	If you want to create .NET bindings, then click **Browse** and select the location where Microsoft .NET is installed.	
Service library client bindings	Select the check boxes for the kind of clients that you want to connect to Teamcenter using services.	When you generate service code, the kinds of binding files you select on this dialog box are created. Services are Teamcenter actions that can be called by external clients, such as check in, check out, and so on. Services are used by system administrators to connect their company's applications to Teamcenter. Client binding files are used to connect the client to the Teamcenter server and are written in the language of the client, for example, C++, Java, or .NET.

8. Click **Finish**.

   The wizard creates the project, and the Business Modeler IDE reads in the data model from the dependent template XML files.

   If there are any errors, they are displayed in the **Console** view.

   > Caution:
   >
   > Resolve all errors before you make any changes using Business Modeler IDE. If you do not resolve any errors encountered while reading in the data model, you risk extending corrupted data.

If you want to change the project settings later, in the **BMIDE** workspace window, right-click the project and choose **Properties**, and select **Teamcenter**.

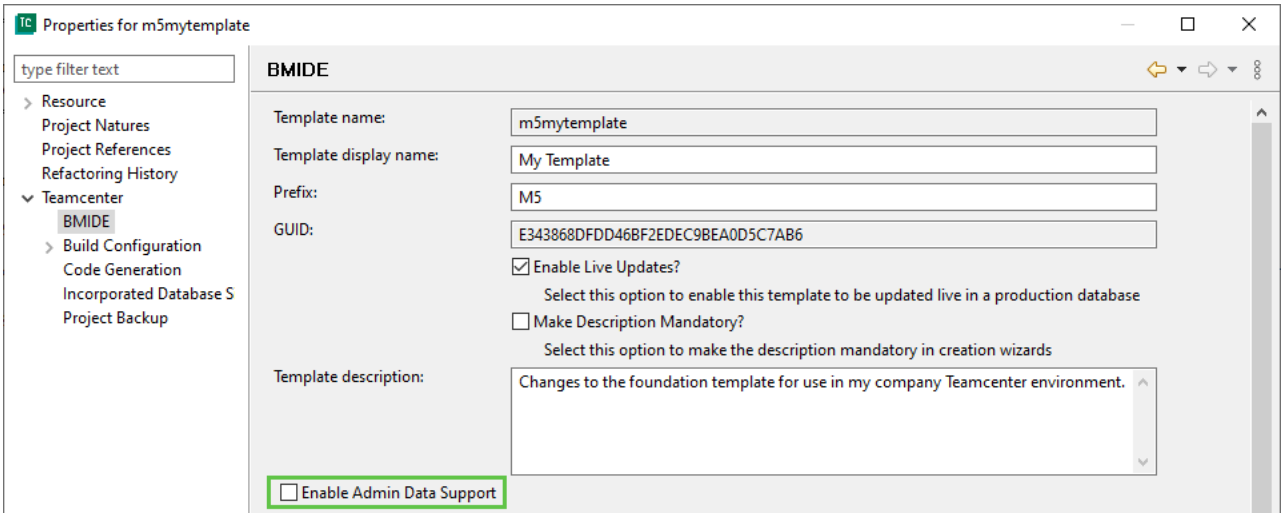
© 2025 Siemens

Enable admin data in an existing project

You can enable admin data support in an existing project.

1. In the BMIDE, right-click on the project and select **Properties**.

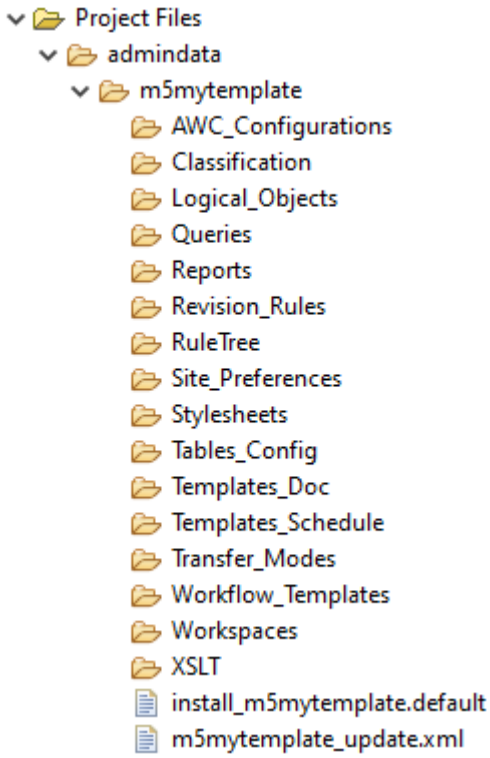
2. Check the **Enable Admin Data Support** check box.



3. Click **Apply and Close**.

   The admin data folders and script files are added to your project

© 2025 Siemens



Use

The administration data scripts are run *after* the data model import has completed. If you wish to import administration data before data model changes, you must create a separate project for that administration data, and then import that project first.

Command line

Instead of using the BMIDE, you can add the admin data folders and script files from the command line instead using the bmide_admin_data utility.

Add admin data to project

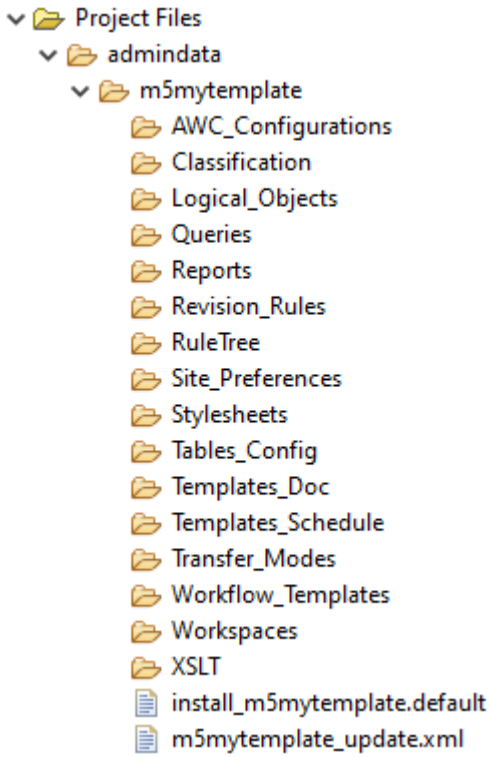
Copy your administration data files into their respective folders in your project.

1. Create your administration data documentation report.

   For example: `generate_admin_data_report -u=adminuser -p=adminpass -g=dba -adminDataTypes=all -outputDir=c:\temp\admindata`

2. Copy your admin data files into their respective folders in your BMIDE template project.

© 2025 Siemens



3. Populate the provided script files with your import utility commands.

> Example:
>
> To import a **tcmaccess_preference_merge.xml** file, use the preferences_manager utility.
>
> ```
> preferences_manager -u=${TC_USER_Name} -pf=$
> {TC_USER_PASSWD_FILE} -g=dba -mode=import -action=MERGE
> -scope=SITE -context=Teamcenter -file=${TC_INSTALL_DIR}/p9test2/
> admindata/Site_Preferences/tcmaccess_preference_merge.xml
> ```

Create a Business Modeler IDE composite software project

You can combine multiple, currently loaded Business Modeler IDE template projects into a composite software project. The composite software project can later be packaged into a single software package containing multiple templates for installation to Teamcenter.

1. In Business Modeler IDE, open the templates you want to include in the composite software project.

2. Choose **BMIDE→New Composite Software Project**.

3. In the **New Composite Software Project** dialog box, in the **Prefix** box, type a 2- to 4-character prefix that will be affixed to the name of all objects created in the composite software project

© 2025 Siemens

outside of its component templates to **ensure unique naming**. The first character must be an uppercase letter, and in the second, third, or fourth character position, type a number between 4 and 9.

4. In the **Project Name** box, enter a name for the software package file that will be created when this project's **extensions are packaged for distribution**. This name must begin with the project prefix to differentiate it from other templates.

5. **Project Display Name** defaults to your project name exclusive of the prefix. This is the template name that appears in Teamcenter Environment Manager when this package is installed on other servers.

6. In the **Project description** box, type a description of the project. The description appears in Teamcenter Environment Manager for the package created from this composite project.

7. In the list of currently loaded template projects, select the template projects you want to include in the composite software project.

8. If you want to create the project in a location other than your default workspace, clear the **Use default location** check box and click **Browse** to choose another location.

   For example, if you are using a source control management (SCM) system to manage your XML source files, you may want to create the project in a location where the SCM can recognize it.

   Caution:

   If you choose the location, you must follow these rules:

   - The destination directory must be named the same as the project.
   - Do not create the project in another project's directory.
   - Do not create the project in the parent directory of your workspace.

9. Click **Finish**.

   The wizard creates the project.

   If there are any errors, they are displayed in the **Console** view.

   Caution:

   Resolve all errors before using the Business Modeler IDE. Otherwise, you may risk extending corrupted data.

© 2025 Siemens

Add a Business Modeler IDE project template to a composite software project

After you have created a composite software project, you can add more currently loaded Business Modeler IDE template projects to the composite software project, or you can remove template projects from the composite software project.

1. In Business Modeler IDE, open the composite software project and the templates you want to include in the composite project.

2. Right-click the composite project and choose **Properties**.

3. In the **Properties** dialog box, choose **Teamcenter→BMIDE**.

4. In the list of currently loaded template projects, select the template projects you want to include in the composite software project.

Deploying templates

Introduction to deploying templates

After you make changes to the data model, you can **deploy** them to a server using the Deploy wizard. Choose **BMIDE→Deploy Template** on the menu bar, or select the project and on the main toolbar click **Deploy Template**  . This process is also known as *live update*. All your extensions are rolled up from your individual extension files into a single template and placed in the database.

If you are deploying live updates to production servers, you can also use the **Deployment Page** by right-clicking a live update template project and choosing **Open Deployment Page**.

Use the Deploy wizard in two different situations:

- **Deploy a template to a test server**

  Live update to a test server when you want to verify your custom data model before packaging it into an installable template and installing it to a production server. This is recommended in most situations.

  > Caution:
  >
  > Do not use live update to a test server when your customizations include the following:
  >
  > - Libraries

© 2025 Siemens

If you have customizations, you cannot use live update because customizations have libraries. Instead, you must package your template and install it using Teamcenter Environment Manager (TEM).

- Localizations

  Do not use live update to place localization changes on a server. Doing so could result in the following error:

  ```
  Error Code: 515062  Error Message: Class referenced
  ```

  Instead of using live update, install localized templates using Teamcenter Environment Manager (TEM).

- **Deploy data to a production server**

  Live update to a production server when you have data to place on the server, such as LOVs and rules. You can live update **non-schema data** that must be updated on a regular basis. In this situation, create a template that only contains data that can be updated live, and use a source control management (SCM) system to manage the versioning of the template source file.

  Note:

  If your process requires you to add both schema and non-schema data to a production server, you must use an SCM system and two Business Modeler IDE clients. One client should be used for deploying non-schema data on a regular basis, and the other for maintaining schema data to be packaged into a installable template. Use the SCM to synchronize both sets of definitions to update the production system with the installable template.

Deploy a template

Deploying a template to a test server.

Deploy a template when you want to send it to a test server prior to installing it to a production server, or when you want to send non-schema data such as LOVs to a production server.

If you are deploying live updates to production servers, you can also use the **Deployment Page** by choosing **BMIDE**>**Deployment Page** or right-clicking a live update project and choosing **Open Deployment Page**.

Note:

If you have customizations, you cannot use live update to a test server because customizations have libraries. Instead, you must package your template and install it using Teamcenter Environment Manager (TEM).

© 2025 Siemens

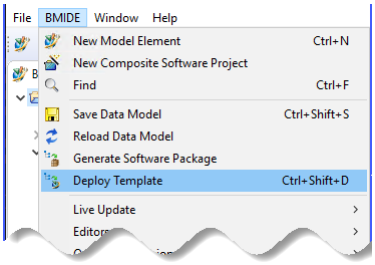
1. The person performing the deployment should log on to Teamcenter and ensure that the Teamcenter server is running. (This user should be the only user logged on to the test Teamcenter server.) Also check that the **BMIDE_ALLOW_FULL_DEPLOY_FROM_CLIENT** preference on the server is set to **TRUE**. By default, the preference is set to **TRUE** to allow deployment.

   To access preferences in the My Teamcenter application within the Teamcenter rich client, choose **Edit**>**Options** and click **Search** at the bottom of the **Options** dialog box.

2. To save any uncommitted changes to the data model, choose **BMIDE**>**Save Data Model**, or on the main toolbar click **Save Data Model**.

3. On the menu bar, choose **BMIDE**>**Deploy Template**.

   You can also select the project in a view and on the main toolbar click **Deploy Template**.



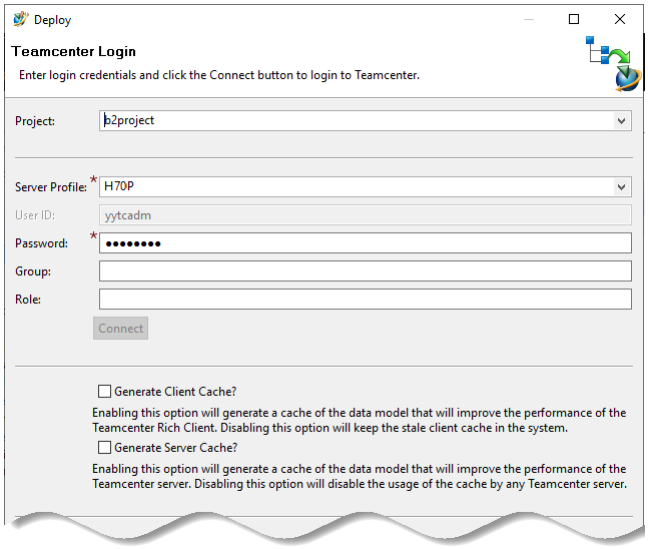
4. In the **Teamcenter Login** dialog box, enter the following information:

For this option	Do this
Project	Click the arrow to select the project to deploy.
Server Profile	Click the arrow to choose the server to deploy the extensions to. If a **server connection profile** has already been created, an existing server profile is loaded. To create profiles, choose **Window**>**Preferences**>**Teamcenter**>**Server Connection Profiles**.
User ID	Type the ID of the authorized user on the Teamcenter server.
Password	Type the password for the authorized user on the Teamcenter server.
Group	(Optional) Type the group the user is assigned to. For example, type **dba** for the database administration group.
Role	(Optional) Type the role the user is assigned. For example, type **DBA** for the database administrator role.

5. Click **Connect**.

   The Business Modeler IDE client connects to the server.

© 2025 Siemens



6. Select the following check boxes to regenerate cache as part of deployment. Be aware that deployment takes longer if you select these options because it takes extra time to update the cache.

   - **Generate Client Cache?**

     Runs the **generate_client_meta_cache** utility to cache metadata for clients. Select this whenever you use live update so that the changes are cached on clients when they connect to the server. This can improve performance for clients. The utility is usually run during installation and upgrade, and when deploying from the Business Modeler IDE.

   - **Generate Server Cache?**

     Runs the **generate_metadata_cache** utility to cache metadata for servers. This cache contains business object types, property descriptors, and constants and reduces the memory footprint for **tcserver** instances.

     Warning:

     Never live update schema changes to a production server. Only live update schema changes to test servers.

     Note:

     If you do not select the **Generate Server Cache?** check box when you deploy schema changes to a test server (for example, changed business objects and properties), then when you log on to the test server, you may see the following message:

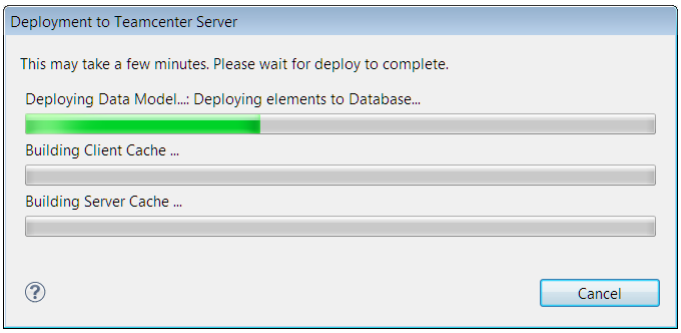
© 2025 Siemens

```
The schema file is out of date. Please regenerate.
```

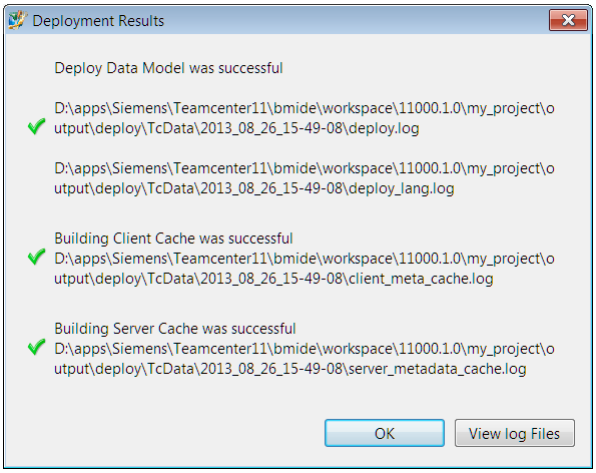
To force generation of the shared server cache, run the **generate_metadata_cache** utility.

7. Click **Finish**.

The **Deployment to Teamcenter Server** dialog box displays the progress of the deployment.

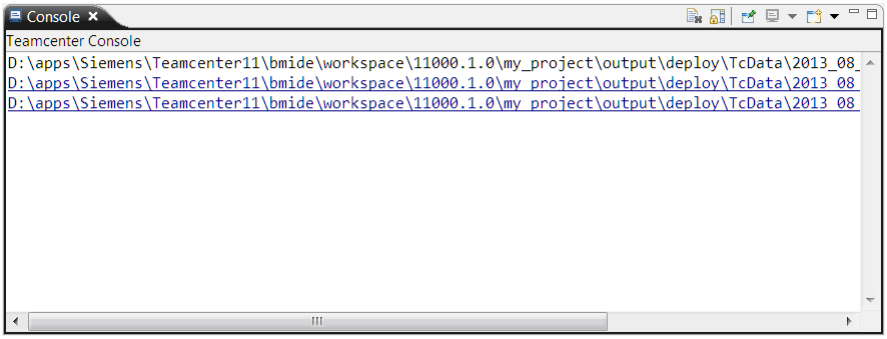


When deployment is done, the **Deployment Results** dialog box displays the outcome of deployment.



The **Console** view displays links to the log files. If there are any deployment errors, take corrective action and attempt to deploy again.

© 2025 Siemens



For every deployment, a new directory with a timestamp is generated in the **output\deploy** folder under your **Project Files** folder. This folder contains the templates and log files generated during the deployment. View the **deploy.log** file to see the results of the deployment.

You can also locate the deployment files in your workspace. To find the *workspace* location, in the **Advanced** perspective, choose **File**>**Switch Workspace**.

For example, on a Windows system, they are saved by default to:

install-location`\bmide\workspace\`*version**project*`\output\deploy`

On Linux, users need to have permissions to the workspace directory.

By default, project backup is enabled at deployment. To change your project's backup settings, right-click the project and choose **Properties**>**Teamcenter**>**Project Backup**.

8. Verify the data model changes are in the server by launching the rich client.

Retrieve deployment archive files

When a deployment is executed either from the Business Modeler IDE or Teamcenter Environment Manager (TEM), deployment archive files are saved in the database in **BMIDE Deploy Archive Dataset (Fnd0BMIDEDeployArchive)** datasets in the **Fnd0DeployArchiveResource** folder. To archive the deployment files, the Business Modeler IDE calls the deployment archive service operation, and TEM calls the **deploy_archive** utility. You can turn off this deployment archival functionality by setting the **DEPLOY_ARCHIVE_ACTIVE** environment variable to **false** (or **off**, **no**, or **0**).

To examine the deployment history or determine why deployments are failing, you can retrieve these archived datasets.

1. In the rich client, use the **BMIDEDeployArchiveRecovery** saved query, or search for **BMIDE Deploy Archive Dataset** datasets, or search for the **Fnd0DeployArchiveResource** folder.

© 2025 Siemens

Archive dataset files are displayed in the **Search Results** view. Archives for Business Modeler IDE deployments are named **deploy_BMIDE_***user-id_date-time-stamp*, and archives for TEM deployments are named **deploy_TEM_***user-id_date-time-stamp*.

2. In the **Search Results** view, right-click the deployment archive file, choose **Named References**, select the file, and click **Download**.

3. Unzip the archive file.

   Each archive consists of:

   - **log_files_archive.zip** file

     Contains all of the **business_model_updater** and **business_model_extractor** logs and the **tc_install** or **tc_upgrade** logs that are created during the deployment.

   - **compare_history_files_archive.zip** file

     Contains the **model***, **delta***, **model_lang***, and **delta_lang*** files for this deployment

Generate a software package for distribution

So that your Business Modeler IDE customizations and extensions of the data model can be distributed and deployed to Teamcenter environments, you generate a software package. To install the package to a Teamcenter environment, you can use either Teamcenter Environment Manager (TEM) or Deployment Center.

The package comprises a folder structure containing the files that define the extensions. If you have coded extensions in C++, then by default the C++ library built for the target platform (Linux or Windows) is included as part of the package.

If you have added multiple BMIDE template projects to a composite software project, then you can generate a single package for the included template projects. Templates within a software package can be individually selected for installation.

When frequent scheduled package builds are desired, such as nightly during an intense development cycle, use the bmide_generate_package utility to generate packages.

1. Choose **BMIDE**>**Generate Software Package**.

2. In the **Generate Software Package** dialog box, perform the following steps:

   a. In the **Project** box, select the project to package.

   b. Specify the location for the package files.

© 2025 Siemens

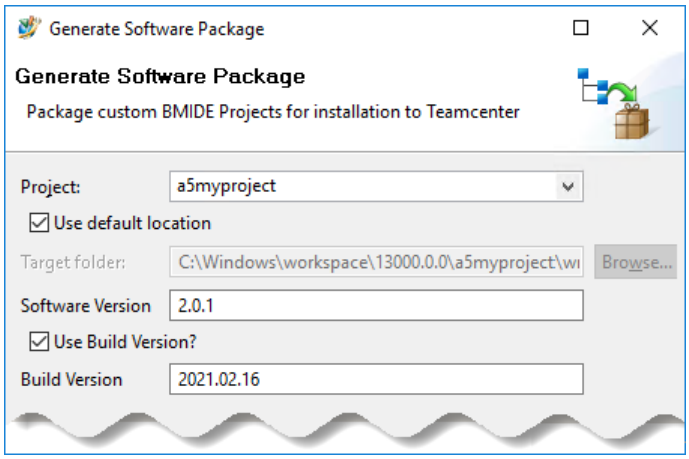
Use default location

Places package files within your workspace project folder, under **output\packaging**.

Target folder

If you want to set the folder where the package files are placed, then clear the **Use default location** check box, and to the right of the box click **Browse** to choose the folder.

c. In the **Software Version** box, use the default value or type the next incremented value based on your **software versioning scheme**. You may enter up to four integers separated by decimal points.



Ensure that the combination of project name, platform, and software and build version number does not duplicate the name of a package that has already been uploaded to your Deployment Center software repository.

By design, Deployment Center scans a package only once. On subsequent passes through the repository, Deployment Center does not rescan a package named the same as a previously scanned package.

Software packages that do not have build version numbers are treated as more recent than software packages that have a matching software version number plus any build version number.

d. (Optional) To add a build version number to the package name, select the **Use Build Version** check box.

In the **Build Version** box, use the default value or type the next incremented value based on your **build versioning scheme**. You can enter up to four integers separated by decimal points. Recommended practice is to use a numeric Year.Month.Day pattern.

e. (Optional) To build the package excluding any platform-specific files, such as compiled libraries that are specific to an OS, select **Support all platforms?**. The package will then be deployable

© 2025 Siemens

to a Teamcenter business logic server on any platform (wntx64 or lnx64x). If you want to include platform-specific content, do not select the **Support all platforms?** option.

This option is useful when your software package only has a BMIDE template and no platform-specific files.

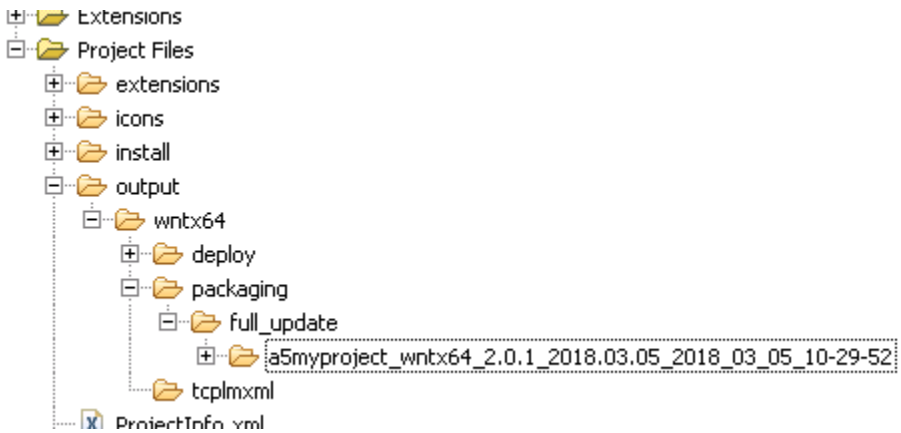
3. Click **Finish**.

Software package files

By default, the template files are saved to the **[workspace]\output\[platform]\packaging\full_update** folder.

Within the **full_update** folder, a unique folder is generated each time a package is created. The folder name comprises [**project name**]_[*target platform*]_[**software version**]_[**build version**]_[*system-generated date and time*]. The folder name includes **build version** only when the **Build Version** box is used while creating the package.

For example:



Sub Folder	File	Description
artifacts	*template-name*_**admindata.zip**	Contains your administration data files and their install scripts. If your project does not contain administration data, this zip file will not be present.
	*template-name*_**icons.zip**	Contains all the icon files.
	*template-name*_**install.zip**	Contains all the support files for installing and upgrading your extensions and any data files that were stored in the *project*/**install** folder.
	*template-name*_**template.zip**	Contains the template definitions (*template-name*_**template.xml**) and the dependency file (*template-name*_**dependency.xml**).

© 2025 Siemens

Sub Folder	File	Description
dc_contributions	*template-name*_**config.json**	Contains the information necessary for Deployment Center to recognize the template and handle the template for installation and upgrade.
	*template-name***DCBundle_***language-code_ country-code* **.xml**	Contains the localized text for the feature file so that Deployment Center can display the feature description in the localized version.
tem_contributions	**feature_***template-name***.xml**	Contains the information necessary for TEM to recognize the template and handle the template for installation and upgrade.
	*template-name***Bundle_***language-code_ country-code***.xml**	Contains the localized text for the feature file so that TEM can display the feature description in the localized version.

If you have generated C++ classes or services artifacts, the following files are added:

Sub Folder	File	Description
	*template-name*_**rtserver.zip**	This file contains C++ run-time libraries (*template-name*_**rtserver.xml**).
	template-name **_soa_client_kit.zip**	This file contains new services.
Web_tier		This folder contains new services web tier deployment files for .NET and Java EE servers.

Versioning software packages

To identify the software packages that you generate in a development cycle, you use the combination of a pair of number patterns: **Software Version** and **Build Version**.

Software Version identifies the target release. A typical pattern is *major_release.minor_release.patch*. This value remains constant during the development cycle.

A **Software Version** value is always required.

Build Version identifies internal iterations generated as the project is developed. Recommended practice is to use a numeric *Year.Month.Day* pattern.

A **Build Version** value should be included while the package is under development, and should be incremented each time you prepare a package for testing.

© 2025 Siemens

Caution:

Ensure that the project name, software version, and build version combination does not duplicate that of a package which has already been uploaded to your Deployment Center software repository.

By design, Deployment Center scans a package only once. On subsequent passes through the repository, packages with a previously scanned name, platform, and software and build version number combination are not rescanned. Deployment Center does not consider the system-generated date and time in the package folder name when scanning software packages.

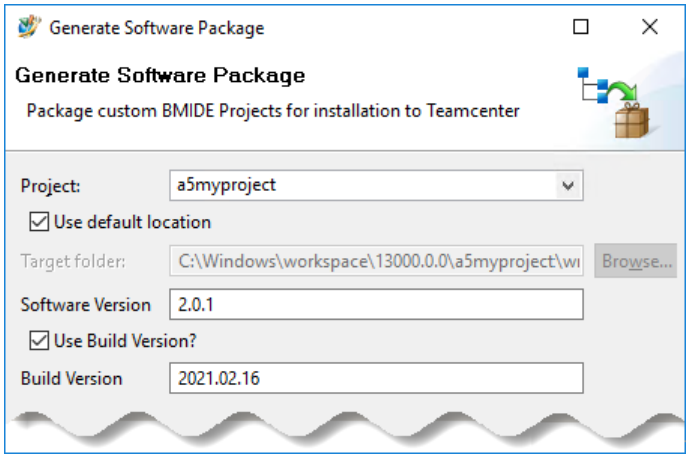
The version combination has a significant effect when some version of software in the package is already installed in a Teamcenter environment.

If the environment has	and you attempt to install	the result is	Comments
Software_1.0	Software_2.0 _2018.03.05	Installation succeeds.	Both versions of the software are shown in the list of installed software.
Software_2.0 _2018.03.05	Software_2.0 _2018.03.07	Installation succeeds.	In the list of installed software, the build number is updated.
Software_2.0 _2018.03.07	Software 2.0	Installation succeeds.	Software that does not have a build version number is treated as the final version. In the list of installed software, the build number is removed from the software.
Software_2.0	Software_2.0 _2018.03.07	Installation fails.	You cannot install an earlier version of installed "final version" software.

Version numbering details

The **Generate Software Package** dialog box displays the latest values used for the project. You can leave the values unchanged, or you can specify a higher value. Both **Software Version** and **Build Version** allow up to four whole-number groups separated by decimal points.

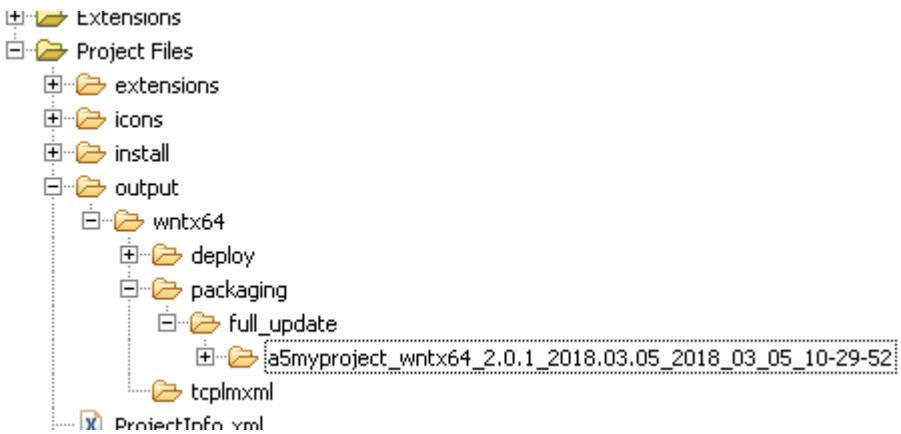
© 2025 Siemens



The versions you specify are included in the output folder name. The folder name comprises [**project name**]_[*target platform*]_[**software version**]_[**build version**]_[*system-generated date and time*]

Example:

Values as entered in the dialog box shown above result in the folder name **a5myproject_wntx64_2.0.1_2018.03.05_2018_03_05_10-29-52**.



Versioning packages in a typical development cycle

1. Add definitions to your Business Modeler IDE template project. Deploy your changes in your development environment and test as needed.

2. When you are ready to test your changes in a test environment, increment the build number and generate a software package of the project.

3. Install the software to a test environment.

   You can use either Deployment Center or Teamcenter Environment Manager to install the software.

© 2025 Siemens

4. Test the definitions against use cases.

5. Repeat from Step 1 until you are ready to release the software to a production environment.

6. Generate the production version package without a build number.

Install a template using TEM

After you package extensions, install the resulting template to a production environment using Teamcenter Environment Manager. You can also use this procedure to install a third-party template. You could also install a template using Deployment Center, or the **tem** command line utility with its `-install` argument.

1. Ensure that you have a good back up of the Teamcenter environment.

2. Copy the template files from the **packaging** directory on your Business Modeler IDE client to a directory that is accessible by the server.

   By default, packaged template files are located in the Business Modeler IDE workspace directory in the **output\packaging** folder under the project. To find the *workspace* location, choose **File→Switch Workspace**. For example, on a Windows system, they are saved by default to:

   install-location`\bmide\workspace\`*version*`\`*project*`\output\packaging`

   On Linux, users must have permissions to the workspace directory.

3. Start Teamcenter Environment Manager (TEM).

4. In the **Maintenance** panel, choose **Configuration Manager** and click **Next**.

5. In the **Configuration Maintenance** panel, choose **Perform maintenance on an existing configuration** and click **Next**.

6. In the **Configuration** pane, select the configuration from which the corporate server was installed. Click **Next**.

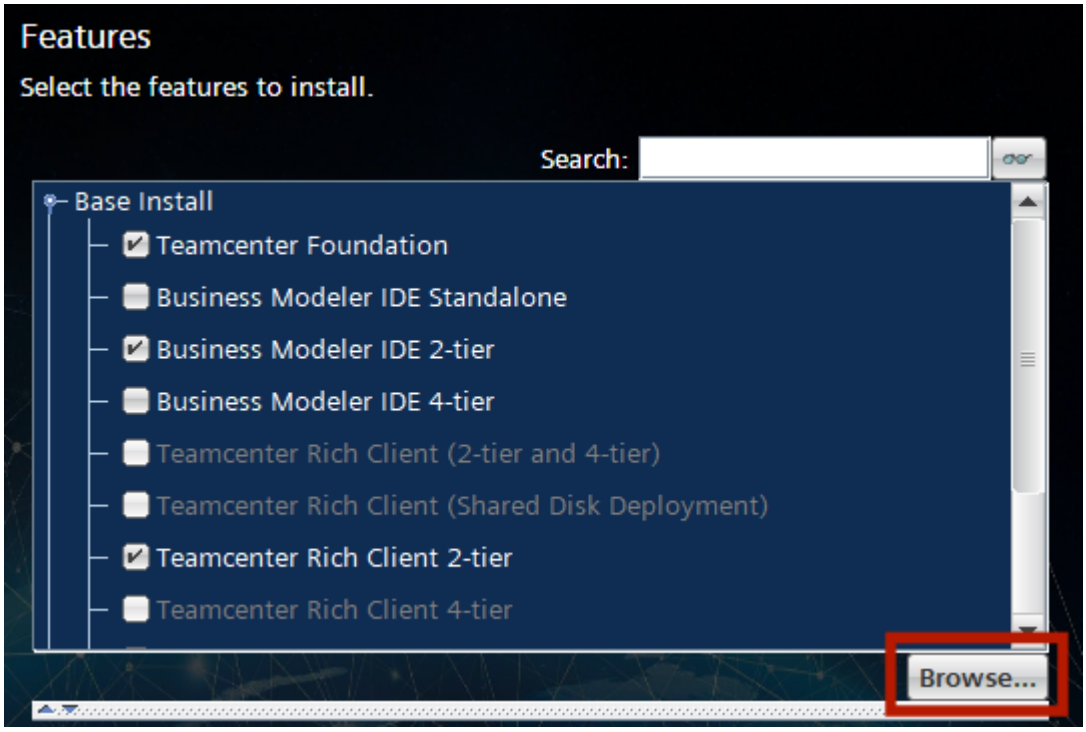
7. In the **Feature Maintenance** panel, under the **Teamcenter** section, select **Add/Remove Features**. Click **Next**.

   If you already installed a template to the database and want to update the template, under the **Teamcenter Foundation** section, select **Update the database**. This option should not be used to install a new template but only to update an already installed template.

   Use the **Add/Update templates for working within the Business Modeler IDE client** option under **Business Modeler Templates** only if you want to add a dependent template to your Business Modeler IDE.

© 2025 Siemens

8. In the **Features** panel, click the **Browse** button beneath the features list on the right side of the panel.



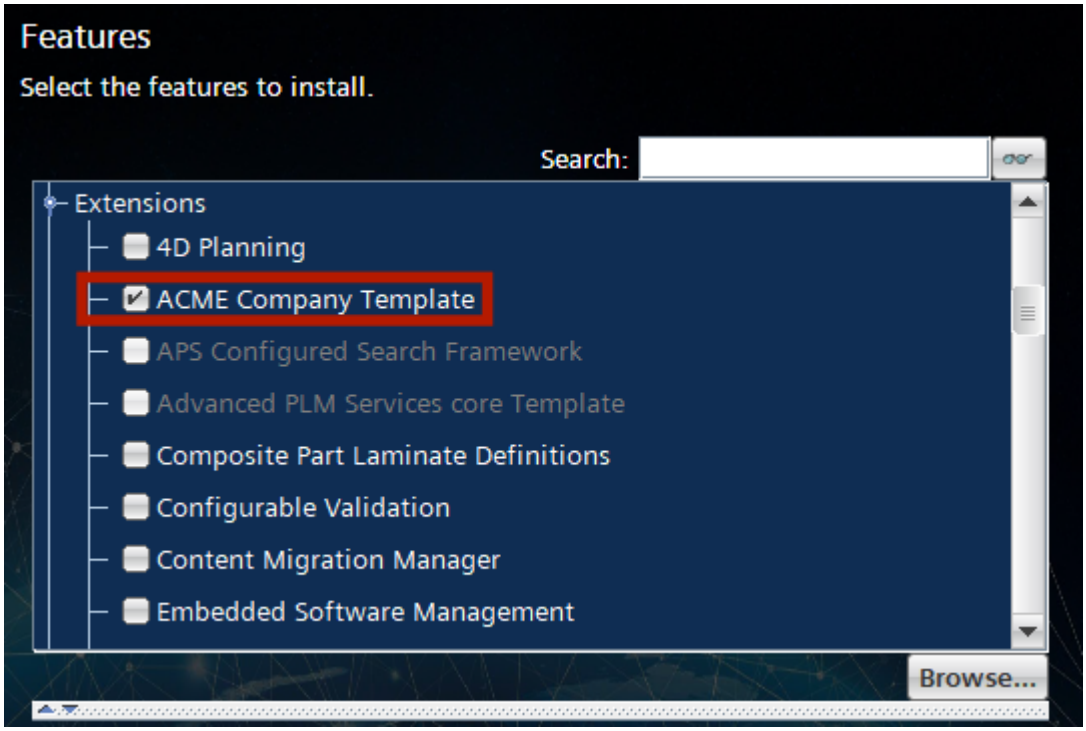
9. Browse to the directory where you have copied the template files. In the **Files of type** box, ensure that **Feature Files** is selected so that you see only the installable template (feature) file. Select your template's feature file (**feature_***template-name***.xml** in the **tem_contributions** directory) and click the **Select** button.

   The template appears as a new feature under **Extensions** in the **Features** panel.

   You can change the location of the feature in the **Features** panel and add a new group to place the feature under.

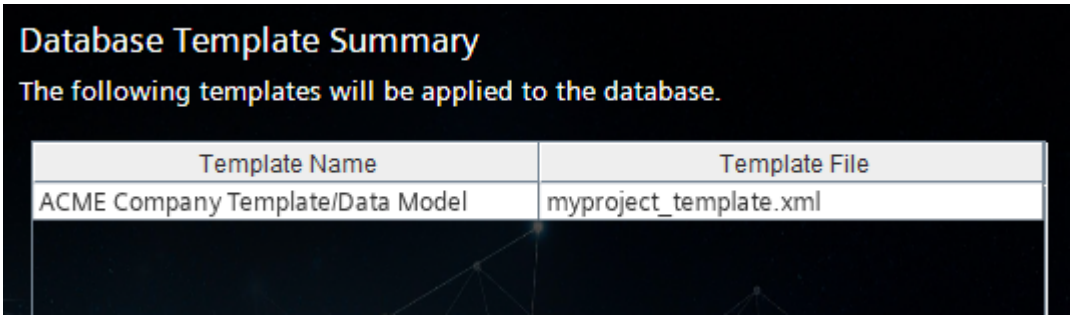
10. Select the new template in the **Features** panel. Click **Next**.

© 2025 Siemens



11. In the **Teamcenter Administrative User** panel, enter your user name and password to log on to the server. Click **Next**.

12. The **Database Template Summary** panel displays the list of templates that are installed as part of your template install. Click **Next**.



13. In the **Confirmation** panel, click **Start**. The new template is installed.

> Note:
>
> If the installation fails because of invalid data model, perform the following steps:
>
> a. Fix the incorrect data model and repackage the template.

© 2025 Siemens

b. Locate the *template-name*_**template.zip** in your project's **packaging** directory and unzip it to a temporary location. Copy the following files to the server in the *TC_ROOT*/**install/***template-name* folder:

*template-name*_**template.xml**
*template-name*_**dependency.xml**
*template-name*_**tcbaseline.xml** (if the file exists)

c. Launch Teamcenter Environment Manager in the maintenance mode and continue with recovery.

14. To verify the installation of the new template, confirm that the *TC_DATA***model** directory on the Teamcenter server contains the new template files.

Also log on to the server and confirm that you can create instances of your new data model.

Note:

To have libraries read on the user system, the **TC_LIBRARY** environment variable must be set to the platform-specific shared library path. This environment variable is set to **LD_LIBRARY_PATH** on Linux systems. The platform is detected when the Teamcenter session is initiated.

Update a template using TEM

If you already installed a template as a new feature and want to update it because you have added more data model definitions to it, perform the following steps in the Teamcenter Environment Manager (TEM).

Note:

You can also update a template using the **tem** command line utility, for example.

```
tem -update -full -templates=template-name-1,template-name-2 -path=location-of-template-files
-pass=password
```

1. Ensure that you have a good back up of the Teamcenter environment.

2. Copy the packaged template files from the **packaging** directory on your Business Modeler IDE client to a directory that is accessible by the server.

By default, packaged template files are located in the Business Modeler IDE workspace directory in the **output\packaging** folder under the project. To find the *workspace* location, choose **File→Switch Workspace**. For example, on a Windows system, they are saved by default to:.

install-location\bmide\workspace*version**project*\output\packaging

© 2025 Siemens

3. Start Teamcenter Environment Manager (TEM).

4. In the **Maintenance** panel, choose **Configuration Manager** and click **Next**.

5. In the **Configuration Maintenance** panel, choose **Perform maintenance on an existing configuration** and click **Next**.

6. The **Configuration** panel displays the installed configuration. Click **Next**.

7. In the **Feature Maintenance** panel, under the **Teamcenter Foundation** section, select **Update Database (Full Model - System Downtime Required)**. Click **Next**.

   > Note:
   >
   > Use the **Add/Update Templates for working with the Business Modeler IDE Client** option under **Business Modeler** only if you want to add or update a dependent template to your Business Modeler IDE.

8. Click **Next**

9. In the **Teamcenter Administrative User** panel, enter your user name and password to log on to the server. Click **Next**.

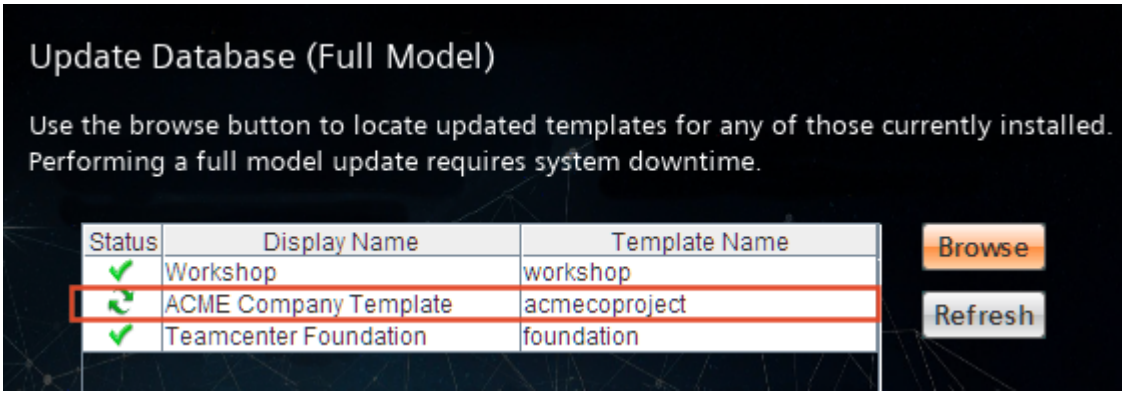
   The **Update Database** panel displays currently installed templates.

10. Click the **Browse** button to navigate to the directory where the packaged template files are located. Select the updated **feature_***template-name***.xml** file.

    > Note:
    >
    > If you are fixing a COTS template (for example, the Foundation template) using a new template file provided in a patch, you must copy the template's **feature_***template-name***.xml** and the *template-name***_install.zip** files to the same temporary directory containing the new *template-name***_template.zip** file.

    The template displays a refreshed status icon .

© 2025 Siemens



11. Click **Next**.

12. In the **Confirmation** panel, click **Next**.

    The new template is installed.

13. To verify the installation of the revised template, log on to the server and confirm that you can create instances of your new data model.

Install or update a template using the tem utility

You can use the **tem** command line utility to install features or update a template without user interaction with the Teamcenter Environment Manager (TEM) graphical user interface (GUI). If you perform frequent data model updates and find that using the TEM GUI slows your process, the command line way of performing updates allows you to write a script for the process and thereby reduce the manual effort involved in doing the update.

However, not all features can be installed or updated using the command line utility. Only those features that are data models or have subfeatures with data models apply. If a qualifying feature has an associated rtserver, Web, (and so on) subfeature, then those subfeatures are installed as well, assuming the required standard features are installed. If any feature or subfeatures contain panels other than the **Admin User** panel, none of the features can be installed. One additional limitation is that you cannot install any feature that requires screen input. (For example, the ClearCase Integration feature prompts for the ClearCase server.)

For example, the rich client cannot be installed or updated using the **tem** command line utility because it is not a data model, nor does it contain a data model subfeature. However, the Teamcenter Automotive Edition feature can be installed using the utility because it contains a data model subfeature and none of the subfeatures have panels other than **Admin User** panel.

The following examples illustrate how to use the utility:

- Installation

  Use the **-install** argument to install a template:

© 2025 Siemens

```
tem -install -features=feature-name -path=location-of-template-files -pass=password
```

- Update

  Use the **-update** argument to update a template:

  ```
  tem -update -full -templates=template-name-1,template-name-2 -path=location-of-template-files
  -pass=password
  ```

- Live update

  Use the **-live** flag with the **-update** argument to perform a live update:

  ```
  tem -update -live -templates=template-name-1,template-name-2 -path=location-of-template-files
  -pass=password
  ```

- Dry run

  Use the **-dryrun** flag with the **-install** or **-update** argument to validate the process before committing the changes to the database:

  ```
  tem -update -full -dryrun -templates=template-name-1,template-name-2 -path=location-of-template-
  files -pass=password
  ```

Live updates

Introduction to live updates

Live updates is the revision on a live running system of nonschema data such as lists of values (LOVs) that require frequent update. The live updates functionality in the Business Modeler IDE allows an administrator to revise data in the production database without shutting down the production server. It also provides tighter control on which elements get updated live in a production database.

Use one of the following processes to update live data:

- **Single administrator**

  Use this process if you are a single administrator who makes live updates and distributes them to servers. In this process, make your updates on a preproduction server before sending the updates to production servers. This process is recommended in most situations.

- **Multiple administrators**

  Use this process if you have multiple administrators who make live updates and distribute them to servers. In this process, there is no preproduction server.

© 2025 Siemens

Caution:

Siemens Digital Industries Software recommends using a single administrator if possible to better control the live updates distributed to servers.

Tip:

If you must update many property values at once, consider using the **attribute_export** and **tcxml_import** utilities to update the values of object properties in your Teamcenter database in bulk.

Single administrator versus multiple administrators in a live updates environment

Typically, Business Modeler IDE developers create a template that contains the data model extensions. The server administrator then uses Teamcenter Environment Manager (TEM) to update the production database with the custom template.

Previously, if further updates needed to be made to the custom template in the database (for example, new LOV values need to be added), the developer first updated the custom template in the Business Modeler IDE with the new LOV values; then the server administrator shut down the production server and used TEM to install the template to update the production database. However, the recommendation to shut down the production server each time prior to updating the custom template in the database was too restrictive for administrators who wanted to update live data (for example, LOV values) on a regular basis. Therefore, the live updates functionality was devised to allow administrators to update nonschema data on running production servers.

Now, a Business Modeler IDE developer works on a template project, tests the changes on the test site, and during the next system downtime, packages the changes and sends them to the server administrator. In addition to the developer, there is a live data Business Modeler IDE administrator. This administrator works on the live update project. During system downtime, this administrator collects the changes (packages) from the developers and performs a TEM deployment of these changes to the preproduction and production servers. When the server is running, this administrator works on the live update project to add any necessary live updates.

In the single administrator environment, there is only one live updates administrator who is responsible for distributing updates to all sites, both preproduction and production.

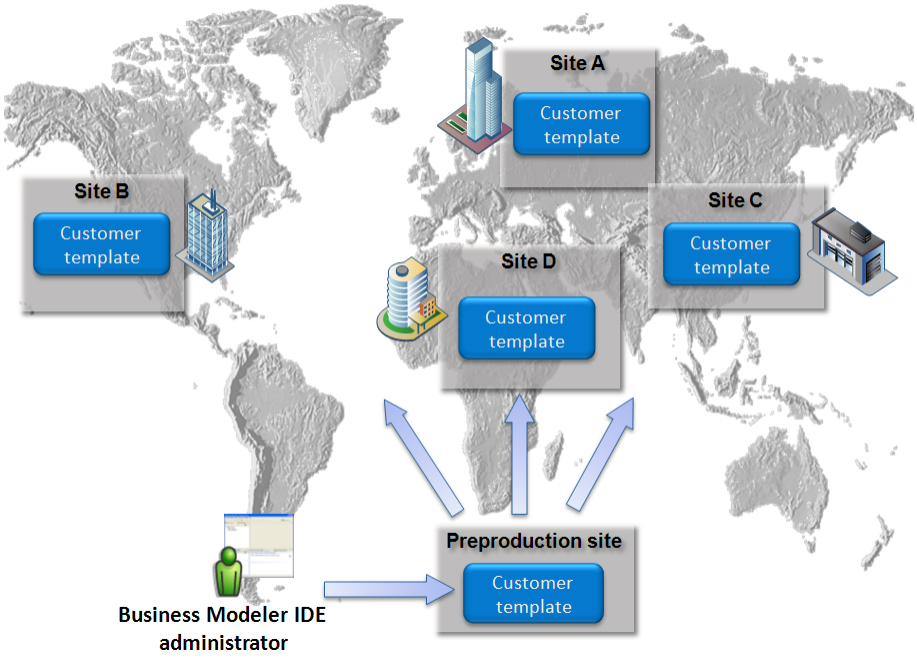
In the multiple administrator environment, there may be more than one live updates administrator per site, and they share the responsibility of keeping their respective sites updated.

- Single Business Modeler IDE administrator of live updates

  Use this process if you are a single administrator who makes live updates and distributes them to servers. In this process, make your updates on a preproduction server before sending the updates to production servers. This process is recommended in most situations.

© 2025 Siemens

- Control all updates with one administrator.
- Perform live updates in one environment.
- Prevalidate updates.
- Package the template and deploy to each site through Teamcenter Environment Manager (TEM).

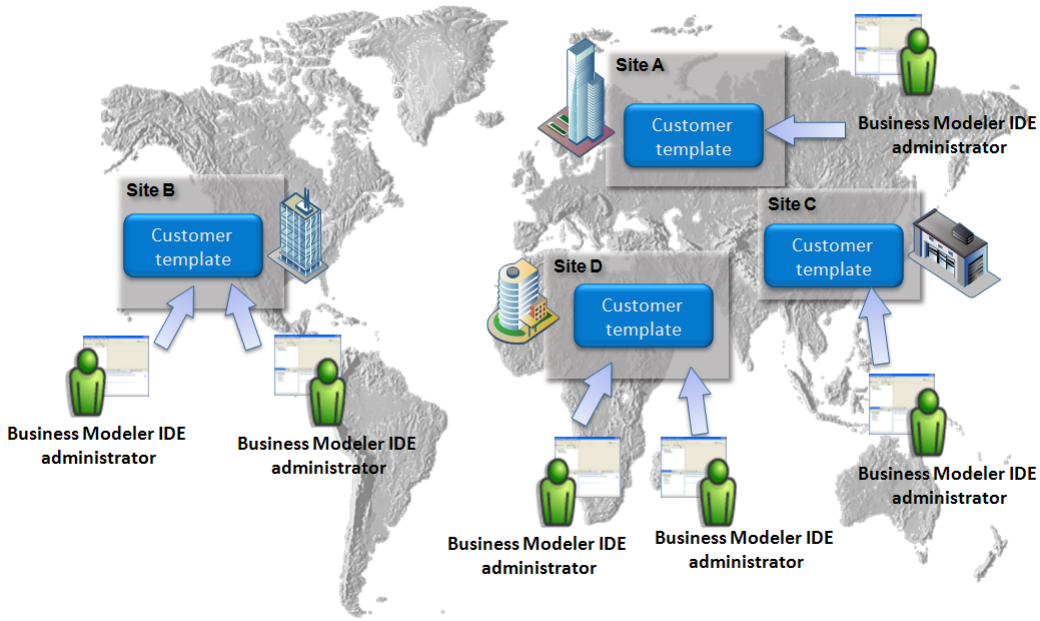


- Multiple Business Modeler IDE administrators of live updates

  Use this process if you have multiple administrators who make live updates and distribute them to servers. In this process, there is no preproduction server.

  - Perform live updates with multiple administrators.
  - Clients at each site update live data.

© 2025 Siemens



Live updates for a single administrator

Live update process: single administrator

Follow this process when you are a single administrator who performs live updates to a production server.

1. In the developer environment, **ensure that a template is enabled for live updates and deploy it to the server**. A custom template is enabled for live updates when the **Enable Live Updates?** check box is selected on the template project.

2. In Teamcenter, **configure the Live Update preference** to select the data model elements to update on the server.

3. In the live updates administrator environment, **install the Business Modeler IDE client** on a machine with 2 GB RAM.

4. **Create a live update project** by choosing **File→New→New Live Update Project**.

5. Make changes to data in the template and **perform live updates** on the preproduction and production servers by clicking **BMIDE** on the menu bar and choosing **Deploy Template** or **Live Update→Deployment Page**.

6. If other sites also require the data updates you have created, such as vendors or partners, you can **package the template and send it to each site** so the updates can be installed using Teamcenter Environment Manager (TEM).

© 2025 Siemens

7. In the developer environment, **incorporate the latest live updates** from the production site by running the Incorporate Latest Live Update Changes wizard.

Enable a template for live updates and deploy it

You must create a Business Modeler IDE template project and install it to the server before you create a live updates project. The Business Modeler IDE template project holds the schema data such as business objects and classes that can only be installed to a production server in a template using Teamcenter Environment Manager (TEM); the live updates project contains the nonschema elements such as LOVs and rules that can be installed to a production server using live update.

1. Create the Business Modeler IDE template project.

   Choose **File→New→New Business Modeler IDE Template Project**.

2. Enable the Business Modeler IDE template project for live updates.

   a. Right-click the Business Modeler IDE template project, choose **Properties**, and choose **Teamcenter→BMIDE**.

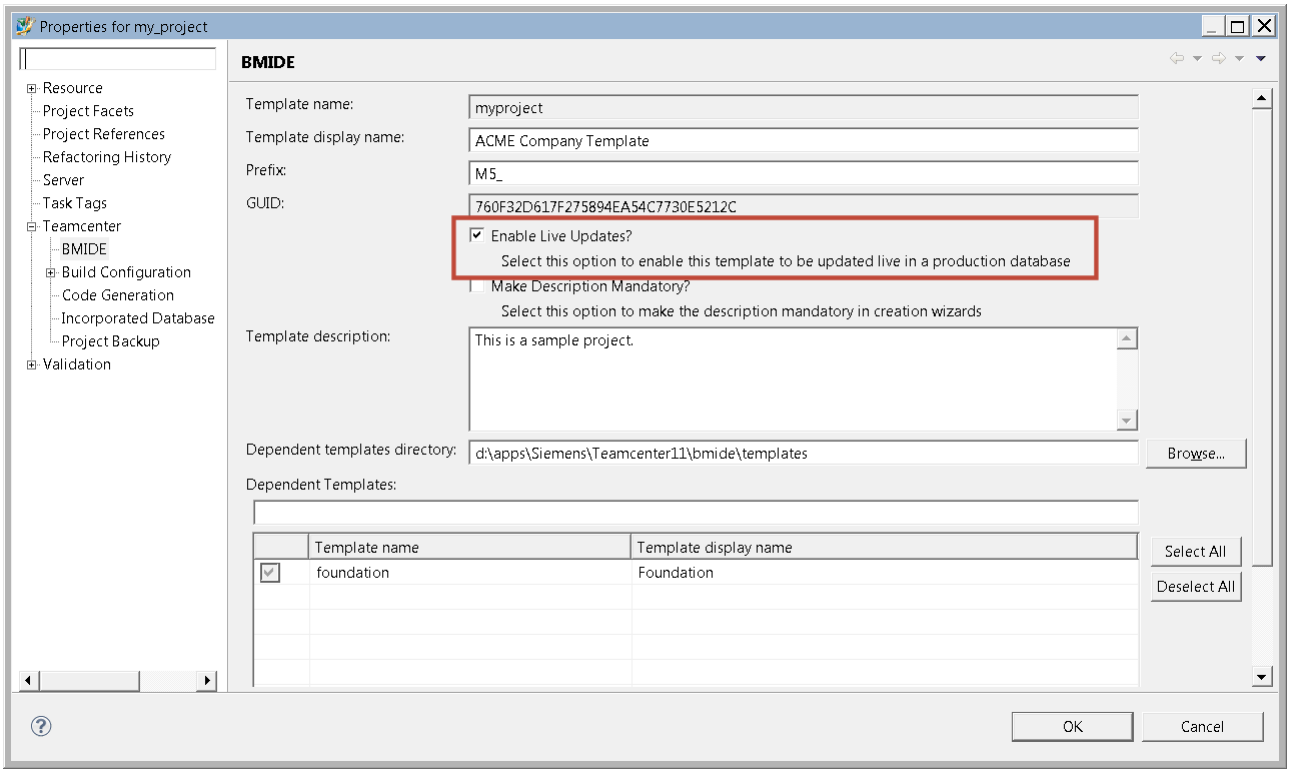
   b. In the **BMIDE** dialog box, select the **Enable Live Updates?** check box.

      This specifies that the Business Modeler IDE template project can have a live updates project created in conjunction with it. This option *must* be selected to enable the creation of a live updates project.

      > Note:
      >
      > This check box is selected by default when you create a new template project.

   c. Click **OK**.

© 2025 Siemens



> Note:
>
> If you want to distribute this template later to a site that you do *not* want to be able to receive live updates, clear the **Enable Live Updates?** check box. For example, to distribute this template to a supplier who should not receive live updates, provide this template with the box cleared. You may also have some servers in your company that should not receive live updates, and in these situations, you can clear this box and then install the template to those servers.

3. Install the Business Modeler IDE template project to the server.

   - Test server

     If you are installing to a test server only, choose **BMIDE→Deploy Template** on the menu bar.

   - Production server

     Package the template and **install it using Teamcenter Environment Manager (TEM)**.

Configure the Live Update preference

Use the **Live Update** preference to select the data model elements to enable for live update.

© 2025 Siemens

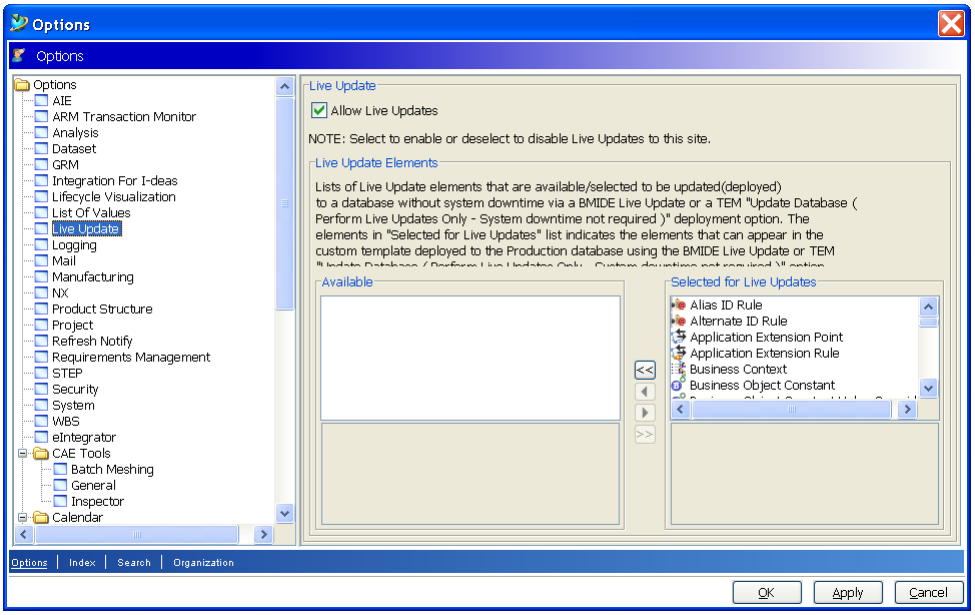
1. In the My Teamcenter application in the rich client, choose **Edit→Options** and select **Live Update** in the left pane. (You can also choose **Edit→Options→Search** and search for the **Live Update** preference.)

2. To select the data elements to allow for live update, move them from the **Available** list on the left to the **Selected for Live Updates** list on the right.

3. Click **Apply**.

4. If you have a four-tier environment, recycle servers to ensure that each warm server receives the latest preference settings. For example, in the Teamcenter Management Console, click **Restart Warm Servers**.

When you perform a live update, only the selected data model elements are updated on the server.



Note:

When incorporating live updates from a production site, before you extract the latest live updates, it is a good practice to first block anyone from making any more live updates to ensure you get the latest. Do this by clearing the **Allow Live Updates?** check box in the **Live Update** preference dialog box in the rich client at the production site. After you finish incorporating the updates, select the **Allow Live Updates?** check box again.

When you select the **Allow Live Updates?** check box, it sets the **BMIDE_ALLOW_LIVE_UPDATES** preference to **true**.

© 2025 Siemens

Install the Business Modeler IDE client for the live updates administrator

The Business Modeler IDE must be installed for the live updates administrator on a machine with a minimum of 2 GB of RAM. This is to accommodate incorporating live data changes. If there is less than 2 GB of RAM, performance when incorporating live data changes is degraded significantly, or the incorporation may not complete at all.

1. Install the Business Modeler IDE on a machine with a minimum of 2 GB of RAM.

2. **Allocate memory to the Business Modeler IDE** using the **Xmx** setting in the **BusinessModelerIDE.ini** file.

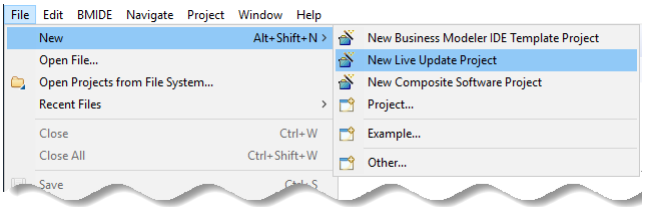
> Caution:
>
> Because Java standards require that no more than 25 percent of total RAM be allocated to virtual memory, if the amount allocated to the Business Modeler IDE is higher than 25 percent of total RAM, memory disk swapping occurs when incorporating live data changes, severely impairing system performance.

Create a live update project

A *live update project* is a project that manages live data in an already-installed custom template on the server. A live update data project holds only data to be deployed to a running production server. Create a live update project for each custom template to which you want to make live updates on the server.

1. Ensure that the template is installed on the server, and that the server is **enabled to receive live data updates**.

2. Choose **File**>**New**>**New Live Update Project**.



3. In the **Teamcenter Login** dialog box, specify option values for connecting to the server.

For this option	Do this
Server Profile	Select the server connection profile to use.
Password	Type the server access password for the profile's **User ID**.
Group	(Optional) Type the group name for the profile's **User ID**.
Role	(Optional) Type the role name for the profile's **User ID**.

© 2025 Siemens

4. Click **Connect**.

   The Business Modeler IDE client connects to the server.

5. If there are multiple templates on the server enabled for live updates, click **Next**.

   From the **Template Name** list, select the template to use for live updates.

6. Click **Finish**.

The project is created and named *template-name*_**live_update**.

If a live update project with the same name already exists, a number is appended to the name of the new live update project (for example, *template-name*_**live_update1**).

Perform live updates

After you create a live update project, you can revise data, such as lists of values (LOVs), and use the **Deploy Template** feature to distribute the updates to a preproduction server. A *preproduction* server is a Teamcenter server that has an exact copy of the templates that the production sites have. Use this server to test a live update before deploying the live update to the production servers.

After testing, you can use the **Deployment Page** to deploy live updates to production servers. Only use this if you are working in a single administrator environment.

> Note:
>
> If you are working in a **multiple administrator environment** with many people creating live updates for distribution to multiple servers, you must package updates into templates for distribution and run the Incorporate Latest Live Update Changes wizard.

1. Create data in the live update project.

   > Note:
   >
   > Typically, administrators create lists of values (LOVs) in a live update project, but **many other kinds of data can be created**.
   >
   > You cannot create schema elements such as business objects or properties in the live update project. Those objects have disabled buttons so that you cannot create or change those objects.

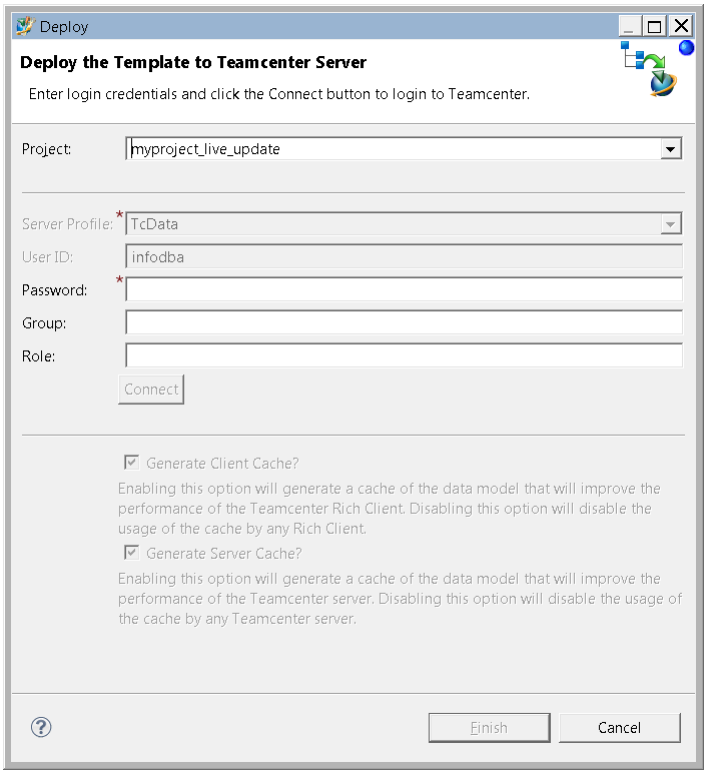
2. Revise data, such as lists of values (LOVs).

3. Deploy to a preproduction server for testing.

© 2025 Siemens

a. Right-click the live update project and choose **Deploy Template**, or choose **BMIDE→Deploy Template** on the menu bar.

b. Type the password, click the **Connect** button, and when a connection is established, click **Finish**.



> Note:
>
> The system checks if the live update project is synchronized with the server. If there is data on the server that is not in the live update project, the Synchronize wizard runs, allowing you to resolve the conflicts.

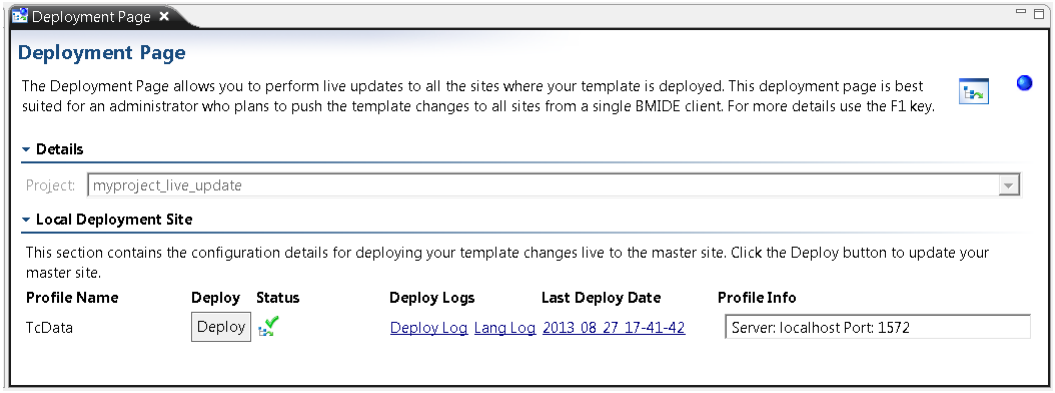
c. See the links to the logs in the **Console** view to verify the success of the update.

d. To verify the live update, log on to the preproduction server and confirm that you can create instances of your newly revised data model.

4. Deploy to production servers.

a. After you finish testing and are ready to deploy to the production servers, choose **BMIDE→Live Update→Deployment Page** or right-click the live update project and choose **Open Deployment Page**.

© 2025 Siemens



b. Click the **Deploy** button to deploy the template to production servers.

c. To verify the live update, log on to a production server and confirm that you can create instances of your newly revised data model.

> Note:
>
> To see the data changes, end users must log off and log on again and wait for servers to cycle through the changes.

Package a live update project for installation to other sites

If other sites also require the live updates you have created, such as vendors or partners, you can package the template and send it to the administrators of each site. Administrators install the template to their sites using the **Update the Database (Perform Live Updates - System downtime not required)** option in Teamcenter Environment Manager (TEM).

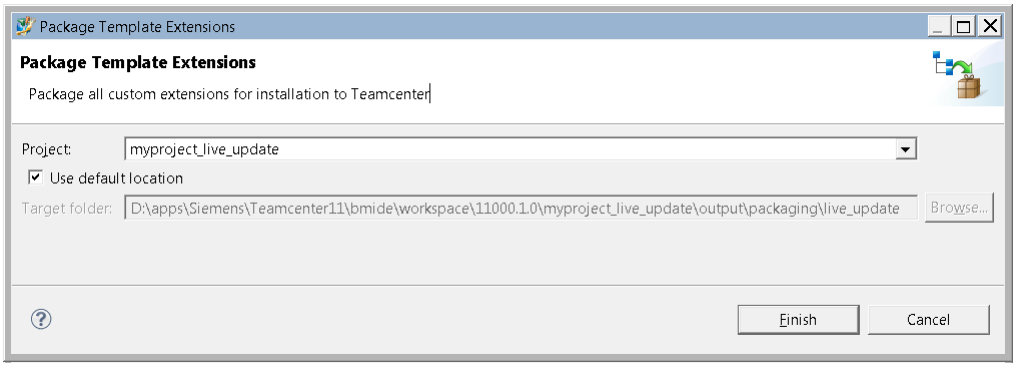
> Note:
>
> You can also live update a template using the **tem** command line utility, for example:
>
> ```
> tem -update -live -templates=template-name-1,template-name-2 -path=location-of-template-files
> -pass=password
> ```

1. Package the template

   a. Choose **BMIDE→Generate Software Package**.

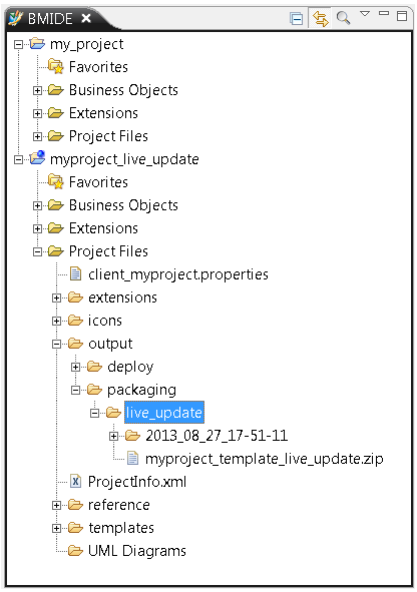
   b. Click the arrow in the **Project** box to select the live update project to package.

© 2025 Siemens



c. Click **Finish**.

The packaged files are saved in the **\output\packaging\live_update** folder. Packaged files are placed in timestamped folders to keep track of packages. The most recent package is in the top folder.



2. Copy the *template-name*_**template_live_update.zip** file from the **packaging** directory on your Business Modeler IDE client to a directory that is accessible by the administrators at other sites.

> Note:
>
> By default, packaged template files are located in the Business Modeler IDE workspace directory in the **output\packaging** folder under the project. To find the *workspace* location, choose **File→Switch Workspace**. For example, on a Windows system, they are saved by default to:.
>
> *install-location*`\bmide\workspace\`*version*`\`*project*`\output\packaging\live_update`

© 2025 Siemens

3. Administrators at other sites perform the live update by installing the live updates using TEM. In this scenario, TEM has already been used to install the template that the live updates are intended for.

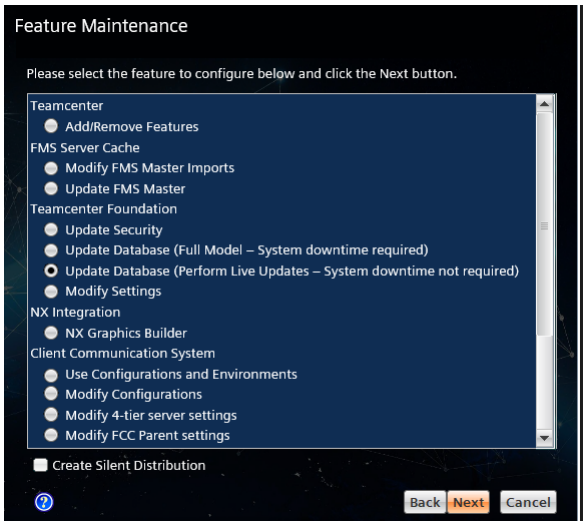
   a. Start Teamcenter Environment Manager (TEM).

   b. In the **Maintenance** panel, select **Configuration Manager** and click **Next**.

   c. In the **Configuration Maintenance** panel, select **Perform maintenance on an existing configuration** and click **Next**.

   d. The **Configuration Selection** panel displays the installed configuration. Click **Next**.

   e. In the **Feature Maintenance** panel, under the **Teamcenter Foundation** section, select **Update the Database (Perform Live Updates Only – System downtime not required)**. This updates the database with live updates that contain only nonschema data such as LOVs and rules.



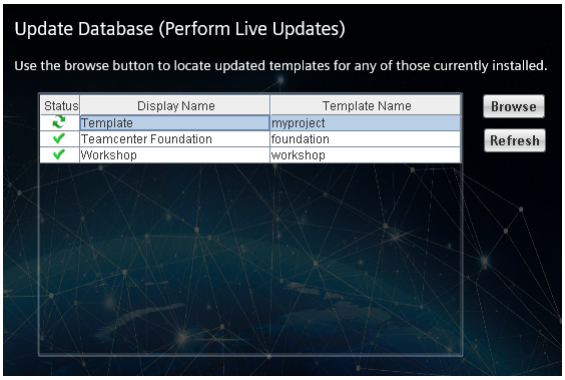
   f. Click **Next**

   g. In the **Teamcenter Administrative User** panel, type your user name and password to log on to the server. Click **Next**.

      The **Update Database** panel displays currently installed templates.

   h. Click the **Browse** button to navigate to the directory where the packaged template files are located. Select the updated *template-name*_**template_live_update.zip** file.

© 2025 Siemens



i. Select the template in the table to receive live updates and click **Next**.

j. In the **Confirmation** panel, click **Start**.

The updates are installed.

> Note:
>
> The system checks if the live update project is synchronized with the server. If there is data on the server that is not in the live update project, the update fails. You must synchronize the data model from the Business Modeler IDE, repackage, and attempt the update once more.
>
> If installation of the live updates fails, check the message in the TEM panel. Installation may have failed because the server you are attempting to install to does not have the **Live Update** preference set to accept live changes. In this case, you must ask the administrator of that production server to change the preference to accept the live updates.

k. To verify the installation of the revised template, log on to the production server and confirm that you can create instances of your newly revised data model.

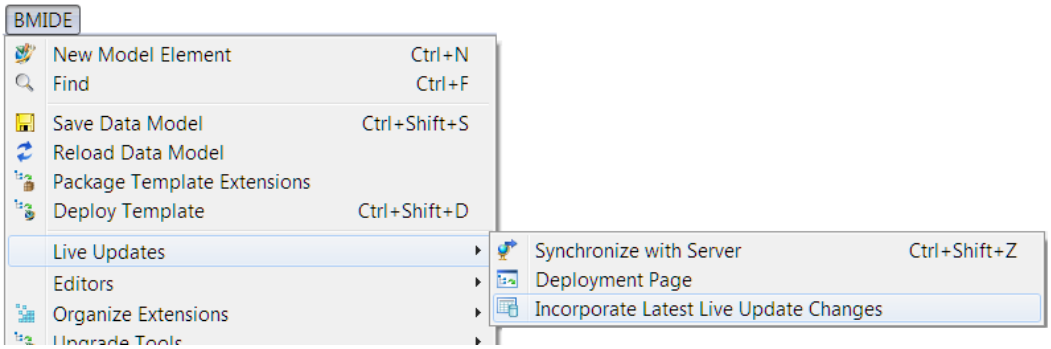
> Note:
>
> To see the live data changes, end users must log off and log on again, and wait for servers to cycle through the changes.

Incorporate latest live updates

At the next system downtime, you must incorporate all of the latest updates from the production environment into your standard template project. Run the Incorporate Latest Live Update Changes wizard to either update directly from the production server or update from a template file obtained from the production server using the **package_live_updates** utility.

1. Choose **BMIDE→Live Update→Incorporate Latest Live Update Changes**. Click **Next**.

© 2025 Siemens



2. Perform the following steps in the **Incorporate Latest Live Update Changes** dialog box:

   a. Click the arrow in the **Project** box to select the standard project into which you want to incorporate the live updates.

   b. Under **Database Site**, select the mode to provide the data model:

   - **Teamcenter Server**

     Select if you want to incorporate all the custom live updates data model on a server into your project.

     This option accesses the server directly to obtain the templates from the server.

   - **Template Package**

     Select if you want to incorporate all the custom live updates data model from a template file into your project. Use this option if you cannot access the server directly.

     Note:

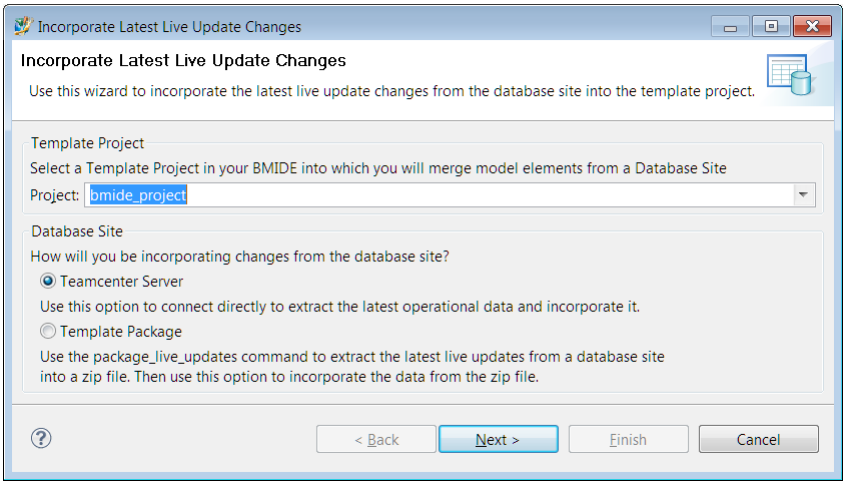
     Use the **package_live_updates** utility to generate the template package file from the server, for example:

     ```
     package_live_updates -u=jsmith -p=jsmithpassword -g=dba
      -template=testproject -dir=d:\scratch
     ```

     This packages all data model in the template, including all the live updates in the template.

   c. Click **Next**.

© 2025 Siemens



d. Depending on your selection, perform one of the following in the **Source Model** dialog box:

- If you selected **Teamcenter Server**, in the dialog box, type your user name and password and click **Connect** to log on to the server.
- If you selected **Template Package**, click the **Browse** button to the right of the **Live Update Zip** box to locate the packaged template's ZIP file.

e. Click **Next**.

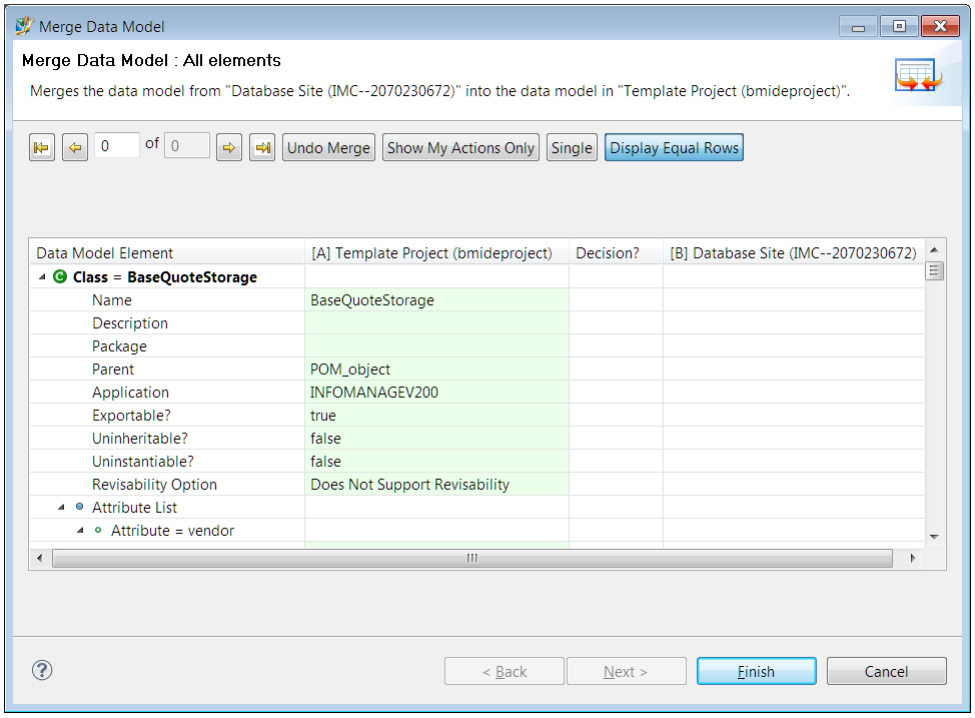
The system checks if the live update project is synchronized with the server. If there is data on the server that is not in the live update project, the Merge Data Model wizard runs, allowing you to resolve the conflicts.

> Caution:
>
> Be patient. The incorporation may take some time.
>
> To improve performance, Siemens Digital Industries Software recommends your machine have a minimum of 2 GB RAM to allow for other processes, with a minimum of 1 GB **RAM allocated to the Business Modeler IDE**.

3. In the **Merge Data Model** dialog box, resolve any conflicts that the merge compare identified between the data model in your target project and in the source.

> Note:
>
> Because there are many different scenarios you may encounter when performing a merge, samples are provided in the **Merge Samples** wizard. For tutorials, choose **BMIDE→Tools→Merge Samples**.

© 2025 Siemens



4. Click **Finish**.

   The merged data model is saved into your project.

   > Note:
   >
   > If multiple database sites are merged into your project, verify them by viewing the **Incorporated Database Site Information** dialog box on the **project properties**. Right-click the project, choose **Properties**, and in the left navigation pane, choose **Teamcenter→Incorporated Database Site Information**.
   >
   > The **Incorporated Database Site Information** dialog box displays the IDs, names, and the deploy consistency stamp for the merged sites. (To see a site's ID, use the Organization application in the rich client.)

5. Once you incorporate live update changes, you must perform a full model update.

   a. Package your template by choosing **BMIDE→Generate Software Package**.

   b. Start Teamcenter Environment Manager (TEM).

   > Note:
   >
   > You can also live update a template using the **tem** command line utility, for example:

© 2025 Siemens

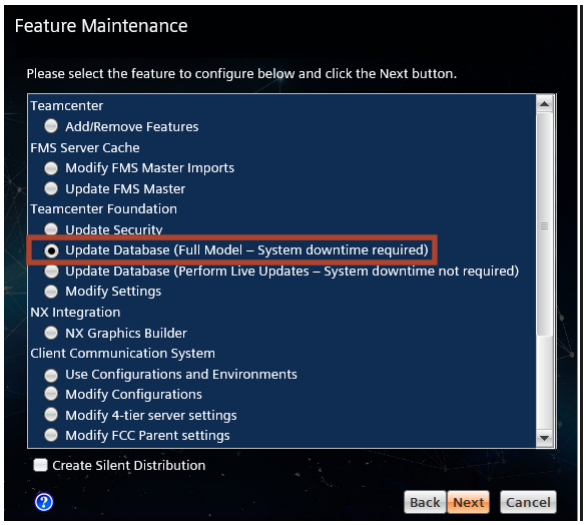
```
tem -update -live -templates=template-name-1,template-name-2 -path=location-of-template-files
-pass=password
```

c. In the **Maintenance** panel, select **Configuration Manager** and click **Next**.

d. In the **Configuration Maintenance** panel, select **Perform maintenance on an existing configuration** and click **Next**.

e. The **Configuration Selection** panel displays the installed configuration. Click **Next**.

f. In the **Feature Maintenance** panel, under the **Teamcenter Foundation** section, select **Update the Database (Full Model - System downtime required)**. This updates the database with the full data model in the standard (non live update) template.



g. Click **Next**

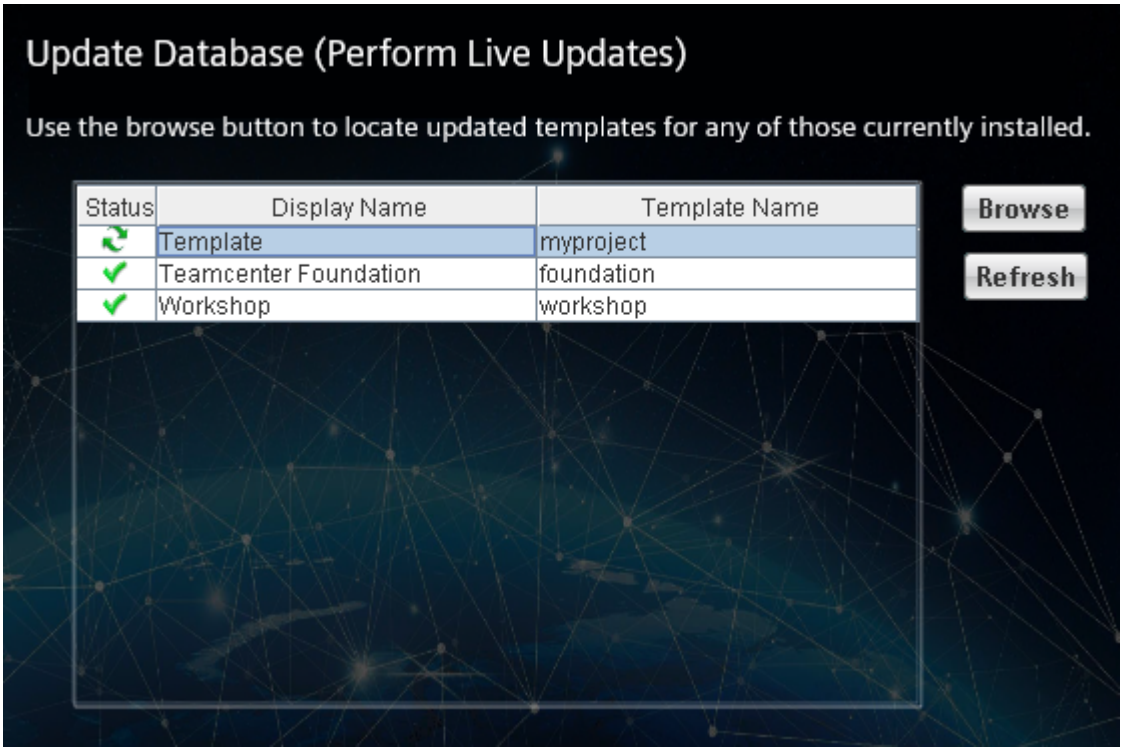
h. In the **Teamcenter Administrative User** panel, type your user name and password to log on to the server. Click **Next**.

The **Update Database** panel displays currently installed templates.

i. Click the **Browse** button to navigate to the directory where the packaged template files are located. Select the updated feature XML file.

j. Click **Next**.

© 2025 Siemens



k. In the **Confirm Selections** panel, click **Next**.

The template is installed.

> Note:
>
> TEM verifies that your template package contains the latest live updates that were performed on the production environment. If the template package does not contain the updates, TEM blocks the update and advises you on the actions to take. This is to protect you from inadvertently deleting the administrator updates.

Live updates for multiple administrators

Live update process: multiple administrators

Follow this process when you are one of many administrators who perform live updates to a production server.

1. **Set up the multiple administrator environment**.

2. **Synchronize data** in the live updates project with data on the server.

3. **Incorporate the latest live updates from the production site** by running the **package_live_updates** utility and provide the updates to the developer environment.

© 2025 Siemens

Set up the multiple administrator environment

Just as for a single administrator environment, in a multiple administrator environment, ensure that a Business Modeler IDE client is installed for each administrator and have each administrator create a live update project.

1. Ensure that the custom template installed on the server is **enabled to receive live updates**.

2. Ensure that the **Live Update preference is configured**.

3. **Install the Business Modeler IDE client for each live updates administrator** on a machine with a minimum of 2 GB of RAM.

4. Have each administrator **create a live update project**.

   During creation of the live update project, the latest live update configuration is downloaded from the server to the administrator's new live update project.

Synchronize data in a multiple administrator environment

Each administrator's Business Modeler IDE must be synchronized to ensure that the administrator is viewing an accurate display of the configurations in the system.

For example, administrator 1 adds a **Steel** value to a materials LOV and deploys his changes, and administrator 2 adds an **Aluminum** value to the same LOV. When administrator 2 tries to deploy the template, the Business Modeler IDE automatically synchronizes the project and downloads the latest model from the database and displays a comparison to the configurations in the project.

Perform a synchronization when the system shows an out-of-synchronization error message when you deploy live updates using the Business Modeler IDE or TEM.

1. When you receive an out-of-synchronization message, select the project to be synchronized and choose **BMIDE→Live Update→Synchronize with Server** or click the **Synchronize** button  on the toolbar.

   The Merge Data Model wizard displays the differences between your Business Modeler IDE and the database.

2. Review the changes and resolve any conflicts that the merge compare identified between the data model in your target project and in the source. To automatically perform the merge, click the **Auto Merge** button.

© 2025 Siemens

Note:

Because there are many different scenarios you may encounter when performing a merge, samples are provided in the **Merge Samples** wizard. For tutorials, choose **BMIDE→Tools→Merge Samples**.

Incorporate the latest live updates from the production sites using the package_live_updates utility

At the next system downtime, you must incorporate all of the latest live updates from the production environment into the developer environment. Use the package_live_updates utility to generate the template package file from each administrator's server site. If you do not do this, Teamcenter Environment Manager (TEM) prevents you from making a system update to each of the sites that you omitted from the latest live updates.

1. At each site, run the **package_live_updates** utility, for example:

   ```
   package_live_updates -u=jsmith -p=jsmithpassword -g=dba
   -template=testproject -dir=d:\scratch
   ```

   This packages all data model in the template on the server, including all the live updates in the template.

2. Incorporate changes from each resulting packaged template one by one, using the **Incorporate Latest Live Update Changes** wizard. Select the **Live Update Zip** option to select the packaged template file.

Localization and live update

Caution:

Do not use live update to place localization changes on a production server. Instead of using live update, install localized templates using Teamcenter Environment Manager (TEM) or Deployment Center.

Because localization changes can contain changes to the schema, and schema changes cannot be live updated to a production server, attempting to use live update to deploy localization changes could result in the following error:

```
Error Code: 515062  Error Message: Class referenced
```

This error occurs in a couple of notable situations.

© 2025 Siemens

Example:

You want to add the **Localization** button to a property, so you set the **Localizable** property constant to **true**. Usually setting a constant value does not result in changes to any classes or attributes. However, in this case, an additional hidden attribute is created on the class used by the associated business object.

- On the **Item** business object, on the **object_desc** or **object_name** property, you set the **Localizable** property constant to **true** and then attempt to live update the change. The 515062 error occurs.

Example:

To make a list of values available for setting a property value in the client, in the Business Modeler IDE you attach an LOV to a property. The attachment results in a hidden attribute being placed on the source business object of the property. If you attempt the live update the change, the 515062 error occurs.

Live updates reference

Data elements that can be updated live

The data elements that can be updated live are listed and described in the **Live Update** pane in the rich client **Options** dialog box.

To see the lists, in the rich client, choose **Edit→Options**, and then in the **Options** category, select **Live Update**.

To see an element description, select the element in one of the lists.

You can further restrict the elements by moving them in that dialog box from the **Selected for Live Updates** list to the **Available** list. To re-enable them, move them back.

Synchronize data model

Synchronize the data model to ensure your live update project and dependent templates have the most up-to-date data model.

1. Select the project to be synchronized and choose **BMIDE→Live Update→Synchronize with Server**, or click the **Synchronize** button  on the toolbar.

   The **Teamcenter Login** dialog box is displayed.

2. In the **Project** box, select the live update project you want to synchronize.

© 2025 Siemens

3. Type the password in the **Password** box.

4. Click **Connect**.

   The connection to the server is established.

5. Click **Finish**.

   The data model is synchronized.

6. If there are changes to the custom template, the Merge wizard is shown based on the live update synchronization preference settings to help resolve merge conflicts:

   - If the preference is set to **Always show Merge Tool during synchronization** (the default), the Merge wizard is shown to help reconcile the conflicts.

   - If the preference is set to **Show Merge Tool only for conflicts that need my resolution**, if there are no change conflicts, all elements of the data model are automerged, and the Merge wizard is not shown. If there are change conflicts, all other elements except for the change conflict elements are automerged, and the Merge wizard is shown.

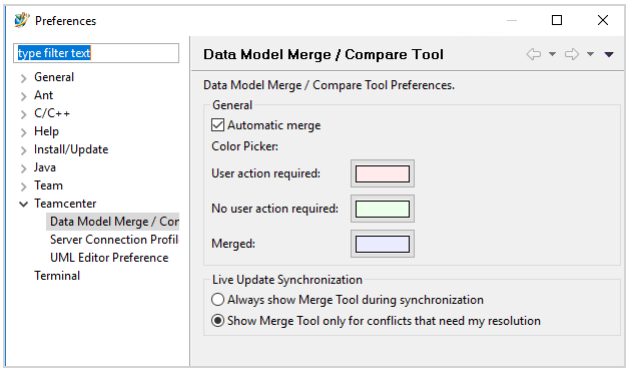
Set data model merge and compare preferences

The Incorporate Latest Live Update Changes wizard displays differences between your live update project and what is in the database. You run the wizard to ensure your live update project has all the latest custom live updates elements. You have a few choices for the appearance and operation of the wizard.

Perform the following steps to set preferences for the Incorporate Latest Live Update Changes wizard:

1. Choose **Window**>**Preferences**.

2. In the **Preferences** dialog box, choose **Teamcenter**>**Data Model Merge / Compare Tool**.



© 2025 Siemens

3. Select **Automatic Merge** to automatically merge all differences except for the change conflict differences when the wizard is launched. If the check box is cleared, no merges are automatically performed when the wizard is launched.

4. In the **Color Picker** pane, select the color for identifying the following conditions:

User action required	Manual merge action needed. The default color is red.
No user action required	No merge action needed. The default color is green.
Merged	A merge action has been performed. The default color is blue.

5. In **Live Update Synchronization**, select one of the options:

   - **Always show Merge Tool during synchronization**

     Displays the Merge wizard during synchronization even if the tool can automatically reconcile all data model differences without user intervention. This option allows you to inspect the incoming changes made by other users to the live update data in the database. In this mode, you must merge each element.

     If the synchronization command runs but there are no elements to be merged, the Merge wizard is not shown. A message states that there are no differences to be merged. You must click **OK** in this dialog box so that the database site information can be saved.

   - **Show Merge Tool only for conflicts that need my resolution**

     Automatically reconciles all data model differences without user intervention. This option allows you to let the Business Modeler IDE take care of any merges, letting you focus on any merge conflicts where you must make a decision.

     - If any change conflict differences are found, the Incorporate Latest Live Update Changes wizard is launched so you can address the conflicts.

     - If there are no change conflict differences, the Incorporate Latest Live Update Changes wizard is not displayed and the Business Modeler IDE automatically synchronizes all data model differences.

Make live updates visible to end users

After live updates have been installed on a production server, the updates are visible to end users who log on after the update. However, users logged on when the update is made must log off and log on again to ensure that the servers cycle through the changes.

© 2025 Siemens

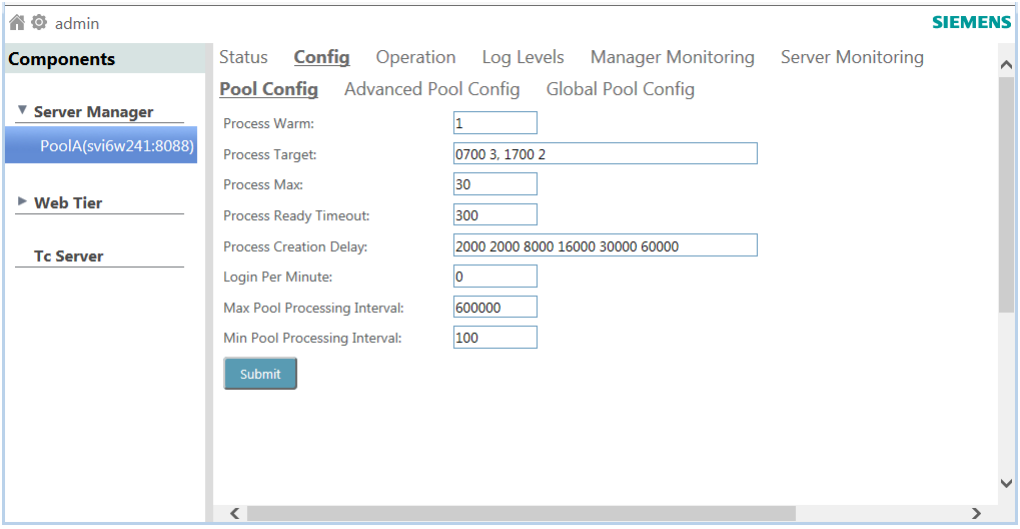
When each user logs on, a server is started and assigned to the client. During startup, the types, properties, and LOVs in the database are built into a cache for performance reasons. Each server has its own cache. When a user updates an LOV to the database, each client still sees the original LOVs and rules because server caches are not updated. Administrators ask users to log off and log on again to see the changes so that a new cache is built.

In four-tier environments, administrators can use the server manager to restart servers at off-peak hours to ensure all live update changes are reflected in servers.

A system administrator can facilitate warm server refresh.

1. Launch the Teamcenter Management Console

2. Under **Server Manager**, select the pool, click **Config**, and click **Pool Config**.



3. In the **Process Warm** box, type **0** and click the **Submit** button.

   This clears unassigned warm servers.

4. In the **Process Target** box, type the time and number of assigned servers, for example, **0700 3, 1700 2** and click the **Submit** button.

5. Wait for the manager to remove the warm servers and restore the **Process Warm** and **Process Target** boxes to desired values.

Package live updates in the database

Run the **package_live_updates** utility to package the data model for a template that resides on the server. The template package file that is created includes all the live updates in the template. This is similar to packaging a template within the Business Modeler IDE, but this utility is run on the Teamcenter server. This utility can only be run on a template that is enabled for live updates deployment.

© 2025 Siemens

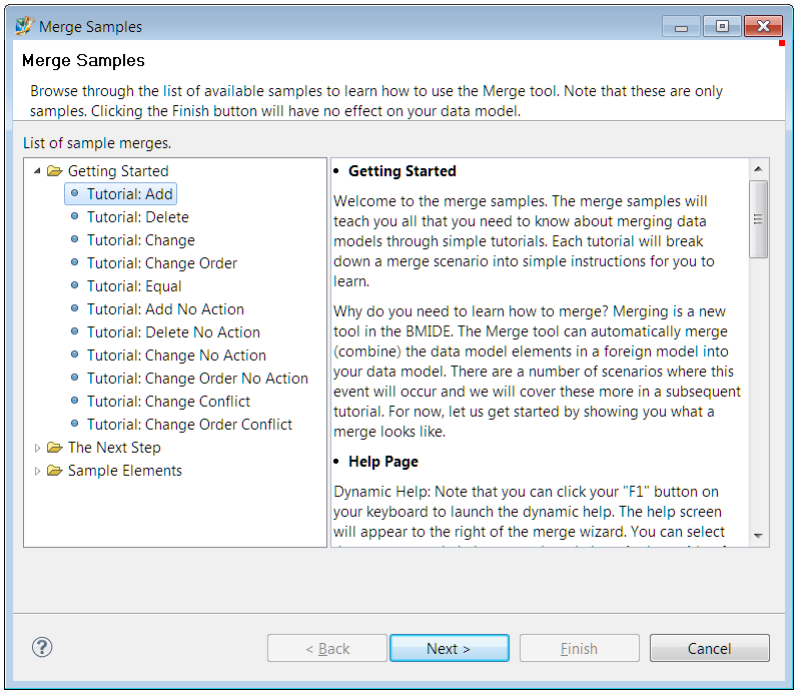
For example:

```
package_live_updates -u=user-id  -p=password -g=group
-template=template-name -dir=directory
```

After running this utility, you can **merge the live updates** that the template contains into a template on your system by running the merge wizard.

Study the merge samples

To become familiar with merge scenarios, study the merge samples. Choose **BMIDE→Tools→Merge Samples**.



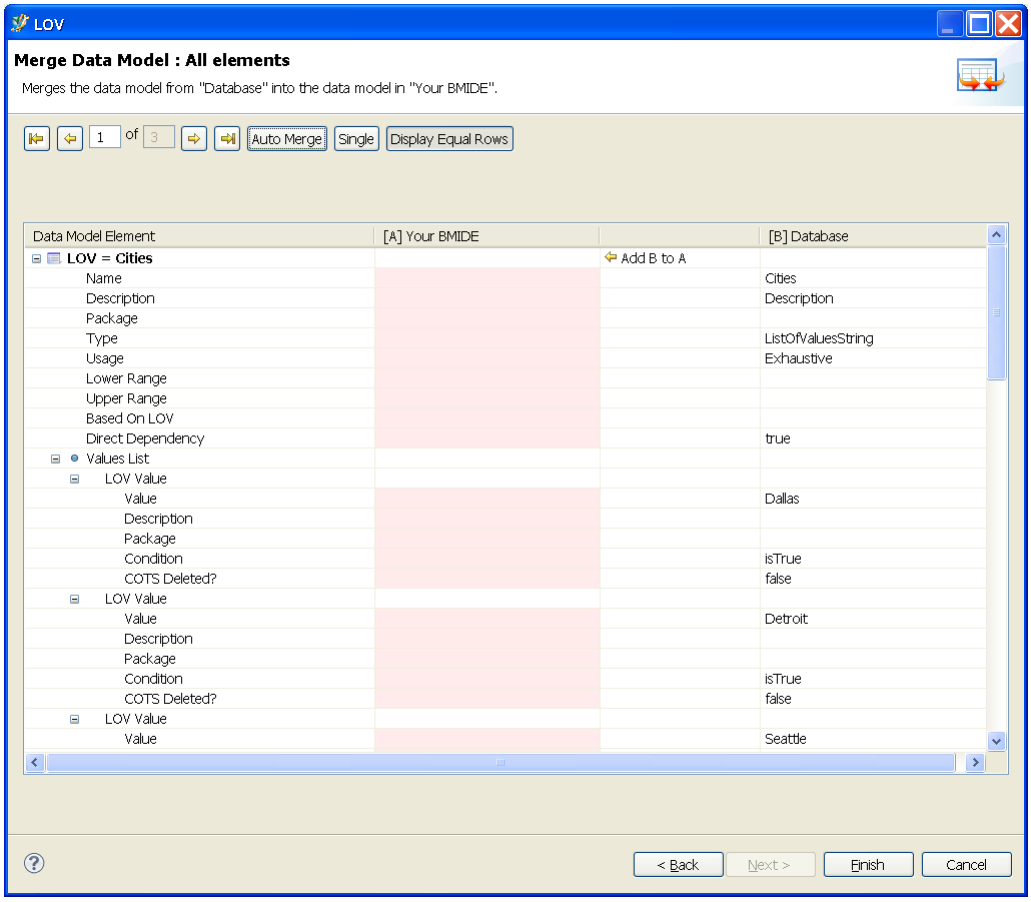
Key features of the merge tool are:

- Automerge

  Click the **Auto Merge** button to let the tool merge all changes for you.

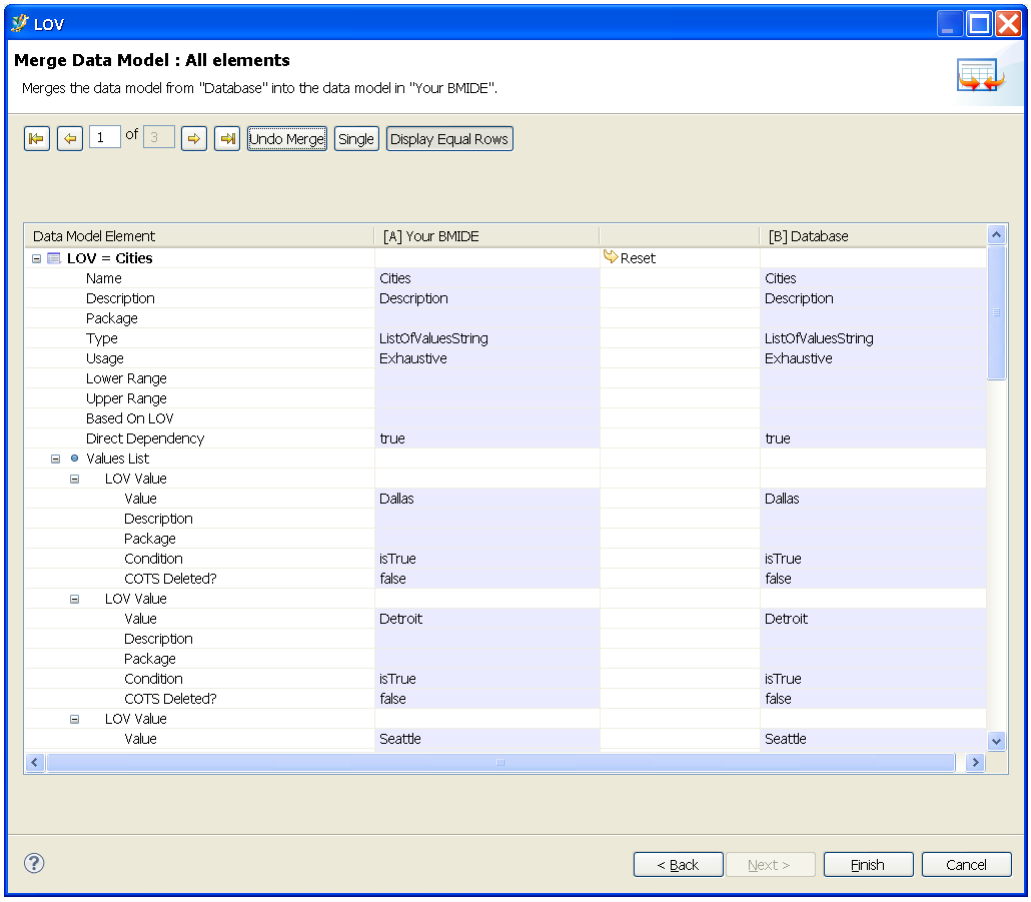
  Following is an example before automerge.

© 2025 Siemens



Following is an example after automerge.

© 2025 Siemens



- Stepping

  Click the forward and backward arrows to step through each change.

- Filtering

  To see only the rows where there are differences, click the **Display Equal Rows** button so that it is not selected. This is a toggle button. When it is not selected, only rows with different values are shown. When it is selected, all rows are shown, including all those with equal values.

Configure FMS to connect to multiple sites

If you deploy to multiple sites using the **Deployment Page**, you must configure FMS to point to the different sites.

1. In this example, set up the Business Modeler IDE to use site A, for which you have created a live update project. Using the **Deployment Page**, you want to deploy to site B and site C.

2. For site B, open the *TC_ROOT***fsc\fmsmaster**.**xml** file. The file has entries for the **fsc id** values (which include the host and port) and the **enterpriseid** value. Note these values, for example:

© 2025 Siemens

```
<fscgroup id="mygroup">
      <fsc id="FSC_MYHOST_usermywork" address="http://MYHOST:4544"
ismaster="true">
        <volume id="00f34da72a9e826b7aa8" enterpriseid="-2106885464"
root="d:\Siemens\myworkshop\volume1" priority="0" />
        <transientvolume id="63ae660c47df33be2d024a346fe4512c"
enterpriseid="-2106885464"
root="C:\\Temp\\transientVolume_mywork" />
      </fsc>
      <clientmap subnet="127.0.0.1" mask="0.0.0.0">
        <assignedfsc fscid="FSC_MYHOST_usermywork" priority="0" />
      </clientmap>
</fscgroup>
```

3. Launch Teamcenter Environment Manager (TEM) for site A and in the **Feature Maintenance** panel under **FMS Server Cache**, select **Modify FMS Master Imports** and click **Next**.

4. On the **Ext. Sites** page, click the **Add** button and add the information for site B (the site ID, the FSC ID, and the host and port that you noted in the previous step). Keep the priority set at **0**.

5. Complete the process and restart the FSC service for site A.

6. Repeat the steps for site C:

    a. Open the **fmsmaster*.xml** file for site C.

    b. Obtain the enterprise ID, the FSC ID, the host and port values.

    c. Launch TEM on site A and add site C.

    d. Restart the FSC for site A.

Merge samples

To access merge samples, choose **BMIDE→Tools→Merge Samples**.

Browse through the list of available samples to learn how to use the merge tool. Note that these are only samples. Clicking the **Finish** button has no effect on your data model.

Basic template tasks

Add a template to Business Modeler IDE reference templates

Whenever you add a template to the server, in order to configure or extend that template, you must add the same template to the Business Modeler IDE reference templates folder. To add templates to your Business Modeler IDE reference templates folder, you use Teamcenter Environment Manager (TEM).

1. Start Teamcenter Environment Manager (TEM).

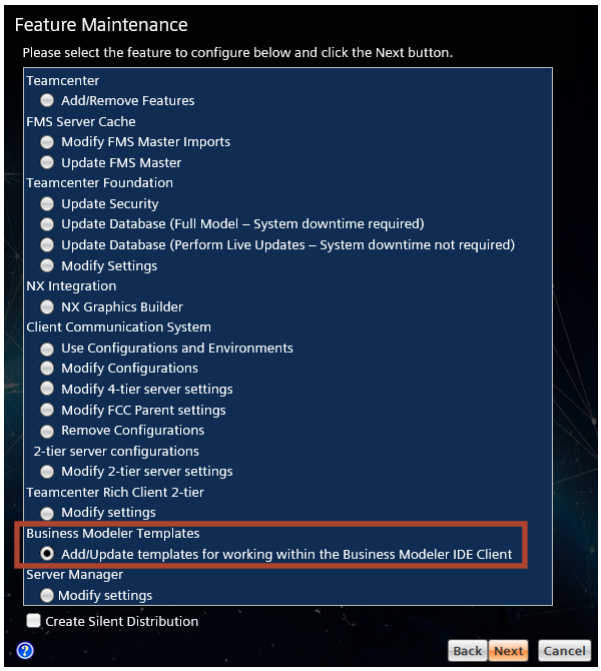
© 2025 Siemens

2. In the **Maintenance** panel, select **Configuration Manager** and click **Next**.

3. In the **Configuration Maintenance** panel, select **Perform maintenance on an existing configuration** and click **Next**.

4. In the **Configuration** panel, select the configuration from which the corporate server was installed. Click **Next**.

5. In the **Feature Maintenance** panel, under the **Business Modeler IDE** section, select **Add/Update Templates for working within the Business Modeler IDE Client**.

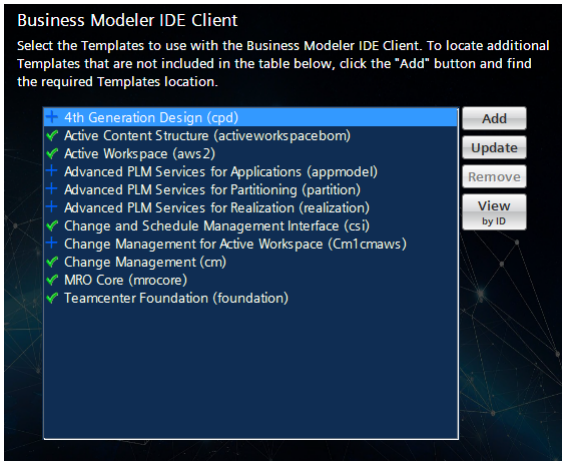


6. Click **Next**.

   The **Business Modeler IDE Client** panel displays templates currently installed to the Business Modeler IDE reference templates folder.

7. In the **Business Modeler IDE Client** panel, click the **Add** button to select standard Teamcenter templates.

© 2025 Siemens



> Tip:
>
> To select your own templates (for example, from a vendor) click the **Browse** button and locate the directory where the template files are located. (For example, the standard Siemens Digital Industries Software templates are located in the installation source in the **tc** directory.)

Click **Next**.

8. In the **Confirmation** panel, click **Next**.

   The new templates are installed to the Business Modeler IDE reference templates folder.

The templates are now available for addition to your custom projects.

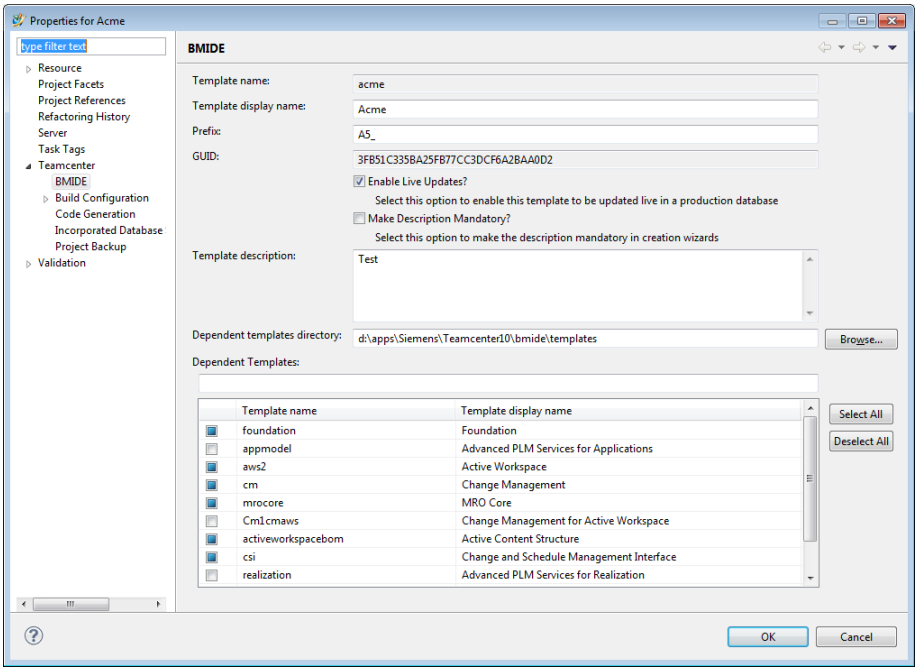
Add a dependent template to a Business Modeler IDE project

To add dependent templates to your project so that you can configure or extend them, perform the following steps in the Business Modeler IDE.

1. Right-click the project and choose **Properties**.

2. In the **Properties** dialog box, in the left pane choose **Teamcenter** > **BMIDE**.

© 2025 Siemens



3. In the **Dependent Templates** section, select the new templates.

4. Click **OK**.

View Business Modeler IDE template project properties

You can view and change project properties, such as the template display name and the location of generated code.

1. Right-click the project and choose **Properties**.

2. Choose **Teamcenter** > **BMIDE** in the left pane. The **BMIDE** dialog box allows you to set the general properties of the project.

   a. In the **Template display name** box, you can change the name your template displays in Teamcenter Environment Manager when this template is installed on a production server.

   b. In the **Prefix** box, you can change the **naming prefix for new objects** created in the template.

      When you name a new data model object, a prefix from the template is automatically affixed to the name to designate the object as belonging to your organization, for example, **A4_**.

© 2025 Siemens

> Caution:
>
> Do not change the prefix in this box without considering the consequences. Changing the prefix adds the new prefix to the names of all objects created in the future, but it does not change prefixes already placed on objects in the template. This can result in two different prefixes on objects in the same template.

c. The **GUID** box shows the globally unique identifier (GUID) automatically assigned to this project. You cannot change this value.

d. Select the **Enable Live Updates?** check box to specify that the Business Modeler IDE template project can have a live updates project created in conjunction with it. This option *must* be selected to **enable the creation of a live updates project**.

e. Select the **Make Description Mandatory?** check box to make the **Description** box required when creating new data model elements in the Business Modeler IDE.

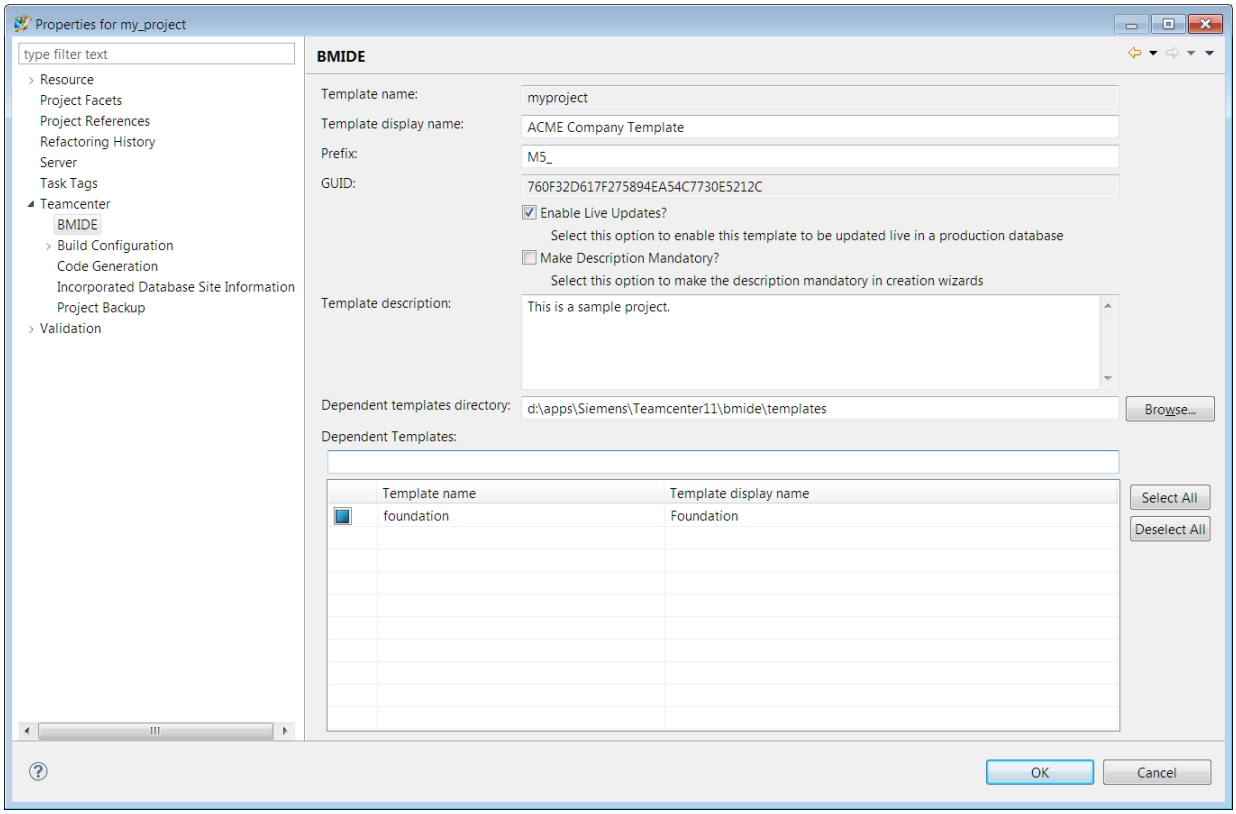
f. In the **Template description** box, you can change the description of the template that appears in Teamcenter Environment Manager.

g. In the **Dependent templates directory** box, you can change the directory where the base templates are read from. These are the templates on top of which you are adding your custom template.

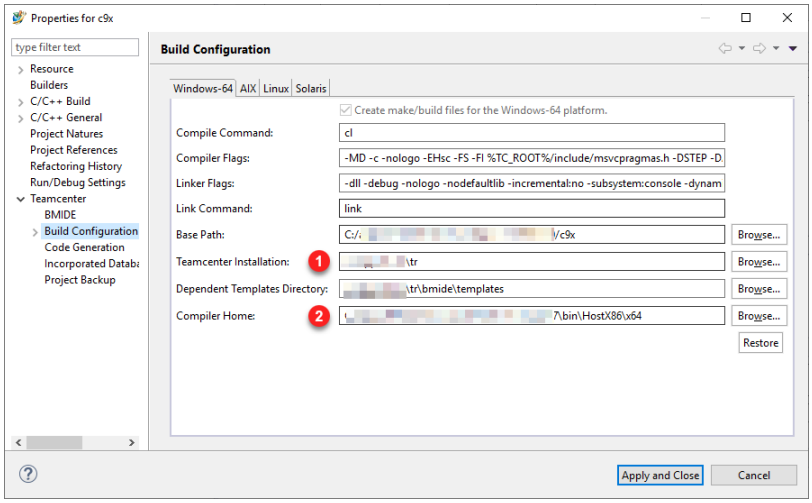
> Note:
>
> When you upgrade the Business Modeler IDE to a new version, you can select an existing project and use this box to point to the directory where the new model files are installed.

h. In the **Dependent Templates** pane, you can choose the templates used to provide the data model for the project.

© 2025 Siemens



3. Choose **Teamcenter→Build Configuration** in the left pane. The **Build Configuration** dialog box is used to set up C++ code generation. You only need to enter this information if you plan to write code.



a. Click the **Browse** button to the right of the **Teamcenter Installation** box to select the location where the Teamcenter server is installed, for example, *install_location***Siemens****Teamcenter***version**.

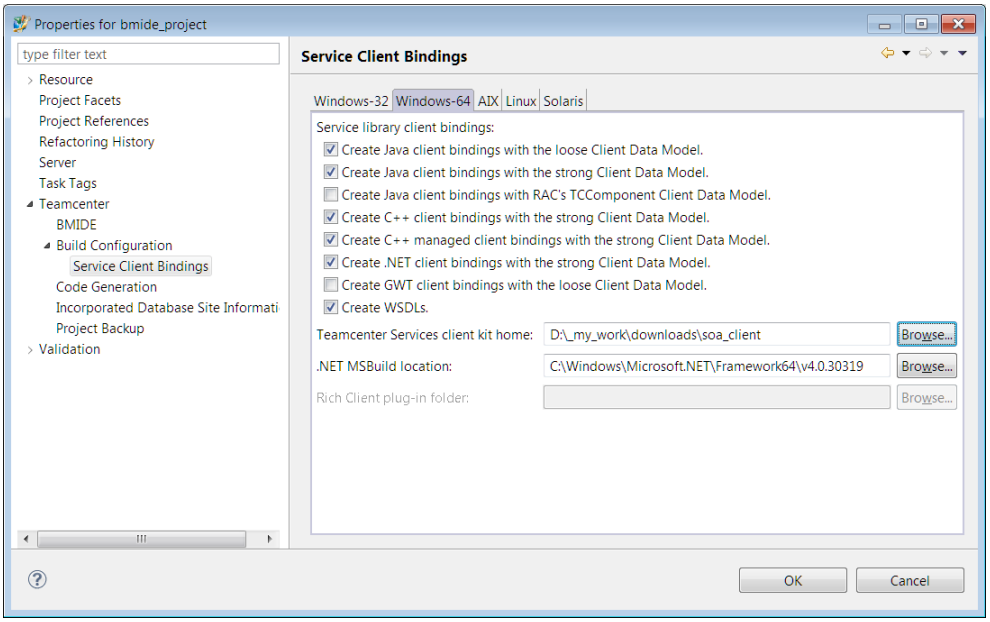
© 2025 Siemens

b. Click the **Browse** button to the right of the **Compiler Home** box to select the location where your C++ compiler is located. For example, if your platform is Windows and you are using Microsoft Visual Studio, browse to *compiler-install-path***VC\bin**.

   For information about supported C++ compilers, see Hardware and Software Certifications knowledge base article on Support Center.

c. You may add additional compiler or link options for your environment. Use the **Restore** button to restore all defaults

4. Choose **Teamcenter→Build Configuration→Service Bindings** in the left pane. The **Service Bindings** dialog box is used to set the kinds of client binding files to create when you generate services code. Services are used by system administrators to connect their company's applications to Teamcenter. Client binding files are used to connect the client to the Teamcenter server and are written in the language of the client, for example, C++, Java, or .NET.



a. In the **Service library client bindings** section, select one or more of the check boxes for the type of client applications that you want to expose these service operations to.

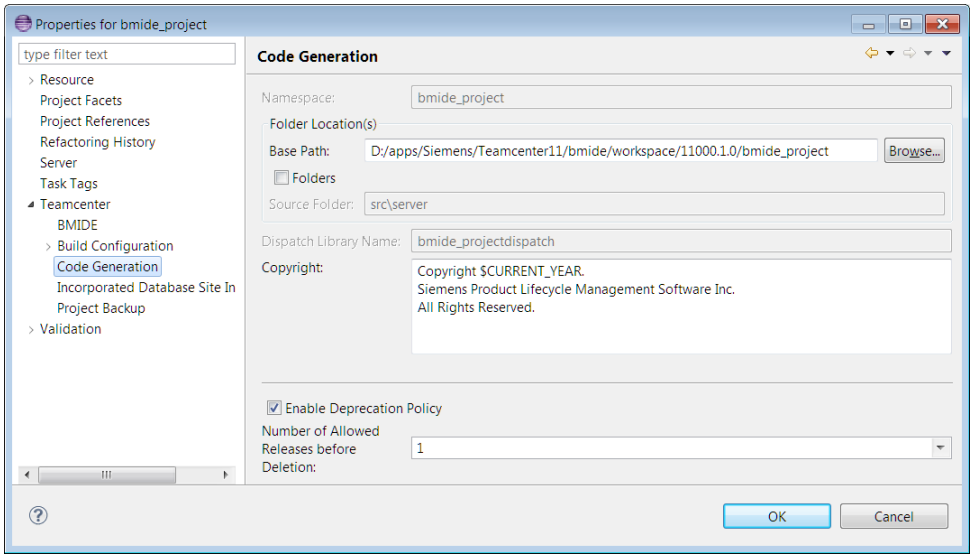
b. Click the **Browse** button to the right of the **Teamcenter Services client kit home** box to select the location where the service oriented architecture files have been extracted from the **soa_client.zip** file, for example, *install_location***soa_client**.

c. If you want to create .NET bindings, click the **Browse** button to the right of the **.NET MSBuild location** box to select the location where Microsoft .NET is installed.

d. If you selected **Create Java client bindings with RAC's TCComponent Client Data Model**, click the **Browse** button to the right of the **Rich Client plug-in folder** box to select the location where the rich client is installed.

© 2025 Siemens

5. Choose **Teamcenter→Code Generation** in the left pane. Use the **Code Generation** dialog box to change the location where generated code is placed.



a. The **Namespace** box represents the predetermined namespace for all generated code. Namespaces allow for grouping code under a name to prevent name collisions. Collisions result when different libraries may use the same class names for different classes. Each template can specify a namespace associated with the C++ classes and with data types created in the template.

   For example, the custom classes for the Widget template can specify **Widget** as the namespace for its classes. This allows both the Foundation and Widget templates to have a class called **Item** by using their different namespaces as the differentiator.

b. Click the **Browse** button to the right of the **Base Path** box to change the location where the generated code is placed.

c. Select the **Folders** check box if you want to change names of the folders to contain generated code.

   In the **Source Folder** box, type the name of the folder to contain the source files where you write business logic. The default is **src\server**.

d. The **Copyright** box displays the copyright text placed into each generated file.

e. Select the **Enable Deprecation Policy** check box to allow for removal of obsolete objects from the project. *Deprecation* means announcing that something is no longer supported and that it will be removed in a future product release.

f. Click the arrow in the **Number of Allowed Releases before Deletion** box to select how many releases before objects can be deleted from the project.

© 2025 Siemens

g. Click **OK**.

If you changed any templates to be read by the project, the IDE reads in the data model from the template XML files. If there are any errors, they are displayed in the **Console** view.

> Caution:
>
> Resolve all errors before using the Business Modeler IDE. Otherwise, you may risk extending corrupted data.

6. Choose **Teamcenter→Incorporated Database Site Information** in the left pane. Use this dialog box to see the site name, operational deploy consistency stamp, and date updated for each site you have incorporated in live update. If you have multiple database sites to track, the project properties page shows the consistency stamp and date updated information for each site. This page helps you remember the sites that you have incorporated or failed to incorporate.

   Clicking the **Update Data Model From Site** runs the **Incorporate Latest Live Update Changes** wizard.

   Whenever an administrator performs a live update on a database, the production database's data model is not synchronized with the development environment data model. At the next system downtime, if you package and deploy from the development environment to the production environment, the update is blocked because that your packaged template does not contain the latest operational updates.

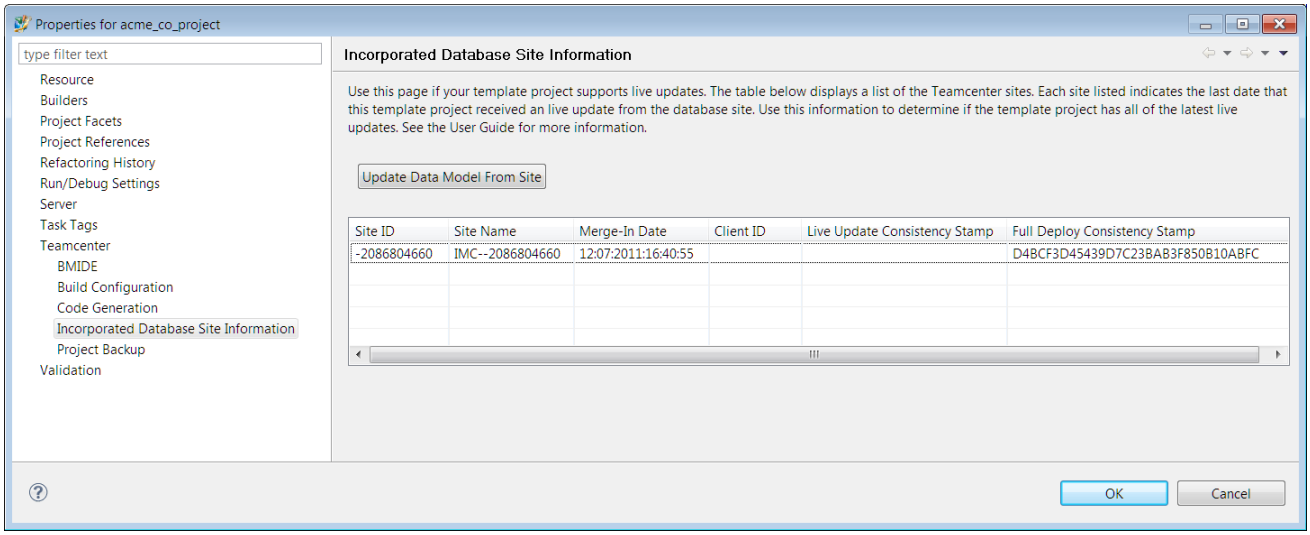
   To ensure that you always have the latest operational data updates, wait until all operational data updates have been performed at the production site. Then incorporate the latest through one of the two options (directly from the server or through a package file). When you deploy from your development environment, Teamcenter Environment Manager (TEM) validates that your template includes the latest updates. This is done through consistency stamp tracking.

   For each live update, a new consistency stamp (CS) is updated in the production database. When you use the Incorporate Latest Operational Data Changes wizard from the development Business Modeler IDE to incorporate the operational data updates from the production database, the wizard also takes a snapshot of the consistency stamp and stores it with the Business Modeler IDE template in the development environment. For each site you incorporate, you can see the site name, operational deploy consistency stamp, and date updated on the **Incorporated Database Site Information** dialog box.

   You can compare the consistency stamp in the Business Modeler IDE to the consistency stamp in the database site by extracting a report using a command prompt at the database site:

   ```
   bmide_manage_templates -u=username -p=password -g=dba -option=list
   ```


© 2025 Siemens



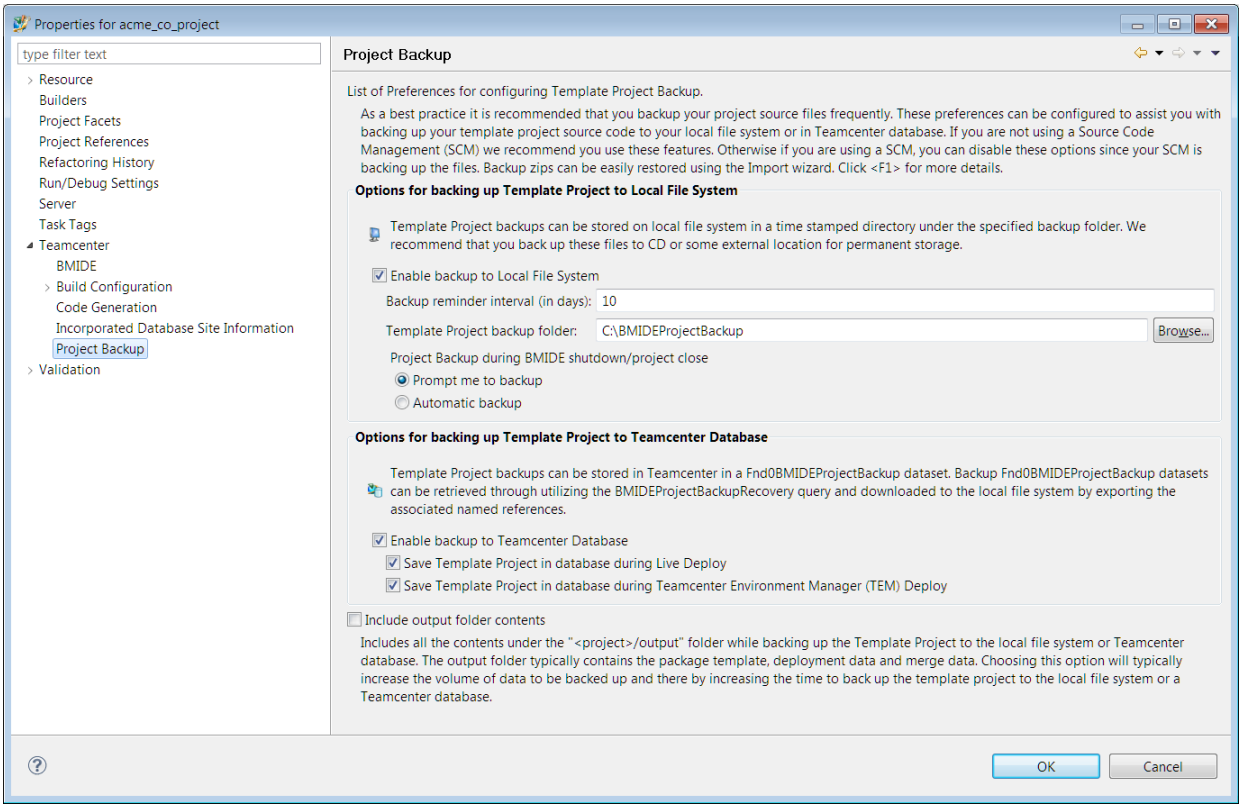
7. Choose **Teamcenter→Project Backup** in the left pane. Use the **Project Backup** dialog box to set how project data is saved in the system.

   a. Select the **Enable backup to Local File System** check box to save the backup files to your computer at Business Modeler IDE shutdown or project close.

   b. Select the **Enable backup to Teamcenter Database** check box to save the backed-up data in a **Fnd0BMIDEProjectBackup** dataset when you perform a live update or an installation using Teamcenter Environment Manager (TEM).

   c. Select the **Include output folder contents** check box to back up the contents of your project's **output** folder.

© 2025 Siemens



Edit the template feature file

Each template has a **feature file** that contains information about the template. You can edit this feature file to change how the feature is displayed in the Teamcenter Environment Manager (TEM) **Features** panel.

1. **Install the template using TEM**.

   By default, the new template is displayed as a feature in the **Features** panel under the **Extensions** group.

2. To change the text that is displayed when you point your mouse arrow over the feature name (in the example, **My Feature**), edit the **TextID** node in your template's *name***Bundle***language_country***.xml** file, for example:

   ```
   <TextID name="template-name.description" text="This is my new
   feature."/>
   </IDMap>
   ```

3. To add a new group under the **Extensions** group to hold the new feature:

   a. Navigate to where TEM is installed: *install-location***install****install**.

© 2025 Siemens

b. Copy an existing **group_*name*.xml** file and rename it, for example, **group_mygroup.xml**.

c. Edit the new group file to create a new group name, for example:

```
<group>
   <name>My Group</name>
   <group>package</group>
   <bundle>TcBundle.xml</bundle>
</group>
```

This results in the following grouping in the **Features** panel:

Extensions
My Group
My Feature

Push a template to the reference directory

You can create a template and copy it into the template reference directory so that you can build another project on top of it. This is useful when you work on two or more templates simultaneously that have dependencies on each other.

The reference directory is where templates are stored that you build your projects on top of, and is located at *install-location***bmide\templates**. You point to the reference directory when you first create a project.

1. Choose **BMIDE→Tools→Push Template to Reference Directory**.

   Click **Next**.

2. In the **Push Template** dialog box, perform the following steps:

   a. In the **Project** box, click the arrow to open a list of projects, and then select the project for which you want to generate the template files.

   b. If you want to generate the template files in a directory other than the default directory, then clear the **Use default location** check box, and then to the right of the **Target folder** box click the **Browse** button and choose the directory.

   c. Click **Finish**.

   The template files are created in the indicated directory.

3. Right-click the project you want to work on and choose **Reload Data Model**. This reloads all data model elements from the project source and the templates in the reference directory. Check the console for any conflicts and resolve them.

© 2025 Siemens

Now you can create a project that builds on top of this new template.

Improving performance by closing and opening projects

The IDE supports multiple projects and multiple data models. Each project uses additional memory. To optimize performance, you may want to have only one project open at a time. To close a project, right-click the project and choose **Close Project**. When a project is closed, it displays a closed folder symbol and the data model is removed from the views.

To reopen a project, right-click the project and choose **Open Project**. This reloads the data model and makes it accessible in the various views. You can also reload the data model for a project by right-clicking a project and choosing **Reload Data Model**.

Deleting projects

To stop working with a project, Siemens Digital Industries Software recommends that you close the project by right-clicking it and choosing **Close Project**.

However, if you want to remove the project altogether, Siemens Digital Industries Software strongly recommends you *back up the project files first* to ensure they are archived. A project contains the master definitions for any data model created with the project, and if you delete a project, those master definitions are gone for good.

After you archive the project files, you can delete the project. Right-click the project and choose **Delete**. If you need to use an archived project again, you can **import** it.

Templates reference

Templates overview

A template is a container that holds all of the Business Modeler IDE data model definitions for a specific customer, site, industry, or application. A template data model is comprised of schema, business objects, business rules, and configuration definitions.

Teamcenter has a base template called **foundation** that contains all of the core definitions to run the default Teamcenter corporate server installation. The foundation template is required at a minimum when installing Teamcenter. Additional templates with their own data model can be installed on top of the foundation template to create a more robust system. In effect, all of the definitions in each installed template are combined to give the customer the net result of all data model extensions in one system (for example, all classes, attributes, business objects, business rules, LOVs, and so on).

A template can have dependencies. Each new template must declare at a minimum that it is dependent on the foundation template, and can also declare additional dependencies on other templates. If a template is dependent on other templates, TEM ensures during the installation of the template that all other dependent templates are installed before your template is added to the database.

© 2025 Siemens

Creating a template project

Templates are created through the Business Modeler IDE client. A template is managed via a project in the Business Modeler IDE. A project is an Eclipse term that means a top-level directory in the user's workspace. A workspace is the directory that the Business Modeler IDE uses to manage all resources. From the Business Modeler IDE client user point of view, a template project is a directory that contains all of the resources and source code related to building and deploying the template project.

Because the Business Modeler IDE workspace can manage more than one project at a time, the Business Modeler IDE can manage more than one template project at a time. This means that two or more independent data models can managed and displayed simultaneously by the Business Modeler IDE. Typically, a user of the Business Modeler IDE client may only work with one template at a time. However, some field services users or internal developers may want to work on multiple templates.

To create a new template project, choose **File→New→New Business Modeler IDE Template Project**.

When creating a new template project, name each of the following:

- Template name

  This is the name of the template. The name must be a globally unique name that identifies the template. The name must consist of letters or digits and must be all lowercase. This name can be a total of 180 character length. When coming up with a template name, you should choose a descriptive name.

- Template display name

  This is the displayable name of the template. This name is displayed to the users of TEM when picking templates to install. The name can consist of characters, digits, and spaces, and can use a mixture of uppercase and lowercase letters. The name should be globally unique so as to identify your template from others in TEM.

- Template description

  This is the description that appears in Teamcenter Environment Manager for this template.

Aligning Siemens Digital Industries Software template development and customer template development

Development by customers and by Siemens Digital Industries Software is aligned by using the Business Modeler IDE to create new data model definitions. By using the Business Modeler IDE, both groups can create data model extensions and create templates. In addition, they can leverage productivity enhancements and validation tools to ensure that templates are adding extensions within acceptable limits and can upgrade safely from release to release.

© 2025 Siemens

Adding extensions to the template

After a template project is created, you can extend the data model and business rules by creating new definitions using the Business Modeler IDE. Your data model extensions are saved in the template project.

One of the benefits of using the Business Modeler IDE is the strict validation that it performs on the extensions in your template.

- After the Business Modeler IDE first loads the dependent extensions, each of your template extension definitions is loaded and validated against the existing model. If any validation errors occur, they are displayed in the **Console** view in the Business Modeler IDE.

- You can also load and validate your model using the **Reload Data Model** menu command (available in most views within the Business Modeler IDE).

This validation tool becomes very important whenever you add a template dependency, manually edit a source file during a SCM merge from another user, add a new version of the dependent templates, or upgrade to the next version. Whenever any of these conditions occur, right-click a Business Modeler IDE project view and choose the **Reload Data Model** command to reload your model and have it validated. If no errors occur, you can be sure that your template can install correctly to a server.

Synchronize all data directories with the latest templates

If your Teamcenter environment has more than one data directory installation referring to the same Teamcenter database, you must synchronize all data directories with the templates. This ensures the clips rules file, the PLM XML schema file, and .des files are all regenerated in the data directories any time a template is installed or updated in the database.

For example, if you have a Teamcenter database installed with a single data directory, and you create additional data directories to connect to the same database, any future template installation or update to the database should ensure all data directories are synchronized.

Follow these steps for every data directory installation any time a template is installed or updated in the database from any of the multiple data directory installations. In this example, the data directories are **data1** and **data2**. Your templates in the database were changed in **data1**, and you must synchronize **data2**.

1. Launch Teamcenter Environment Manager in maintenance mode.

2. Select the configuration where you created the **data2** directory.

3. If you made a template installation from **data1**, do the same in **data2** by selecting **Add/Remove features**. Because the template was already installed from a different data directory, **data1**, the template installation from **data2** directory does *not* update the database but only the files in the **data2** directory.

© 2025 Siemens

4. If you made a template update from **data1**, do the same in **data2** by selecting **Rebuild/Update the database**. Because the template was already updated in the database from a different data directory, **data1**, the template update from **data2** directory does *not* update the database but only the files in the **data2** directory.

5. If you have additional data directories, repeat steps 1 to 4.

Remove a template

Remove a template process

Follow this process to remove a template from your Business Modeler IDE project from your Teamcenter installation. Through manual steps and the **bmide_remove_template** utility, the process also removes instances of the data model elements in your database. You can remove a custom, third-party, or any COTS template except the Foundation template.

1. Remove the dependency on the template to be removed, if a dependent template exists. For example, your environment (the Business Modeler IDE project and your database) may have a **vendor** template that has the Teamcenter Foundation and **customer** dependent templates. First, you must remove **customer** dependent template from the **vendor** template.

   a. Open the **vendor** template and from the project **Properties** dialog box, clear the **customer** check box to remove the dependency, and click **OK**.

   b. Reload the **vendor** template project by right-clicking the project and choosing **Reload Data Model**. If there are any errors, they are displayed in the **Console** view.

   > Caution:
   >
   > Resolve all errors before using the Business Modeler IDE. Otherwise, you may risk extending corrupted data.

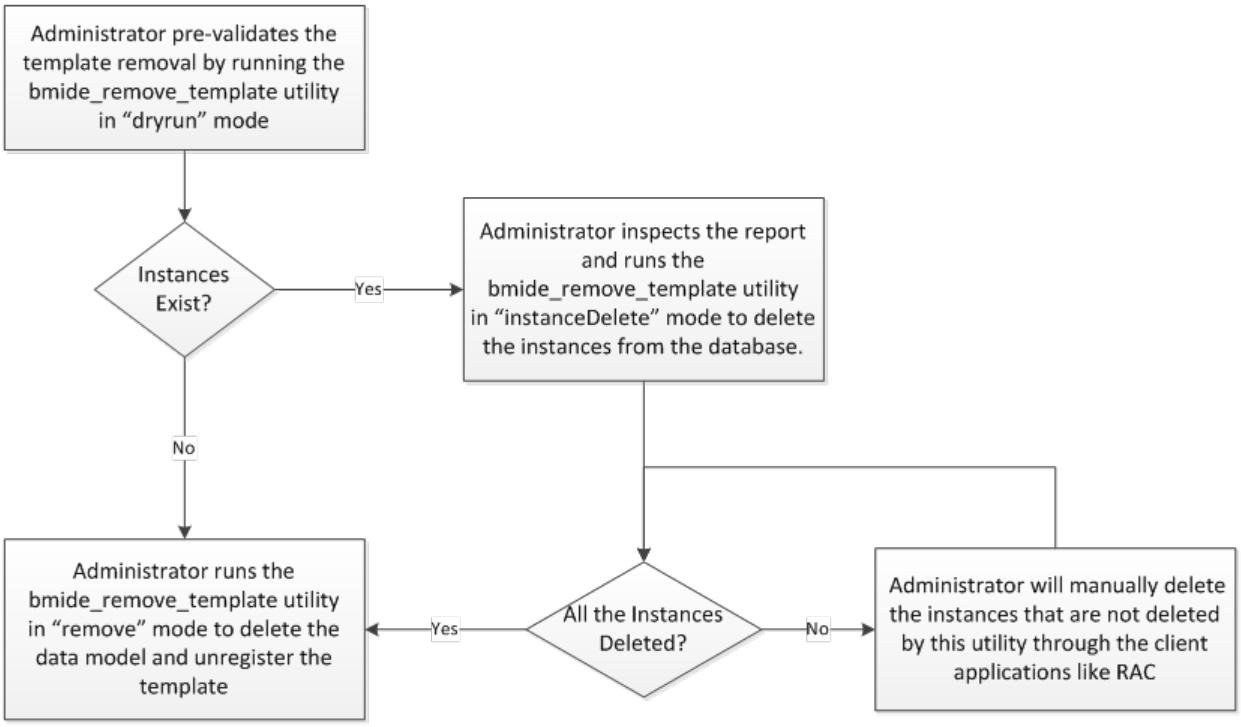
   c. Delete the **customer** template project from the **master.xml** file in the **extensions** folder.

   d. Remove the **customer** template from the Business Modeler IDE dependent templates directory.

   e. **Package the template** and **install your custom template to the test server database by using Teamcenter Environment Manager (TEM)**.

2. Remove the template by performing multiple steps with the **bmide_remove_template** utility. These steps are necessary to inspect a report of the template's data model instances, remove the data model instances (automatically and some must be removed manually), and finally remove the template from your database. If there are any instances in the database of the data model being removed, the **bmide_remove_template** utility does not remove the template.

© 2025 Siemens



a. **Prevalidate that the template can be removed**. Run the **bmide_remove_template** utility in **dryrun** mode to report any instances of the data model being removed or validate that the template can be removed.

b. **Remove business object instances**. Run the **bmide_remove_template** utility in **instanceDelete** mode to delete any instances of the data model being removed. As needed, manually remove referenced instances with the rich client.

c. Remove the template. Run the **bmide_remove_template** utility in **remove** mode. There are requirements before removing a template:

- Database instance data from the template must be removed before the template can be removed.
- Other templates that are dependent on the template must be removed before the template can be removed.

d. Repeat the steps with the **instanceDelete** and **remove** modes until the template is removed by the **bmide_remove_template** utility.

3. Remove the template related files from the *TC_DATA* and *TC_ROOT* folders.

- Remove the *template-name* folder from the *TC_DATA***install****models.xml** file.
- Update the TEM related files in the *TC_ROOT***install** folder:

© 2025 Siemens

- Remove all entries to *template-name* in the **configuration.xml** file.
- Remove all entries to *template-name* in the **uninstall.xml** file.
- Delete the *template-name* folder.
- Delete the **feature_***template-name***.xml** file from the *TC_ROOT***install\install\modules** folder.
- Delete the *template-name* bundle files from the *TC_ROOT***install\install\lang** folder.

Prevalidate the template removal

Run the **bmide_remove_template** utility in **dryrun** mode to validate if a template can be removed. No instance deletion processing occurs in this mode. The **bmide_remove_template** utility performs validation checks and generates a summary report to analyze the data model instances in the database for the specified template. A Teamcenter administrator must run this utility.

The **bmide_remove_template** utility also generates a delta XML file that contains the data model elements to be removed. The utility compares the model extracted from the database with the consolidated model without the selected template to generate the delta XML file. This file is the input to the **bmide_instance_delete** utility, which attempts to delete instances in the database that are from these data model elements.

Before a template can be removed, all of the model elements defined in the template must be removed from the Teamcenter database. The model element to be removed are identified in the delete section of the delta XML file. Any existing instances of those model elements in the Teamcenter database must be deleted before the model element can be removed. Instances of these data model elements must be deleted from the Teamcenter database (other data model elements are ignored):

- Standard types
- Form types
- Dataset types
- Note types
- Tools

Remove a template

Run the **bmide_remove_template** utility in **remove** mode to remove the data model definitions from the database and unregister the template. A Teamcenter administrator must run this utility.

Before removing a template, note the following points:

© 2025 Siemens

- You can remove a custom, third-party template or any COTS template except the Foundation template.
- Before a business object definition can be removed, the business object instances must be deleted. You must first delete all instances of each definition in the template that is to be removed. If you cannot or do not want to delete the instances, you cannot remove the template.
- Data model elements defined in any template can be inherited by creating subtype data model definitions in a template. Instances of subtype data model definitions also must be deleted from the database. 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. The non-data model elements, such as workflow templates, organization, and libraries, are not removed.
- Before removing a template, Siemens Digital Industries Software strongly recommends that you back up your existing Teamcenter environment.

Currently, Teamcenter Environment Manager (TEM) does not support the removal of the Business Modeler IDE templates.

The Teamcenter utilities to remove a template do not support:

- Adding a new template and removing another template in a single deploy action.
- Removing a template during an upgrade while installing a patch.
- Cleaning up business object references in Access Manager and saved queries.
- Updating TEM-related files from the *TC_ROOT***install** folder:
  - Updates to **configuration.xml** and **uninstall.xml** files.
  - Removing the feature and bundle files.
- Removing the libraries or binary files that were installed with a template feature.

Remove business object instances

The **bmide_instance_delete** utility is run as a command line utility in a Teamcenter environment. This utility finds and deletes instances from a Teamcenter database before attempting to remove the template. A Teamcenter administrator must run this utility. Before deleting instances, Siemens Digital Industries Software strongly recommends that you back up your existing Teamcenter environment.

The **bmide_instance_delete** utility supports two methods:

- A delta XML file with the **-file** argument.

© 2025 Siemens

  - The **-file** argument specifies the location of a delta XML file created by the **bmide_remove_template** utility.
  - Only qualified model elements that exist in the delete section of the delta XML file are processed:
    - Standard types
    - Form types
    - Dataset types
    - Note types
    - Tools

- A formatted instance list with the **-instances** argument.
  - The **-instances** argument specifies a formatted list of model elements to process. Single or multiple values (comma separated) are permitted.
  - The format is *type-qualifier-name*:*model-element-name*. For example:

    ```
    -instances="Type:MyItem"

    -instances="Type:MyItem,Type:YourItem"
    ```

The **bmide_instance_delete** utility generates a **Database Instance Summary report**. The contents of the report depends on the **-mode** argument:

- **count**

  Instances are not deleted from the Teamcenter database in this mode. The **Database Instance Summary** report from the **count** mode allows you to see how many instances of the business object definition exist in the database.

- **dryrun**

  Instances are not deleted from the Teamcenter database in this mode. The **Database Instance Summary** report from the **dryrun** mode allows you to analyze the data to be deleted before you actually delete the data. After you confirm that all data can be deleted, run the **bmide_instance_delete** utility in **delete** mode to delete the instances.

- **export**

  Instances are not deleted from the Teamcenter database in this mode. The **Database Instance Summary** report from the **export** mode allows you to see the details of the instances (and their

© 2025 Siemens

references) of the business object definition that exist in the database that are exported out for backup purposes. To delete instances so that you can remove the Business Modeler IDE template, run the **bmide_instance_delete** utility in **dryrun** mode first so that you can analyze the data to be deleted before you actually delete the data. After you confirm that all data can be deleted, run the **bmide_instance_delete** utility in **delete** mode to delete the instances.

- **delete**

  Instances are deleted from the Teamcenter database in this mode, with exceptions that are not deleted as follows:

  - Instances that are part of a BOM structure.
  - Instances that have replica data exported to another Teamcenter site.
  - If a reference property needs to be set to **NULL** to disconnect the relationship to an object that needs to be deleted, but that reference property is defined as **NULLS_NOT_ALLOWED**.

  The **Database Instance Summary** report from the **delete** mode allows you to see detailed information on whether the instances (and their references) of the business object definition that exist in the database are deleted. For each instance that could not be deleted, the report lists the error message why the instance could not be deleted.

  > Tip:
  >
  > The **-tcxmlOutput** argument is recommended when deleting instances because the delta XML file is a backup of the deleted instances.

Database Instance Summary report

An instance deletion report is generated from the **bmide_instance_delete** utility. The content of this report depends on the **-mode** argument:

- When **-mode** is **count**, the summary report is formatted as in this example:

```
Database Instance Summary:

The following table summarizes the amount of database instance
information for each model element identified for deletion.

Total        BMIDE Element        Definition
-----------------------------------------------------------------
  12,988    Type            Item
  16,002    Type            Item Revision
  16,002    Type            Item Master
  16,002    Type            Item Revision Master
       4    Tool            Notepad
```


© 2025 Siemens

```
-----------------------------------------------------------------
  61,234 Total
-----------------------------------------------------------------
```

- When **-mode** is **dryrun**, the summary report is formatted as in this example:

```
Database Instance Summary: 21 (Total)

Total        Instances that         Instances that          BMIDE Element    Definition
                can be Deleted         cannot be Deleted
-------------------------------------------------------------------------------------
------
       1           1                    0                 Type        Item
       4           4                    2                 Type        Item Revision
       3           3                    1                 Type        Item Master
       1           1                    0                 Type        Item Revision
Master
      12           4                    0                 Tool        Notepad
-------------------------------------------------------------------------------------
------
      21              18                    3 Totals
-------------------------------------------------------------------------------------
------


=====================================================================================
======

Database Instance List

Number    Deleted?           Type                    Name
-------------------------------------------------------------------------------------
------
      1   Yes                Item Revision Master    000091/A
      2   Yes                Item Revision           b2myitem1 (Item/Rev ID is
000091/A)
      3   Yes                Item Master             000091
      4   Yes                Item                    b2myitem1 (Item ID is 000091)
...

=====================================================================================
======
Database Instance Details:


Item:
    Display Name : Item
    Object Type  : Type
    Reporting 1 of a total of 1 instances...

      1) Instance Name        : 0805_item (Item ID is 000003)
         Instance UID         : xlRRfr4DAAgcRA
         Owning User          : admin
         Can this be Deleted? : Yes

         Reference Count : 5
           1. Reference Name (Type): <Empty> (IMAN_master_form)
              Reference UID        : wsVVglXR$my_DB
```


© 2025 Siemens

```
              Owning User          : <No Owning User>
              Can this be Deleted? : Yes
...
```

- When **-mode** is **export**, the summary report is formatted as in this example:

```
Database Instance Summary:

Total        BMIDE Element        Definition
------------------------------------------------------------------------------------------
------
  12,988        Type             Item
  16,002        Type             Item Revision
  16,002        Type             Item Master
  16,002        Type             Item Revision Master
       4        Tool             Notepad
------------------------------------------------------------------------------------------
------
  60,998 Total
------------------------------------------------------------------------------------------
------

==========================================================================================
======

Database Instance Details:
...
```

- When **-mode** is **delete**, the summary report is formatted as in this example:

```
Database Instance Summary: 21 (Total)

Total        Instances that         Instances that         BMIDE Element     Definition
                are Deleted            are not Deleted
------------------------------------------------------------------------------------------
------
       1           1                    0                  Type        Item
       4           4                    2                  Type        Item Revision
       3           3                    1                  Type        Item Master
       1           1                    0                  Type        Item Revision
Master
      12           4                    0                  Tool        Notepad
------------------------------------------------------------------------------------------
------
      21              18                    3 Total
------------------------------------------------------------------------------------------
------

==========================================================================================
======

Database Instance List

Number    Deleted?           Type                    Name
------------------------------------------------------------------------------------------
------
      1   Yes                Item Revision Master    000091/A
      2   Yes                Item Revision           b2myitem1 (Item/Rev ID is
```


© 2025 Siemens

```
000091/A)
      3   Yes               Item Master            000091
      4   Yes               Item                   b2myitem1 (Item ID is 000091)
...

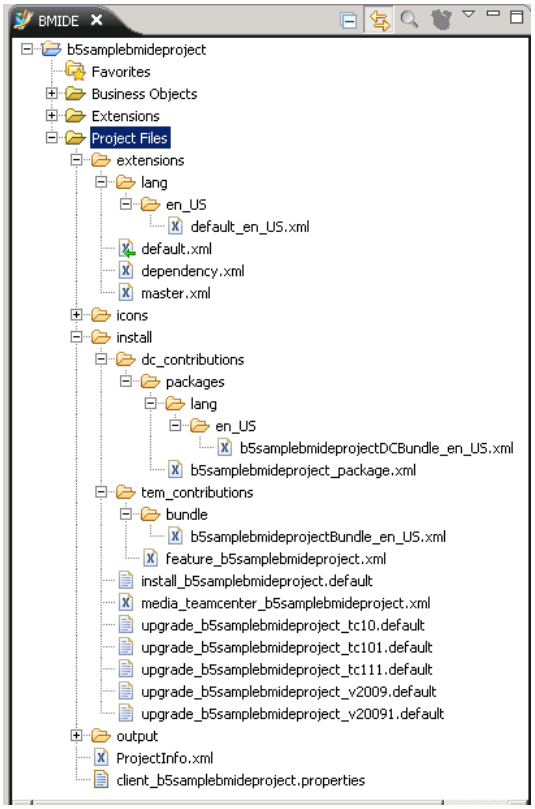
================================================================================
======

Database Instance Details:
...
```

Files in a template project

Overview of files in a template project

After you create the Business Modeler IDE template project, you see the following file structure. Notice that many of the files include the template name embedded in the name of the file.



Template project directory

The project directory is the top-level directory for the template. The Business Modeler IDE expects this structure to exist so that it can manage the template project correctly.

The template project directory and all of its contents should be managed in a source control management system (SCM). This is recommended because the template project is treated as source

© 2025 Siemens

code. An SCM is the best way to manage source code, especially when two or more users are making concurrent changes. An SCM provides robust tools that integrate with the Business Modeler IDE for handling file merges, graphical file differences, and syncing to the latest code checked in from other users.

Project extensions directory

The **extensions** directory contains three types of files: source files, the dependency file, and the master file.

- Source files

  Source files are XML files that contain the template extensions to the data model and business rules. The definitions can be saved into any source file. The Business Modeler IDE can load definitions in any order.

  The user of the Business Modeler IDE can determine the number of source files, the file names, and structure. Subdirectories can also be created under the extensions directory to suit the needs of the template developers.

  If developers use an SCM and two or more of them change a file in their own SCM branch, they must manually merge the XML changes. It is suggested that more source files are better than one. Developers should spread out their definitions into different files based on functionality or business objects, or any developer defined organization. Spreading out definitions makes it easier to manage file changes and avoid merge conflicts with files if there is more than one developer working on a template.

- **dependency.xml**

  The **dependency.xml** file contains the following:

  - Name of the template.
  - Names of the dependent templates.
  - Custom prefixes designated for the project.
  - Display name of the template.
  - GUID of the template.

  The optional flag indicates if the template is optional for Teamcenter. Only the foundation template is required (**optional=false**). All other templates are optional (**optional=true**).

- **master.xml**

© 2025 Siemens

The **master.xml** file is a specialized XML file. This file contains the master include list of all XML source definition files for your template project.

The order of the files does not matter because the Business Modeler IDE can load definitions in any order. However, the Business Modeler IDE loads faster if the definitions are in order of dependency. For example, if **Class2** is parented under **Class1**, **Class1** should be defined before **Class2**.

Project install directory

The **install** directory contains all the files that support install and upgrade of the template in TEM and Deployment Center.

- **Feature file**

  A feature file is used to present your template as a choice in TEM for install and upgrade. All templates must be installed through a feature file. A feature file declares the name of your template, a localized description, its dependencies, and the zip files that go with the template. The feature file also contains the commands necessary to get your template installed or upgraded.

- **Media file**

  The media file contains project information, such as the project display name, prefix, GUID, and description. To see the project information that is placed in this file, right-click the project and choose **Properties**, and then choose **Teamcenter** in the left pane of the **Properties** dialog box.

- **Bundle files**

  The bundle files are used to provide localizable strings for your TEM feature file and Deployment Center package file. Typically, this file includes the description for your template feature.

- **Installation script**

  The installation file is required for each template feature. The installation file contains a set of ordered commands that installs your template into a Teamcenter server.

- **Upgrade scripts**

  The upgrade files assist TEM with upgrading your template. Each template can provide upgrade scripts, but they are not required. They are only required if you have additional objects to be installed beyond what is managed inside your template (for example, process templates). If you do need an upgrade script, a separate script must be provided for each version of upgrade that is supported.

Output directory

The **output** directory is a directory where files are generated from the template source code. Typically the files in this directory do not need to be placed under control of an SCM because they would be

© 2025 Siemens

potentially different for each user. Some directories are only created when using the Business Modeler IDE.

- **deploy** directory

  When the deploy wizard is used, the Business Modeler IDE consolidates all of the individual sources files into a template and places the template and dependency files in this directory before it sends it to the server for processing.

- **merges** directory

  This directory contains data on data merged into the project from the database during live updates.

- **packaging** directory

  The **packaging** directory is used by the Generate Software Package wizard. The wizard places the template into a package that is then used by TEM to install or upgrade the template on a server.

- **reports**

  This directory contains generated reports.

- **tcplmxml** directory

  This directory is used by Global Multi-Site to generate the TC XML file based on the model in the Business Modeler IDE.

- **upgrade**

  This directory contains files related to upgrading the template project to the most recent version of the Business Modeler IDE.

.project file

This file is used by the Business Modeler IDE and contains specific information about the type of template project the directory contains. Without this file, the Business Modeler IDE cannot recognize this directory as a template project directory.

ProjectInfo.xml file

The **ProjectInfo.xml** file contains additional project information, such as settings used for building and compiling. To see the project information that is placed in this file, right-click the project and choose **Properties**, and then choose **Teamcenter** in the left pane of the **Properties** dialog box.

© 2025 Siemens

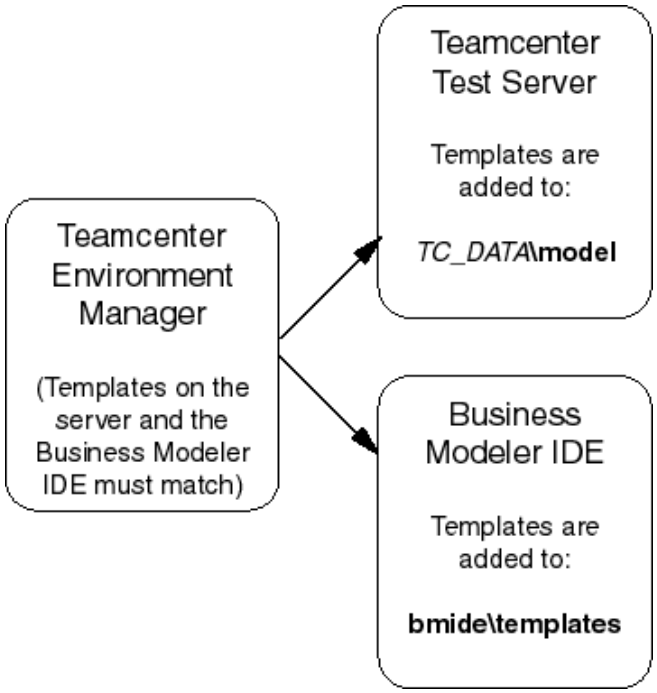
Templates installation reference

Templates installation process

Install the initial template

To successfully use the Business Modeler IDE, you must understand how to install templates to the Business Modeler IDE and the server. The key thing to remember is that the Business Modeler IDE and the server are completely separate, and if you install a template to the server, you must separately install the same template to the Business Modeler IDE.

When you use Teamcenter Environment Manager (TEM) to initially install a server and the Business Modeler IDE, you must install the same templates to both. The templates on each must match (see the figure below).



1. Run the Teamcenter Environment Manager and proceed to the **Features** panel.
2. Under **Base Install**, select **Teamcenter Foundation** and **Business Modeler IDE 2-tier** or **Business Modeler IDE 4-tier**.
3. Under **Extensions**, select other templates as needed.
4. Click **Next**.
5. In the **Business Modeler IDE Client** panel, select the **Teamcenter Foundation** template and the same templates that are installed to the server.

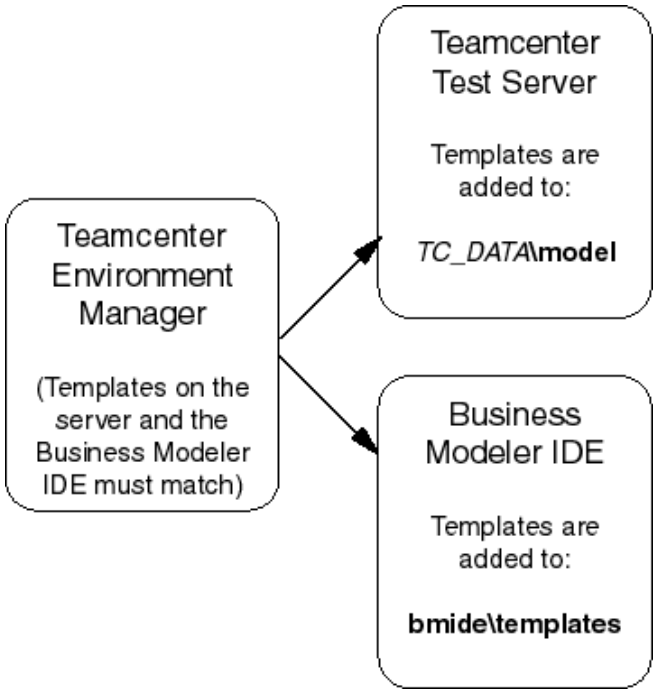
© 2025 Siemens

6. After you install the Business Modeler IDE, run the Business Modeler IDE and create a project by selecting **File→New→New Business Modeler IDE Template Project**.

   When you create the project, in the **Dependent Templates** pane, select the same templates that are installed on the server.

Install a subsequent template

After you initially install templates to a server and the Business Modeler IDE, you can subsequently go back and add templates (see the following figure). Because the templates on a server and the Business Modeler IDE must match, when you install a new template to an existing server, you must install the same template to the Business Modeler IDE.



1. Perform the following steps to install a new template to the server (test or production):

   a. Run the Teamcenter Environment Manager and proceed to the **Feature Maintenance** panel.

   b. Select **Add/Remove Features** and click **Next**.

   c. In the **Features** panel, select the template, or click **Browse** to locate the template.

   > Note:
   >
   > Standard Teamcenter templates are located on the Teamcenter software distribution image in the **tc** folder.

2. Perform the following steps to install a new template to the Business Modeler IDE:

© 2025 Siemens

a. Run the Teamcenter Environment Manager and proceed to the **Feature Maintenance** panel.

b. Select **Add/Update Templates for working within the Business Modeler IDE Client** and click **Next**.

c. In the **Business Modeler IDE Client** panel, click **Add** to add a standard Teamcenter template, or click **Browse** to locate a custom template.

d. Perform the following steps in the Business Modeler IDE to add the new template to your project:

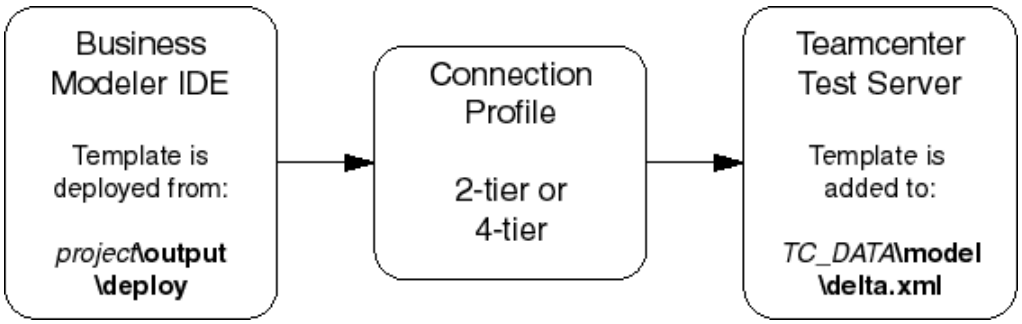
   A. Right-click the project and choose **Properties**.

   B. Choose **Teamcenter→BMIDE** in the left pane.

   C. In the **Dependent Templates** pane, select the new template.

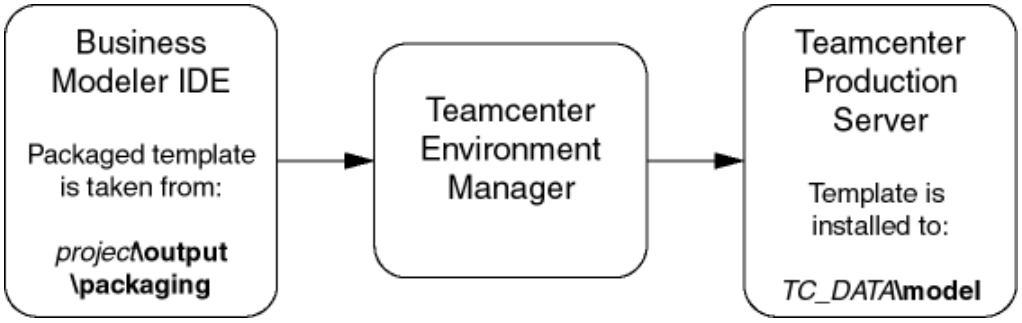
Live update a template

To send your template to a Teamcenter server for testing purposes, choose **BMIDE→Deploy Template** on the menu bar. This is also known as *live update*. You can also use live update to send operational data (such as LOVs) to a production server.



Install a template to a production server

To install your template to a Teamcenter production server, package it using the Business Modeler IDE and install it using the Teamcenter Environment Manager (see the following figure).



© 2025 Siemens

1. In the Business Modeler IDE choose **BMIDE→Generate Software Package**.

   The packaged template is saved to:

   project\output\[operating system type]\packaging

2. Copy the template files to a directory that is accessible by the production server.

3. In Teamcenter Environment Manager on the production server, proceed to the **Feature Maintenance** panel.

4. Select **Add/Remove Features** and click **Next**.

5. In the **Features** panel, click **Browse** to locate the template in your project workspace.

   The template is added to the **Features** panel under the **Extensions** group.

6. In the **Features** panel, choose the new feature in the **Extensions** group.

Behind the scenes of the Foundation template installation

To install the Foundation template, you should start the TEM wizard and select **Create a new installation of this product**. In the **Solutions** panel, select **Corporate Server** and continue with the remaining configuration pages.

After you click the final **OK** button, the TEM wizard installs all selected features. Remember that each packaged feature consists of the following files on the installation source:

install\modules\feature_base.xml
install\lang\en\TcBundle_*language-code_country-code***.xml**
tc\foundation_template.zip
tc\foundation _install.zip

> Note:
>
> The Foundation feature file is called **feature_base.xml**.

When the administrative user selects the **Corporate Server** option, TEM does the following:

1. Copies the feature file to the *TC_ROOT***\install\module** directory.

2. Copies the bundle file to the *TC_ROOT***\install\install\lang***lang* directory

3. Unzips the contents of the **foundation_template.zip** file to the *TC_ROOT***\install\foundation** directory.

© 2025 Siemens

4. Unzips the contents of the **foundation_install.zip** to the *TC_ROOT***install\foundation** directory.

5. Creates a *TC_ROOT***model** directory.

6. Creates a *TC_DATA***model\baselines** directory.

7. TEM executes the **<pre-install>** and **<install>** sections of the Foundation feature file, which does the following:

   a. Copies the **foundation_template.xml** and **foundation_dependency.xml** files from the *TC_ROOT*/**install/foundation** directory to *TC_DATA*/**model** directory.

   b. Copies the **foundation_tcbaseline.xml** file from the *TC_ROOT***install\foundation** directory to the *TC_DATA*/**model/baselines** directory.

   c. Adds the Foundation template to the *TC_DATA***model\master.xml** file.

   d. Creates an empty *TC_DATA***model\model_backup.xml** file.

   e. Consolidates all definitions in the templates listed in the **master.xml** file to a *TC_DATA***model\model.xml** file.

   f. Executes a compare tool which loads the **model.xml** and **model_backup.xml** files into a Java model, compares the two models, and writes a **delta.xml** file that contains the differences between the two models.

      > Note:
      >
      > At the first installation, the **model.xml** file contains the Foundation template extensions, and the **model_backup.xml** file contains no definitions. But for subsequent installations, the **model_backup.xml** file represents the current state and the **model.xml** file represents the expected state after installation. Therefore, the differences between the two files would be all of the Foundation template definitions.

   g. Executes the **populate_new_db.default** script, which reads the **delta.xml** file to populate the database.

Installing optional templates

You can select more than one optional template to install simultaneously through TEM, or you can choose to install one template at a time. Either way the process is the same. This example uses the **tcae** template (Automotive Edition).

The **tcae** template is bundled as follows on the installation source:

install\modules\feature_tcae.xml

© 2025 Siemens

install\lang*language-code***\tcaeBundle_***language-code_country-code***.xml**
tc\tcae_template.zip
tc\tcae _install.zip

When you select the **Teamcenter Automotive Edition** option, TEM does the following:

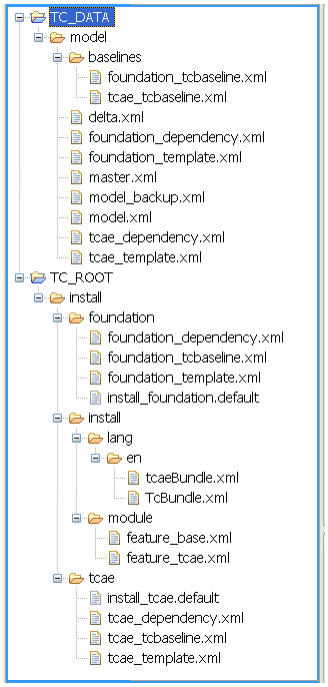
1. Copies the feature file to the *TC_ROOT***\install\module** directory.
2. Copies the bundle file to the *TC_ROOT***\install\install\lang***lang* directory.
3. Unzips the contents of the **tcae_template.zip** to the *TC_ROOT***\install\tcae** directory.
4. Unzips the contents of the **tcae_install.zip** to the *TC_ROOT***\install\tcae** directory.
5. TEM executes the **<pre-install>** and **<install>** sections of the selected feature file, which does the following:
    a. Copies the *template-name***_template.xml** file and the *template-name***_dependency.xml** file from the *TC_ROOT***\install***template-name* directory to the *TC_DATA***\model** directory.
    b. Copies the *template-name***_tcbaseline.xml** file from the *TC_ROOT***\install***template-name* directory to the *TC_DATA***\model\baselines** directory.
    c. Adds the *template-name* to the *TC_DATA***\model\master.xml** file.
    d. Deletes the existing *TC_DATA***\model\model_backup.xml** file.
    e. Renames the existing *TC_DATA***\model\model.xml** file to *TC_DATA***\model\model_backup.xml**.
    f. Consolidates all definitions in the templates listed in the **master.xml** file to a *TC_DATA***\model\model.xml** file.
    g. Executes a compare tool that loads the **model.xml** and **model_backup.xml** files into a Java model, compares the two models, and writes a **delta.xml** file that contains the differences between the two models.

    > Note:
    >
    > The **model.xml** file contains the foundation and **tcae** template extensions, and the **model_backup.xml** file contains the Foundation definitions. Therefore the differences between the two files would be all of the **tcae** template definitions.

    h. Execute the **install_***template-name***.default** script, which synchronizes the data model in the database with the differences in the **delta.xml** file.

© 2025 Siemens

When complete, the *TC_DATA* directory contains the files shown in the following example (Automotive Edition).



Installing custom templates

The process for installing a custom template is the same as installing Siemens Digital Industries Software optional templates but with one exception. The packaging wizard puts all four components of the package in the same location, whereas the Teamcenter installation source places these files in different locations. In this case the TEM wizard knows that all four files are located together and copies them into the *TC_ROOT* directory accordingly:

feature_*template-name***.xml**
*template-name***Bundle_***language-code_country-code* **.xml**
*template-name***_template.zip**
*template-name***_install.zip**

Template upgrading during Teamcenter upgrade

During Teamcenter release upgrade, TEM upgrades any template installed in the Teamcenter environment that is included in the distribution. Upgrade means synchronizing the data model definitions in a server to the latest template definitions in the new release.

TEM automatically identifies the templates that must be upgraded and preselects them in the TEM features panel. The autoselected features are checked, and editing is disabled so that they cannot be unselected. Any distributed templates that the user wants to add to the Teamcenter environment in addition to the ones that already exist in the environment must be added after the upgrade is completed.

© 2025 Siemens

In the following example, during release upgrade of a Teamcenter environment, TEM upgrades both the Foundation and **tcae** templates.

Example:

The Foundation and **tcae** templates are packaged on the installation source as follows:

- Foundation

  install\modules\feature_base.xml
  install\lang\en\TcBundle_*language-code_country-code***.xml**
  tc\foundation_template.zip
  tc\foundation _install.zip

- **tcae**

  install\modules\feature_tcae.xml
  install\lang*language-code***\tcaeBundle_***language-code_country-code***.xml**
  tc\tcae_template.zip
  tc\tcae _install.zip

When the administrative user uses TEM to upgrade the Teamcenter release, TEM does the following:

1. For each selected template, TEM performs the following:

   a. Copies the **feature_***template-name***.xml** file to the *TC_ROOT***\install\module** directory.

   b. Copies the *template-name***Bundle_***language-code_country-code***.xml** file to the *TC_ROOT***\install\install\lang***lang* directory.

   c. Unzips the contents of the *template-name***_template.zip** to the *TC_ROOT***\install***template-name* directory.

   d. Unzips the contents of the *template-name***_install.zip** to the *TC_ROOT***\install***template-name* directory.

2. Creates a new *TC_DATA* directory.

3. Creates a new *TC_DATA***\model** directory.

4. Creates a new *TC_DATA***\model\baselines** directory.

5. For each template feature selected, TEM performs the following:

© 2025 Siemens

a. Copies the *template-name*_**template.xml** and the *template-name*_**dependency.xml** file from the *TC_ROOT***install***template-name* directory to *TC_DATA***model** directory.

b. Copies the *template-name*_**tcbaseline.xml** from the *TC_ROOT***install***template-name* directory to *TC_DATA***model****baselines** directory.

c. Adds the *template-name* to the *TC_DATA***model****master.xml** file.

6. Consolidates all definitions in the templates listed in the **master.xml** file to a *TC_DATA***model****model.xml** file.

7. Runs the **upgrade_runner** utility.

Upgrade_runner utility actions during Teamcenter upgrade

During the Teamcenter upgrade, TEM executes the **upgrade_runner** utility, which gives each template listed in the **master.xml** file an opportunity to hook in optional utilities to create additional data or migrate data. The runner checks each **upgrade_** *template-name_version*.**default** file to see if it has any utilities listed in the various sections. Seven sections are provided so that each template can add data or migrate data at specific segments in the upgrade.

For this section	The upgrade_runner utility does this
Pre Schema Additions	Executes any utilities in the portion of the corresponding **upgrade_***template-name_version*.**default** file.
Post Schema Additions	1. Executes the **business_model_updater** utility using the **delta.xml** file to add any new schema definitions (classes or attributes). 2. For each template listed in the **master.xml**, it executes any utilities in the **SECTION: Post Schema Additions** section of the corresponding **upgrade_***template-name_version*.**default** file.
Post Schema Changes	1. Executes the **business_model_updater** utility using the **delta.xml** file to change any schema definitions (classes or attributes). 2. For each template listed in the **master.xml**, it executes any utilities in the **SECTION: Post Schema Changes** section of the corresponding **upgrade_***template-name_version*.**default** file.
Post Non-Schema Additions	1. Executes the **business_model_updater** utility using the **delta.xml** file to add any new non-schema definitions (types, LOVs, business rules, tools, status, and so on).

© 2025 Siemens

For this section	The upgrade_runner utility does this
	2. For each template listed in the **master.xml** file, executes any utilities in the **SECTION: Post Non-Schema Additions** section of the corresponding **upgrade_***template-name_version***.default** file.
Post Non-Schema Changes	1. Executes the **business_model_updater** utility using the **delta.xml** file to change any non-schema definitions (types, LOVs, business rules, tools, status, and so on). 2. For each template listed in the **master.xml** file, executes any utilities in the **SECTION: Post Non-Schema Changes** section of the corresponding **upgrade_***template-name_version***.default** file.
Post Non-Schema Deletions	1. Executes the **business_model_updater** utility using the **delta.xml** file to delete any non-schema definitions (types, LOVs, business rules, tools, status, and so on). 2. For each template listed in the **master.xml** file, executes any utilities in the **SECTION: Post Non-Schema Deletions** section of the corresponding **upgrade_***template-name_version***.default** file.
Post Schema Deletions	1. Executes the **business_model_updater** utility using the **delta.xml** file to delete any schema definitions (classes or attributes). 2. For each template listed in the **master.xml** file, executes any utilities in the **SECTION: Post Schema Deletions** section of the corresponding **upgrade_***template-name_version***.default** file.

Behind the scenes of a template update

Once a custom template has been installed through TEM, it can be updated at any time using the TEM maintenance mode. For example, a customer creates their own template with data model definitions. After they finish developing and testing their extension definitions, they install their template using TEM into a production server. After some time passes, additional requirements are processed and they add additional extensions to the template and want to put this updated template into production. The TEM maintenance mode can be used to update an existing template in this case.

When a **template is updated**, TEM does the following:

1. Unzips the *template-name***_template.zip** and *template-name***_install.zip** files to the corresponding locations in the *TC_ROOT***\install** directory.

2. Copies the *template-name***_template.xml** and *template-name***_dependency.xml** files from the *TC_ROOT***\install***template-name* directory to *TC_DATA***\model** directory.

© 2025 Siemens

3. Copies the *template-name*_**tcbaseline.xml** from the *TC_ROOT***install***template-name* directory to *TC_DATA***model****baselines** directory.

4. Deletes the existing *TC_DATA***model****model_backup.xml** file.

5. Extracts the full data model from the database and writes into a new *TC_DATA***model****model_backup.xml** file.

6. Consolidates all definitions in the templates listed in the **master.xml** file to a *TC_DATA***model****model.xml** file.

7. Executes a compare tool which loads the **model.xml** and **model_backup.xml** files into a Java model, compares the two models, and writes a **delta.xml** file that contains the differences between the two models.

   > Note:
   >
   > The **model.xml** file contains the latest template extensions and the **model_backup.xml** file contains the previous definitions from the database. Therefore, the differences between the two files would be the differences between the two older customer template and the latest customer template.

8. Executes the **business_model_updater** to load the definitions in the **delta.xml** file which synchronizes the database to the latest template definitions.

Behind the scenes of the template packaging process

After you populate the template with extensions, you can package the template so that it can be installed to a server. Packaging refers to the bundling of the components in the template project. The template is packaged into a format that TEM recognizes so that TEM can install the template into the server, or for usage with the Business Modeler IDE client as a reference template.

Use the packaging wizard if you want to do one of the following:

- Use TEM to install or upgrade the template into a test or production server.
- Distribute the template to other sites or customers who want to share your extensions.

To launch the package wizard, choose **BMIDE→Generate Software Package**.

When you **package a template**, the packaging wizard does the following:

1. Prompts the user to save any unsaved changes to all source files in the extensions directory.

2. Updates the feature file with the rtserver feature if there are any generated C++ classes or services artifacts.

© 2025 Siemens

3. Copies the *template-name*_**config.json** and *template-name***DCBundle**_*language-code_country-code*.**xml** files to the *project***output\packaging** directory.

4. Copies the **feature**_*template-name*.**xml** and *template-name***Bundle**_*language-code_country-code*.**xml** files to the *project***output\packaging** directory.

5. Creates a *project***output\packaging***template-name**_**template.zip** using the following process:

    a. Creates a new file called *template-name*_**template.xml**. This file is generated by reading the contents of each source file listed in the *project***extensions\master.xml** file and appending to the new *template-name*_**template.xml** file.

    b. The *project***extensions\dependency.xml** file is copied to a new file called *project***output\packaging***template-name**_**dependency.xml**.

6. Creates a *project***output\packaging***template-name**_**install.zip** using the following process:

    a. Copies the *project***install\install**_*template-name* files into the ZIP file.

    b. Copies all of the *project***install\upgrade**_*template-name_version* files into the ZIP file.

    c. Copies any other data files in the *project***install** directory to the ZIP file.

When the packaging wizard is complete, the following files are located in the *project***output\packaging** directory:

- **feature**_*template-name*.**xml**

  This file contains the information necessary for TEM to recognize the template and how to handle the template for install and upgrade.

- *template-name***Bundle**_*language-code_country-code*.**xml**

  This file contains the localized text for the feature file so that TEM can display the feature description in the localized version.

- *template-name*_**config.json**

  This file contains the information necessary for Deployment Center to recognize the template and how to handle the template for installation and upgrade.

- *template-name***DCBundle**_*language-code_country-code*.**xml**

  This file contains the localized text for the feature file so that Deployment Center can display the feature description in the localized version.

© 2025 Siemens

- *template-name*_**icons.zip**

  This ZIP file contains all the icon files.

- *template-name*_**template.zip**

  This ZIP file contains the template definitions (*template-name*_**template.xml**), and the dependency file (*template-name*_**dependency.xml**).

- *template-name*_**install.zip**

  This ZIP file contains all the support files for installing and upgrading your template:

  install_*template-name*.**xml**
  Any data files.

Once the template package is created, the bundled files must be transported to the server so that the TEM or Deployment Center can locate and install the templates.

If you have any custom utilities that are called in the installation or upgrade scripts, ensure that the appropriate platform binaries for these utilities are placed into the *TC_ROOT***bin** directory before using TEM or Deployment Center to install or upgrade the template.

Behind the scenes of live update

Live update enables a Business Modeler IDE user to deploy data model extensions directly from the Business Modeler IDE to a running Teamcenter server. You can run live update by choosing **BMIDE→Deploy Template**. In most instances, live update is appropriate for test servers where a Business Modeler IDE user is developing and testing extensions before putting them into production. However, you can also deploy some types of data directly to a production server.

Live update has the advantage of not requiring a packaged template and can deploy through a two-tier or four-tier environment. A disadvantage of live update is that it does not execute the extra utilities listed in your **install**_*template-name*.**default** file or **upgrade**_*template-name_version*.**default** file. Therefore, if you need Teamcenter to process the upgrade or installation utilities, you must package your template and install it through TEM. If you are only making data model changes to the template and want to deploy them quickly to test the changes, you can use live update. Another disadvantage of live update is that only the session that makes the changes can see the changes. All other users of the test server must log off and back on to see the data model changes.

The Deploy wizard does the following:

1. Consolidates all XML files listed in the **master.xml** into a single *project***output\packaging***template_name**_template.xml** file.

2. Copies the *project***extensions\dependency.xml** file to *project***output\packaging***template-name**_dependency.xml** file

© 2025 Siemens

3. Creates an SOA (services-oriented architecture) connection to the specified server using the login credentials, and sends the *template-name*_**template.xml** and *template-name*_**dependency.xml** to the server using FMS.

4. The SOA service writes the files to the *TC_DATA***model** directory.

5. The SOA service adds the template name to the *TC_DATA***model\master.xml** file if it is not already included.

6. Deletes the existing *TC_DATA***model\model_backup.xml** file.

7. Renames the *TC_DATA***model\model.xml** file to *TC_DATA***model\model_backup.xml**.

8. Consolidates all definitions in the templates listed in the **master.xml** file to a *TC_DATA***model\model.xml** file.

9. Executes a compare tool which loads the **model.xml** and **model_backup.xml** files into a Java model, compares the two models, and writes a **delta.xml** file that contains the differences between the two models.

   > Note:
   >
   > The **model.xml** file contains the latest template extensions and the **model_backup.xml** file contains the previous definitions from the database. Therefore, the differences between the two files are the differences between the two older customer template and the latest customer template.

10. Lists all executed actions in a log file. If any elements listed in the **delta.xml** file fail to execute, the failed elements are also listed in the log file.

11. Extracts the data model contents of the database to the *TC_DATA***model\model.xml** file so that the next live update is able to compute the differences against a file that reflects the accurate contents of the database. This step is executed only if there are unprocessed elements in the delta file.

12. Sends the log file back through the SOA service and places it in the *project***output\packaging** directory, and names the file with a timestamp.

Template artifacts

Feature file

Feature file description

The feature file is used for the following purposes:

© 2025 Siemens

- A feature file is required by TEM so that you can present your template as a choice in TEM to be installed into the corporate server.
- If a customer chooses your template for the corporate server, the extensions in your template are combined with the other choices of the user and added to the corporate server database.
- When you choose your template for the corporate server, also choose to install your template as a reference template when using the Business Modeler IDE to extend your own template. That way a customer can see all the extensions provided in your template and add their own.
- A feature file is also used by TEM for upgrade.
- A feature file ensures that your template is chosen for upgrade if it was previously installed into the database.
- If you provide extensions from a previous release, ensure that your template is selected during upgrade. Once a template is installed, it must always persist in future releases. You can keep providing updates, but it can never be removed.
- A feature file contains a list of the dependent feature templates that should be installed before your template can be installed.

Feature file naming convention

The feature file must be named as follows: **feature_***template-name***.xml** where *template-name* is your template name. All characters should be lowercase.

Following are examples of feature file names:

feature_tcae.xml
feature_gmo.xml
feature_mytemplate.xml

The Business Modeler IDE application creates this default feature file for a given template when the project is created.

Sample feature file

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Document   : feature_template-name.xml
     Description: This XML is used by TEM to install or upgrade the "template-name"
solution.-->
<feature>
    <name value="template-name"/>
    <property name="feature_name" value="template-name"/>
    <group value="package"/>
    <guid value="049B06807BDA0239E0A054B39495661A"/>
    <bundle value="${feature_name}Bundle.xml"/>
    <description value="${feature_name}.description"/>
```


© 2025 Siemens

```
    <include file="dataModelDependency.xml"/>
    <relation/>
    <feature>
<!-- Feature: Data Model -->
        <name value="Data Model"/>
        <property name="feature_id" value="datamodel"/>
        <property name="bmide_optional" value="false"/>
        <property name="template_name" value="${feature_name}"/>
        <property name="template_file" value="${template_name}_template.xml"/>
        <removable value="false"/>
        <root value="true"/>
        <bundle value="${template_name}Bundle.xml"/>
        <description value="${template_name}.description"/>
        <guid value="1D388F052C95BE2EFCC70355B229BA03"/>
        <include file="coreDataModel.xml"/>
        <include file="generatePlmxmlSchema.xml"/>
    </feature>
</feature>
```

Feature file header section

The following example shows header information and contains a description comment about the purpose of the feature file:

```
<?xml version="1.0" encoding="UTF-8"?>

<!--
    Document   : feature_<template_name>.xml
    Description: This XML is used by TEM to install or upgrade the <template_name>
solution.
-->
```

Feature section

The feature section contains the following. (The following list is presented in alphabetical order.)

- The **bundle value** tag indicates the localization file that should be used with the feature to provide the display name for the feature in TEM panels. All **bundle files** are named with the following format *template_name***Bundle.xml**. This tag uses a **${template_name}** variable to determine the bundle name, so there is no need to edit this tag. Never change this value.

  ```
  <bundle value="${template_name}Bundle.xml"/>
  ```

- The **relation/depends name** tag specifies the data model features that your template is dependent on. This ensures that the dependent templates are installed prior to your feature if the user selects your template. By default all templates are dependent on the Foundation template at a minimum, so the Foundation template is implicitly specified (by including the **dataModelDependency.xml** file). You must add an additional **<depends>** tag for each dependent template. The name key should specify the template name, and the value should specify the GUID. The GUID can be obtained by looking in the dependency files for each template. If you are dependent on another template that also has its own dependencies, you need not specify all dependencies. You only need to provide your template's dependencies. TEM takes care of the rest.

© 2025 Siemens

```
<depends name=”foundation” value="8C061DD51E13E0CB9DC4687B1A3348BE"/>
```

Never change this value.

- The **description value** tag indicates which key in the **bundle.xml** file to use for getting the feature's description. Never change this value.

  ```
  <description value="template.description"/>
  ```

- The **feature** tag specifies the subfeatures under a feature.

- The **group value** tag indicates that your feature should appear in TEM under the **Extensions** option. This is the option where all template features appear. If you have suboptions for a feature, you must create a **group_***name***.xml**. The group name and the parent group name must be specified.

  ```
  <group value="package"/>
  ```

  For example, you have a solution called **solution1** that has **template1**, **template2** and **template3** options. **solution1** is a high-level group name and not installable template by itself. The **group_solution.xml** file must be created with the following entries:

  ```
  <group>
     <name>solution-display-name</name>
     <parent>foundation</parent>
  </group>
  ```

  In **feature_template1.xml**, **feature_template2.xml**, and **feature_template3.xml**, the group tag is:

  ```
  <group value=”solution”/>
  ```

  This solution appears as follows in TEM:

  Corporate Server
  Solution1
  Template1
  Template2
  Template3

- The **guid** tag specifies the GUID for your feature (template). If a unique GUID has not been assigned, then you should get one assigned from the TEM team and update the value. A GUID is automatically created by the Business Modeler IDE application and assigned to the feature file when the project is created.

  ```
  <guid value="C0E54180924C29FE0B89C9D3CDCC7A17"/>
  ```

- The **include** tag allows you to include XML content in a separate file within a feature XML. There are no file name restrictions. This also reduces the feature file size by taking the common elements and keeping them in separate files.

© 2025 Siemens

- The **property name="feature_id"** tag specifies the subfeature type. This tag is used by TEM to determine the subfeature type. For example, **"datamodel"** represents a datamodel (template) feature, **"rtserver"** represents a run-time server component, and **"rac"** represents a rich client add-on component.

- The **property name="template_name"** tag specifies the template name. This tag is used by TEM to determine which feature files contain templates.

  ```
  <property name=”template_name” value=”template-name”/>
  ```

- The **removable value** tag indicates whether TEM lets you remove this feature. By default no template feature can be allowed to be removed since it would be difficult to remove all instances of each extension. Therefore, all template features can never be removed.

  ```
  <removable value="false"/>
  ```

  Never change this value.

- The **root value** tag indicates that the user who installs the feature must have administrative privileges on the machine. This tag is only used by Windows. Because the feature is used on all platforms, it should always be provided. Because each feature can potentially make changes to services, this tag should always be set to true.

  ```
  <root value="true"/>
  ```

  Never change this value.

- The **template_match** tag provides the data model object to match for template matching. This is required for any template feature that may have been installed prior to Teamcenter 2007. A feature can provide as many **template_match** tags as needed to appropriately match your feature with the database. For example, it may not be sufficient to try to match only one data model object.

  ```
  <property name=”template_match_1” value=”ImanType,type_name, ALIAS_PROJECT”/>
  <property name=”template_match_2” value=”Tool,object_name, AUTOSTUDIO_TOOL”/>
  ```

  The match works by providing a class, attribute, and value. TEM queries for the class and attribute pair, and if it finds that one of the rows matches the value you provided, there is a match on this tag.

  Examples:

  template_match_1
  template_match_2
  template_match_3

  Edit these tags as needed.

© 2025 Siemens

Files section

The files section of the feature file (in the **coreDataModel.xml** file) provides two commands to unzip a ZIP file provided with the feature. The command gives the location of the ZIP files with respect to *TC_CDROM*. Each ZIP file should be zipped specifically so that when unzipped, the file contents end up in the correct locations in *TC_ROOT*. The relative path of the files in the ZIP file is with respect to *TC_ROOT*. For customer templates, the Business Modeler IDE zips these files accordingly. For Siemens Digital Industries Software templates, these ZIP files are placed into an installation kit to accomplish the same things. Do not edit this section.

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

</files>
```

Install section

```
<preinstall>

</preinstall>

<install>

<gui>
    <panel class="com.teamcenter.install.tceng.base.gui.AdminUserPanel"
    factory="acquirePanel"/>

</gui>

<code>
    <tcexec cmd="bmide_pretemplateinstall${script_ext}" parms="-u=${adminUser.user}
    -p=${adminUser.password} -g=dba -templates=${template_name}"/>
    <condition property="template_installed" value="true" else="false">
   <checktemplate value="${template_name}"/>
    </condition>
    <tcexec script="${tcroot.path}/install/${template_name}/install_$
{template_name}.default"
    whenfalse="${template_installed}"/>
    <tcexec cmd="business_model_updater" parms="-u=${adminUser.user}
    -p=${adminUser.password} -g=dba -mode=upgrade -update=types
    -file=${tcroot.path}/install/${template_name}/${template_file}"
    whentrue="${template_installed}"/>

</code>

</install>
```

The **preinstall** and **install** sections (in the **coreDataModel.xml** file) are called by TEM during the first installation of your template. The **install** section commands take care of copying your zipped template files to the *TC_DATA***model** directory, and then ensures that the correct tools are run to add your template extensions to the data model. The **template_installed** condition is used by TEM to see if the template is already installed in the database. The TEM installer installs the template only if it is not

© 2025 Siemens

already installed. If the template is already installed, **business_model_updater** is run to install any types in the database to populate the **.des** files. This may be when installing an alternate *TC_DATA* to an existing database.

There is no need to edit these sections. The empty **preinstall** section is provided as a placeholder in case it is needed in the future.

Update section

```
<preupgrade>

</preupgrade>

<upgrade>

</upgrade>
```

The preupgrade and upgrade sections are called by TEM during the upgrade of your template. The preupgrade and upgrade sections are placeholders for any commands needed for upgrading. Currently these sections are empty. TEM knows that if your feature is a template feature, it automatically runs the appropriate upgrade runner script utility provided by the Business Modeler IDE. The upgrade runner calls sections in your upgrade script (depending upon the version of the upgrade).

There is no need to change or remove the sections. There are provided in case they are needed for future needs.

Bundle file

Bundle file description

A bundle file is used to provide localized text to TEM. This file contains your feature's description that are displayed in the various TEM panels where the user can select your template.

Your bundle should contain a text name **template.description** that corresponds to the **feature file**. The feature file is looking for the localized description of your feature in the bundle file by the name of **template.description**. Ensure that have corresponding entries in both the feature file and bundle file.

```
<description value="$(template_name).description"/>
```

In the bundle file, be sure to update the comments section and the **TextID** to reflect your template:

```
<TextID name="template_name.description" text=" Installs the XX Solution application."/>
```

Bundle file naming convention

The bundle file must be named *template-name***Bundle_***language-code_country-code***.xml**. Replace *template-name* with your template name. Only the **B** in **Bundle** should be upper case; all other characters must be lowercase.

© 2025 Siemens

Examples:

tcaeBundle_en_US.xml
gmoBundle_en_US.xml
mytemplateBundle_en_US.xml

If you follow this format, your bundle file automatically gets picked up by the **feature.xml** file entry that declares the bundle file based on your template name.

Sample bundle file

```
<?xml version="1.0" encoding="UTF-8"?>

<!--
bcprt
This software and related documentation are proprietary to UGS Corp.
COPYRIGHT 2006 UGS CORP.  ALL RIGHTS RESERVED
ecprt
-->

<!--
    Document   : template_nameBundle.xml
    Description: Localized text for template_name template feature
-->

<IDMap>
   <TextID name="template_name.description" text="Installs the XX Solution
application."/>

</IDMap
```

Installation file

Installation file description

Every template must have an installation script. This script is run by TEM when your template feature is selected by the user to install into a Teamcenter corporate server. This install script has very specific content and ordering of commands to get your Business Modeler IDE extensions installed into the database. This order and content cannot be changed.

The installation script is separated into sections labeled with the **# SECTION:** comment and the name of the section. TEM must install the template's data model extensions in a specific order, therefore these sections must be kept in the order that they are given and this order and the content should never be changed.

The solid and dotted lines that separate the sections are intentional and should not be changed. The solid lines indicate the sections that contain utilities that the templates cannot alter. These are required so that the Business Modeler IDE utilities can install the template data model. The dotted line sections are areas where the template can add additional utilities to support the installation of the template. For example, you may want to load process template objects.

© 2025 Siemens

To add any utilities, only add them 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.

SECTION	Type	Description
SECTION: Utilities	Open for adding utilities	Adds utilities that are required after the schema has been updated with the latest classes and attributes.
SECTION: Utilities - Post BMIDE	Open for adding utilities	Adds utilities that are required after the non-schema data model objects have been updated (that is, business objects, LOVs, business rules, extension points, and so on).

Installation file naming convention

The installation script must be named **install_***template-name***.default**. Replace *template-name* with your template name. All characters should be lowercase. You must ensure that you follow this naming convention; otherwise your script does not run.

Examples:

install_tcae.default
install_gmo.default
install_mytemplate.default

Sample install file

```
# install_test.default:
#
#
#
# This script installs the Data Model for test solution.
# ____________________________________________________________________________
#


# ..............................................................................
# SECTION: Utilites
# ..............................................................................
#    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>
```


© 2025 Siemens

```
<pre-non-schema-add>

</pre-non-schema-add>

# ..........................................................................
# 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.
# ..........................................................................
#
<post-non-schema-add>

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

Media file

Sample media file

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!--
================================================================================
=======
Copyright 2016.
Siemens Product Lifecycle Management Software Inc.
All Rights Reserved.
================================================================================
=======
-->
<media>
    <product_id>Teamcenter</product_id>
    <product_display_name>Teamcenter</product_display_name>
    <application_id>b5samplebmideproject</application_id>
    <application_display_name>Sample BMIDE Project</
application_display_name>
    <version>1.0:1</version>
    <customer>All</customer>
    <rtm_type>Major</rtm_type>
    <parent_release/>
    <platform>wntx64</platform>
    <description>This is a sample template project.</description>
    <application_dependency/>
        <application id="Foundation" release="11.4"/>
    <supported_upgrade_paths/>
</media>
```


© 2025 Siemens

Update file

Update file description

If your template is packaged in a feature and installed to the environment, then during a Teamcenter update, TEM or Deployment Center automatically synchronizes your Business Modeler IDE data model extensions and any related data objects to the latest release.

In addition to the Business Modeler IDE data model extensions in your template, you may have additional data that needs to be added. The update file is a script used to accomplish that data addition. While an update script is not required if you have no additional things to add during an update, you should provide an update script even if it has no special commands in it, both for consistency with other templates and so that you can readily add things to the script later if needed.

Update file content details

The update script is separated into sections using the **<section>** element. TEM or Deployment Center needs to update the template's data model extensions in a specific order. Therefore, these sections must be present and must be kept in the order that they are given; this order and the label content should never be changed.

Within each section, you define your commands using the **<command>** element.

Example:

This is an example of the update file that is created when you make a new BMIDE template project.

```
<update_default_script>
    <section name="pre-schema-add" description="...">
        <!-- <command sourceVersion="The release version from which an update will
be made"
                      originVersion="The release version in which the command has
been introduced"
                      cmd="command to be executed"
                      description="The description of the command"/> -->
    </section>
    <section name="post-schema-add" description="..."/>
    <section name="post-schema-change" description="..."/>
    <section name="pre-non-schema-add" description="..."/>
    <section name="post-non-schema-add" description="..."/>
    <section name="post-non-schema-change" description="..."/>
    <section name="post-non-schema-delete" description="..."/>
    <section name="post-schema-delete" description="..."/>
</update_default_script>
```

Whenever you add utilities, the utilities must go between the specific opening and closing tags. Any commands outside these tags are not executed. Do not alter the tags in any way. The update runner is looking specifically for these tags and does not recognize any new ones or tags by another name or spelling.

© 2025 Siemens

Example:

This is an example of a single section in an update file that has commands added. Notice how the **<section />** has been split into **<section>** and **</section>**.

```
<section name="" description="">
    <command cmd="preferences_manager -u= -pf= -g= -mode= -action= -scope=
-context= -file=" description=""/>
    <command cmd="plmxml_import -u= -pf= -g= -mode= -action= -scope= -context=
-file=" description=""/>
</section>
```

Section information

SECTION:	Description
pre-schema-add	Use this section to add any commands that should migrate data before the Business Modeler IDE tools have added the new classes and attributes.
post-schema-add	Adds any commands that should migrate data after the Business Modeler IDE tools have added the new classes and attributes, but before the Business Modeler IDE tools modify or delete any existing classes and attributes.
post-schema-change	Adds any commands that should migrate data after the Business Modeler IDE tools have changed any existing classes and attributes, but before they delete any existing classes and attributes, and before they add, change, or delete non-schema objects. Use this section if your template needs to install any workflow templates, transfer modes, or any other data that must be installed before the Business Modeler IDE tools install data model objects.
pre-non-schema-add	Add the utilities that install new transfer modes or process templates in this section.
post-non-schema-add	Adds any commands that should migrate data after the Business Modeler IDE tools have added data model objects.
post-non-schema-change	Adds any commands that should migrate data after the Business Modeler IDE tools have changed any existing data model objects.
post-non-schema-delete	Adds any commands that should migrate data after the Business Modeler IDE tools have deleted any existing data model objects.
post-schema-delete	Adds any commands that should migrate data after the Business Modeler IDE tools have added, changed, or deleted data model objects.

© 2025 Siemens

Command information

Example:

The command should always run regardless of the source release.

I want my command to always run, regardless of which source release I am updating from. Certain default commands should always run regardless of the source release. For example, database finalization commands or commands that use the file as an input can be modified in any release based on needs, such as the preference file.

Syntax: For such commands, omit the **sourceVersion** attribute from the command.

```
<command originVersion = "2312" cmd = "<command to be executed>"
description = "<Description of the command>" />
```

Example:

The command should run while updating from Teamcenter 13.x and Teamcenter 14.x only

If the command must run during the update from Teamcenter 13.x and Tc14.x (two supported update pathways) to Teamcenter 2312 and will not be needed in the future, for example, during an update from Teamcenter 2312 to Teamcenter 2512, then the following syntax should be used:

Syntax: For such commands, use the **sourceVersion** attribute as "13x,14x".

```
<command sourceVersion = "13x,14x" originVersion = "2312" cmd =
"<command to be executed>" description = "<Description of the
command>" />
```

Example:

The command should always run while updating from Teamcenter 13.x only

If the command must run only during the update from Teamcenter 13.x to Tc2312 and is not required starting from Teamcenter 14.x, then the following syntax should be used:

Syntax: For such commands, use the **sourceVersion** attribute as "13x".

```
<command sourceVersion = "13x" cmd = "<command to be executed>"
description = "<Description of the command>" />
```

Example:

The command should always run while updating from specific releases

If the command must run only during the update from Teamcenter 13.x and 14.0, 14.1, 14.2 to Teamcenter 2312 and is not required starting from Teamcenter 14.3, then the following syntax should be used:

© 2025 Siemens

Syntax: For such commands, use the **sourceVersion** attribute as "13x,14.0,14.1,14.2".

```
<command sourceVersion = "13x,14.0,14.1,14.2" cmd = "<command to be
executed>" description = "<Description of the command>" />
```

Example:

The command should run from Teamcenter 2312 to the future updates

For example, if the command was added in Teamcenter 2512 and all earlier supported releases should run it upon update, the syntax would be as follows:

```
<command sourceVersion = "14x,2312,2406,2412,2506" originVersion =
"2512" cmd = "<command to be executed>" description = "<Description
of the command>" />
```

Update file naming convention

The update script must be named *template-name*_**update.xml**. Replace *template-name*. All characters should be lowercase. You must ensure that you follow this naming convention. The TEM update utility expects the script to be named this way, otherwise your script will not run correctly.

If you do need an update script, you must provide a separate update script for each version of Teamcenter that you support. This is because the TEM wizard does not know what version of the product you are upgrading from and only executes the appropriate script that matches that release.

The *version* tag should be replaced with one of the versions in the table. Do not make up a version name. These names are provided by TEM as the officially supported versions. No other versions work. If you do not see a version that you require, please contact your Siemens Digital Industries Software representative for the correct version string.

Sample update file

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!--
@<COPYRIGHT>@
==========================================================================
Copyright 2024.
Siemens Digital Industries Software.
All Rights Reserved.
==========================================================================
@<COPYRIGHT>@

Steps for updating c9x to the latest Teamcenter release
-->

<update_default_script>
    <section name="pre-schema-add" description="Use this section to add any commands
that should migrate data before the BMIDE tools have added the new classes and
attributes.">
        <!-- <command sourceVersion="The release version from which an update will be
```


© 2025 Siemens

```
made" originVersion="The release version in which the command has been introduced"
cmd="command to be executed" description="The description of the command"/> -->
    </section>
    <section name="post-schema-add" description="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."/>
    <section name="post-schema-change" description="Use this section to add any commands
that should migrate data after 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. 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. The creation of types for pre-tc2007 releases
should be added in this section."/>
    <section name="pre-non-schema-add" description="Add the utilites that install new
Transfer modes or Process templates in this section."/>
    <section name="post-non-schema-add" description="Use this section to add any
commands that should migrate data after the BMIDE tools have added the Business Objects,
Business Rules, LOVs, Change objects, Validation Data, Tools, etc."/>
    <section name="post-non-schema-change" description="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."/>
    <section name="post-non-schema-delete" description="Use this section to add any
commands that should migrate data after the BMIDE tools have deleted any existing
Business Objects, Business Rules, LOVs, Change objects, Validation Data, Tools, etc."/>
    <section name="post-schema-delete" description="Use this section to add any commands
that should migrate data after the the BMIDE tools have added/changed/deleted Classes/
Attributes Business Objects, Business Rules, LOVs, Change objects, Validation Data,
Tools, etc."/>
</update_default_script>
```


© 2025 Siemens

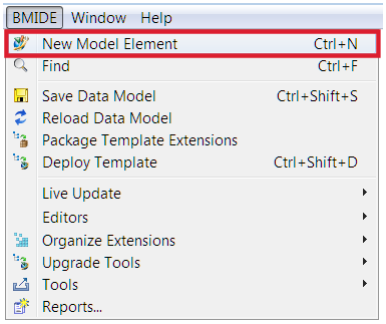
© 2025 Siemens

11. Creating data model elements

Add a new model element

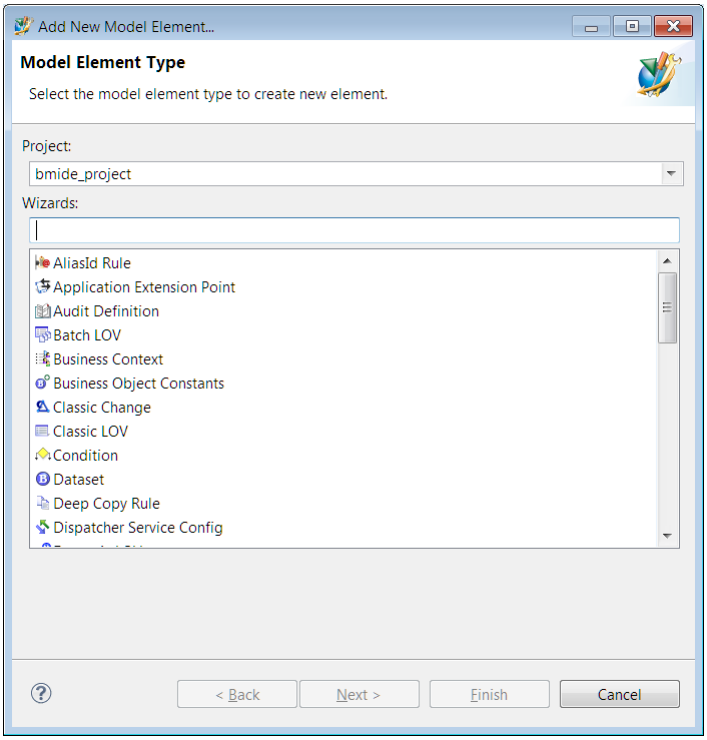
The Add New Model Element wizard allows you to create custom model elements.

1. On the menu bar, choose **BMIDE→New Model Element**.



2. Select the model element you want to create and click **Next**.

   The creation wizard for the selected model element runs.



© 2025 Siemens

Business Modeler IDE administration tasks

The Business Modeler IDE is intended for business analysts who are responsible for tailoring Teamcenter for use at their companies. Users of the IDE must have a thorough understanding of how to perform end-user tasks with Teamcenter and have some knowledge of the data model items used in Teamcenter.

Use the IDE to create:

Data model elements	Description
Business objects	The fundamental objects used to model business data. Create business objects to represent product parts, documents, change processes, and so on. Business objects were formerly known as *types* in Engineering Process Management.
Classes	The persistent representations of business objects in the database. A class is the logical data model and maps the storage of a business object to the database. Every class has a business object by the same name. A class can have attributes defined on it. Every attribute defined on the class results as a property on the business object. Classes support inheritance. Any attribute defined on a parent class is inherited by its children.
Properties	Item characteristics, such as name, number, description, and so on. Properties are attached to business objects. All children of a business object inherit their parents' properties.
Lists of values (LOVs)	Pick lists of values. They are commonly accessed by Teamcenter users from a menu at the end of a data entry box.
Business rules	Directives that govern how objects are handled, including how they are named, what actions can be undertaken on them, and so on. Creating rules is also known as business modeling.
Constants	Configuration points within the data model. They provide consistent definitions that can be used throughout the system.
Options	Miscellaneous objects such as change objects, notes, and so on.
Code generation objects	Objects used for software development, and include data types, libraries, releases, and services. These are used when you want to write C++ code.

After you create these new data model objects, you can **deploy** them to a test server. And after you verify the new data model on the test server, you can **package** it for installation to a production server.

© 2025 Siemens

Business objects

Create business objects

Introduction to creating business objects

To create a new business object in the **BMIDE** view of the Business Modeler IDE, right-click a business object in the **Business Objects** folder and choose **New Business Object**. The **Business Objects** folder is used for working with *business objects*, the fundamental objects that model business data. You can create business objects and add properties to them. You can also add operations or business rules to business objects.

The most common business objects you create are children of the **Item**, **Form**, and **Dataset** business objects. The **process for creating business objects** is the same for most kinds of business objects.

> Tip:
>
> When you create a new business object, it is often exposed in the dialog boxes accessed from the **File→New** menu in the My Teamcenter application in the rich client. To find where to add the business object in the Business Modeler IDE tree of business objects, search in the tree for the objects shown in these **File→New** menu dialog boxes. You can also create business objects in the My Teamcenter application by choosing **File→New→Other**.

Where classes are the logical data model, business objects are the objects that the user works with in the clients. Typically, most business objects have a storage class that helps map the attributes to the database. However, run-time business objects do not have any persistent storage and therefore do not have storage classes.

Business objects get their properties from two locations. Any attribute defined on the storage class is derived as a property on the business object. These properties are called persistent properties. Additional properties such as compound properties, relation properties, and run-time properties can be defined directly on the business object.

Business objects also support inheritance. Any compound property, relation property, and run-time property defined on a parent business object is inherited by its children.

Business objects can also have behavior attached to them in the form of operations and business rules. Operations and rules defined on parent business objects are also inherited by the children. Typical business rules include GRM rules, deep copy rules, naming rules, revision naming rules, LOVs, business object constants, property constants, IRDCs, business object display rules, propagation rules, and extension rules.

Process for adding a new business object

When you add a new business object using the Business Modeler IDE, you select a parent business object under which to create it. The child business object inherits the properties and behaviors of the parent.

© 2025 Siemens

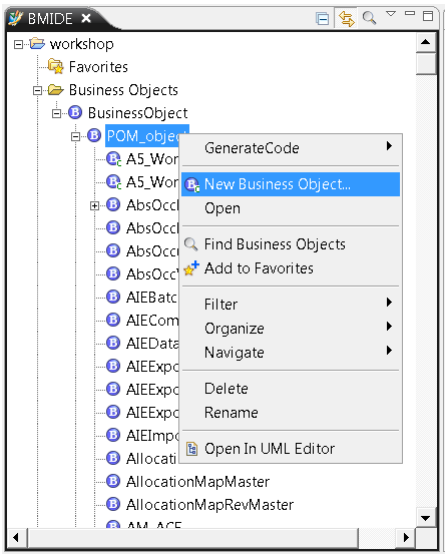
You can add a primary or secondary business object. A *primary* business object has the same name as its associated storage class. A *secondary* business object uses the storage class of its parent business object. When you add a primary business object, a corresponding class is created to hold data for the primary business object.

The following steps are the generic procedure for adding a new business object, whether primary or secondary:

1. In the **Business Objects** folder, browse to a business object under which you want to create the new object. To search for a business object, you can click the **Find** button at the top of the view.

2. Right-click the business object you want to use as the parent and choose **New Business Object**.

   The New Business Object wizard runs.



   There are several other ways to launch the New Business Object wizard:

   - To create children of the **Item** or **Dataset** business objects, choose **BMIDE→New Model Element**. You cannot create any other type of business object with the **New Model Element** wizard.

   - Right-click a business object displayed in the **UML Editor** and choose **Add Business Object**.

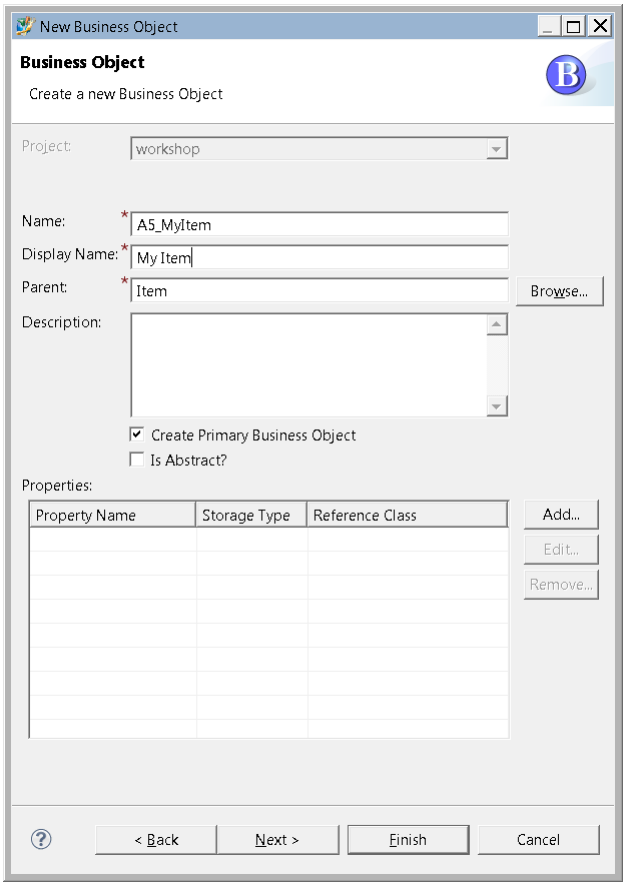
   - Drag **Business Object** from the UML Editor palette into the UML editor.

     When you choose this method, the New Business Object wizard allows you to select the type of business object to create, for example, **Item**, **Form**, and so on.

3. Fill in the dialog boxes presented by the New Business Object wizard. For example, name the business object and indicate its parent.

© 2025 Siemens

The wizard can present different dialog boxes depending on the parent business object selected. See subsequent topics about adding business object types that have different dialog boxes.



Note:

a. Select **Create Primary Business Object** to add a new class that stores the data for the new business object. Clear this option to set the storage class as the same used by the parent business object.

   A *primary* business object has its own storage class. A *secondary* business object uses the storage class of its parent business object. You can change a secondary business object to a primary business object one by right-clicking the secondary business object and choosing **Convert to primary**.

b. Select **Is Abstract?** if instances of the business object will not be created in user interfaces such as the rich client. For example, you may want to select this if the business object is intended as a folder for child business objects.

c. The **Store as lightweight object** check box indicates if the object is stored in its own database table outside of the **POM_object** database table. This improves performance

© 2025 Siemens

of POM object handling. Initially only a limited number of internal classes may be stored as lightweight objects. This check box is read-only.

4. Click **Finish**.

   The new business object appears in the **Business Objects** folder. A **c** on the business object symbol indicates that it is a custom business object.

   If you create a primary business object, the corresponding class appears in the **Classes** view in the **Advanced** perspective.

5. To add the business object to your **Favorites** folder, right-click the business object and choose **Add to Favorites**.

   If you create a child of the **Item** or **Document** business object, you can find the master, revision, and revision master business objects that were also created. Click the **Find** button at the top of the **BMIDE** view and search for the new business object name.

6. To save the changes to the data model, choose **BMIDE→Save Data Model**, or click the **Save Data Model** button on the main toolbar.

7. After you create a business object, you can deploy your changes to the test server. Choose **BMIDE→Deploy Template** on the menu bar, or select the project and click the **Deploy Template** button on the main toolbar.

8. After deployment, test your new business object in the Teamcenter rich client by creating an instance of it.

   For example, if you created a new business object as a child of the **Item** business object, in the My Teamcenter application, choose **File→New→Item**. (You can also create business objects in the Teamcenter rich client by choosing **File→New→Other**.)

   Your new business object appears in the **New Item** dialog box. Choose your new business object and create an instance of it.

Note:

- If you select the **Is Abstract?** check box during creation of the new business object, you cannot create an instance of the business object.
- Not all children of the **Item** business object appear in the **File→New→Other** menu. For example, **ADSPart**, **ADSDesign**, and **ADSDrawing** and their subtypes appear in other menus.

© 2025 Siemens

Note:

You can **create your own custom icon** to assign to the new business object.

Common business objects you can subclass

You can subclass from specific business objects in the data model to get the behavior you want in Teamcenter.

To create a new business object, right-click a business object in the **Business Objects** folder of the Business Modeler IDE and choose **New Business Object**. (When you create a new primary business object, a class of the same name is created that stores the business object information.)

There are many business objects listed in the **Business Objects** folder. Following are some of the most commonly subclassed business objects in the Foundation template, shown in their tree structure.

Note:

Not all objects you want to create are business objects. Use the **Extensions\Options** folder in the Business Modeler IDE to create the following: contexts, note types, occurrence types, view types, status, storage media, or tools.

POM_object

Represents storage classes in the database.

ImanRelation

Represents relationships between objects.

POM_application_object

Represents Teamcenter application classes in the database.

WorkspaceObject

Holds the objects that can be displayed in the end-user workspace.

AppearanceGroup

Persists occurrence groups.

Dataset

Represents file types.

© 2025 Siemens

Folder

Organizes groups of objects.

Form

Displays properties.

GeneralDesignElement

Models a connectable interface of a product.

GeneralDesignElementLink

Models connectivity between one or more general design element (GDE) objects. This object serves the same functional purpose as a **PSConnection** business object except that it is not revisable.

Item

Represents parts and documents.

When you create a new subclass of the **Item** business object or one of its children, the following are automatically created: **ItemMaster**, **ItemMasterS**, **ItemRevision**, **ItemRevisionMaster**, **ItemRevMasterS**. The same types of master forms are also created for all the children of the **Item** business object.

Architecture

Represents product structures.

GPA

Represents a generic physical architecture type.

Design

Represents designs.

Document

Represents documents.

Specification

Represents a collection of requirements specifications.

© 2025 Siemens

RequirementSpec

Represents a requirements specification.

Drawing

Represents part drawings.

OP

Represents a measurement operation being done at a station in a plant. It aggregates all the inspection features that are measured at that operation.

Part

Represents parts.

PSConnection

Models connectivity between one or more general design element (GDE) objects.

Schedule

Represents a group of planned tasks that must be performed to complete a project.

ScheduleTask

Represents a planned task that must be completed to make progress in the schedule.

SpecElement

Represents an individual requirement specification element.

Paragraph

Represents a requirements paragraph.

Requirement

Represents a specific requirement.

ItemRevision

© 2025 Siemens

Represents a revision of an **Item** business object.

Design Revision

Represents a revision of a **Design** business object.

DocumentRevision

Represents a revision of a **Document** business object.

Drawing Revision

Represents a revision of a **Drawing** business object.

OPRevision

Represents a revision of an **OP** business object.

Part Revision

Represents a revision of a **Part** business object.

ScheduleRevision

Represents a revision of a **Schedule** business object.

ScheduleTaskRevision

Represents a revision of a **ScheduleTask** business object.

ReleaseStatus

Represents the status of a release.

Item business objects

Introduction to item business objects

Items are the fundamental object used to model data in Teamcenter. They are commonly used to identify an element of a product (component, assembly, end item) or other data such as procurement specification, test procedure, standard part, shop tooling, change, and so on.

Data modeled using the **Item** business object is generally revision controlled data and all revisions of the information must be maintained, tracked, and recoverable. Data must also be modeled using the Teamcenter concept of item if you want to build a structure of the items such as a bill of material (BOM) for items that represent a product.

© 2025 Siemens

In the initial setup for Teamcenter, two types of items are provided:

- **Item**

  Generally used for data that represents an element of a product that is CAD defined and for which product structure (BOM) data is maintained in the system.

- **Document**

  Generally used for all other data that is considered revision controlled but not necessarily considered as a product or is not defined using CAD applications.

Create an Item business object

Item is the most common business object under which you create a new business object. Use the **Item** business object or its children when you want to create business objects to represent product parts. When you create a business object using **Item** (or one of its children) as the parent, in addition to the new business object, you also create an item master form, an item revision, and an item revision master form.

> Note:
>
> When you create a child of the **Item** business object, a child **ItemRevision** business object is created automatically. You cannot create a child of the **ItemRevision** object directly.

1. Choose one of these methods:

   - On the menu bar, choose **BMIDE→New Model Element**, type **Item** in the **Wizards** box, and click **Next**.

   - Click the **Find** button at the top of the **BMIDE** view, search for the **Item** business object, right-click it, and choose **New Business Object**.

2. In the **Business Object** dialog box, enter the following information:

   a. The **Project** box defaults to the already-selected project.

   b. In the **Name** box, type the name you want to assign to the new business object in the database.

      When you name a new data model object, a prefix from the template is automatically affixed to the name to designate the object as belonging to your organization, for example, **A5_**.

   c. In the **Display Name** box, type the name as you want it to appear in the user interface.

© 2025 Siemens

d. The **Parent** box displays the business object you already selected as the parent business object.

e. In the **Description** box, type a description for the new business object.

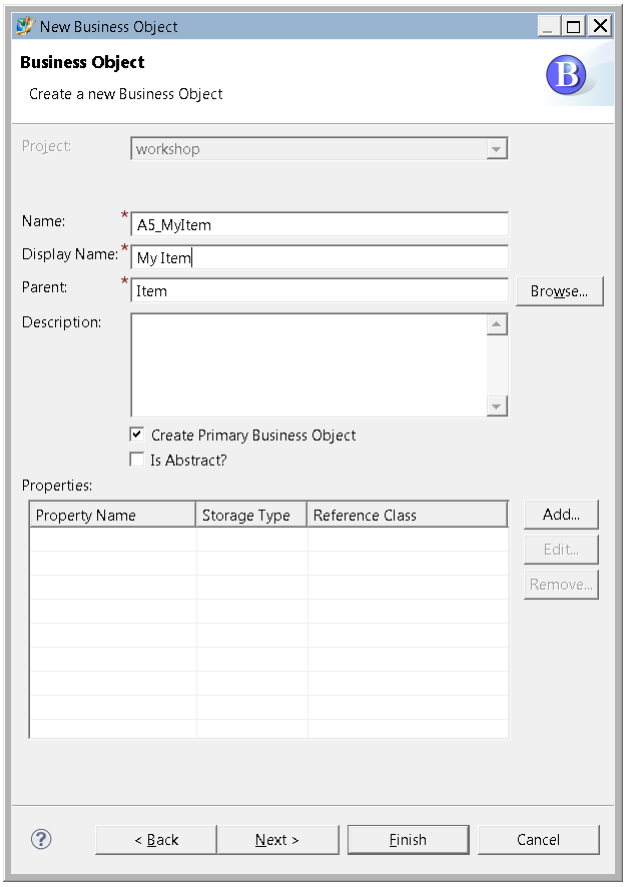
f. Select **Create Primary Business Object** to make a new class that stores the data for the new business object. Clear this option to set the storage class as the same used by the parent business object.

g. Select **Is Abstract?** if instances of the business object will not be created in user interfaces such as the rich client. For example, you may want to select this if the business object is intended as a folder for child business objects.

h. Click the **Add** button to the right of the **Properties** table to **add a property to the business object**.

The New Property wizard runs.

i. Click **Next**.



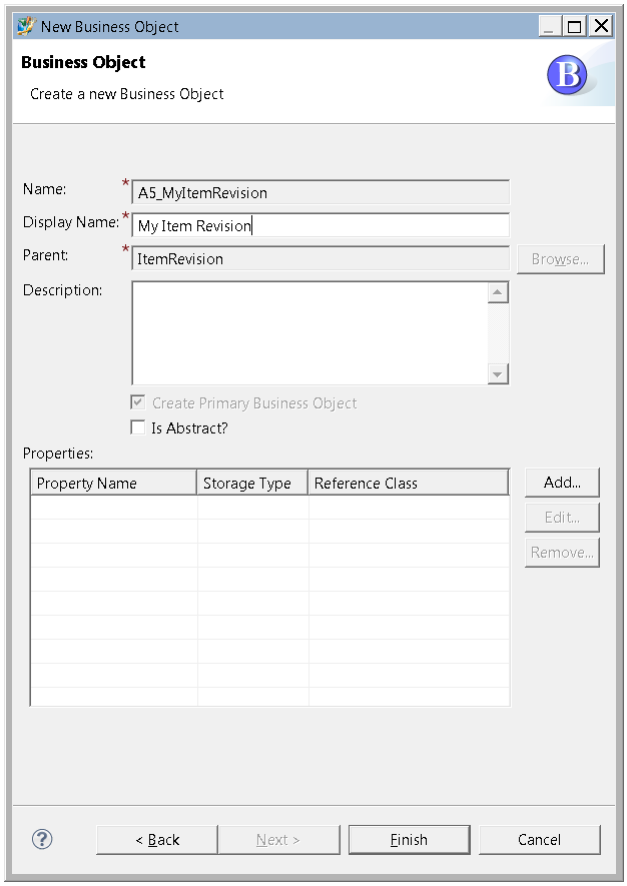
3. The **Business Object** dialog box displays the name of the revision to be created in the **Name** box, and displays the parent of the revision in the **Parent** box. You cannot change these values.

© 2025 Siemens

a. In the **Display Name** box, type the name of the item revision as you want it to appear in the user interface.

b. In the **Description** box, type a description for the new business object revision.

c. Click the **Add** button to the right of the **Properties** table to add a property to the revision business object.

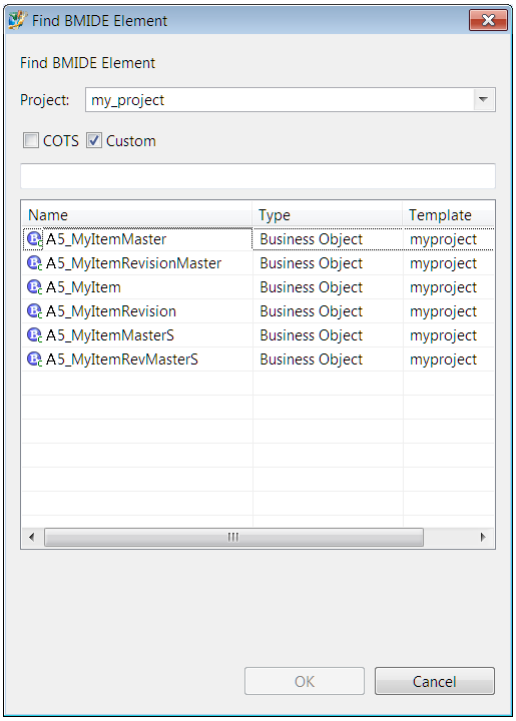


d. Click **Finish**.

The new business object appears in the **Business Objects** folder. A **c** on the business object symbol indicates that it is a custom business object.

When you create a custom item business object, the following business objects are also created: *custom-item***Master**, *custom-item***MasterS**, *custom-item***Revision**, *custom-item***RevisionMaster**, and *custom-item***RevMasterS**. To find the master, revision, and revision master that you created, click the **Find** button at the top of the **BMIDE** view and search for the new business object name, for example, **A5_MyItem**.

© 2025 Siemens



> Note:
>
> The **Master** form holds properties for the item, and the **RevisionMaster** form holds properties for the item revision. For style sheets used on the form display, type inheritance is not allowed. A style sheet registered on a parent form display is not used for its subtypes, and different properties are shown on the custom **RevisionMaster** form than are shown on the **RevisionMaster** form of the parent business object. If you want to show the same properties on the custom **RevisionMaster** as on the parent **RevisionMaster**, you must create a new style sheet for the custom form display.

4. Perform additional steps to configure the new item business object type:

   - **Configure the properties that appear in the new item dialog box** when an end user creates an instance of the item business object.

     For example, to make the **Configuration Item** check box appear in the **New Item** wizard, open the item business object type, click the **Operation Descriptor** tab, click the **CreateInput** tab, and select the **is_configuration_item** property. On the **Property Constants** tab, select the **Visible** property constant.

   - Set **business object constants** for the new item business object type.

     For example, to allow checkout of a new item when it is created, set the **Fnd0AllowCheckOutOnCreate** business object constant to **true**.

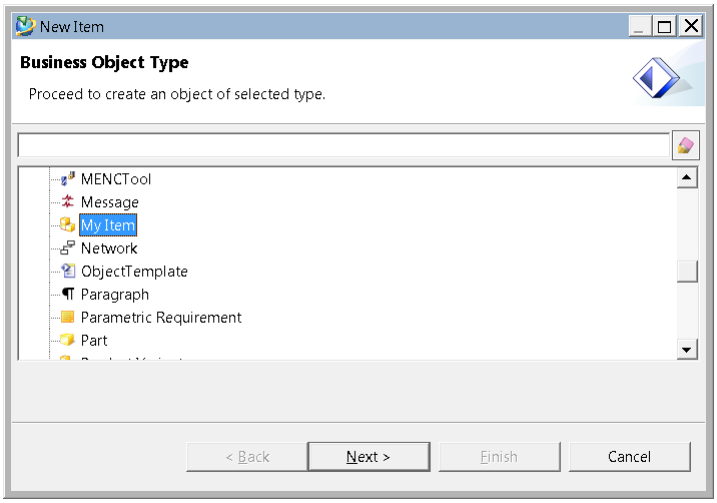
© 2025 Siemens

5. To save the changes to the data model, choose **BMIDE→Save Data Model** on the menu bar, or click the **Save Data Model** button on the main toolbar.

6. Deploy your changes to the test server. Choose **BMIDE→Deploy Template** on the menu bar, or select the project and click the **Deploy Template** button on the main toolbar.

7. After deployment, test your new business object in the Teamcenter rich client by creating an instance of it.

   For example, in the My Teamcenter application, choose **File→New→Item**. (You can also create business objects in the Teamcenter rich client by choosing **File→New→Other**.) Your new business object appears in the **New Item** dialog box. Choose your new business object and create an instance of it.



Note:

If you select the **Is Abstract?** check box during creation of the new business object, you cannot create an instance of the business object.

Tip:

You can **create your own custom icon** to assign to the new business object.

Basic item data model

An item in Teamcenter is a structure of related objects. The basic structure of any item consists of the following minimum objects:

Item
ItemMaster (Form business object)
ItemMasterS
ItemRevision

© 2025 Siemens

ItemRevisionMaster (Form business object)
ItemRevMasterS

- **Item**

  Collects data that is globally applicable to all revisions of the item. You typically add custom attributes directly to the custom item business object.

- **ItemMaster** (Form)

  A form object that is often used to extend the stored property data for an item to include data unique to the customer.

- **ItemMasterS**

  The storage object for the item master form.

- **ItemRevision**

  Collects data that is applicable to a single revision of the item.

- **ItemRevisionMaster** (Form business object)

  A form object that is often used to extend the stored property data for an **ItemRevision** object to include data unique to the customer.

- **ItemRevMasterS**

  The storage object for the item revision master form.

Extending item business objects

The Business Modeler IDE can be used to define more **Item** business objects in addition to those provided with base Teamcenter.

Reasons for extending the **Item** business objects are:

- Having more types of **Item** business objects may be a useful approach of categorizing data making it easier for users to find data and understand the differences between different kinds of data stored in the system.

- Different rules for naming conventions, deep copy rules, and so on, can be configured for one type of **Item** business object compared to another type.

- Default process model association for one type of **Item** business object versus another is easier to implement.

© 2025 Siemens

- Different designs for the **Item Master** and **ItemRevision Master** forms may be desired. Each type of **Item** business object can have unique and different master form definitions.

In the following example, a new type of **Item** business object (**EndItem**) has been defined so that the customer can define customer specific attribute data that Teamcenter stores for this type of data.

EndItem
EndItem Master (Form business object)
EndItem Revision
EndItem Revision Master (Form business object)

How do I hide item types so that when users create a new item, these types are not seen?

To hide item types from creation, make business object display rules. You must create a rule for every item type that you want to hide from end users.

Form business objects

Introduction to form business objects

Form business objects manage underlying product information and control how this information is displayed to users.

There are two basic ways to manage product information using form business objects: store product information in a form definition class in the database, or store product information as a text file in a Teamcenter volume. You can display a form to users as a simple list of properties, or you can use a form definition file to create a custom form.

By default, forms are displayed as a simple list of properties. The form is similar to the **Properties** dialog box. However, you can only use this technique if you are storing your information in a form definition class in the database. Otherwise, it is not possible to determine which properties to list.

Create a form business object

Use the **Form** business object or its children to create a form to hold attributes. All **Item** business objects have a form associated with them, but these are typically created when you use the New Business Object wizard to create a new **Item** business object. Use the following procedure to create additional **Form** business objects to use with your **Item** and **ItemRevision** business objects.

1. Click the **Find** button at the top of the **BMIDE** view and search for the **Form** business object.

2. In the **Business Objects** folder, right-click the **Form** business object or one of its children and choose **New Business Object**.

   The Create New Form Business Object wizard runs.

3. In the **Create New Form Business Object** dialog box, enter the following information:

© 2025 Siemens

a. The **Project** box defaults to the already-selected project.

b. In the **Name** box, type the name you want to assign to the new business object in the database.

When you name a new data model object, a prefix from the template is automatically affixed to the name to designate the object as belonging to your organization, for example, **A4_**.

c. In the **Display Name** box, type the name as you want it to appear in the user interface.

d. The **Parent** box displays the business object you already selected as the parent business object.

e. In the **Description** box, type a description for the new business object.

f. Choose one of the following in the **Storage Class** pane:

- Click **Use new class** to create a new storage class to store the attributes.

  In the **Class Name** box, type the name for the new form storage class. Click the **Browse** button to the right of the **Parent Name** box if you want to select a different class to be the parent of the new form storage class.

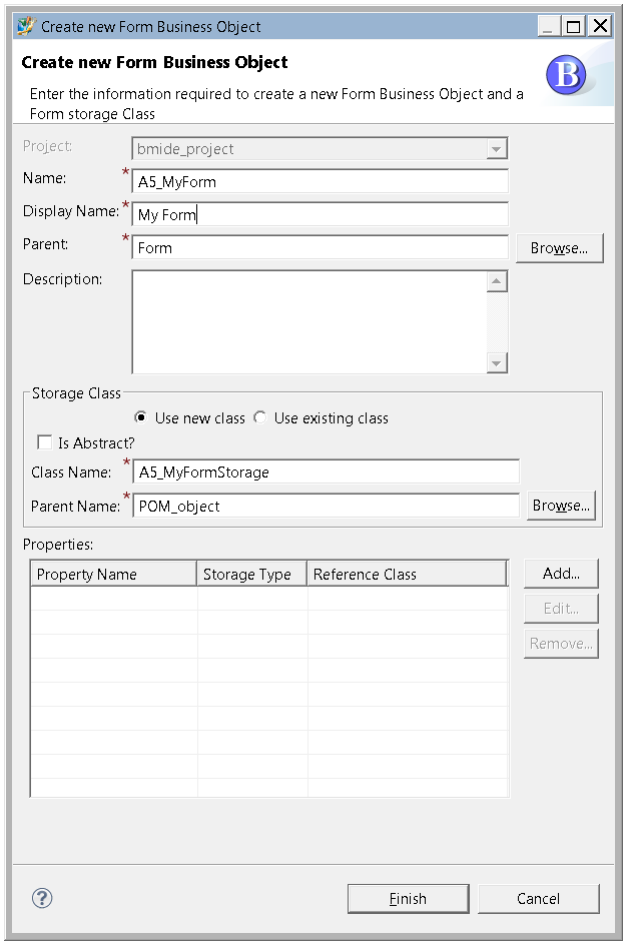
- Click **Use existing class** to use an existing class to store the attributes.

  Click the **Browse** button to the right of the **Class Name** box if you want to select a different class to be the form storage class.

g. Click the **Add** button to the right of the **Properties** table to **add a property to the business object**.

The New Property wizard runs.

© 2025 Siemens



h. Click **Finish**.

The new business object appears in the **Business Objects** folder. A **c** on the business object icon indicates that it is a custom business object.

4. To save the changes to the data model, choose **BMIDE→Save Data Model**, or click the **Save Data Model** button on the main toolbar.

5. Deploy your changes to a test server. Choose **BMIDE→Deploy Template** on the menu bar, or select the project and click the **Deploy Template** button on the main toolbar.

6. After deployment, test your new business object in the Teamcenter rich client by creating an instance of it.

   For example, in the My Teamcenter application, choose **File→New→Form**. (You can also create business objects in the Teamcenter rich client by choosing **File→New→Other**.) In the **New Form** dialog box, choose the new form business object from the list of available forms. Create the instance of the form and click **OK**.

© 2025 Siemens

Hide master forms

When item or item revisions are created in the rich client, master forms are automatically created and attached. By default in the rich client, the creation dialog box has a page for entering master form values, and the master form appears in rich client views such as the **Home** view navigation tree.

Hiding the master form page from the creation dialog box

If for some reason you want to exclude the master form page from the creation dialog box, perform the following procedure. The master form is still created.

> Tip:
>
> If any of the properties on the master form are mandatory, or if they need to be edited by the user, but you still prefer to hide the form page in the creation dialog box, then you can use the Business Modeler IDE to create a compound property on the corresponding **Item** or **ItemRevision** business object that points to the attribute on the master form.

1. Open the **Item** or **ItemRevision** business object or one of its children.
2. Click the **Operation Descriptor** tab.
3. On the **CreateInput** tab, select the **IMAN_master_form** property (for item business objects) or the **IMAN_master_form_rev** property (for item revision business objects).
4. Click **Edit**.
5. Clear the **Visible** check box.

Hiding the master form elsewhere in the rich client

To hide a master form from appearing elsewhere in the rich client, such as in the **Home** view navigation tree, edit the *business-object*_**DefaultChildProperties** preference to remove the **IMAN_master_form** or **IMAN_master_form_rev** value.

This example procedure shows how to easily remove the **IMAN_master_form** value from the **Item_DefaultChildProperties** preference. Typically, you would probably want to repeat the procedure and remove the **IMAN_master_form_rev** value from the **ItemRevision_DefaultChildProperties** preference.

1. In the rich client, choose **Edit→Options** to display the **Options** dialog box.
2. In the options tree, under the **General** category, select **Item Revision**.
3. On the **General** tab, in the **Shown Relations** list, select **Item Masters** and then click .

© 2025 Siemens

Item Masters moves to the **Available Relations** list.

4. Click **OK**.

End-user form creation

End users create forms in the Teamcenter rich client using one of the following methods:

- Choose **File→New→Form**. This is used to create a stand-alone form object in a container (like a folder) or a form associated with an existing item or item revision object. End users select the type of form they are creating from the list of form types on the **New Form** dialog box.
- Choose **File→New→Item**. When an item is created, at least two form objects are also created: the item master and item revision master.
- Choose **File→New→Change**. When change objects are created, additional forms may be automatically created for the change revision object. How many forms and of what type are set when you create new change objects.
- Use a workflow handler. Form objects may be created automatically during a workflow process by using the **EPM-create-form** action handler.

To view a form in the Teamcenter rich client, select the form object and click the **Viewer** tab.

Threshold properties must exist on the new Form storage classes

If the standard form business object has any threshold properties and you change the form storage class on that standard form type, the form type does not include the threshold properties. This causes exceptions in the Business Modeler IDE, loader, comparator, and so on.

A *threshold property* is property on a form that is derived from its form storage class. The property has one of the following: LOV attachment, naming rule attachment, property business modeler operation, extension attachment, and so on. A property is referred to as a threshold property if it has required functionality connected to it such as business rules, list of values, or extension rules.

When business rules, lists of values, and so on, are defined on a property by the owner of a Business Modeler IDE template, they are assumed to be present on the property and there may even be code logic that assumes the functionality is present on the property. If a subsequent customer switches the form storage class and chooses a class with a different set of attributes, the original threshold properties are not available on the **Form** business object. This creates problems where threshold properties become invalid or are dangling.

If the customer has a **Form** type with threshold properties and later switches the **Form** storage class on that form type and generates the custom template, loading of the custom template in the Business Modeler IDE client fails and throws one of the following model errors:

© 2025 Siemens

`The Form Storage Class on the form type` *form-name* `cannot be changed to the class` *new-form-storage-class* `unless the class has the following threshold properties:` *threshold-properties* `Add the missing properties from the original form storage class to this new form storage class.`

`The Form Storage Class cannot be removed from the form type` *form_name* `because the form requires the following threshold properties:` *threshold-properties*`. Either set this form to point to the original form storage class or use a new class that adds the threshold properties from the original form storage class.`

If either of the two errors occur in your Business Modeler IDE, you must manually add the missing threshold properties to the new form storage class in the custom template. The property definition must be identical to the original property definition. This must be performed manually by editing the XML source files in your template to add the missing threshold property to your form storage class. After you complete the edit, save the file, close it, and perform a **Reload Data Model** action in the Business Modeler IDE. This reloads the XML and performs the validation to ensure there are no errors.

If after adding the missing threshold properties, you do not want to see these attributes on this form business object, set the **Visible** property constant to **False** on the form property that corresponds to the attribute.

Changing the form storage class

Switching the storage class of a **Form** business object is not allowed.

In previous versions, you could switch the storage class of a **Form** business object if the existing storage class did not meet your requirements, which led to dangling data model objects. For example, if a template contained a **Form** business object with business rules and a list of values configured on the form properties and the dependent template switched storage classes, the business rules and list of values attachments from the parent template became invalid and the database contained dangling data model objects.

Perform the following steps to resolve the issue:

1. Define a new property on the **Form** business object with the Business Modeler IDE.

2. If the existing form properties are not useful, hide them by setting the **Visible** property constant to **false** on the form property that corresponds to the attribute.

Note that if you use the Business Modeler IDE Deploy wizard to commit this visibility change to the server, the server must be reset to see these changes in the clients.

© 2025 Siemens

Dataset business objects

Introduction to dataset business objects

Datasets are objects used to manage file data associated with external software applications. They typically consist of a single application data file or logical groupings of application data files. There are numerous types of datasets predefined in Teamcenter. However, your site may need to add more types to be able to manage your site's specific application data files and the viewing/editing software applications associated with these files.

Create a dataset business object

Use the **Dataset** business object to represent a file from a specific software application. For example, files created in Microsoft Word are represented by the **MSWord** dataset object, text files are represented by the **Text** dataset object, and so on.

You can **associate a tool with each dataset type** so that the appropriate software application launches when you open a file in Teamcenter.

1. Choose one of these methods:

   - On the menu bar, choose **BMIDE→New Model Element**, type **Dataset** in the **Wizards** box, and click **Next**.

   - Click the **Find** button at the top of the **BMIDE** view, search for the **Dataset** business object, right-click it, and choose **New Business Object**.

2. In the **Business Objects** folder, right-click the **Dataset** business object and choose **New Business Object**.

   The New Dataset wizard runs.

3. In the **Dataset** dialog box, enter the following information:

   a. The **Project** box defaults to the already-selected project.

   b. In the **Name** box, type the name you want to assign to the new business object in the database.

      When you name a new data model object, a prefix from the template is automatically affixed to the name to designate the object as belonging to your organization, for example, **A4_**.

   c. In the **Display Name** box, type the name as you want it to appear in the user interface.

   d. In the **Parent** box, **Dataset** is already selected as the parent business object.

© 2025 Siemens

e. In the **Description** box, type a description of the new business object.

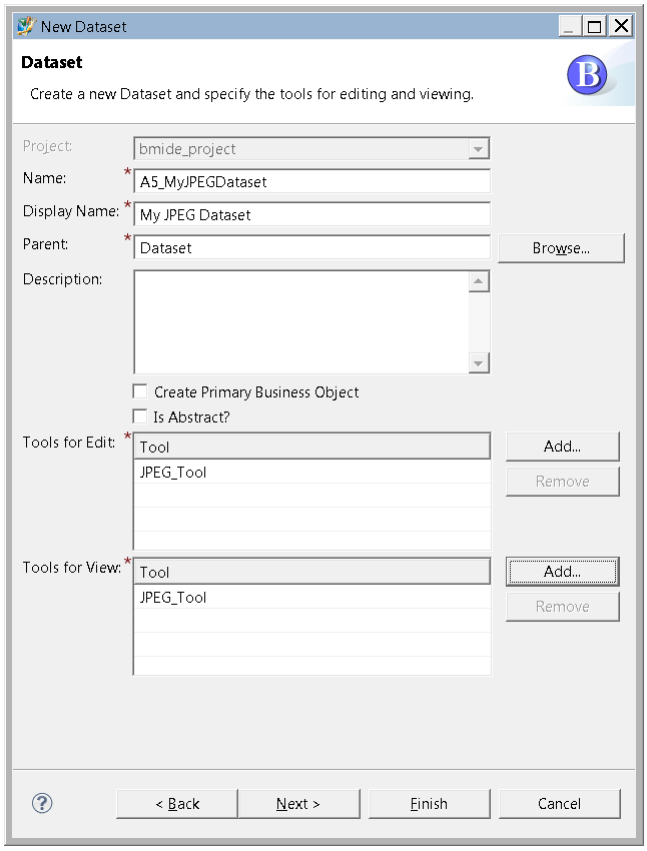
f. Select **Create Primary Business Object** to make a new class that stores the data for the new business object. Clear this option to set the storage class as the same used by the parent business object (in this case, the **Dataset** class).

g. Select **Is Abstract?** if instances of the business object will not be created in user interfaces such as the rich client. For example, you may want to select this if the business object is intended as a folder for child business objects.

h. To the right of the **Tools for Edit** pane, click the **Add** button to select the software application to launch when the dataset is selected in Teamcenter. If the tool does not exist, you must create one using the **Tool** option.

i. To the right of the **Tools for View** pane, click the **Add** button to select the software application to be used to view the dataset files. If the desired tool does not exist, you must create one using the **Tool** option.

j. Click **Next** if you want to add references and parameters to the dataset. Otherwise, click **Finish** to complete the dataset creation.



4. If you clicked **Next**, the **References** table appears. Click the **Add** button to the right of the **References** table to add file name references to associate with the dataset.

© 2025 Siemens

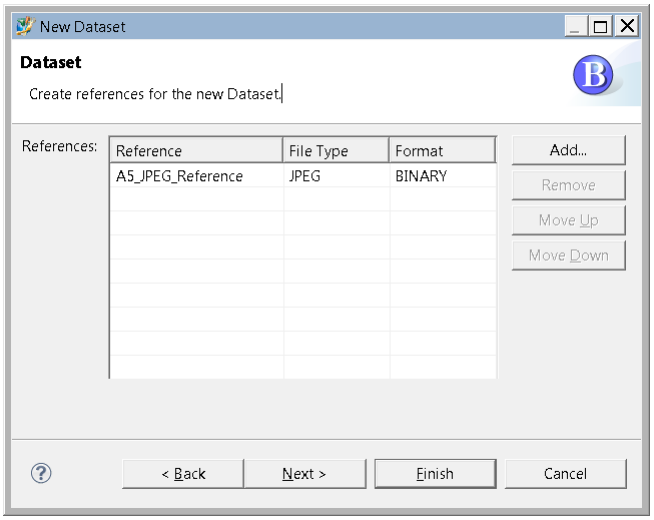
The Add Dataset Reference wizard runs. Perform the following steps in the **Dataset Reference** dialog box:

a. For **Reference**, type a unique name for this file reference.

b. For **File Type**, enter the file name extension (for example, **txt**, **pdf**, **doc**, and so on).

c. For **Format**, choose whether the files are binary, object, or text.

d. Click **Finish**.



5. Click **Finish** in the New Dataset wizard to complete the dataset creation, or if you want to change the action taken when the dataset is launched, click **Next**.

6. If you clicked **Next**, the **Tool Action** table appears. Click the **Add** button to the right of the **Tool Action** table to modify the action that a software application takes when a dataset is launched.

   The Dataset Tool Action Definition wizard runs. Perform the following steps:

   a. Click the **Browse** button to the right of the **Tools** box to select the tool you previously chose to use for viewing or editing this dataset. (The only available tool is the one you previously selected.

   b. Click the arrow in the **Operations** box to choose the operation to use with the dataset (for example, **Open**, **OpenUsing**, **Print**, **PrintUsing**, or **Send**).

      The **OpenUsing** operation allows the user to select the tool to use for opening the dataset, and the **PrintUsing** operation allows the user to select the tool to use to print the dataset.

   c. To set the file name extensions to associate with the action, click the **Add** button to the right of the **References** table.

© 2025 Siemens

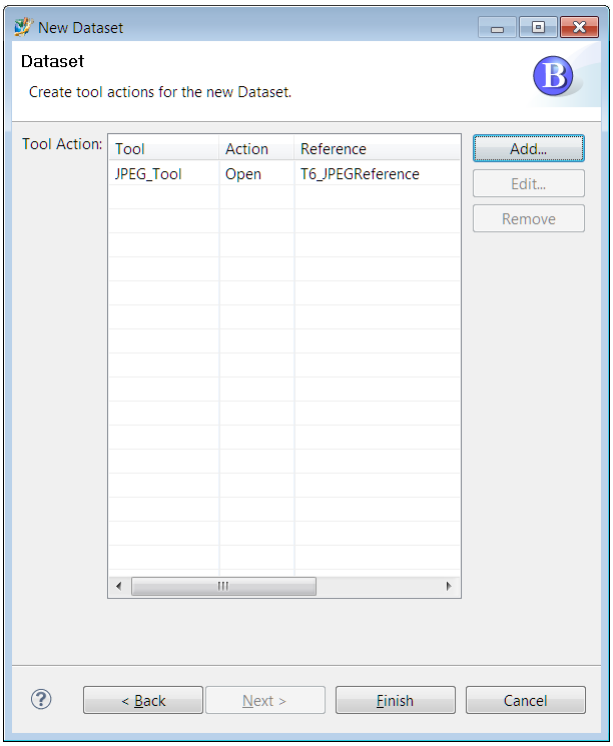
The Add Reference wizard runs. In the **Reference** dialog box, click the arrow in the **Select the Reference Name** box to choose the file name reference for this tool action. Select the **Export** check box to export the dataset out of the database.

d. To add parameters to be passed with the action, click the **Add** button to the right of the **Parameters** box.

The Add Parameter wizard runs. Select a parameter and click **Finish**.

e. Click **Finish** in the **Dataset Tool Action Definition** dialog box.

f. Click **Finish** in the **New Dataset** dialog box.



7. To save the changes to the data model, choose **BMIDE→Save Data Model**, or click the **Save Data Model** button on the main toolbar.

8. Deploy your changes to a test server. Choose **BMIDE→Deploy Template** on the menu bar, or select the project and click the **Deploy Template** button on the main toolbar.

9. After deployment, test your new business object in the Teamcenter rich client by creating an instance of it.

For example, in the My Teamcenter application, choose **File→New→Dataset**. Click the **More...** button and choose the new dataset business object from the list of available ones.

© 2025 Siemens

> Note:
>
> To make the custom dataset appear on the list of available dataset types, open the **TYPE_DISPLAY_RULES_list_types_of_subclasses** preference and add **Dataset** to it.

You can also create an instance of the custom dataset in the Teamcenter rich client by choosing **File→New→Other**.

Named references

Named references are files attached to a dataset object. A single dataset object may have one or more named references. To view the named references of a dataset from the Teamcenter rich client, in My Teamcenter, select the dataset and choose **View→Named References**, or right-click the dataset and choose **Named References**.

Named references are not to be confused with **dataset references**, which are the kind of applications associated with a dataset type.

Dataset file creation process in the Teamcenter rich client

When a new dataset is created in the Teamcenter rich client, unless a dataset file is simultaneously imported, the dataset has no named references or files stored on a Teamcenter volume. When a Teamcenter rich client user opens the new dataset, the following chain of events takes place:

1. If the user has write access to the dataset object, an implicit (automatic) checkout occurs. Thereby the user maintains exclusive write access to the data during the modification session.

2. What happens next is not directly visible to the user:

    a. Teamcenter creates a temporary operating system (OS) directory.

    b. Because initially there is no file associated with the dataset, Teamcenter creates a 0-size file and exports the file to the temporary OS directory. The format of the file (binary/text) and the extension of the file are as defined in the references part of the dataset business object definition.

    c. Teamcenter executes the OS software application associated with the dataset tool definition and passes the file to the application as a parameter.

3. The user sees the following:

    a. The OS software application executes with the 0-size file loaded.

    b. The user modifies the file using the OS software application.

© 2025 Siemens

c. When a modified file is saved (without terminating the application), the modified file is written back to the temporary OS directory overwriting the initial exported 0-sized file. Each save performed (without terminating the application) continues to overwrite the file in the temporary directory without updating the data file stored in Teamcenter.

d. The user terminates the OS software application.

4. What happens next is not directly visible to the user:

a. Teamcenter senses that the application process has terminated and checks the temporary OS directory to see if the file it originally exported there has been changed or if new files have been added to the directory.

b. If there are changed or new files, Teamcenter imports the files and creates the next version of the dataset object.

c. The implicit checkout on the dataset object is reversed (implicit checkin).

If the dataset is opened again, a similar process occurs except that the current file (or set of application data files) for the dataset is exported instead of a 0-size file.

Configure tools to open or view dataset files in external applications

So that users can directly open or view a data file (or a logical grouping of data files) associated with an instance of a dataset business object in a software application external to the Teamcenter rich client, administrators can define and assign tools to the dataset business object type. The list of tools appears in the Teamcenter rich client when the user selects a dataset object and chooses **File**>**Open With** or **File**>**View With**.

Data files can potentially be opened or viewed by multiple applications. To provide flexibility for opening or viewing data files, you can add multiple tools with differing media types to a dataset business object type definition.

The application that a tool launches depends on a combination of two settings:

1. In the tool definition, the media type.

2. On the client machine, the application associated with the media type.

> Note:
>
> Other than expressly defined Teamcenter applications such as Lifecycle Visualization, external software applications are not launched from Active Workspace.

Set the media type for a tool
Add a tool to a dataset business object type
Set up media types on a rich client workstation

© 2025 Siemens

Set the media type for a tool

To set the media type for a tool, perform the following steps in the Business Modeler IDE.

1. Expand the **Extensions**>**Options**>**Tool** folder and double-click the tool.

   The tool properties are displayed in a new view.

2. In **MIME/TYPE**, enter the media type and subtype separated by a slash (/).

Add a tool to a dataset business object type

To add a tool to a dataset business object type so that the tool will be available in the rich client, perform the following steps in the Business Modeler IDE.

1. Expand the **Business Objects**>**Dataset** folder, locate the dataset business object to which you want to add the tool, and double-click the object.

   The object properties are displayed in a new view.

2. On the **Dataset Properties** tab, add the tool to the **Tools for Edit** and **Tools for View** tables.

3. On the **ToolActions** tab, repeat the following steps to add **Open** and **View** actions using the tool.

   a. Next to the **Tool Action** table click **Add**.

   b. In the **Dataset Tool Action Definition** dialog box, set the following:

For this setting	Do this
Tools	Click **Browse** and choose the tool to be used for the action.
Operations	Choose the **Open** or **View** action.
References	A. Click **Add**. B. In the **Reference** dialog box, select the **Export** check box and then click **Finish**.

   c. In the **Dataset Tool Action Definition** dialog box, click **Finish**.

4. Save and deploy the data model.

Set up media types on a rich client workstation

To ensure that a tool can be used to launch the intended application for a dataset file, on the workstation where the rich client runs, associate the media type with the intended application.

© 2025 Siemens

- In Linux operating systems, a Teamcenter-supplied *.mailcap* file in the *portal* folder of the rich client installation maps media types to applications. The media type for a tool must be specified in the *.mailcap* file in order for the tool to launch an application.

  Example:

  /usr/tc/Tc123_1106_4T/portal/.mailcap

```
##########################################################################
##########
# Modification Log:
# Date         Name                    Description
# ----         -----------             -----------------
# 20-Oct-2005  Vilas Tharakan              Add htm and html
mime-types and history.
# 20-Oct-2005  Vilas Tharakan            Checked the support for
Linux
# 17-Nov-2005  roudebus                  Rename and move to mailcap
folder
# $HISTORY$
##########################################################################
##########
text/plain; \
/usr/bin/gedit
text/htm; \
/usr/bin/mozilla
text/html; \
/usr/bin/mozilla
application/ugs-portal; \
/usr/tc/Tc123_4T_1209/portal

application/pvportal; \
/usr/tc/Tc123_4T_1209/portal/start_vis

application/x-visnetwork; \
/usr/tc/TcVis123_1210/bin/vmulaunch
```

- In Windows operating systems, the association of a media type with an application is defined in the Windows registry. The following example shows an easy way to define the association.

  Example:

  1. In Windows File Explorer, locate a file with the filename extension for the data files normally associated with a dataset business object type instance.

  2. Right-click the file, and from the context menu choose **Open with**>**Choose another app**.

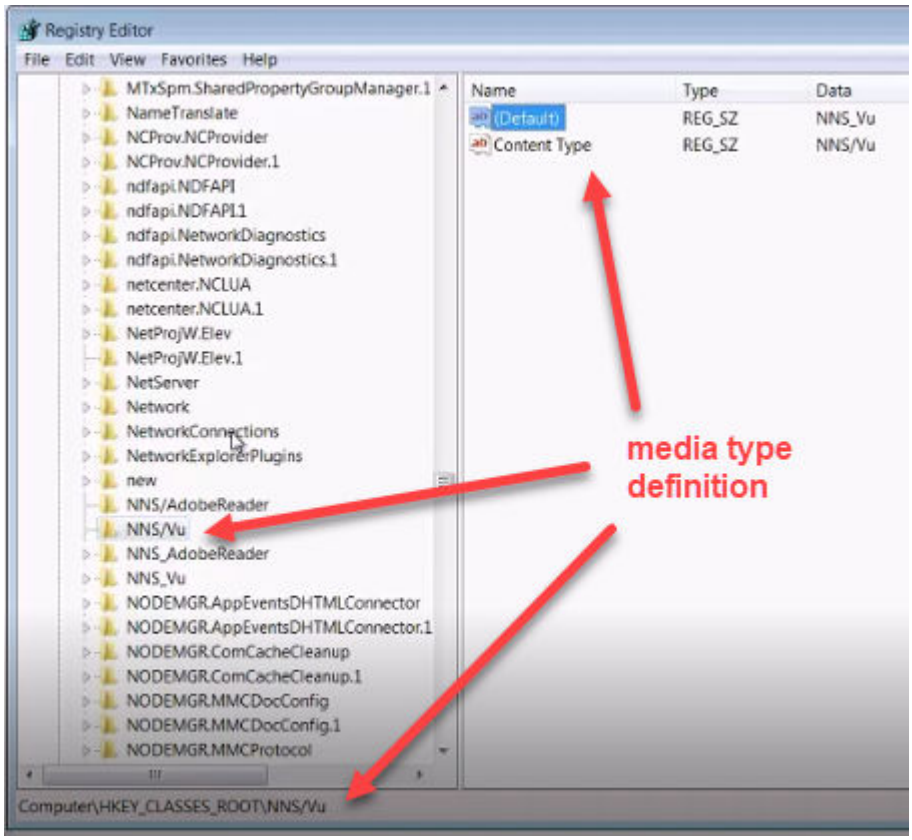
© 2025 Siemens

3. In the **How do you want to open this file?** dialog box, mark the checkbox for **Always use this app to open** *extension* **files**.

4. Select an app and then click **OK**.

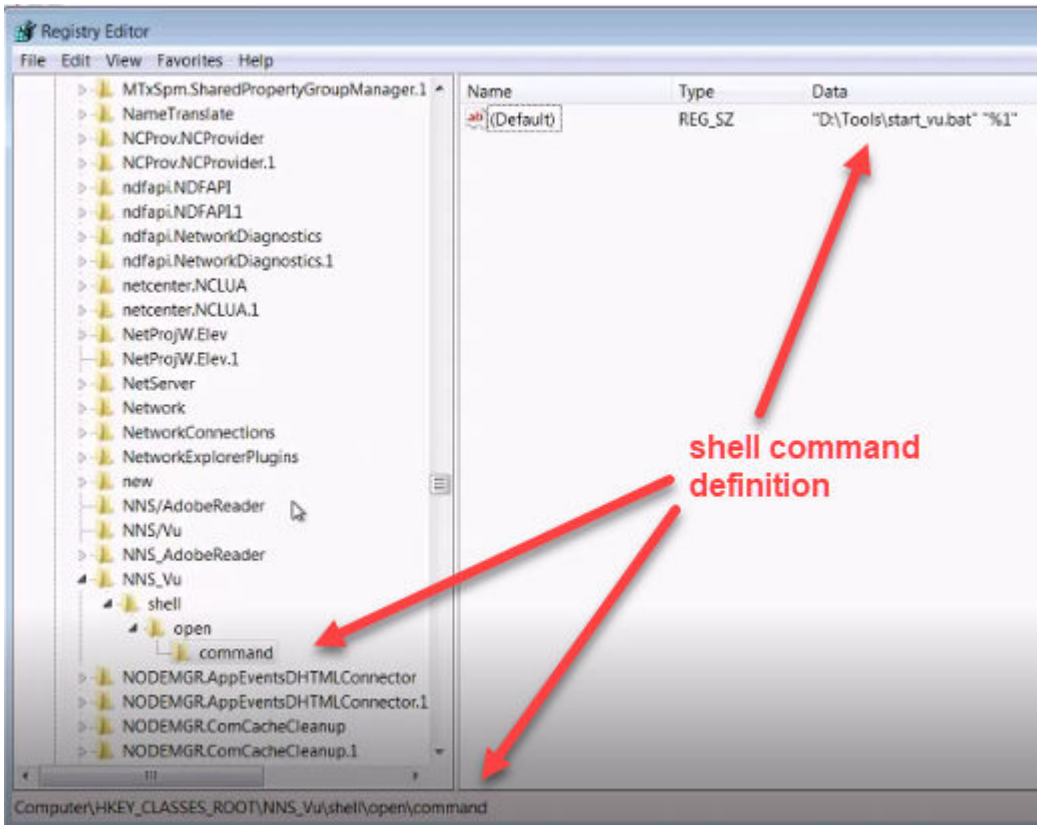
A user may want to view a dataset file using one application, but open and edit it in another. To enable launching a dataset file with a given filename extension in a choice of multiple applications, define distinct media types for the applications.

1. On the machine where the rich client runs, in the Windows registry under HKEY_CLASSES_ROOT, add a unique media type (*type/subtype*).



2. Associate the new media type with a shell command for an application.

© 2025 Siemens



3. In BMIDE, define a tool for the new media type.

4. Add tools with appropriate media types, and their corresponding tool actions, to the dataset business object type.

Configure dataset file upload security

Dataset business objects are configured to accept certain file types for upload. For example, the **PDF** dataset accepts ***.pdf** files, and the **HTML** dataset accepts ***.htm** or ***.html** files. This is configured on the **References** tab in the Business Modeler IDE.

However, some datasets are configured to accept any file type (***.*** or ***). This poses a security risk, because executable files could be uploaded that can contain viruses and malware.

You can strengthen security by allowing an administrator to prevent datasets from accepting any file type. This is accomplished with the **Fnd0DatasetFileExtensionRestrict** business object constant. The administrator can set this constant on a dataset and list the file types to block, for example, executables such as **.exe**, **.bat**, and **.cmd**.

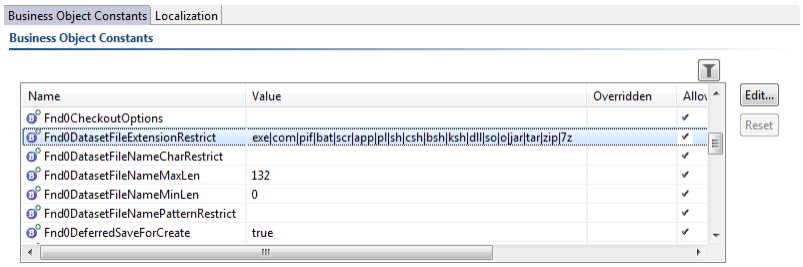
You can also use the following business object constants to restrict files from being uploaded based on file name length, characters, or patterns:

Fnd0DatasetFileNameCharRestrict

© 2025 Siemens

Fnd0DatasetFileNameMaxLen
Fnd0DatasetFileNameMinLen
Fnd0DatasetFileNamePatternRestrict

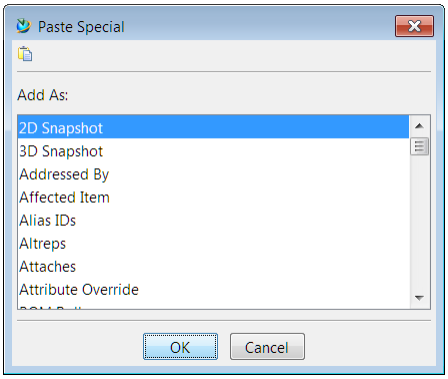
To change the value of a business object constant on a dataset business object, open the dataset business object, click the **Dataset Properties** tab, and scroll down to the **Business Object Constants** table. Select the business object constant and click the **Edit** button.



Relation business objects

Introduction to relation business objects

To see the available relations that can be used to relate source and target objects, in the rich client My Teamcenter application, copy a source object, select a target object, and choose **Edit→Paste Special**.



Following are some common types of relation business objects. These are all children of the **ImanRelation** business object.

- Specification relations

  The **IMAN_specification** business object defines this relation. Specification relations are detailed methods, designs, processes, and procedures used to satisfy requirements. A specification relationship can only be established with an item revision, not an item. Although requirements may remain fairly constant for a product (item), actual manufacturing methods, designs, processes, and procedures may change drastically from model to model (item revisions).

- Requirement relations

© 2025 Siemens

The **IMAN_requirement** business object defines this relation. Requirement relations are criteria that must be satisfied by this item or item revision. However, requirements often do not specify how this criteria should be satisfied. For example, a requirements relation may specify maximum weight for an item revision but not how to construct it. Extend the business object to create your own specification relations.

- Manifestation relations

  The **IMAN_manifestation** business object defines this relation. Manifestation relations are non-defining snapshots of a particular aspect of an item or item revision at a particular moment in time. For example, numerically controlled (NC) program files are a common manifestation. Consider that they represent one aspect of an item revision (that is, machining information) and that this information is only accurate as long as the item revision does not change. If the item revision does change, the NC program files may no longer be accurate and may need to be re-created.

- Reference relations

  The **IMAN_reference** business object defines this relation. Reference relations describe a general non-defining relationship of a workspace object to an item or item revision. This relation type can be thought of as a miscellaneous relation type. Typical examples of reference relations are white papers, field reports, trade articles, customer letters, lab notes, and so on.

Tip:

- The *business_object*_**default_relation** preference specifies the default relation that is created when an object is pasted under an instance of the business object.
- If you want to **create your own kind of relation business object**, extend the **ImanRelation** business object.

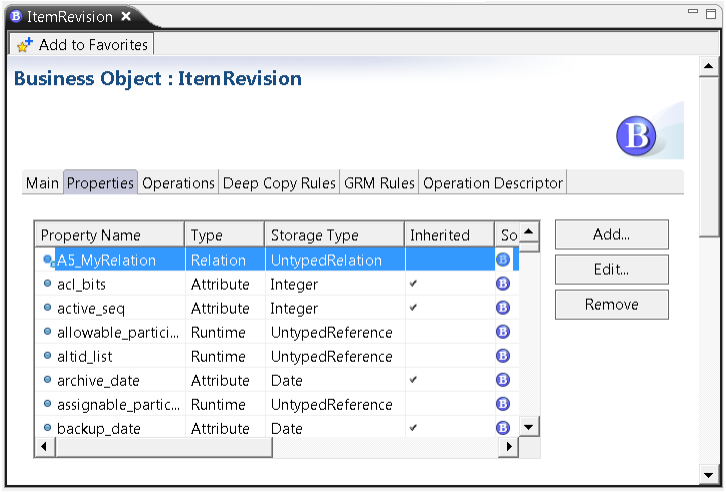
Create a relation business object

Items in Teamcenter are related to one another using relation business objects, which are children of the **ImanRelation** business object. You can create your own relation business objects as children under this business object.

1. To create a new relation business object, right-click the **ImanRelation** business object or one of its children and choose **New Business Object**.
2. To make properties required or visible on the new relation business object, open the business object and click the **Operation Descriptor** tab.
3. Make the new relation business object appear on the target business object.
   a. Open the target business object (for example, **ItemRevision**).
   b. Click the **Properties** tab and click the **Add** button to the right of the properties table.

© 2025 Siemens

c. Choose **Relation** as the property type, click the **Browse** button to the right of the **Relation Business Object** box, and select the new relation business object.



Identifier business objects

Create an identifier business object

Identifier business objects contain attributes, such as supplier name, address, and cost, which are used with **alias ID rules** and **alternate ID rules**.

1. Click the **Find** button at the top of the **BMIDE** view. Type **Identifier** in the search box and click **OK**.

   The **Identifier** business object is selected in the **Business Objects** folder.

2. In the **Business Objects** folder, right-click the **Identifier** business object and choose **New Business Object**.

   The New Identifier wizard runs.

3. In the **Identifier** dialog box, enter the following information:

   a. The **Project** box defaults to the already-selected project.

   b. In the **Name** box, type the name you want to assign to the new business object in the database.

   When you **name a new data model object**, a prefix from the **template** is automatically affixed to the name to designate the object as belonging to your organization, for example, **A4_**.

   c. In the **Display Name** box, type the name as you want it to appear in the user interface.

© 2025 Siemens

d. In the **Parent** box, **Identifier** is already selected as the parent business object.

e. In the **Description** box, type a description of the new identifier.

f. Select **Create Primary Business Object** to make a new class that stores the data for the new business object. Clear this option to set the storage class as the same used by the parent business object (in this case, the **Identifier** class).

g. Select **Is Abstract?** if instances of the business object will not be created in user interfaces such as the rich client. For example, you may want to select this if the business object is intended as a folder for child business objects.

h. Click the **Add** button to the right of the **Properties** table to add a property to the business object.

The New Property wizard runs.

For instructions about how to fill in the **Persistent Property** dialog box, see *Add a persistent property*.

i. Click **Next**.

4. Create a supplemental **Identifier** business object, which is comparable to a revision. In the **Identifier** dialog box, enter the following information:

a. The value in the **Name** box defaults to the name of the master identifier with **Rev** added to the end.

b. In the **Parent** box, **Identifier** is already selected as the parent business object.

c. In the **Description** box, type a description of the new identifier revision.

d. Click the **Add** button to the right of the **Properties** table if you want to add properties to the new business object.

For instructions about how to add properties see *Add a persistent property*.

e. Click **Finish**.

The master and supplemental **Identifier** business object are created.

5. To save the changes to the data model, choose **BMIDE→Save Data Model**, or click the **Save Data Model** button on the main toolbar.

6. Deploy your changes to a test server. Choose **BMIDE→Deploy Template** on the menu bar, or select the project and click the **Deploy Template** button on the main toolbar.

© 2025 Siemens

Identifier business objects can only be used in the context of alias ID rules and alternate ID rules.

Alias identifiers

Alias identifiers are used to store part numbers and other attribute information for similar parts and they can be associated with many items or item revisions.

Alias identifiers allow you to store information about external entities. For example, parts can be stored according to their internal naming conventions and also according to the naming conventions of other companies, such as suppliers. You can also use alias IDs to maintain a cross reference of the relationship between other manufacturer's part numbers and your own.

In this scenario, each supplier is considered a separate context or a single context called *suppliers* and can be created. You then create an identifier business object and associated attributes, such as supplier name, address, and cost. Contexts are defined by your administrator and you assign them when you **create an alias ID**.

Assign an alias identifier in the rich client by choosing **File→New→ID**.

Alternate identifiers

Alternate identifiers store information about part numbers and attributes of the same part from different perspectives. They allow different user communities to identify and display an item or item revision according to their own rules rather than by the rules of the user who created the object.

Assigning alternate identifiers to a part at different stages of development and production allows you to maintain a history of the life cycle of the part.

The alternate identifier functionality is enabled by creating identifier business objects, accessed by users of the rich client from the **Revise**, **New Item**, and **Save As** dialog boxes or by choosing **File→New→ID**.

Create a run-time business object

Business objects can be either persistent or run-time business objects. Persistent business objects are stored in the database in their associated storage class. The run-time business objects are not stored in the database but are calculated at run time.

Run-time objects (children of the **RuntimeBusinessObject** object) function much like standard business objects. Although run-time business objects are not persisted in the database as business objects, you can add properties to the run-time business objects, and those properties can be made visible in the user interface.

You can create custom run-time business object types in the Business Modeler IDE on the **RuntimeBusinessObject** business object.

© 2025 Siemens

Warning:

Although the Business Modeler IDE allows you to create run-time business objects on the children of the **RuntimeBusinessObject** business object, you should not create custom run-time business objects at this level. Only create custom business objects directly on the **RuntimeBusinessObject** business itself.

Note:

You can also create custom run-time business objects using the **TCTYPE_create_objects()** ITK interface method and the **DatamanagementService::createObjects** Teamcenter Services operation method.

1. Click the **Find** button at the top of the **BMIDE** view and search for the **RuntimeBusinessObject** business object.

2. Right-click the **RuntimeBusinessObject** business object and choose **New Runtime Business Object**.

3. Perform the following steps in the **Runtime Business Object** dialog box:

   a. In the **Name** box, type the name you want to assign to the new business object in the database.

      When you name a new data model object, a prefix from the template is automatically affixed to the name to designate the object as belonging to your organization, for example, **A5_**.

   b. In the **Display Name** box, type the name as you want it to appear in the user interface.

   c. Select **Is Abstract?** if instances of the business object will not be created in user interfaces such as the rich client.

   d. In the **Description** box, type a description for the new business object.

   e. Click **Finish**.

      The custom run-time business object is created.

4. To save the changes to the data model, choose **BMIDE→Save Data Model** on the menu bar, or click the **Save Data Model** button on the main toolbar.

5. Deploy your changes to the test server. Choose **BMIDE→Deploy Template** on the menu bar, or select the project and click the **Deploy Template** button on the main toolbar.

6. After creating custom run-time business object types, you can perform the following:

© 2025 Siemens

- Add properties to the custom run-time business objects. In this way, you can create an object that is only persisted in the session to store some information used by a customization or external application.

- Add properties to the **Operation Descriptor** tab to allow the properties to be displayed when creating instantiations of the custom run-time business objects.

- Override some operations on the custom run-time business objects, such as **finalizeCreateInput** and **validateCreateInput**.

Create an application interface business object

Children of the **AppInterface** business object are used to transfer data between Teamcenter and an external application. A new entry for each application interface type is added to the **Tools→Send Data To** dialog box in the rich client applications that support application interfaces.

The Application Interface Viewer allows you to control data exchanges between Teamcenter and an external application.

1. Click the **Find** button at the top of the **BMIDE** view. Type **AppInterface** in the search box and click **OK**.

   The **AppInterface** business object is selected in the **Business Objects** folder.

2. In the **Business Objects** folder, right-click the **AppInterface** business object or one of its children and choose **New Business Object**.

   The New Application Interface wizard runs.

3. In the **Application Interface** dialog box, enter the following information:

   a. The **Project** box defaults to the already-selected project.

   b. In the **Name** box, type the name you want to assign to the new business object in the database.

      When you name a new data model object, a prefix from the template is automatically affixed to the name to designate the object as belonging to your organization, for example, **A4_**.

   c. In the **Display Name** box, type the name as you want it to appear in the user interface.

   d. The **Parent** box defaults to the already-selected parent business object.

   e. In the **Description** box, type a description of the new business object.

   f. Click the arrow in the **Available Tool** box to select the application interface to share data with.

© 2025 Siemens

g. Click the **Browse** button to the right of the **Transfer Mode for Import XML** box. The **Teamcenter Repository Connection wizard** prompts you to log on to a server to look up its available transfer modes. Select the transfer mode to use when importing objects into your database for this application interface.

h. Click the **Browse** button to the right of the **Transfer Mode for Export XML** box to select the transfer mode to use when exporting objects from your database for this application interface.

i. Select the **Is Logical Incremental Change Required** check box if you need to create an incremental change object for this application interface. This object documents the differences between the two states of a product structure.

j. Select **Create Primary Business Object** to make a new class that stores the data for the new business object. Clear this option to set the storage class as the same used by the parent business object (for example, the **AppInterface** class).

k. Select **Is Abstract?** if instances of the business object will not be created in user interfaces such as the rich client. For example, you may want to select this if the business object is intended as a folder for child business objects.

l. Click **Next**.

4. In the **Application Interface** dialog box, enter the following information:

a. Click the **Add** button to the right of the **View List** pane to choose the view objects that can be configured by the external application.

For instructions about how to add your own view objects, see *Add a view type*.

b. Click the **Add** button to the right of the **Object List** pane to choose the business objects that can be sent to the external application.

c. Click **Finish**.

The new business object appears in the **Business Objects** folder. A **c** on the business object symbol indicates that it is a custom business object.

5. To save the changes to the data model, choose **BMIDE→Save Data Model**, or click the **Save Data Model** button on the main toolbar.

6. Deploy your changes to a test server. Choose **BMIDE→Deploy Template** on the menu bar, or select the project and click the **Deploy Template** button on the main toolbar.

7. After deployment, test your new business object in the Teamcenter rich client.

© 2025 Siemens

In My Teamcenter, select a dataset and choose **Tools→Export→Objects**. In the **Export** dialog box, click the **AppInterface** button, and then click the **New Application Interface** button  to the right of the **Application Interface** box. Select your new application interface business object on the left of the **New Application Interface** dialog box and click **OK** to create an instance of it.

Create an intermediate data capture business object

An **IntermediateDataCapture** business object holds data from other sources. An intermediate data capture (IDC) can hold structure contexts or collaboration contexts, BOM lines, or occurrence groups. When you create an IDC business object, specify a transfer mode for importing and exporting objects into the database. **IntermediateDataCapture** is a child of the **AppInterface** business object.

1. Click the **Find** button at the top of the **BMIDE** view. Type **IntermediateDataCapture** in the search box and click **OK**.

   The **IntermediateDataCapture** business object is selected in the **Business Objects** folder.

2. In the **Business Objects** folder, right-click the **IntermediateDataCapture** business object and choose **New Business Object**.

   The New Intermediate Data Capture wizard runs.

3. In the **Intermediate Data Capture** dialog box, enter the following information:

   a. The **Project** box defaults to the already-selected project.

   b. In the **Name** box, type the name you want to assign to the new business object in the database.

      When you name a new data model object, a prefix from the template is automatically affixed to the name to designate the object as belonging to your organization, for example, **A4_**.

   c. In the **Display Name** box, type the name as you want it to appear in the user interface.

   d. In the **Parent** box, **IntermediateDataCapture** is already selected as the parent business object.

   e. In the **Description** box, type a description of the new business object.

   f. Click the **Browse** button to the right of the **Transfer Mode for XML** box. You are prompted to log on to a test server to look up its available transfer modes. Select the transfer mode to use for capturing data and exporting it to your database.

   g. Select **Is Abstract?** if instances of the business object will not be created in user interfaces such as the rich client. For example, you may want to select this if the business object is intended as a folder for child business objects.

© 2025 Siemens

h. Click **Finish**.

The new **IntermediateDataCapture** object appears in the tree of business objects.

4. To save the changes to the data model, choose **BMIDE→Save Data Model**, or click the **Save Data Model** button on the main toolbar.

5. Deploy your changes to a test server. Choose **BMIDE→Deploy Template** on the menu bar, or select the project and click the **Deploy Template** button on the main toolbar.

6. After deployment, test your new business object in the Teamcenter rich client by creating an instance of it.

   For example, in the My Teamcenter application, create a collaboration context by choosing **File→New→Collaboration Context**. Then select the collaboration context object you created and choose **Tools→Intermediate Data Capture**. Select your new intermediate data capture business object on the left of the **New Intermediate Data Capture** dialog box, select the **Open on create** check box and click **OK**.

   The new collaboration context is opened in the Multi-Structure Manager page with the IDC transfer mode applied.

Create a structure context business object

A **StructureContext** business object is a BOM or assembly structure contained in a collaboration context. The structure context can contain occurrence groups, items, and item revisions.

1. Click the **Find** button at the top of the **BMIDE** view. Type **StructureContext** in the search box and click **OK**.

   The **StructureContext** business object is selected in the **Business Objects** folder.

2. In the **Business Objects** folder, right-click the **StructureContext** business object or one of its children and choose **New Business Object**.

   The New Structure Context wizard runs.

3. In the **Structure Context** dialog box, enter the following information:

   a. The **Project** box defaults to the already-selected project.

   b. In the **Name** box, type the name you want to assign to the new business object in the database.

   When you name a new data model object, a prefix from the template is automatically affixed to the name to designate the object as belonging to your organization, for example, **A4_**.

© 2025 Siemens

c. In the **Display Name** box, type the name as you want it to appear in the user interface.

d. The **Parent** box defaults to the already-selected parent business object.

e. In the **Description** box, type a description of the new business object.

f. Select the **Composition?** check box if the structure context is also a composition. A composition is a special kind of structure context that contains components added from other structure contexts.

g. Select **Create Primary Business Object** to make a new class that stores the data for the new business object. Clear this option to set the storage class as the same used by the parent business object.

h. Select **Is Abstract?** if instances of the business object will not be created in user interfaces such as the rich client. For example, you may want to select this if the business object is intended as a folder for child business objects.

i. Click **Finish**.

The new business object appears in the tree of business objects.

4. To save the changes to the data model, choose **BMIDE→Save Data Model**, or click the **Save Data Model** button on the main toolbar.

5. Deploy your changes to a test server. Choose **BMIDE→Deploy Template** on the menu bar, or select the project and click the **Deploy Template** button on the main toolbar.

6. After deployment, test your new business object in the Teamcenter rich client by creating an instance of it.

   For example, in the My Teamcenter application, choose **File→New→Structure Context**. Choose the new business object from the list of available ones. Create an instance of the object and click **OK**.

Create a general design element business object

A **GeneralDesignElement** (GDE) business object, also known as an *item element*, is used to model any non-physical feature, such as electrical connections, weld points, mating relationships, and so on. You can manage, release, and instantiate item elements in a product structure. The **GeneralDesignElement** business object is a child of the **Form** business object.

1. At the top of the **BMIDE** view, click **Find**. In the search box, type **GeneralDesignElement** and click **OK**.

   In the **BMIDE** view **Business Objects** folder, the **GeneralDesignElement** business object is selected.

© 2025 Siemens

2. Right-click the **GeneralDesignElement** business object and choose **New Business Object**.

3. In the **New Item Element** dialog box, enter the following information:

For this parameter	Do this
Name	Type the name that you want to assign to the new business object in the database. The name must begin with the template prefix.
Display Name	Type the name as you want it to appear in the user interface.
Parent	Accept the default parent business object.
Description	Type a description of the new business object.
Maximum Instances per Interface	Enter the maximum number of occurrences allowed for this GDE. You can enter any positive integer, **0**, or **-1** (infinite).
Create Primary Business Object	Clear this option. By design, only secondary **GeneralDesignElement** objects are listed in the rich client **File→New→Item Element** menu. If for some reason you prefer to create a primary GDE object, for example to apply some custom code, it would be listed in the rich client **File→New→Other** menu.
Is Abstract?	Select if instances of the business object will not be created in user interfaces such as the rich client. For example, you may want to select this if the business object is intended as a folder for child business objects.
Class Name	Type the name of the class to store the attributes for the new business object.
Parent Name	Accept the default class **POM_object** unless you know that you need to inherit properties from a specific class.
Attributes	To add attributes to the business object, to the right of the table, click **Add**.
View List	To choose the **views** to associate with this GDE, to the right of the list, click **Add**.

4. Click **Finish**.

The new **GeneralDesignElement** object appears in the tree of business objects.

© 2025 Siemens

Create a general design element link business object

A **GeneralDesignElementLink** business object is a type of non-revisable electromechanical connection. It is also known as a **GDELink** object, and it is used to model connectivity in an electrical device within wire harness modeling.

1. Click the **Find** button at the top of the **BMIDE** view. Type **GeneralDesignElementLink** in the search box and click **OK**.

   The **GeneralDesignElementLink** business object is selected in the **Business Objects** folder.

2. In the **Business Objects** folder, right-click the **GeneralDesignElementLink** business object or one of its children and choose **New Business Object**.

3. In the **Create** dialog box, enter the following information:

   a. The **Project** box defaults to the already-selected project.

   b. In the **Name** box, type the name you want to assign to the new business object in the database.

      When you name a new data model object, a prefix from the template is automatically affixed to the name to designate the object as belonging to your organization, for example, **A4_**.

   c. In the **Display Name** box, type the name as you want it to appear in the user interface.

   d. The **Parent** box defaults to the already-selected parent business object.

   e. In the **Description** box, type a description for the new business object.

   f. Enter the following information in the **Storage Class** pane:

      A. Click **Use new class** to create a new storage class to store the properties and attributes.

         To choose an existing class as the form storage class, click the **Use existing class** button and click **Browse**.

      B. In the **Class Name** box, type the name you want for the form storage class.

      C. If you previously chose the **Use new class** button, in the **Parent Name** box, select the class you want to be the parent of new form storage class.

   g. If you want to add a property to the form storage class, click the **Add** button on the right side of the **Properties** dialog box.

© 2025 Siemens

The New Property wizard runs. After the attribute is added, you can change values as desired by clicking cells in the attribute table.

h. Click **Finish**.

The new business object appears in the business objects tree. A **c** on the business object symbol indicates that it is a custom business object.

4. To save the changes to the data model, choose **BMIDE→Save Data Model**, or click the **Save Data Model** button on the main toolbar.

5. Deploy your changes to a test server. Choose **BMIDE→Deploy Template** on the menu bar, or select the project and click the **Deploy Template** button on the main toolbar.

6. After deployment, test your new business object in the Teamcenter rich client by creating an instance of it.

   For example, in the My Teamcenter application choose **File→New→Connection→Non Revisable**. Click the **More...** button and choose the new business object from the list of available ones. Create the instance and click **OK**.

Business object icons

Add or change a business object icon

For children of the WorkspaceObject business object, you can change the default rich client icon for an existing business object, or add an icon to a newly created business object. You can also decorate (overlay) the icon with images when specified conditions exist.

Example:

An icon of a pencil is added for a custom business object type.

An x is overlaid on the icon when an instance of the business object is checked out.

A check mark is overlaid on the icon when an instance of the business object is checked in.

The following procedures show how to:

- Add an icon to a custom business object type.
- **Vary the primary icon, or overlay decorations onto the primary icon, based on a property value**.

© 2025 Siemens

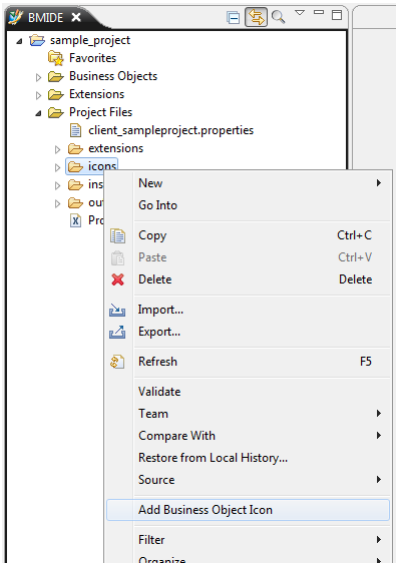
Add an icon to a custom business object type

1. Use a graphics editing tool to create a primary icon to represent the business object. Make the icon 16x16 pixels in size, and give the image a transparent background.

   If you want to add decorations to designate the business object's state, also create these as 16x16 pixel-sized icons with transparent backgrounds. These decoration images are overlaid onto the business object icons.

2. Add the icons to your Business Modeler IDE project.

   In the **Project Files** folder, right-click the **icons** folder and choose **Add Business Object Icon**.



   For this example, the following icons are added to the **icons** folder:

Image	File name	Description
	**A5_pencil_16.png**	Primary icon that represents the business object
	A5_xmark_16.png	Overlay icon decoration that shows that the business object is checked out
	**A5_checkmark_16.png**	Overlay icon decoration that shows that the business object is checked in

3. Specify the primary icon for instances of a business object.

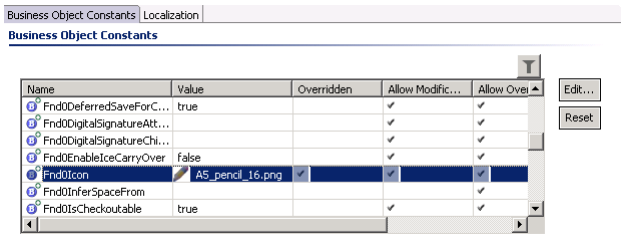
   a. Open the business object for which you want to add or change the primary icon.

   b. On the **Main** tab in the **Business Object Constants** table, select the **Fnd0Icon** business object constant and click **Edit**.

© 2025 Siemens

c. In the **Modify Business Object Constant** dialog box, to the right of the **Icon** box, click **Browse** and select the primary icon you want to represent the business object.

The icon appears in the **Fnd0Icon** business object constant row of the table.

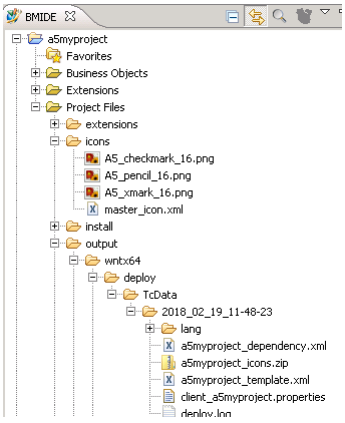


d. To save the changes to the data model, choose **BMIDE**>**Save Data Model**, or on the main toolbar click **Save Data Model** .

4. Deploy the icons to a server in one of two ways:

- **Deploy** to a test server using the Deploy Template wizard.
- **Package the template** using the **Generate Software Package** wizard and **deploy to a production server** using Teamcenter Environment Manager (TEM) or Deployment Center.

When you deploy or package the template, the icons are placed in a *project*_**icons.zip** file.



When icons are deployed to a server, the zipped template icon files are placed in the *TC_DATA***model\icons** directory, and the manage_icon_files utility is used internally to upload the icons into the **Fnd0IconResource** dataset instances in the database. To search for these datasets using the rich client, search for the **Icon Resource Dataset** type.

At this point, the icon displays on all instances of the business object in the rich client.

© 2025 Siemens

Vary the primary icon, or overlay decorations onto the primary icon, based on a property value

If you want to vary the primary business object icon or overlay decorations onto the primary business object icon based on some property value, then you can associate a property renderer with the **object_string** property of a business object.

In the following example, you overlay an image on the primary icon based on the **checked_out** property value.

> Note:
>
> Only String type properties can be set as the basis for rendering (**checked_out** is the basis for rendering in this example definition). Reference type properties or customer-defined Compound type properties are not supported for defining icon rendering. For example, properties like "owning_group", "owning_user", "owning_project", and "release_status" are reference type, so they cannot be used for icon rendering.

1. If a suitable property renderer does not already exist, then create a property renderer.

   a. Start the New Property Renderer wizard in one of these ways:

   - On the menu bar, choose **BMIDE→New Model Element**, type **Property Renderer** in the **Wizards** box, and click **Next**.
   - Open the **Extensions** folder, right-click the **Property Renderers** folder and choose **New Property Renderer**.

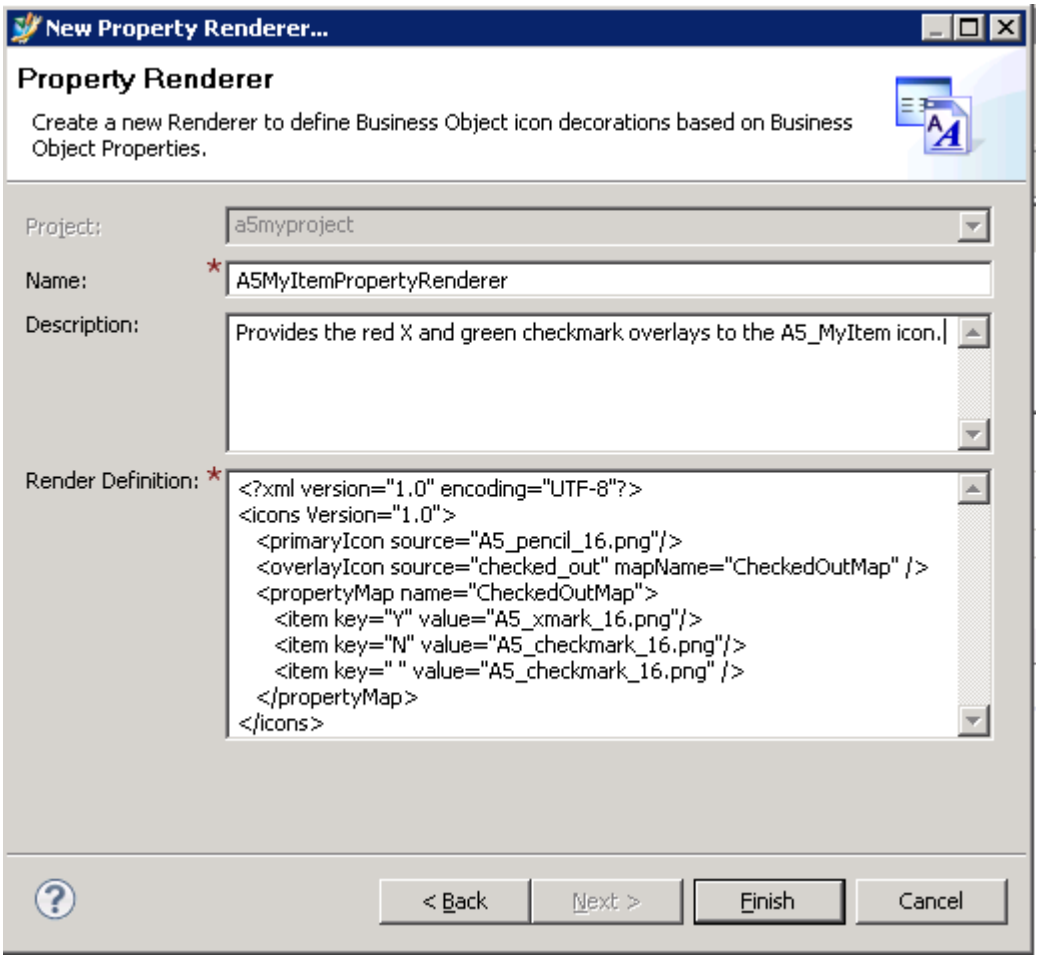
   b. In the **Name** box, type a name for the renderer, and in the **Description** box, type the purpose of the renderer.

   c. In the **Render Definition** box, type a well-formed XML **rendering definition**.

   For this example, the following definition places overlay icons on the primary icon depending on whether the business object is checked out.

```
<?xml version="1.0" encoding="UTF-8"?>
<icons Version="1.0">
   <primaryIcon source="A5_pencil_16.png"/>
   <overlayIcon source="checked_out" mapName="CheckedOutMap" />
   <propertyMap name="CheckedOutMap">
      <item key="Y" value="A5_xmark_16.png"/>
      <item key="N" value="A5_checkmark_16.png"/>
      <item key=" " value="A5_checkmark_16.png" />
   </propertyMap>
</icons>
```


© 2025 Siemens



d. Click **Finish**.

2. Attach the property renderer to the **object_string** property of the business object you want to affect.

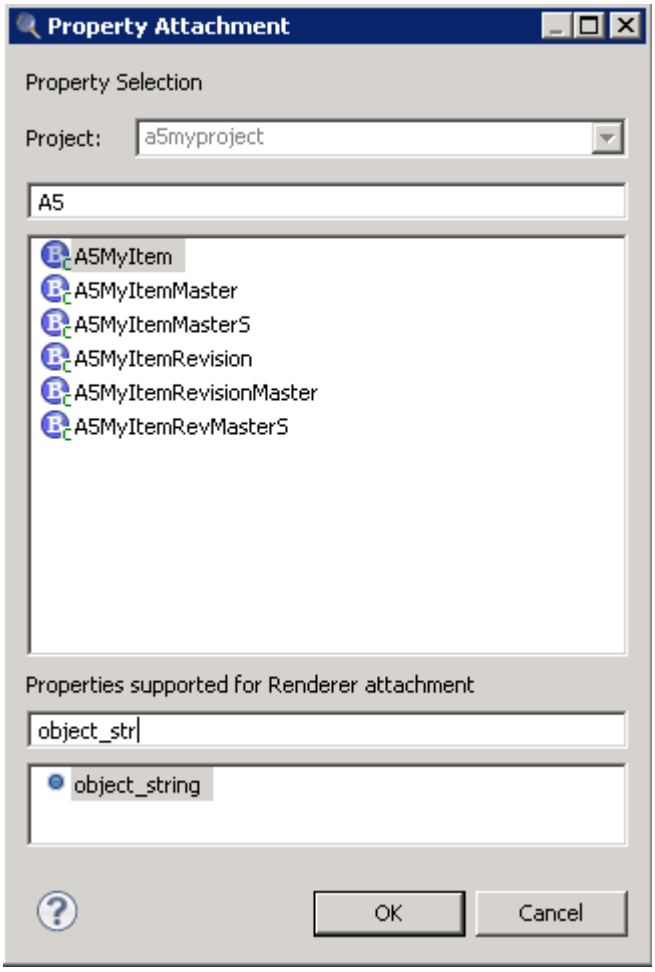
   a. Open the property renderer that you want to attach to a business object.

   b. To the right of the **Property Renderer Attaches** list, click **Attach**.

   c. In the **Property Renderer Attachment** dialog box, next to the **Property** box, click **Browse** and select the business object that you want to attach the renderer to.

   d. In the **Properties supported for Renderer Attachment** list, select the **object_string** property.

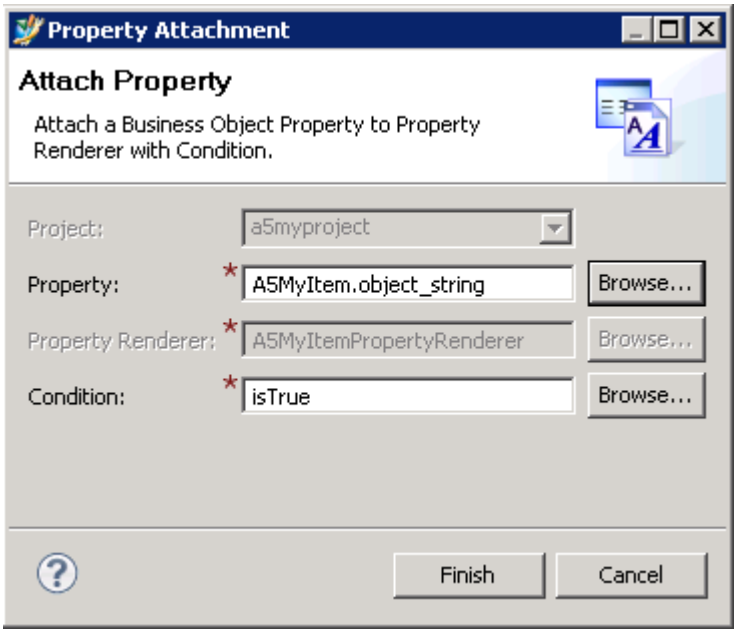
© 2025 Siemens



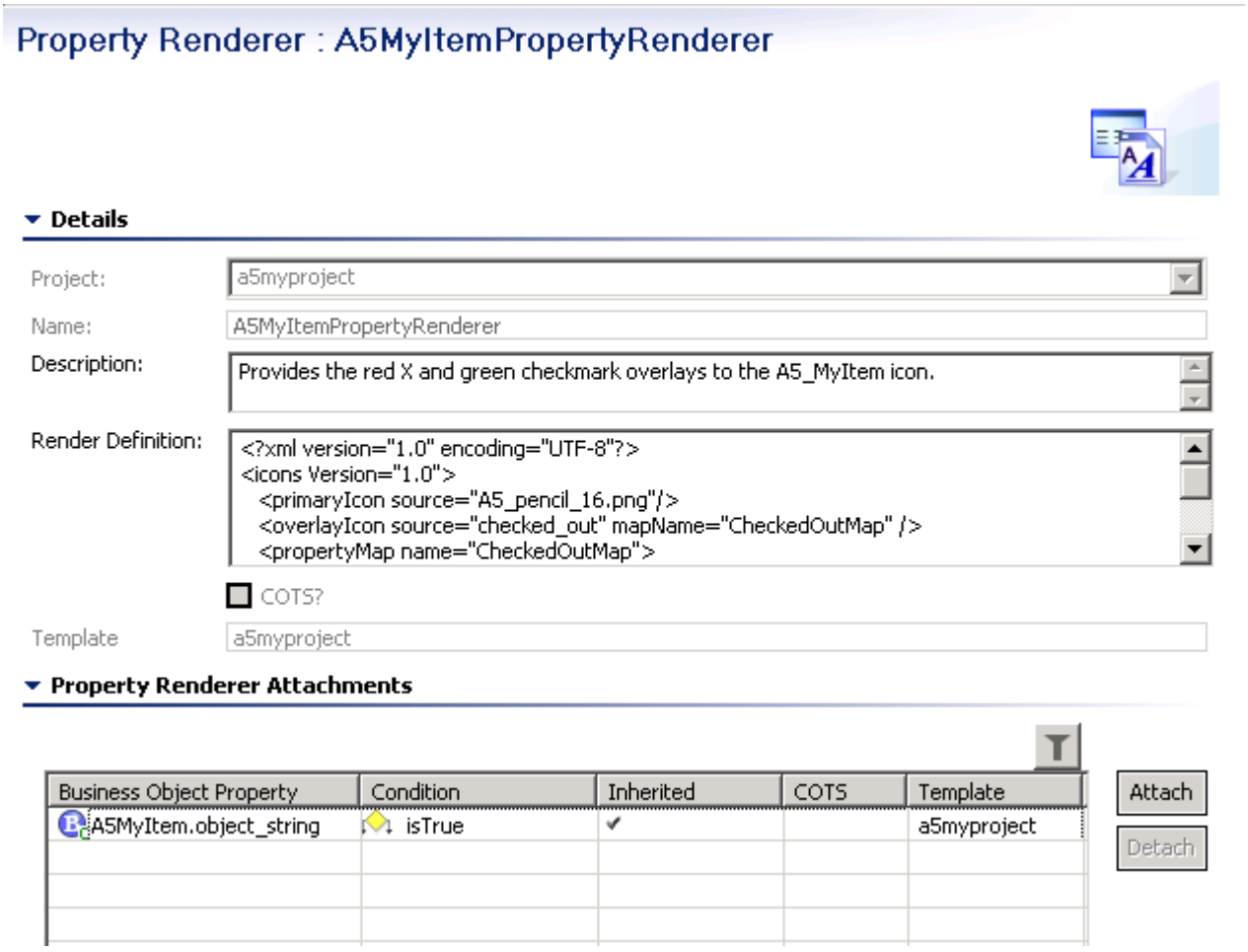
e. Click **OK** to close the **Property Attachment** dialog box.

f. For this example, to add an overlay icon, choose the **isTrue** condition to identify that the renderer applies whenever an object_string property exists for the object.

© 2025 Siemens



g. Click **Finish**.



© 2025 Siemens

3. To save the changes to the data model, choose **BMIDE→Save Data Model**, or click the **Save Data Model** button on the main toolbar.

4. Deploy the new data model to a server. You can use either of the following ways:

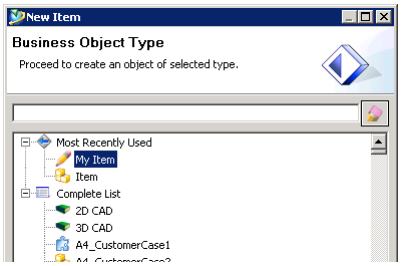
   - **Deploy** to a test server using the Deploy Template wizard.
   - **Package the template** using the **Generate Software Package** wizard and **deploy to a production server** using Teamcenter Environment Manager (TEM) or Deployment Center.

View the icons in the rich client

1. Create an instance of the business object in the rich client.

   Example:

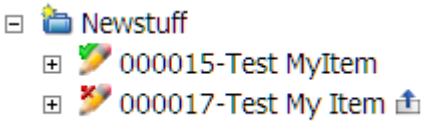
   In the My Teamcenter perspective, choose **File>New>Item** and choose the custom business object in the example. Note the new icon on the business object.



2. If any decorations are defined for the icon, then change the state of the business object instance to view them.

   Example:

   When the custom business object is checked in, a check mark appears on the icon. When the custom business object is checked out, an X appears on the icon.



Render definition elements and examples

A *render definition* defines the primary and overlay icons to display for an instance of a business object. The definition applies to a business object if you associate the render definition with an appropriate

© 2025 Siemens

business object property, typically the **object_string** property, when you **Add or change a business object icon**.

Render definitions must contain well-formed XML using the following elements:

Element	Description
icons	The root icon render definition element. It contains **propertyMap**, **primaryIcon**, and **overlayIcon** elements. It can also contain a **version** attribute to support versioning of the XML.
primaryIcon	Defines the base icon. It contains a string **source** attribute that identifies the icon dataset name, and can contain a **visibleWhen** clause as well as **overlayIcon** elements that are limited to this specific primary icon. The **source** element and **visibleWhen** clause are described in the following table. If **visibleWhen** expressions are used within the **primaryIcon** elements, then multiple **primaryIcon** elements can be defined within the **icons** element. However, only one **primaryIcon** element **visibleWhen** clause should evaluate to true in a given case. If multiple primary icon **visibleWhen** expressions evaluate to true, then an error occurs.
propertyMap	Maps values of the property to which the property renderer is attached (the **key** attribute) to icon dataset names (the **value** attribute). The property map **key** values must be exhaustive of allowed values for the property. If the property value is allowed to be empty, then an empty property value must be included in the propertyMap. An empty value for the icon dataset name results in no icon being overlaid for the given property value (**key**). Note: Only String type property values are supported. Other types, for example Array and Boolean, are not supported.
overlayIcon	Defines an icon that is overlaid on top of a primary icon. As with a **primaryIcon**, the **overlayIcon** contains a string **source** attribute that identifies the icon dataset name. Typically, the overlay icon is small with a transparent background so as not to completely obscure the primary icon. The overlay icon can be placed as a child element of **primaryIcon** so it is limited to the specific primary icon, or it can be a sibling element to the **primaryIcon** element(s), thus applying to whichever **primaryIcon** element evaluates to true.

The **primaryIcon** and **overlayIcon** elements can contain the following attribute and clause:

Attribute or clause	Description
source	Specifies the dataset name containing the icon. This can be done in any of three ways:

© 2025 Siemens

<table>
<tr><th>Attribute or clause</th><th>Description</th></tr>
<tr><td></td><td>
<table>
<tr><th>Method for specifying the dataset name</th><th>Example</th></tr>
<tr><td>Directly name the dataset.</td><td><primaryIcon source="A5_icon.png"/>
<overlayIcon source="A5_icon.png"/></td></tr>
<tr><td>Refer to a propertyMap element.</td><td>For a primaryIcon element, provide only the propertyMap name. The value for the property to which the renderer is attached is used in mapping to an icon dataset.
<primaryIcon source="a5_MyStatusMap"/>
For an overlayIcon element, provide an object constant from which to get the value to map to an icon dataset.
<overlayIcon source="fnd0state" mapName="a5_MyStatusMap"/></td></tr>
<tr><td>(For a primaryIcon element) Point to the Fnd0Icon business object constant, which specifies the object icon.</td><td><primaryIcon source="Fnd0Icon"/>
This method works only when the render definition is attached to an object for which Fnd0Icon is defined.</td></tr>
</table>
</td></tr>
<tr><td>visibleWhen</td><td>Provides an expression that evaluates to either true or false. If a visibleWhen clause is used, then the containing element (either primaryIcon or overlayIcon) is used only if its visibleWhen clause evaluates to true.
The expression that the visibleWhen clause contains is passed to the Eclipse expression engine for evaluation. All cached properties for the given business object, as well as metadata such as type constants, are used as context for evaluating the expression.
Tip:
For more information about the Eclipse command core expressions used to evaluate the visibleWhen clause, see the Eclipse wiki.
http://wiki.eclipse.org/
You can use the currentTcComponent variable to access the TCComponent method, as shown in this example from the Fnd0sm_scheduletask_icon property renderer:
<pre><visibleWhen>
  <with variable="currentTcComponent">
    <test property="com.teamcenter.rac.kernel.TCComponent.property"
     args="task_type" value="1" />
  </with>
</visibleWhen></pre></td></tr>
</table>

© 2025 Siemens

Attribute or clause	Description

	You can use a Boolean property to evaluate a renderer to resolve to true or false. In the following example, **is_configuration_item** is a Boolean property:
	
	`<icons Version="1.0">`
	`  <primaryIcon source="Fnd0sm_schedule_16.png" >`
	`  <overlayIcon source="greenball.png">`
	`  <visibleWhen>`
	`  <with variable="currentTcComponent">`
	`  <test`
	`  property="com.teamcenter.rac.kernel.TCComponent.property"`
	`  args="is_configuration_item" value="True" />`
	`  </with>`
	`  </visibleWhen>`
	`  </overlayIcon>`
	`  </primaryIcon>`
	`</icons>`

Examples of render definitions

Place different overlay icons on the primary icon depending on the value of the `checked_out` property:

```
<?xml version="1.0" encoding="UTF-8"?>
<icons Version="1.0">
  <propertyMap name="checkoutMap">
    <item key="Y" value="A5_small_green_dot"/>
    <item key="N" value="A5_small_red_dot"/>
    <item key=" " value="A5_small_blue_dot"/>
   </propertyMap>

  <primaryIcon source="A5_Requirement.png">
    <overlayIcon source="checked_out" mapName="checkoutMap"/>
  </primaryIcon>
</icons>
```

Place different overlay icons on the primary icon only when the `checked_out` property has a certain value:

```
<?xml version="1.0" encoding="UTF-8"?>
<icons Version="1.0">
  <propertyMap name="CheckedOutMap">
    <item key="Y" value="A5_small_green_dot.png"/>
    <item key="N" value="A5_small_red_dot.png"/>
    <item key=" " value="A5_small_blue_dot.png" />
  </propertyMap>

  <primaryIcon source="A5_draw.png"/>
  <overlayIcon source="checked_out" mapName="CheckedOutMap" >
    <visibleWhen>
      <with variable="current_desc">
        <equals value="overlay" />
      </with>
    </visibleWhen>
```


© 2025 Siemens

```
  </overlayIcon>
</icons>
```

Place an overlay icon on the primary icon only when a `checked_out` property condition is met:

```
<?xml version="1.0" encoding="UTF-8"?>
<icons Version="1.0">
  <propertyMap name="checkoutMap">
    <item key=" " value=""/>
    <item key="Y" value="small_red_dot"/>
    <item key="N" value=""/>
  </propertyMap>

  <primaryIcon source="A5_Email.png"/>
  <overlayIcon source="checked_out" mapName="checkoutMap">
    <visibleWhen>
      <with variable="is_modifiable">
        <not>
          <equals value="Y"/>
        </not>
      </with>
    </visibleWhen>
  </overlayIcon>
</icons>
```

Place different overlay icons on the primary icon when multiple property conditions are met:

```
<?xml version="1.0" encoding="UTF-8"?>
<icons Version="1.0">
  <propertyMap name="IsCheckoutableMap">
    <item key="Y" value="A5_small_blue_dot.png" />
  </propertyMap>

  <propertyMap name="checkoutMap">
    <item key=" " value="A5_small_red_dot.png"/>
    <item key="Y" value="A5_small_green_dot.png"/>
    <item key="N" value="A5_small_red_dot.png"/>
  </propertyMap>

  <primaryIcon source="A5_CompanyAlt.png">
    <visibleWhen>
      <with variable="item_revision">
        <not>
          <equals value="1" />
        </not>
      </with>
    </visibleWhen>
  </primaryIcon>
  <primaryIcon source="A5_Company.png" />
  <overlayIcon source="checked_out" mapName="checkoutMap">
    <visibleWhen>
      <with variable="checked_out">
        <not>
          <equals value="N"/>
        </not>
      </with>
    </visibleWhen>
  </overlayIcon>
```


© 2025 Siemens

```
  <overlayIcon source="fnd0IsCheckoutable" mapName="IsCheckoutableMap">
    <visibleWhen>
      <with variable="IsCheckoutable" >
        <equals value="Y" />
      </with>
    </visibleWhen>
  </overlayIcon>
</icons>
```

Place different overlay icons on the primary icon depending on the IP classification of the object:

```
<?xml version="1.0" encoding="UTF-8"?>
<icons Version="1.0">
  <propertyMap name="ip_classificationMap">
    <item key="Public" value="EMR8_green_overlay_16.png"/>
    <item key="Internal Use" value="EMR8_blue_overlay_16.png"/>
    <item key="Confidential" value="EMR8_yellow_overlay_16.png"/>
    <item key="Development" value="EMR8_purple_overlay_16.png"/>
    <item key="Restricted" value="EMR8_red_overlay_16.png"/>
  </propertyMap>

  <primaryIcon source="Fnd0Icon"/>
  <overlayIcon source="ip_classification" mapName="ip_classificationMap" />
</icons>
```

> Note:
>
> This method works only when the render definition is attached to an object for which Fnd0Icon is defined.

Place an overlay icon on the primary icon when the `currentTcComponent` value returned from the server for the `checked_out` property is `Y`:

```
<?xml version="1.0" encoding="UTF-8"?>
<icons Version="1.0">
  <propertyMap name="YourMap">
    <item key="N" value=""/>
    <item key="Y" value="small_red_dot.png "/>
    <item key=" " value=""/>
  </propertyMap>

  <primaryIcon source="small_green_dot.png"/>
  <overlayIcon source="checked_out" mapName="YourMap">
    <visibleWhen>
      <with variable="currentTcComponent">
        <test property="com.teamcenter.rac.kernel.TCComponent.property"
         args="checked_out" value="Y"/>
      </with>
    </visibleWhen>
  </overlayIcon>
</icons>
```


© 2025 Siemens

Supporting create, save as, and revise operations in clients

Making properties available for operations in clients

In order to perform an operation that involves a business object, a Teamcenter client needs values for certain properties of the business object. So that the properties are available to the client and can be exposed as needed for entry of a value or to support a decision, in the Business Modeler IDE the business object properties needed for the operation must be included in the operation list within the **Operation Descriptor tab**.



This tab	lists properties used for this operation
CreateInput	Creating an instance of an object.
SaveAsInput	Saving a copy of an object instance with a new name.
ReviseInput	Revising an object instance. Relation objects do not have a revision status, so the **ReviseInput** operation input list is not needed and does not appear for a relation business object.

Because operations on business objects can involve associated business objects, the operation lists include properties of both the object and its associated objects. For example, creating an instance of an **Item** business object can also create associated instances of master form, revision, and revision master form business objects. In the rich client, associated objects have additional pages in the object operation dialog box.

Set property Required and Visible constants for client operations

Once you have added business object properties to an operation list in the BMIDE, you set the **Required** and **Visible** property constants to control availability of the property for inclusion in a Teamcenter client user interface during the operation.

Property Constant	Constant value	Implications
Required	true	A value for the property is required. A red asterisk appears for the property in the client operation dialog box. (rich client)

© 2025 Siemens

Property Constant	Constant value	Implications
		The **Visible** constant is automatically set to true.
	false	A value for the property is optional.
Visible	true	The property is available for display in the Teamcenter client. Note: To make the property visible in the operation dialog box, the style sheet for the object operation must include the property.
	false	The property is hidden in the Teamcenter client.

1. In the **BMIDE** workspace, right-click the business object for which you want to make the change, choose **Open**, and in the resulting view click the **Operation Descriptor** tab.

2. Click the tab for the operation input list that you want to change.



3. In the operation input list, select the property for which you want to change the visible or required setting, and then click **Edit**.

4. In the **Modify** dialog box, mark the check box for the **Required** and **Visible** constants that you want to set to **true**.

5. Click **Finish**.

Note:

- Active Workspace supports settings for **CreateInput**, **SaveAsInput**, and **ReviseInput** operations.

  The rich client supports settings for **CreateInput** and **SaveAsInput** operations; the rich client does not support settings specified for the **ReviseInput** operation.

- Use the Fnd0EnableUsageOfDialog business object constant to enable use of generic dialog boxes in the rich client.

- Use of the **CreateInput** operation is not supported for the **BOMLine** business object and its children.

© 2025 Siemens

- For relation business objects, if any input property is made visible, then the **primary_object**, **relation_type**, and **secondary_object** properties must also be visible.

Add business object properties to an operation input list

1. In the **BMIDE** workspace, right-click the business object for which you want to make the change, choose **Open**, and in the resulting view click the **Operation Descriptor** tab.

2. Click the tab for the operation input list that you want to change.



3. To add a property to the list, click the **Add** button to the right of the table.

4. In the **OperationInputProperty** dialog box, select **Add Property from Business Object** and click **Next**.

5. To the right of **Source Property**, click **Browse** and select an existing property.

   Note:

   Configuring compound properties for the **CreateInput** operation on the **Operation Descriptor** tab is not supported by server processing. Therefore, compound properties are excluded from the browse list in the New OperationInput Property wizard.

6. Select **Required**, **Visible** property constants as appropriate.

7. If the **CopyFromOriginal** option is available, then selecting the option causes the operation to initially copy the value of the specified **Source Property** from the original business object. Select or clear the option as appropriate.

   Propagation rules can cause propagation of security-related property values even if **CopyFromOriginal** is not selected.

8. Select a **Usage** type as appropriate.

© 2025 Siemens

In this case	Select this Usage type
The property is not a relation or reference property that references a secondary object.	**None**
The property is associated with a secondary object that is to be created from the value in the **CompoundObjectType** box.	**Type** The **CompoundObjectType** box is displayed. This is the type of secondary object that is to be created. Click **Browse** to select the secondary business object.
The property is associated with a secondary object that is to be created from the value entered in the **CompoundObjectConstant** box.	**Constant** The **CompoundObjectConstant** box is displayed. This is the name of the **business object constant** associated with the primary object that is created. The value of this constant indicates the type of secondary object to be created. Click **Browse** to select the business object constant.

9. In the **Description** box, enter a description of the property.

10. Click **Finish**.

Add runtime properties to an operation input list

You can add a run-time property that is associated with an operation on a source business object. Do this in cases where information required for operation must be specified, but there is no associated source property for it.

Example:

- Some information required for creation needs to be specified in the create dialog box, but there is no associated property for it on the primary object.
- A secondary object needs to be created, but there is no corresponding relation or reference property on the primary object.

The following process for creating a run-time property is similar to **creating a run-time property** using the **Add** button on the **Properties** tab.

1. In the **BMIDE** workspace, right-click the business object for which you want to make the change, choose **Open**, and in the resulting view click the **Operation Descriptor** tab.

2. Click the tab for the operation input list that you want to change.

© 2025 Siemens



3. To define and add a runtime property to the list, to the right of the table click **Add**.

4. In the **New OperationInputProperty** dialog box, select **Define and add a new Runtime Property from Business Object** and click **Next**.

5. Enter values for the new run-time property.

For this attribute	Do this
Name	Type a name for the new run-time property.
Display Name	Type the name to display in the client interface.
Attribute Type	Select the storage type for the attribute from the list of attribute types: **Boolean** Allows two choices to the user (**True** or **False**). **Character** A single character, such as **A**, **B**, **Z**. **Date** A calendar date. A form using this format displays a shortcut date selector. For the date attribute type, the earliest date supported is January 2, 1900. **Double** A double-precision floating point decimal number. For Oracle, the limit for the double property value is 1E-130 to 9E125. For SQL Server, the limit is 2.3E-308 to 1.7E308. **Integer** An integer without decimals from 1 to 999999999. **String** A string of characters. Type the byte length of the string attribute in the associated **String Length** box. For the English alphabet, one character requires one byte. Special characters in Western languages such as French and Portuguese, for example, may require two bytes. For such special characters, and for most characters in multibyte

© 2025 Siemens

For this attribute	Do this
	languages such as Chinese, Japanese, and Korean, a field with a string length of 128 can accommodate a maximum of 64 double-byte characters. **TypedReference** Points to a Teamcenter class. Next to the **Reference Business Object** box, click **Browse** and select the class. **UntypedReference** Points to any class of data. **ExternalReference** Points to data outside of Teamcenter.

6. In **Description**, enter a description of the run-time property.

7. Select **Visible** and **Required** property constants as appropriate.

8. In the **Array Keys** section, specify the properties that apply.

   Array?

   Specifies that the attribute is an array of the specified **Attribute Type** data.

   Unlimited

   Indicates that there is no limit on the number of array elements used for the attribute.

   MaxLength

   Specifies the maximum number of array elements allowed for the attribute.

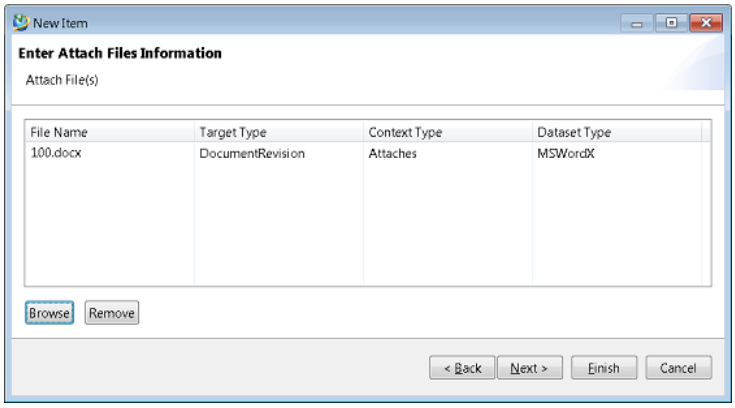
9. Click **Finish**.

Configure file attachment for item creation

When you create items in a Teamcenter client, the new item wizard displays dialog boxes allowing you to define aspects of the new item. You can define the dataset types that can be attached to the new item as well as their default relation.

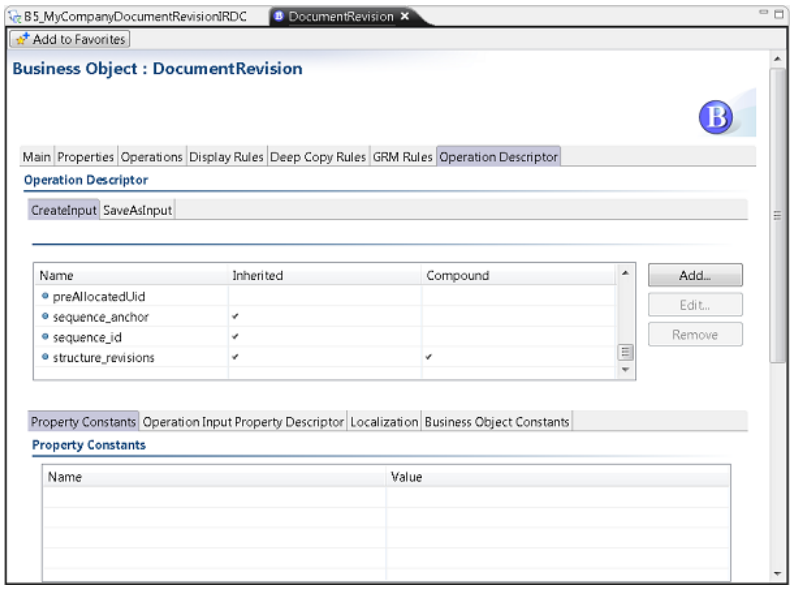
> Tip:
>
> By default, IRDCs allow only one type of dataset to be attached during item creation. If you use IRDCs and want to enable another dataset type, you must follow this procedure to enable it. First add the new dataset type on the **IRDC Dataset Criteria Info** tab of the IRDC. Then perform the following procedure to enable attachment of the dataset type.

© 2025 Siemens



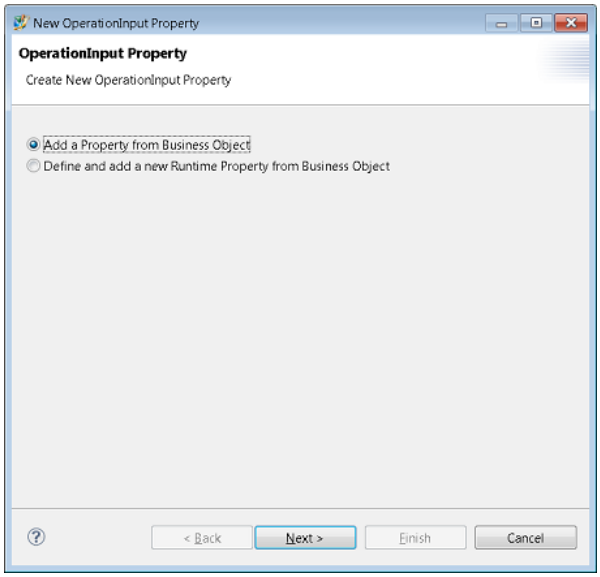
In the following example, by default only the MSWordX file type can be attached to a new document revision. Follow the example to enable MSWord types to be added also.

1. Open the business object to which you want to enable another dataset type (for example, **DocumentRevision**) and click the **OperationDescriptor** tab.



2. Click the **Add** button and select **Add a Property from Business Object**. Click **Next**.

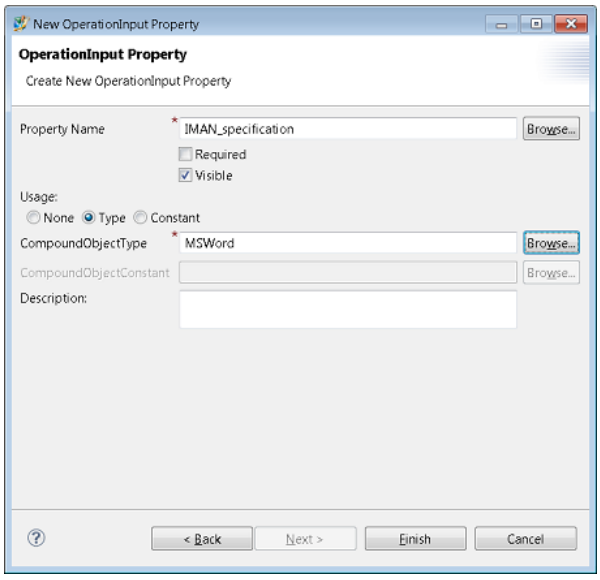
© 2025 Siemens



3. Next to **Property Name**, click **Browse** and select the relationship property that you want to use, for example, **IMAN_specification**.

   To select the type of dataset which you want to attach at creation time, under **Usage**, select **Type** and next to **CompoundObjectType** click **Browse**, for example, **MSWord**.

   Click **Finish**.



Add input properties for a secondary object

You can add properties to a **CreateInput** operation so that a secondary business object's input properties are visible while creating the primary business object. Use the **Dataset** business object to reference a secondary business object, for example, a form.

© 2025 Siemens

1. Start the New Dataset wizard in one of these ways:

   - On the menu bar, choose **BMIDE→New Model Element**, type **Dataset** in the **Wizards** box, and click **Next**.
   - At the top of the **BMIDE** view, click **Find** and search for the **Dataset** business object, and then right-click it and choose **New Business Object**.
   - In the **Business Objects** folder, right-click the **Dataset** business object and choose **New Business Object**.

2. In the **Dataset** dialog box, enter the following information:

For this property	Do this
Name	After the template prefix, type the name you want to assign to the new business object in the database.
Display Name	Type the name as you want it to appear in the user interface.
Description	Type a description of the new business object.
Create Primary Business Object	Choose one of two options: • Select the check box to make a new class that stores the data for the new business object • Clear the check box to set the storage class as the same used by the parent business object (in this case, the **Dataset** class).
Is Abstract?	Select the check box if instances of the business object will not be created in user interfaces such as the rich client. For example, you may want to select this if the business object is intended as a folder for child business objects.
Tools for Edit	Click **Add** to select the software application to launch when the dataset is selected in Teamcenter. If the tool does not exist, then you must create one using the **Tool** option.
Tools for View	Click **Add** to select the software application to be used to view the dataset files. If the desired tool does not exist, then you must create one using the **Tool** option.

3. If you want to add references and parameters to the dataset, then click **Next**. Otherwise, skip to step 5.

© 2025 Siemens

4. If you clicked **Next**, then the **References** table appears. To the right of the **References** table, click **Add** to add file name references to associate with the dataset. The Add Dataset Reference wizard runs.

   Note:

   Datasets may reference files and business objects. You must set **File Type** and **Format** appropriately for a reference file or business object.

   a. In the **Dataset Reference** dialog box, do the following:

For this property	Do this
Reference	Type a unique name for this file reference.
File Type	Enter the file name extension or **OBJECT** when referencing another business object. For a file name extension, enter **txt**, **pdf**, **doc**, and so on.
Format	Choose **BINARY** or **TEXT** for a file reference. Choose **OBJECT** when referencing another business object.

   b. Click **Finish**.

5. In the New Dataset wizard, click **Finish** to complete the dataset creation.

6. Right-click the business object for which you want to make the change, choose **Open**, click the **Operation Descriptor** tab in the resulting view, and in the **Operation** box, select **CreateInput**.

7. To the right of the table, click **Add**.

   The New OperationInput Property wizard runs.

8. In the **OperationInputProperty** dialog box, select **Add Property from Business Object** to add a property. Click **Next** and enter the following information:

For this property	Do this
Property Name	Click **Browse** and select an existing property.
Required	To make the property required for entry in the creation dialog box, select the check box. In the rich client, required properties in the creation dialog box are shown with a red asterisk.

© 2025 Siemens

For this property	Do this
Visible	To make the property display in the creation dialog box, select the check box.
Usage	Select one of the following: **None** — If the property is not a relation or reference property that references a secondary object. **Type** — If this is a property associated with a secondary object that is to be created from the value in the **CompoundObjectType** box. **Constant** — If this is a property associated with a secondary object that is to be created from the value entered in the **CompoundObjectConstant** box.
CompoundObject Type	For **Type** usage. This is the type of secondary object that is to be created. Click **Browse** and select the secondary business object. For example, to save cost data, select **CostDataForm** for a secondary object that is referenced to the dataset.
CompoundObject Constant	For **Constant** usage. This is the name of the **business object constant** associated with the primary object that is created. The value of this constant indicates the type of secondary object to be created. Click **Browse** and select the business object constant.
Description	Type a description of this property.

9. Click **Finish**.

Save, deploy, and test your secondary object definition

1. To save the changes to the data model, choose **BMIDE→Save Data Model**, or on the main toolbar click **Save Data Model** .

2. Deploy your changes to a test server. Choose **BMIDE→Deploy Template** on the menu bar, or select the project and on the main toolbar click **Deploy Template** .

3. After deployment, test your new dataset business object in the Teamcenter rich client to ensure the secondary object properties are visible when you create an instance of the dataset in the Teamcenter rich client.

   a. In My Teamcenter, select the parent object for the business object.

© 2025 Siemens

b. Choose **File→New→Other** to display the **New Business Object** dialog box.

> Note:
>
> The **File→New→Other** must be used to display the **New Business Object** dialog box; do not use **File→New→Dataset**.

c. Select the dataset business object type that your created in the Business Modeler IDE and click **Next**.

d. Click **Next** and enter a name for the new dataset.

e. The wizard displays the **Define business object create information** panel. Provide the necessary information and click **Finish** to create the new business object

Test business object operation input settings

1. To save the changes to the data model, choose **BMIDE→Save Data Model**, or on the main toolbar click **Save Data Model**  .

2. Deploy your changes to a test server. Choose **BMIDE→Deploy Template** on the menu bar, or select the project and on the main toolbar click **Deploy Template**  .

3. After deployment, test to ensure the property is visible or required when you create an instance of the object.

> Example:
>
> If you made a property required for the **CreateInput** operation on the **Item** business object, for example a name, then when a My Teamcenter user chooses **File→New→Item** to create a new **Item** object, a value for the name property is required in the creation dialog box. Required properties are marked with a red asterisk *****. (You can also create business objects in the Teamcenter rich client by choosing **File→New→Other**.)

Test file attachment

If you configured some file attachment at item creation, then perform the following steps to verify that attachment occurs.

1. Choose **File→New→Item**, select **Document** in the **New Item** dialog box, and click **Next**.

2. Assign the ID and give the document a name. Click **Next**.

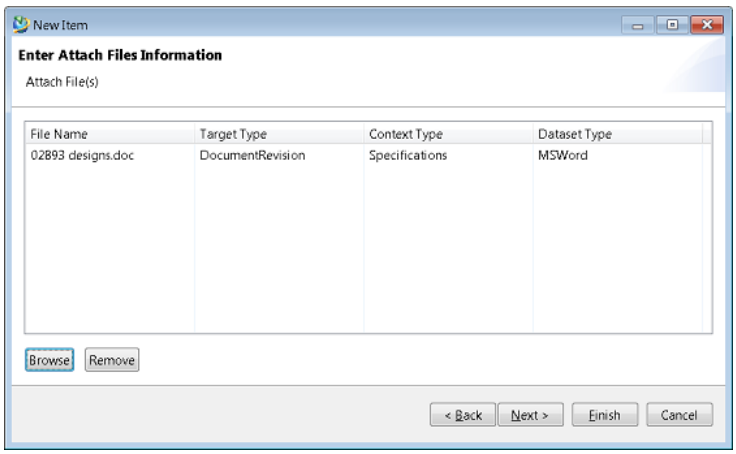
3. Click **Next** until you get to the **DocumentRevision** dialog box.

© 2025 Siemens

> Note:
>
> If this were an IRDC, you may need to select the type of document. For example, if you use the *prefix***DocumentRevisionSampleFSIRDC** sample, click the arrow in the **Document Subject** box and select **Functional Specification**.

4. Click **Next** until you get to the **Enter Attach Files Information** dialog box.

5. Click the **Browse** and select a file of the type that you enabled, for example, a **MSWord** dataset. Notice how it is attached with the specification relationship that you set up.



6. Click **Finish**.

7. Verify that the document revision is created successfully and the dataset is attached to the newly created document revision with the correct relationship.

Business object constants

Introduction to business object constants

A *business object constant* is a defined value that can be attached to a business object in order to supply a default value for object instances.

Reviewing and adjusting the value of an attached business object constant

To see the constants attached and their set values for a specific business object, in the **BMIDE** view expand the **Business Objects** folder as needed, right-click the business object, choose **Open**, and view the **Business Object Constants** table on the **Main** tab.

Attached business object constants are inherited by child business objects, but can be overridden in the business object hierarchy.

© 2025 Siemens

To adjust the value of an attached business object constant this way	Do this
Change the value of the business object constant.	Select the constant in the table, click **Edit**, and change the value in the **Value** box.
Set the constant value back to its default value.	Select the constant in the **Business Objects Constants** table and then click **Reset**.

Changing the value of an attached business object constant reloads the data model. The reload is necessary because all children of a parent business object inherit the parent constants, and resetting the value on a parent business object also resets the value on all the children.

Application of business object constants

You can create business object constants for a number of situations. Some examples are:

- Set the display name of a **Workspace** business object or one of its children.
- Set the maximum number of allowed revisions for an item.
- Define the icon to be used for the business object in the user interface.

Example:

To create a business object constant to show an icon for Microsoft Word datasets. Set the following:

Property	Value
Name	**IconName**
Scope	**MSWord**
Data type	**String**
Default value	**MSIcon.png**

With the business constant defined, a My Teamcenter application developer can link the constant value to Microsoft Word datasets so that the **MSIcon.png** icon is displayed in My Teamcenter.

Adding server code to return a constant's value

When you **create a new constant**, you must also add the code on the server to return the constant's value to the caller, so that the caller can branch the business logic based on the returned value.

The server side code can use the following published ITK to retrieve a business object constant value:

```
int CONSTANTS_get_type_constant_value (
       const char*     constant_name,      /* <I> */
       const char*     type_name,          /* <I> */
```


© 2025 Siemens

```
        char**          value                   /* <OF> */
        );
```

Limiting attachment of business object constants

You can define a constant to have a specific scope so that it is available only on certain business objects. This ensures that server API can retrieve the value properly on just those business objects.

Business object constant details

Details of a business object constant are displayed in the **BMIDE** edit view for the constant. You can include business object constant details in BMIDE data model reports.

Create a business object constant

1. Start the New Business Object Constants wizard in one of these ways:

   - On the menu bar, choose **BMIDE→New Model Element**, type **Business Object Constants** in the **Wizards** box, and click **Next**.
   - Open the **Extensions\Constants** folders, right-click the **Business Object Constants** folder, and choose **New Business Object Constants**.

2. In the **Create Business Object Constant** dialog box, specify the following parameters:

For this parameter	Do this
Name	Type the name you want to assign to the new constant in the database. When you name a new data model object, a prefix from the template is automatically affixed to the name to designate the object as belonging to your organization, for example, **A4_**.
Description	Type an explanation of how the constant is to be used.
Scope	To choose a business object to which the constant applies, click **Add** to start the Define Scope wizard. a. In the Define Scope wizard, click the **Browse** button to the right of the **Business Object Scope** box and select a business object. Remember that business object constants are inherited by sub-business objects. A scope of ***** means that the constant applies to all business objects.

© 2025 Siemens

For this parameter	Do this
	b. Click **Finish**. Repeat the steps to add more business objects as needed.
Data Type	Choose one of the following: • **Boolean** Allows two choices to the user (**True** or **False**). • **String** Indicates that the value is a text string. • **List** Contains a list of values. If you selected the **List** data type, a **Values** table appears. Click the **Add** button to the right of the **Values** table to add values to the list: a. In the **Value** box, type a value for the list. b. To prevent the selected value from being overridden by another template, select **Secured**. c. Click **Finish**. Repeat the steps to add more list values as needed.
Default Value	Enter the initial value of the constant. Entry differs depending on the data type you previously chose: • If you selected the **String** data type, type in the default value. • If you selected the **Boolean** data type, click the arrow to select **True** or **False** for the default value. • If you selected the **List** data type, click **Browse** to select a value from the available ones on the list.

© 2025 Siemens

For this parameter	Do this
	• Note: If the selected default value is marked as **Secured**, this value cannot be overridden by any other template.
Allow Live Updates?	Select this box to indicate that the default value of the constant can be changed by **live updates**.
Allow Live Updates to the Constant Override?	Select this box to indicate that if the default value is already overridden, the overridden value still can be changed by a live update.

3. Click **Finish**.

   The new constant appears under the **Business Object Constants** folder.

   Note:

   You can also see the constant on the business object it is applied to. Right-click the business object and choose **Open**. The constant appears in the **Business Object Constants** table on the **Main** tab.

   To modify the value of the constant, select the constant on the **Business Object Constants** table and click the **Edit** button.

4. To save the changes to the data model, choose **BMIDE→Save Data Model**, or click the **Save Data Model** button on the main toolbar.

5. Deploy your changes to the test server. Choose **BMIDE→Deploy Template** on the menu bar, or select the project and click the **Deploy Template** button on the main toolbar.

6. To verify the constant on the server, run the **gettypeconstantvalue** utility. This utility tests the value of the business object constant on a particular business object in the database. The utility accepts the name of a business object constant and business object, and outputs the value of the constant if present.

Change the value of a business object constant

Business object constants provide default values to business objects. Constants that are attached to business objects are inherited by child objects, but can be overridden in the hierarchy. You can typically change the value of a business object constant for a business object or any of its children.

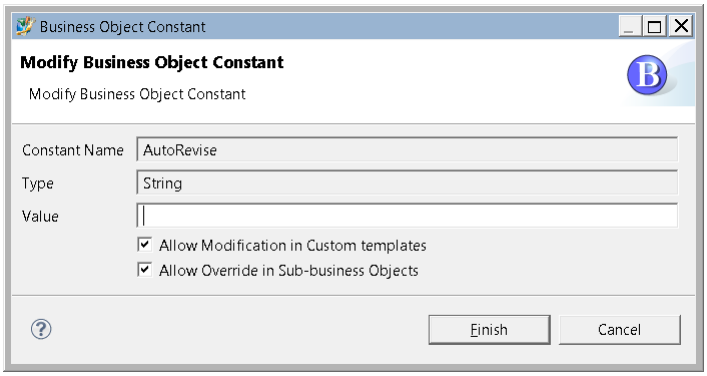
© 2025 Siemens

1. In the **Business Objects** folder, right-click the business object and choose **Open**.

   The constant appears in the **Business Object Constants** table on the **Main** tab.

2. In the **Business Object Constants** table, select the constant that you want to change and click the **Edit** button.

   The **Modify Business Object Constant** dialog box is displayed.



3. Change the value in the **Value** box. How you change it depends on the type of value:

   - Boolean

     Select **True** or **False**.

   - List

     Click the arrow in the **Value** box to select from a list of available values.

   - String

     Type a new value in the **Value** box.

   Note:

   Valid values are dependent on the business object constant. For valid values, view the description of the constant. To see the description of the constant, in the **Extensions** folder, open the **Constants\Business Object Constants** folders and select the constant. The constant details, including a description of the constant, appear in the **Business Object Constant** editor.

4. Select the **Allow Modification in Custom templates** check box to allow the constant attachment value to be changed by a custom template.

© 2025 Siemens

5. Select the **Allow Override in Sub-business Objects** check box to allow the constant attach value to be overridden by child business objects.

6. Click **Finish**.

   The new value for the constant displays in the **Value** column of the **Business Objects Constants** table.

   > Tip:
   >
   > If you want to set the value back to its previous value, click the **Reset** button. This reloads the data model, because all children of a parent business object inherit the parent constants, and resetting the value on a parent business object also resets the value on all the children.

7. To save the changes to the data model, choose **BMIDE**> **Save Data Model**, or on the main toolbar click **Save Data Model**.

   The new constant is saved in the active extension file.

Changing the name of custom business objects

Rename a business object

You may want to rename a business object (type) because you do not like the original name, because you want to add or change a prefix to the name, or because you think that it may at some future point be involved in a **type name collision**. You can rename a business object only when the entire data model is loaded without any validation errors.

> Caution:
>
> Before renaming business objects that have been deployed to a database, read all the documentation for **changing the name of custom business objects**. You can damage your database if you do not perform all the manual steps described in the documentation.

Changing a custom **Item** business object also changes the custom **ItemRevision** and master forms associated with it.

1. In the **Business Objects** folder, right-click the custom business object you want to rename and choose **Rename**.

2. In the Rename Object wizard, click **Next**.

3. In the **New Name** box, type the new name for the business object in the database.

   When you **name a new data model object**, a prefix from the template is automatically affixed to the name to designate the object as belonging to your organization, for example, **A4_**.

© 2025 Siemens

4. In the **Associated Objects** box, type the new name for each associated business object (for example, part revision, part master, and so on). Use the same naming prefix you used for the main business object.

5. Click **Finish**.

   The renamed business object displays in the **Business Objects** folder. There is no need to live update.

6. Run the **change_type_name** utility to change the name of the business object in the database.

   Example:

   To run the utility in dry run mode, enter the following command on a single line:

   ```
   change_type_name -u=username -p=password -g=dba
   -old_type=oldItem -new_type=newItem -dry
   ```

   This runs the utility in dry run mode to estimate how much data would be altered if the old business object is renamed.

Type name collisions

You can extend the Teamcenter architecture with your own custom classes and types. Each of these names must be unique to the system. Typically, when you create a new class or type, you use the Business Modeler IDE, which validates that the new name is unique among all classes or types installed in the system at that time, and enforces a naming pattern that makes future name collisions unlikely. However, if Teamcenter product development subsequently adds classes and types to a patch or new version of Teamcenter that have the same names as your custom classes and types, a collision could potentially occur when your installation of Teamcenter is patched or upgraded.

This type name collision is an issue that blocks a Teamcenter database from being upgraded to a later patch level or version. If a Teamcenter database were operated with a type name collision, data corruption could result. Therefore, if a Teamcenter database has a type collision, the upgrade is prevented until the collision is fixed.

If type collisions have been identified, use the **change_type_name** utility to rename the colliding types in the database.

In addition to the type name change, there may also be data referencing the type. For example, there probably are instances of the type and business rules, list of values, extension points, closure rules, saved queries, and so on, that use the type. Perform additional steps to **make changes for these instances**.

The last of the tasks to perform after running the change_type_name utility is to **rename the business object** in the project template.

© 2025 Siemens

When to run the change_type_name utility

If business object (type) name collisions are identified during Teamcenter installation, patch, upgrade, or deploy custom software operations, the operation cannot proceed. Run the **change_type_name** utility to fix each identified name collision.

1. Review the steps in *Run the change_type_name utility*.

2. Run the **change_type_name** utility in dry run mode by specifying the **-dry** command line option.

   The log file lists all objects that would be modified and the approximate time it would take to complete.

3. Run the **change_type_name** utility to update the type and associated data.

4. Perform the manual steps to be made after running the utility as described in *Run the change_type_name utility*.

5. Rerun the utility in dry run mode. You should see from the log file that there no remaining objects to be modified.

6. Verify that you can successfully log on to Teamcenter.

> Note:
>
> If a problem exists in a particular area of functionality after running the utility, **rerun the utility** to change the type information on any objects that may have been missed.

Objects changed by the change_type_name utility

The **change_type_name** utility changes the following types of data:

Type name
Class name for a primary type
Type name on workspace objects such as items and datasets.
Property rules
Compound properties
Naming rule attachments
Deep copy rules
Business Modeler operations
Extensions
Constants and their attachments
Lists of values
AM rules
Workflow handlers
Saved queries
Property finder formatter used for reports

© 2025 Siemens

Multi-Site objects (stubs and import/export records)
Mapped properties
Filters
Closure rules

The utility does not change the following data that uses the type name:

Preferences
Data residing outside the database like text server files, rich client property files, and so on
Custom code using the old type
Dataset description files (**.des**) in the *TC_DATA***gs_info** directory.

Tasks to perform before running the change_type_name utility

There are a number of tasks that must be performed before you run the **change_type_name** utility to ensure integrity of your data. Review these steps carefully and perform them in order.

1. Back up your database. The utility makes changes directly to the database tables, and making a backup ensures that you can restore the database should problems arise.

2. Ensure that database statistics are up-to-date if you are running Oracle.

3. Add indexes. The utility may need to make changes to **POM_stub** objects. The table for this class can be very large, so you should add two indexes to improve performance. Run the following commands:

   ```
   install -add_index tc-admin-user password group pippom_stub_type 0 POM_stub object_type

   install -add_index tc-admin-user password  pippom_stub_class 0 POM_stub object_class
   ```

   These commands can take a long time to run depending on the size of your database. Allow enough time to create these indexes before you run the utility. These operations may also consume database resources such as temporary tablespaces. Ensure that your database is correctly configured to allow the creation of indexes on a large table.

4. Schedule a time to run the utility and to make the required changes. The utility requires exclusive access to the database to prevent data corruption from occurring. Scalability tests have shown that the utility can modify between 2,000 and 4,000 instances per second depending on database size and hardware resources. To modify a large number of instances, allocate enough time for the operation to complete.

5. Ensure that the **UNDO** tablespace is either sized appropriately or can grow as required. If changes are being made to a large number of instances then the database may require a large **UNDO** tablespace.

© 2025 Siemens

Run the change_type_name utility

Run the **change_type_name** utility from a Teamcenter command prompt.

The utility should be run from an appropriately configured Teamcenter environment. The **change_type_name** utility is available as part of a full Teamcenter installation, and is located in the *TC_ROOT***bin** directory.

Run the command for both Windows and Linux in a Teamcenter shell, for example:

```
change_type_name -u=tc_admin_1 -p=secret -g=dba -old_type=oldItem -new_type=newItem -dry
```

This runs the utility in dry run mode to estimate how much data would be altered if the rename occurs.

When the utility is run, a log file is created in a location pointed to by the **TC_TMP_DIR** environment variable. If the variable is not set then this log file is written to the temporary directory. The location of the log file is also written to the console when the utility is run. The log file contains information about the inputs to the utility, the instructions that were processed, the number of instances that were changed, and timing information. The dry run mode shows the number of instances that would be modified and the approximate time it would take for those modifications.

Tasks to perform after running the change_type_name utility

There are a number of tasks that must be performed after you run the **change_type_name** utility to ensure integrity of your data. Review these steps carefully and perform them in order.

Not all the following steps need to be done. Choose the steps that apply. For example, if you do not have any preferences that use the old business object type, you do not execute the step to modify preferences.

1. Modify preferences.

   Any preferences using the old type name at any level site, group, role, and user must be changed. Preferences are in the database. The administrator can use the **preferences_manager** utility to export site, group, role, and user preferences. Replace each occurrence of the old type with the new type. Then import the XML code to the database using the **OVERRIDE** option.

   The administrator can use **preferences_manager** utility to export and import other group, role, and user preferences using the **-target=***group-name/role-name/user-name* option available with the import and export mode.

2. Modify text server files.

   Change the old type names in all of the XML files in the **TC_MSG_ROOT** directory for your language to the new type names.

3. Modify rich client property files.

© 2025 Siemens

The default properties files, with a .properties extension, are located in the various **com.teamcenter.rac.component.jar** files in the *TC_ROOT***portal\plugins** directory. You may have overridden the default file by creating a **_user.properties** file and wrapping it in an Eclipse plug-in or modified the **.properties** file in the JAR file.

Locate the .properties file that contains the conflicting type and modify it to the new type.

Examples:

- If you include your custom type to be displayed in the Teamcenter viewer, make an entry under the **DatasetViewer.TYPES** section of the following file:

  com.teamcenter.rac.common\com\teamcenter\rac\common\tcviewer\tcviewer.properties

- If you create an icon for your custom type, make an entry in the following file:

  com.teamcenter.rac.common\com\teamcenter\rac\common\common.properties

4. Modify server code.

   - ITK utilities

     Search for the old type name in all of the custom header and custom source files. After you change each occurrence to the new type name, compile and link your code.

   - Workflow handlers

     Search for the old type name in all of the custom header and custom source files and change each occurrence to the new type name. Once completed, compile and link your code.

     > Note:
     >
     > Workflow handler arguments in the database are modified by the utility per the instructions in the **configuration file**. If any of the handler argument patterns are not present in the configuration file, they are not modified. In such a case, you must change the type name manually using the Workflow Designer application in the Teamcenter rich client or by adding instructions to the configuration file for the **EPMHandler** class. If you have already run the utility once, you can **rerun it** with the new instructions in the configuration file.

   - User exits

     Search for the conflict custom type name in all of source files under user exit modules and change each occurrence to the new type name. Compile and link the library.

© 2025 Siemens

- Extensions

  Search for the conflict custom type name in all of the custom XML code and source files. Change each occurrence to the new type name. Compile and link.

5. Modify rich client code.

   Search for the conflict custom type name in all of the custom Java source files under the rich client. Change each occurrence to the new type name. Compile and rebundle the package:

   a. Locate and change the conflict custom type name in all of the Java source files. Compile the files to create new class files.

   b. Use the **jar xvf** command to extract the customized .jar file to any directory to which you have access, for example, **D:\CustFiles**. This creates a directory structure similar to **d:\CustFiles\com\teamcente\rac**.

   c. Copy the new class file to replace the old one under the **d:\CustFiles** directory.

   d. In the **d:\CustFiles** directory, re-create a JAR file of the customization using the **jar cfv** command.

6. Modify the installation and upgrade scripts.

   Replace the old type name with the new type name in any custom written installation or upgrade scripts.

7. Modify attribute mapping.

   You can modify the conflict type name in the customized attribute mappings by exporting the attribute mappings to a file, changing the conflict type name, and importing the file back into Teamcenter.

   a. Open a Teamcenter command prompt.

   b. Change to the **bin** directory within *TC_ROOT*, the installed location of Teamcenter.

   c. Run the **export_attr_mappings** command to output the mappings to a file, for example:

      ```
      export_attr_mappings -u=tc_admin_1 -p=pwd -file=mapping_output_filename
      ```

   d. Edit the mappings file to change the conflicting type name.

   e. Import the edited mapping file back into Teamcenter by running the **import_attr_mappings** command, for example:

      ```
      import_attr_mappings -u=tc_admin_1 -p=pwd -file=mapping_output_filename
      ```


© 2025 Siemens

8. Modify dataset .des files.

   This step is required only if the type being modified is a dataset type or a tool type.

   - Dataset

     Rename the .des file in the **gs_info** directory. For example, if the dataset type name is being changed from **MyDatasetType** to **MyNewDatasetType**, rename the **mydatasettype.des** file in the **gs_info** directory to **mynewdatasettype.des**.

   - Tool type

     Replace the tool type name string in all the .des files to the new tool type name. For example, if the name of a tool is **MyTool** and it is changed to **MyNewTool**, replace all occurrences of **MyTool** in all .des files in the **gs_info** directory to **MyNewTool**.

   > Note:
   >
   > When both the dataset type and the tool type name are changed, follow both steps. If there are multiple *TC_DATA***IMAN_DATA** directories, make the change for one and copy the changed **.des** files to the other *TC_DATA***IMAN_DATA** directory.

9. Make the same changes at all sites that exchange data.

   By default, the utility does not modify all matching objects in the system. This includes those that belong to a different site and stubs representing remote objects. For all sites exchanging data to remain consistent, it is important that the same changes are made at all sites at the same time. Do not run automatic synchronization utilities such as **IDSM** and **data_share** while sites are being modified.

10. Use the Business Modeler IDE client to **rename the type** in Business Modeler IDE templates.

    When you rename a type name in the Teamcenter database, you also must update any Business Modeler IDE template source files that reference the type name.

Configuration file for the change_type_name utility

The **change_type_name** utility processes instructions from a configuration file. This file is in XML format. There are many categories based on the Teamcenter schema classes. Each category can have several instructions.

There are two instructions types:

- **TC_PROCESS_TYPE_BULK**

© 2025 Siemens

Processes a bulk update of data. This is mainly used for types that are likely to have a large number of instances in the database, for example, **Workspace** objects.

- **TC_PROCESS_TYPE_INSTANCE**

  Processes individual type instances. This is used for types that are unlikely to have a large number of instances in the database.

You can add your own instructions for your type if the COTS instructions are not sufficient. For example, you have an embedded string that does not get processed by the COTS instructions, such as a type named **MyType** with a string attribute named **MyAttribute**, or a type name embedded in the attribute such as **This*is*MyType*entry**.

Following is an example of the instruction in the configuration file:

```
<ProcessInstruction type="TC_PROCESS_TYPE_INSTANCE" class=" MyType"
attribute="MyAttribute"
prefix="*" suffix="*"/>
```

Note:

If the log file shows the following message, it means that the instruction was not processed:

```
Processing instruction:
[TC_PROCESS_TYPE_INSTANCE:POM_catia_mfg_userdata:Dataset_Type::]

NOTE: Instruction will not be processed.
Class [POM_catia_mfg_userdata] does not exist.
```

This can be due to two reasons:

- The class or attribute does not exist in the version the utility is run against. If this is the case, you can safely ignore this message.
- If you have changed the configuration file, you may have entered a wrong class name. You must correct the class name in the configuration file.

If, by design, the **MyAttribute** attribute can contain only a dataset type, place the instruction in the **DatasetType** category. If the **MyAttribute** attribute can contain any type, place the instruction in the most general category, for example, **POM_object**.

Rerun the change_type_name utility

If you have successfully run the **change_type_name** utility once and find some objects are not changed because an entry for it did not exist in the configuration file, the utility must be rerun. If some functionality does not work after running the utility, it may be that data pertaining to that functionality did not change.

© 2025 Siemens

For example, you already changed a type name from **MYOLDTYPE** to **MYNEWTYPE**. Revert the type name only, add the new instructions, and rerun the utility.

1. Create a configuration file, for example, **myconfiguration.xml**, containing the following:

```
<?xml version="1.0" encoding="UTF-8"?>
  <Categories>
     <Category categoryName="POMObjectType" className="POM_object">
        <!-- ImanType - This should be the last instruction  -->
        <ProcessInstruction type="TC_PROCESS_TYPE_INSTANCE" class="ImanType"
attribute="type_name"/>
  </Category>
</Categories>
```

2. Run the utility.

```
change_type_name -u=username -p=password -g=group -old_type=MYNEWTYPE
-new_type=MYOLDTYPE -config=myconfiguration.xml
```

   This step reverts the type name only. You must supply the newly created configuration file using the **-config** option on the utility.

3. Add the required instructions to the new configuration file, **myconfiguration.xml**. Refer to the COTS *TC_DATA***change_type_name_config.xml** configuration file for examples.

   > Note:
   >
   > The instruction to change the type should be the last instruction in the configuration file.

4. Run the utility.

```
change_type_name -u=username -p=password -g=group -old_type=MYOLDTYPE
-new_type=MYNEWTYPE -config=myconfiguration.xml
```

   This executes the new instructions and finally changes the old type to the new type. You must supply the newly created configuration file using the **-config** option on the utility.

Convert secondary business objects to primary

When you create a new business object, the **Create Primary Business Object** check box is selected by default in the creation dialog box. This creates a *primary business object* with its own storage class. If you clear this option when you create a new business object, you create a *secondary business object*, one that uses the storage class of its parent business object.

In older versions of Teamcenter, many business objects were secondary business objects, and many have been changed to primary business objects. But some COTS business object types, such as datasets, are still secondary business objects, and there are no plans to change them to primary business objects.

© 2025 Siemens

Secondary business objects result in more traversal of the database to access attribute information, and thus can cause performance issues. To remedy this for a particular custom secondary business object type, you can change it to a primary business object by right-clicking the custom secondary business object in the **Business Objects** folder and choosing **Convert to primary**.

The conversion results in improved performance for **Item**-**Master Form** model, and allows you to add properties to the newly converted primary business object. However, you are under no obligation to convert secondary business objects to primary business objects, and there is no risk if you do not convert secondary business objects. You must convert your custom secondary business objects to primary business objects only if you need to add properties directly to the custom secondary business objects.

> Note:
>
> When a secondary **Item** business object is converted to a primary business object, its secondary **ItemRevision** business object is autoconverted to a primary business object as well. However, master form business objects are not autoconverted.

1. Verify that the business object is secondary.

   Right-click the business object in the **Business Objects** folder, choose **Open**, and look at the **Storage Class** box in the **Main** tab. If the storage class has the same name as the business object, the business object is primary. If it has a different name, it is secondary.

2. Right-click the secondary business object in the **Business Objects** folder and choose **Convert to primary**.

   > Note:
   >
   > The **Convert to primary** context-sensitive menu command is available only if the selected business object satisfies the following conditions:
   >
   > - The selected object is a secondary business object and a custom business object.
   > - The parent of the selected object is a primary business object.
   > - The selected object is not derived under the **ItemRevision** hierarchy.

3. After performing the conversion, the change does not appear immediately in the **Storage Class** box of the business object editor. First, close the business object editor by right-clicking its tab and choosing **Close** or **Close All**. Then right-click the business object in the **Business Objects** folder and choose **Open**. The new storage class is displayed in the **Storage Class** box.

4. Run the **business_model_updater** utility with the **-update=convert_to_primary** flag to migrate the converted business objects to the database, for example:

   ```
   business_model_updater -u=username -p=password -g=group
   -update=convert_to_primary -mode=upgrade
   -file=file-name -log=log-file-name
   ```


© 2025 Siemens

Replace *file-name* with the name of the active extension file that contains the data model definition for the converted business objects, such as the **default.xml** file.

If the number of the instances to be migrated is large, it may take a long time for the command to run. Teamcenter performs the following in the migration:

- Migrates all instances of the converted business objects and their children.
- Creates a new storage class for each converted business object.
- Updates the definitions of the converted business objects to point to their new storage classes.

5. Run the **generate_metadata_cache** utility to regenerate the cache. If you do not regenerate the cache, you can have inconsistent shared memory cache.

6. Package the template and install it using Teamcenter Environment Manager (TEM).

TC_WorkContext business object reference

The **TC_WorkContext** business object is used to define and assign a custom method of setting a work context. The custom method is invoked whenever a work context is set as the current work context. A user in My Teamcenter can create a work context object by choosing **File→New→Work Context**. A user can assign a work context by selecting an item, item revision, or workflow task and choosing **Tools→Assign Work Context**.

To see the operations for the **TC_WorkContext** business object, open it, click the **Operations** tab, and select the **Property Operations** folder. The following table shows valid operations for the **TC_WorkContext** business object.

Business object	Property	Operation	Extension point
TC_WorkContext	allow_subgroups	PROP_ask_value_logical	BaseAction
TC_WorkContext	allow_subgroups	PROP_set_value_logical	BaseAction
TC_WorkContext	group	PROP_ask_value_tag	BaseAction
TC_WorkContext	group	PROP_set_value_tag	BaseAction
TC_WorkContext	project	PROP_ask_value_tag	BaseAction
TC_WorkContext	project	PROP_set_value_tag	BaseAction

© 2025 Siemens

Business object	Property	Operation	Extension point
TC_WorkContext	role	PROP_ask_value_tag	BaseAction
TC_WorkContext	role	PROP_set_value_tag	BaseAction
TC_WorkContext	setting_modifiable	PROP_ask_value_logical	BaseAction
TC_WorkContext	setting_modifiable	PROP_set_value_logical	BaseAction
TC_WorkContext	user	PROP_ask_value_tag	BaseAction
TC_WorkContext	user	PROP_set_value_tag	BaseAction
TC_WorkContext	workcontext_desc	PROP_ask_value_string	BaseAction
TC_WorkContext	workcontext_desc	PROP_set_value_string	BaseAction
TC_WorkContext	workcontext_name	PROP_ask_value_string	BaseAction
TC_WorkContext	workcontext_name	PROP_set_value_string	BaseAction

Classes

Introduction to creating classes

In the **Advanced** perspective, the **Classes** view is used for working with *classes*, the persistent representations of the data model schema.

A class is the logical **data model** and maps the storage of a business object to the database. Every class has a business object by the same name. A class can have attributes defined on it. Every attribute defined on the class results as a property on the business object. Classes support inheritance. Any attribute defined on a parent class is inherited by its children.

The most commonly used classes under which you create new child classes include:

- **POM_object**

  Represents storage classes in the database.

- **POM_application_object**

© 2025 Siemens

Represents Teamcenter application classes in the database.

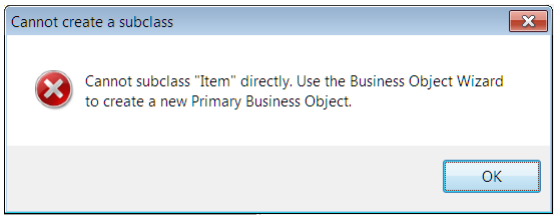
- **WorkspaceObject**

  Represents commonly used classes, such as **Item** and **Document**, in the database.

When you add a class, a matching primary business object is automatically generated. A *primary* business object has the same name as its associated storage class. (A *secondary* business object uses the storage class of its parent business object.)

You can only create new subclasses for the following classes by creating primary business objects: **AppInterface**, **Dataset**, **Form**, **Item**, and **ItemRevision**, and **StructureContext**. If you right-click any of these classes or their children and choose **New Class**, the following message is displayed:

```
Cannot subclass class-name directly.
Use the Business Object Wizard to create a new Primary Business Object.
```



You cannot subclass these classes or their children directly in the **Classes** view. To subclass them, create new business objects under the corresponding business objects of the same name in the **Business Objects** view of the **Advanced** perspective; the new subclasses are created automatically. For example, to create a **MyItemRevision** class, right-click the **ItemRevision** business object in the **Business Objects** view and choose **New Business Object**; the **MyItemRevision** class is automatically created and appears in the **Classes** view.

Add a new class

1. Access the **Advanced** perspective by choosing **Window→Open Perspective→Other→Advanced**.

2. Click the **Classes** tab.

3. In the **Classes** view, select the project in which you want to create a new class.

4. Right-click the project and choose **Organize→Set active extension file**. Select the file where you want to save the data model changes.

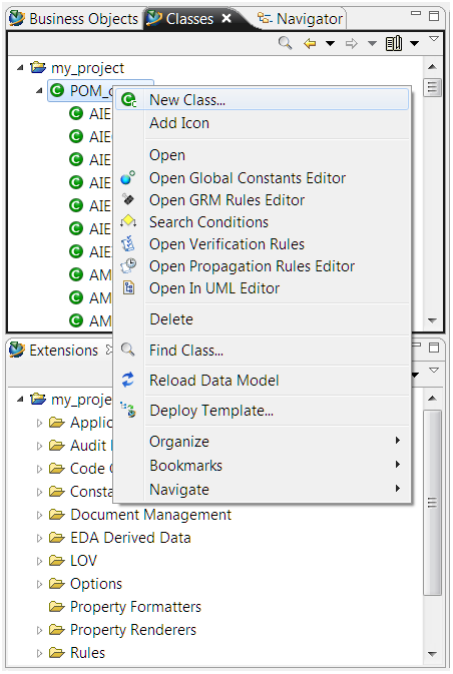
5. Browse to a class under which you want to create the new class. To search for a class, you can click the **Find Class** button at the top of the view.

6. Right-click the class and choose **New Class**. The New Class wizard runs.

© 2025 Siemens

There are other ways to launch the New Class wizard:

- Drag a class from the **Class** view into the UML editor.
- Drag **Class** from the UML editor palette into the UML editor.



7. In the **Class** dialog box, enter the following information:

   a. In the **Name** box, type the name of the class as you want it to appear in the database.

      A prefix from the template is automatically affixed to the name to designate the object as belonging to your organization, for example, **A4_**.

   b. The **Parent** box displays the already-selected parent class.

   c. Select the **Exportable?** check box if the class can be exported using PLM XML.

   d. Select the **Uninheritable?** check box if the class cannot have children classes.

   e. The **Uninstantiable?** check box indicates whether instances of the class can be created in user interfaces such as the rich client. This check box is read-only.

   f. The **Store as lightweight object** check box indicates if the object is stored in its own database table outside of the **POM_object** database table. This improves performance of POM object handling. Initially only a limited number of internal classes may be stored as lightweight objects. This check box is read-only.

© 2025 Siemens

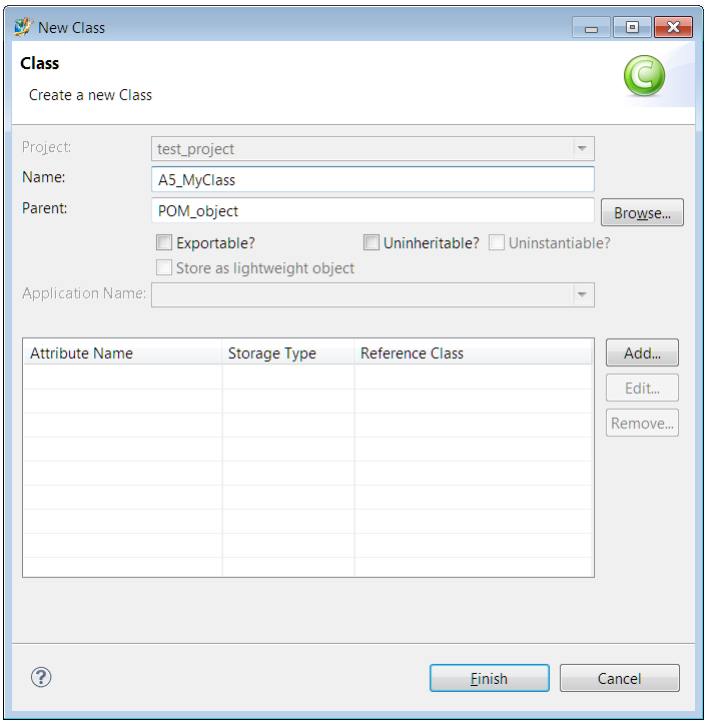
g. Click the arrow in the **Application Name** box to choose the application that can be used to add or edit attributes on the class.

Available applications appear in the **Applications** folder in the **Extensions** view. You cannot configure applications, but you may need to look at them in the **Extensions** view to understand which classes you cannot edit with ITK code.

h. Click the **Add** button if you want to add an attribute to the class.

i. Click **Finish**.

The new class is created, as well as the corresponding primary business object.



8. The new class appears in the **Classes** view. A **c** on the class symbol indicates that it is a custom class. To see the corresponding new business object, open the **Business Objects** view.

To bookmark the class, right-click the class and choose **Bookmarks→Add Bookmark**. To access the class later, click the arrow in the **Bookmarks** button at the top of the **Classes** view.

9. To save the changes to the data model, choose **BMIDE→Save Data Model**, or click the **Save Data Model** button on the main toolbar.

10. Deploy your changes to a test server. Choose **BMIDE→Deploy Template** on the menu bar, or select the project click the **Deploy Template** button on the main toolbar.

© 2025 Siemens

11. To verify your changes, look for the attributes on the new class that are inherited by a business object. Perform the following steps:

    a. In the Business Modeler IDE, **create a new business object** that uses the new class as the parent for the form storage class.

    b. Save the changes by choosing **BMIDE→Save Data Model**.

    c. Deploy to the test server again.

    d. In the Teamcenter rich client, create an instance of the new business object. Open the instance and select its master or revision form. In the **Viewer** tab, you should see the new attributes you created on the new class.

Class attributes

Introduction to attributes

Attributes are class characteristics, such as name, number, description, and so on. An attribute is a persistent information tag assigned to all objects in the same class. **The attributes of the class appear in a table** when you right-click a class in the **Classes** view of the **Advanced** perspective and choose **Open**.

Attributes are of a defined data type, for example, integer, string, and so on. An attribute contains either a *value* (if the attribute is an integer, string, date, or logical data type), or it can contain a *reference* to another class if the attribute is a typed reference or untyped reference. For example, the **object_name** attribute contains a value because it is a string attribute; it can have a value such as **000001/A**. The **last_mod_user** attribute contains a reference, because it is a typed reference attribute; its value points to a user's name.

Attributes can contain one value (scalar) or contain many values (array). An example of a scalar attribute is the **last_mod_user** attribute, because it contains a single value, the user name. An example of an array attribute is the **project_list** attribute, because is contains a list of projects, for example, **Car05**, **Car06**, **Car08**.

Array attributes can be a fixed length array or a variable length array (VLA). Siemens Digital Industries Software strongly recommends that attributes are variable length array (VLA), especially when you expect a large number of business object instances. We do not recommend small, fixed length arrays with a maximum length of eight or less elements as the Teamcenter query performance may decrease.

> Note:
>
> Adding a fixed length array to a business object with a large number of database instances increases the Business Modeler IDE deployment time and impacts rich client performance when the user creates a new instance.

© 2025 Siemens

The user interface shows a display name for the attribute via properties. For example, the **object_desc** property displays as **Description** in the user interface. To be proficient with attributes, you need to know both the internal name of the attribute and its display name. You can change the settings in the rich client to display the internal name of an attribute in the UI. Log on to the rich client as an administrator, choose **Edit→Options**, and in the left pane of the **Options** dialog box, choose **Options→General→UI**. In the right pane, click the **Sys Admin** tab and select **Real Property Name**. To verify the change, select an item in the rich client and choose **View→Properties**.

Add or change attributes on classes

Attributes are item characteristics, such as name, number, description, and so on. You can add or modify attributes on custom classes, and some COTS classes.

1. Access the **Advanced** perspective by choosing **Window>Open Perspective>Other>Advanced**.

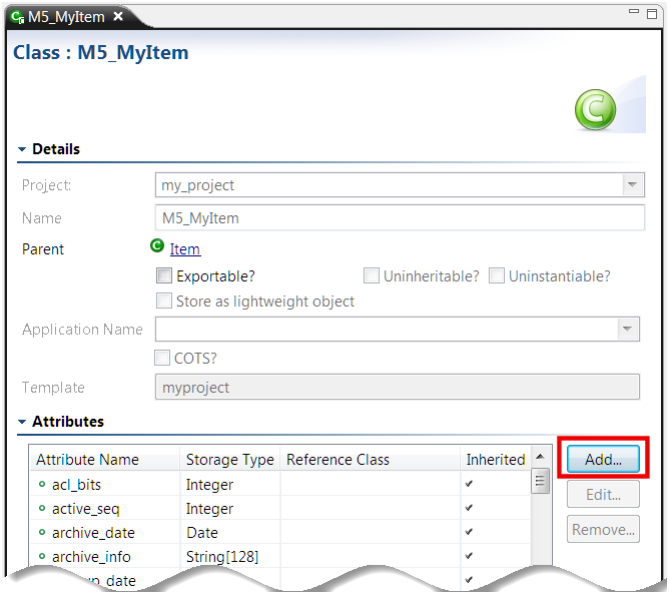
2. Click the **Classes** tab to display the **Classes** view.

3. Select the project in which you want to save the attribute extensions. Right-click the project and choose **Organize>Set active extension file**. Select the file where you want to save the data model changes.

4. Browse to the custom class for which you want to manage attributes. To search for a custom class, you can click the **Find Class** button at the top of the view and select **Custom** on the **Find Class** dialog box.

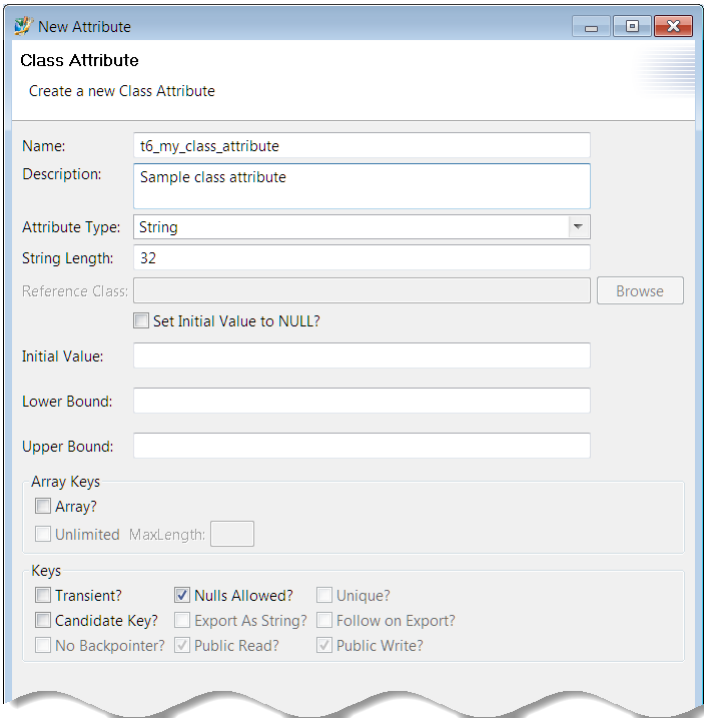
5. Right-click the custom class and choose **Open**. A view displays the class details and attributes.

6. To add an attribute, click the **Add** button to the right of the **Attributes** table.



© 2025 Siemens

The **Class Attribute** dialog box appears.



7. Perform the following steps in the **Class Attribute** dialog box:

   a. In the **Name** box, type the attribute name as you want it to appear in the database.

      When you name a new data model object, a prefix from the template is automatically affixed to the name to designate the object as belonging to your organization, for example, **a4_**. Class attributes begin with a lowercase letter.

   b. In the **Description** box, type a description of the new attribute.

   c. In the **Attribute Type** box, select the storage type for the attribute:

      Boolean

      Allows two choices to the user (**True** or **False**).

      Character

      A single character, such as **A**, **B**, **Z**.

      Date

      A calendar date. A form using this format displays a date selector.

      For the date attribute type, the earliest date supported is January 2, 1900.

      Double

© 2025 Siemens

A double-precision floating point decimal number. (For Oracle, the limit for the double property value is 1E-130 to 9E125. For SQL Server, the limit is 2.3E-308 to 1.7E308.)

ExternalReference

Points to data outside of Teamcenter.

Integer

An integer without decimals from 1 to 999999999.

LongString

A string of unlimited length.

String

A string of characters. Type the byte length of the string attribute in the associated **String Length** box.

For the English alphabet, one character requires one byte. Special characters in Western languages such as French and Portuguese, for example, may require two bytes. For such special characters, and for most characters in multibyte languages such as Chinese, Japanese, and Korean, a field with a string length of 128 can accommodate a maximum of 64 double-byte characters.

TypedReference

Points to a Teamcenter class. Select the class where the attribute is stored in the associated **Reference Class** box.

UntypedReference

Points to any class of data.

d. Select **Set Initial Value to NULL?** if you want the default value to be a null value.

e. If you did not select **Set Initial Value to NULL**, in the **Initial Value** box, type the value to populate the attribute.

f. In the **Lower Bound** box, type the lower numerical limit for a numerical or alphanumerical attribute.

g. In the **Upper Bound** box, type the upper numerical limit for a numerical or alphanumerical attribute.

h. In the **Array Keys** section, check the properties that apply:

Array?

Specifies that the attribute is a string array.

Unlimited

Indicates that there is no limit on the number of characters used for the attribute.

© 2025 Siemens

MaxLength

Specifies the maximum number of characters allowed for the attribute.

> Note:
>
> Siemens Digital Industries Software recommends using **Unlimited** for a variable length array (VLA) especially when you expect a large number of elements.

i. In the **Keys** section, check the properties that apply:

Transient?

Does not persist the attribute in the database.

Nulls Allowed?

Allows an empty value for the attribute.

Unique?

Specifies that the attribute value cannot be duplicated.

Candidate Key?

Specifies that when exporting an object, send this attribute and rely on the receiving site to look up this string in the local database. This is typically used for classes such as **user** where the **user_id** attribute must be located on the importing site.

Export As String?

Specifies that when exporting an object, export this attribute as a string.

Follow on Export?

Exports the referenced object when the attribute is exported.

No Backpointer?

Does not record attribute in the POM backpointer table.

Each time a forward pointer is created for attribute, an inverse record is stored in the POM backpointer table. Therefore, the backpointer table keeps a record of where-referenced attributes, and is used for where-referenced calls and for a check on delete that things are not referenced.

To help system optimization, you may want to use the **No Backpointer** key for those attributes that are unlikely ever to be deleted (such as **owning-user**, **owning-group**, or **dataset-type**). Choosing the **No Backpointer** key saves the POM system from having to check every reference column in the table whenever a where-referenced or delete action is called.

Public Read?

Grants everyone read access to the attribute.

Public Write?

© 2025 Siemens

Grants everyone write privileges on the attribute.

j. When done making changes, click **Finish**.

The new attribute appears in the properties table and is marked with a **c** indicating it is a custom attribute.

8. To change values on the new attribute, click the **Edit** button. You can only change values on your custom attributes. You cannot change values on COTS (commercial off-the-shelf) attributes.

To add another attribute, click the **Add** button. To remove an attribute, click the **Remove** button.

> Note:
>
> Database platforms set the limits on the number of attributes (columns) that can be supported in a class. The practical limit depends on the attribute type and size. To be portable (allowing for future database platform migration or Multi-Site exchanges between Teamcenter sites on alternate database platforms) it is worth checking that the schema (including localization and indexes) can be deployed on all database platforms required.
>
> Microsoft SQL Server has limited chaining for strings in rows exceeding 8K, but definitions that raise warnings are cleared.
>
> Warnings are present in the Business Modeler IDE deployment syslog files with codes 1708, 1945 or 1981. When tables exceed the SQL Server maximum row size they hit a hard error (511) for the table row, and error 1946 for index entry. Both errors are mapped to the **POM_rdbms_error** record (515209).

9. To save the changes to the data model, choose **BMIDE→Save Data Model**, or click the **Save Data Model** button on the toolbar.

10. Deploy your changes to the test server. Choose **BMIDE→Deploy Template** on the menu bar, or select the project click the **Deploy Template** button on the main toolbar.

11. After deployment, test your changed attributes in the Teamcenter rich client.

For example, if you changed attributes on a custom subclass of the **Item** class, in the My Teamcenter application, select a business object that inherits from that custom class and view its attributes in the **Viewer** tab.

Class attributes table

The attributes of the class appear in a table when you right-click a class in the **Classes** view in the **Advanced** perspective and choose **Open**.

© 2025 Siemens

Column	Description	
Attribute Name	Displays the property name for the attribute.	
Storage Type	Displays the storage type of the attribute:	
	Character	A single character, such as **A**, **B**, **Z**.
	Date	A calendar date. A form using this format displays a date selector. Tip: For the date attribute type, the earliest date supported is January 2, 1900.
	Double	A double-precision floating point decimal number. (For Oracle, the limit for the double property value is 1E-130 to 9E125. For SQL Server, the limit is 2.3E-308 to 1.7E308.)
	ExternalReference	Points to data outside of Teamcenter.
	Integer	An integer without decimals from 1 to 999999999.
	Logical	A Boolean value of **True** or **False**.
	LongString	A string of unlimited length. Note: **LongString** attributes cannot be used in queries.
	Short	An integer number without decimals from 1 to 9999. Note: This storage type is deprecated. Although you can no longer create properties of this type, already-existing properties of this storage type are still supported.
	String	A string of characters.
	TypedReference	Points to a Teamcenter class.
	UntypedReference	Points to any class of data.
Reference Class	Displays the reference class for the attribute (for typed reference classes).	
Inherited	Indicates if the attribute is inherited from a parent class.	
Source Class	Lists the parent class that provides the attribute.	

© 2025 Siemens

Column	Description
COTS	Indicates whether the attribute is a standard (COTS) or custom attribute. COTS means *consumer off-the-shelf*, or from a dependent template.
Template	The template in which the attribute is defined.
Initial Value	Lists the initial value of the attribute on a creation window in the Teamcenter user interface. You can change this value if desired.
Lower Bound	The lower numerical limit for a numerical or alphanumerical attribute.
Upper Bound	The upper numerical limit for a numerical or alphanumerical attribute.
Array	Specifies that the attribute is an array. • Fixed length Recommended when the number of elements is known and is not expected to change. • Variable length (VLA) Recommended for large number of elements because the length is determined at run time as needed. Note: Siemens Digital Industries Software recommends variable length array (VLA) especially when you expect a large number of elements.
Array Length	Indicates the length of the array elements.
Public Write	Grants everyone write privileges on the attribute.
Public Read	Grants everyone read access to the attribute.
Nulls Allowed	Allows an empty value for the attribute.
Unique	Specifies that the attribute value cannot be duplicated.
Candidate Key	Specifies that when exporting an object, send this attribute and rely on the receiving site to look up this string in the local database. This is typically used for classes such as **user** where the **user_id** attribute must be located on the importing site.
Transient	Does not persist the attribute in the database.
Follow on Export	Exports the referenced object when the attribute is exported.
Export As String	Specifies that when exporting an object, export this attribute as a string.
No Backpointer	Does not record the attribute in the POM backpointer table. Each time a forward pointer is created for attribute, an inverse record is stored in the POM backpointer table. Therefore, the backpointer table keeps a record of where-

© 2025 Siemens

Column	Description
	referenced attributes, and is used for where-referenced calls and for a check on delete that things are not referenced. To help system optimization, you may want to use the **No Backpointer** key for those attributes that are unlikely ever to be deleted (such as **owning-user**, **owning-group**, or **dataset-type**). Choosing the **No Backpointer** key saves the POM system from having to check every reference column in the table whenever a where-referenced or delete action is called.

Classes that cannot have new attributes

You can **add or modify attributes on custom classes**, and some COTS classes.

You cannot add attributes to the following COTS classes:

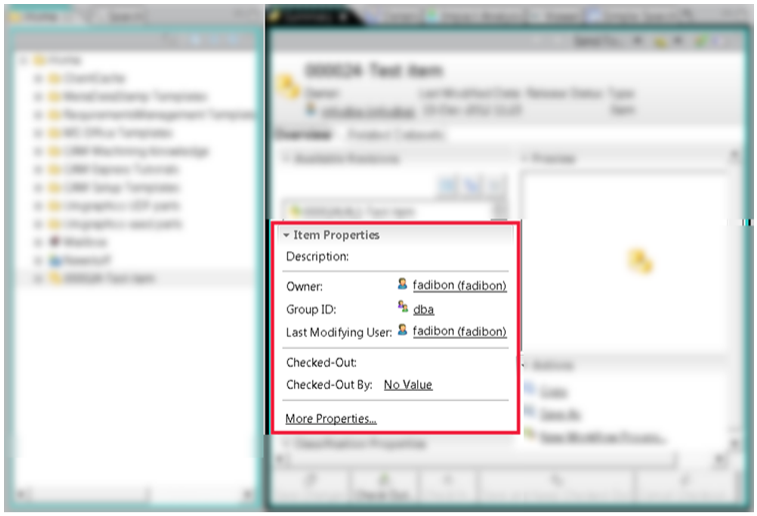
AM_ACE	ExtensionDescriptor	POM_attribute	POM_system_class
AM_ACL	Extension	POM_class	POM_stub
AM_named_tag	Group	POM_data_manager	POM_user
AM_privileges	GroupMember	POM_dd	PropertyBMOperation
AM_tree	GroupSecurityNamedTag	POM_group	Role
BMOperation	ImanType	POM_imc	RoleInSchedule
BusinessRule	NameField	POM_member	TypeBMOperation
DatasetType	NameRule	POM_object	TC_Preferences
EffectivityMode	POM_accessor	pom_session	User
ExtensionPoint	POM_application	POM_site_config	

Properties

Introduction to properties

Business object properties contain information such as name, number, and description. A business object derives its properties from its persistent storage class. In addition to the properties that are derived from the persistent storage class, business objects can also have additional properties such as run-time properties, compound properties, and relation properties. The following image shows an example of some properties for a business object instance in the rich client.

© 2025 Siemens



Following are some are some common tasks you perform when defining properties for a business object:

- To **create properties on a business object**, click the **Properties** tab and then click the **Add** button.
- To **change the display name of properties**, select a property on the **Properties** tab and click the **Localization** tab.
- To **display custom properties in the end-user interface**, use XML rendering style sheets.
- To **define which properties are displayed on creation dialog boxes**, use the **Operation Descriptor** tab.

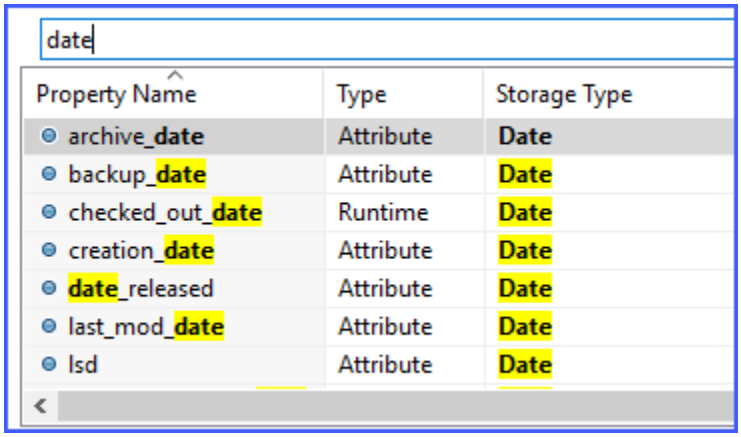
Properties table in the BMIDE

To examine the properties of a business object, open the object and click the **Properties** tab. The properties of the business object appear in a table. Depending on the selected property, additional characteristics of the property appear in tabs and tables below the properties table.

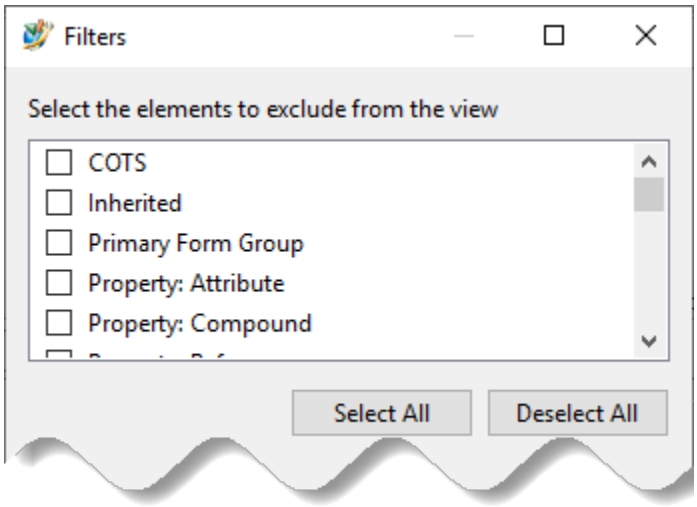
Initally, all the object properties are listed alphabetically by property name. You can use the following means of controlling the display of properties in the table:

- To show only properties with a value that matches a string, in the search box above the table type a string. For example, **date**.

© 2025 Siemens



- To filter displayed properties by element type, right-click within the table and choose **Select Filters**. For example, to hide commercial off-the-shelf properties and thus display only custom properties, mark the COTS box.

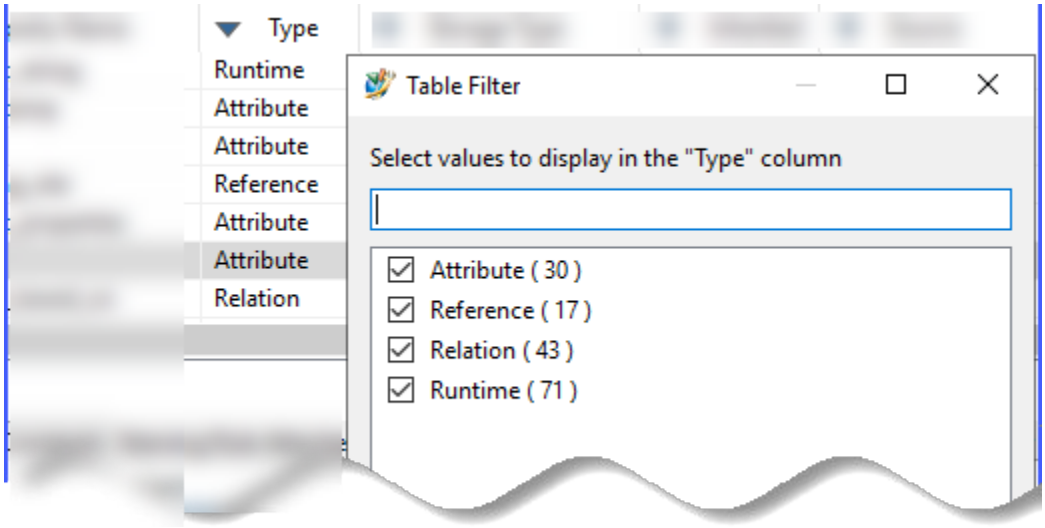


- To first group properties by their **Source** column value before any sort is applied, right-click within the table and choose **Organize by Inheritance**.

- To sort the properties list by a column value, click a column heading. Click the heading again to switch between ascending and descending order.

- To enable filtering displayed properties by the values in any column, click **Enable / Disable table filtering** . A solid triangle appears in column headings. Click a triangle to open a table filtering dialog where you can select or clear values to filter the displayed properties.

© 2025 Siemens



The following table describes content that might be contained in a column.

Column	Description
Property Name	The name of the property (class attribute).
Type	The type of property. **Attribute** A simple value (for example, integer, string, or date). The value is stored in the database as an attribute and mapped to the property. **Compound** A property displayed from one type as if it were the property of that type, though it actually resides on another type. Though run-time properties can be used to display such a property, they require custom coding to do so. Compound properties allow you to create such properties without custom coding. **Reference** A reference to another object. The reference is stored in the database as a reference attribute and mapped to the property. **Relation** A reference to secondary objects of an **ImanRelation** business object. The reference is stored in the database as a relation type and is derived from that **ImanRelation** business object. **Runtime** A property that is defined at run time and attached to types. Run-time properties do not map directly to persistent attributes, references, or relations. Instead, run-time properties derive data from one or more pieces of system information (for example, date and time) that are not stored in the Teamcenter database. Run-time

© 2025 Siemens

Column	Description
	properties can also be used to display a property of one type as if it were a property of another type.
Storage Type	How the property is stored. **Character** A single character, such as **A**, **B**, **Z**. **Date** A calendar date. A form using this format displays a date selector. For the date attribute type, the earliest date supported is January 2, 1900. **Double** A double-precision floating point decimal number. (For Oracle, the limit for the double property value is 1E-130 to 9E125. For SQL Server the limit is 2.3E-308 to 1.7E308.) **ExternalReference** Points to data outside of Teamcenter. **Integer** An integer without decimals from 1 to 999999999. **Logical** A Boolean value of **True** or **False**. **LongString** A string of unlimited length. Prior to Teamcenter 2007.1, the **Note** type was used for unlimited strings. Use the **LongString** type instead. **LongString** properties cannot be used in queries. **Short** An integer number without decimals from 1 to 9999. This storage type is deprecated. Although you can no longer create properties of this type, already-existing properties of this storage type are still supported. **String** A string of characters. **TypedReference** Points to a Teamcenter class.

© 2025 Siemens

Column	Description
	UntypedReference Points to any class of data.
Inherited	Whether the property is inherited from a parent business object.
Source	The parent business object or class that provides the property.
COTS	Whether the property is a standard or custom property. COTS means *commercial off-the-shelf*.
Referenced Type	The typed reference pointing to a Teamcenter class.
Array	Whether the property is an array of the data of the data type (for example, string).
Template	The template that provides the attribute.

Hide properties on a form business object

Forms in the Teamcenter rich client hold additional information about items. If you do not want a property on a form to be visible to end users, you can hide the property by overriding its **Visible** property constant. You can do this for properties on the **Form** business object and its children.

1. Open the **Business Objects** folder.

2. Browse to the child of the **Form** business object that has the property you want to hide. To search for a form business object, you can click the **Find** button at the top of the view.

3. Right-click the form business object, choose **Open**, and click the **Properties** tab in the resulting view.

   The properties of the form appear in a table.

4. In the **Properties** table, select the property you want to hide.

5. In the **Property Constants** table, select **Visible**.

6. Click the **Edit** button to the right of the **Property Constants** table.

   The system displays the **Modify Property Constant** dialog box.

7. Click **Value** to clear the box and click **Finish**.

8. To save the changes to the data model, choose **BMIDE→Save Data Model**, or click the **Save Data Model** button on the main toolbar.

9. Deploy your changes to a test server. Choose **BMIDE→Deploy Template** on the menu bar, or select the project and click the **Deploy Template** button on the main toolbar.

© 2025 Siemens

10. After deployment, test to ensure the property is hidden in the Teamcenter rich client. Open the form and verify that the property is now hidden.

Add properties to a custom form business object

Forms in the Teamcenter rich client use properties to hold additional information about items. You can add additional properties to custom forms that you create.

1. Open the **Business Objects** folder.

2. Browse to a custom form. All custom forms are children of the **Form** business object. To search for the business object, you can click the **Find** button at the top of the view.

3. Right-click the custom form business object, choose **Open**, and click the **Properties** tab in the resulting view.

   The properties of the form appear in a table.

4. Click the **Add** button to the right of the **Properties** table to add properties.

   > Note:
   >
   > Unlike other business objects, which have one directly corresponding storage class, Form business objects have two storage classes: a class corresponding to the business object that the form attaches to (call it class X), and a second class corresponding to the form type (call it class Y). When you add a persistent property to a Form business object, the BMIDE adds a runtime property to the Form business object, and an attribute to class Y.

5. After you add a new property, select the property and ensure that its **Enabled** and **Visible** property constants are set to **true**. This ensures that the property is enabled for use in the user interface.

6. To save the changes to the data model, choose **BMIDE→Save Data Model**, or click the **Save Data Model** button on the main toolbar.

7. Deploy your changes to a test server. Choose **BMIDE→Deploy Template** on the menu bar, or select the project and click the **Deploy Template** button on the main toolbar.

8. After deployment, test to ensure the property is visible on the form. For example, in the Teamcenter rich client, create an instance of the form by choosing **File→New→Form**. Open the form instance and verify that the new property is visible on the form.

© 2025 Siemens

Add properties

Adding properties to an object

When you want to include additional information in your business objects, you can add properties to contain that information. For the various types of properties you can add to a business object, the general procedure is the same.

Once a new property is added to a business object, there are many ways to **display custom properties in the client user interface**. The easiest way is to add the properties using **XMLRenderingStylesheet** datasets.

To add a property to a business object, do the following in the BMIDE:

1. If you intend to add an operation to the new property, set the active library for the property. In the **Extensions** folder, open the **Code Generation\Libraries** folders, right-click the library and choose **Organize**>**Set as active Library**. A green arrow in the library symbol indicates it is the active library.

2. In the **BMIDE** view, search for the business object to which you want to add the property, or expand the **Business Objects** folder as needed to show the business object.

   To search for a business object, you can click the **Find** button at the top of the view.

3. Right-click the business object and choose **Open**.

4. In the resulting view, click the **Properties** tab.

5. To the right of the properties table, click **Add**.

   The **New Property** wizard guides you through the remainder of the process.

   Detailed procedures for various attribute types are covered in the respective topics:

   Add a persistent property
   Change persistent property attribute definitions
   Add a runtime property
   Add a compound property
   Add a complex property
   Add a relation property
   Add a table property
   Add a name-value property

6. After you add a property to an object, to configure the property for use in the object, select the property in the **Property Constants** table and set the following **property constants**:

© 2025 Siemens

For this constant	Do this
Enabled	Set to **true**.
Modifiable	If you want the property to be writable, change the constant from **Read** to **Write**.
Visible	Set to **true**. If you do not want the property to be visible in the user interface, you must explicitly set the value to **false**. This is because default value for the Visible constant is *true*. For configured **runtime** or **compound** properties, the default value for the Visible constant takes precedence over its inherited value.

a. In the **Property Constants** table, select the constant you want to change and click the **Edit** button.

7. Save your changes to the data model.

8. If you created a property operation, generate code.

   In the **Business Objects** folder, right-click the business object to which you added the property operation and choose **Generate Code**>**C++ classes**.

9. If you generated code, write an implementation in the generated *business_object***Impl.cxx** file.

10. Deploy your changes to a test server.

11. After deployment, test your new property in the Teamcenter rich client.

    For example, if you added a property on a custom child of the **Form** class, in the My Teamcenter application, select an instance of that custom form business object and view its properties in the **Viewer** tab.

    If you added a compound property to a business object, you should see the same property on both the source object's and the target object's properties page in the rich client.

Add a persistent property

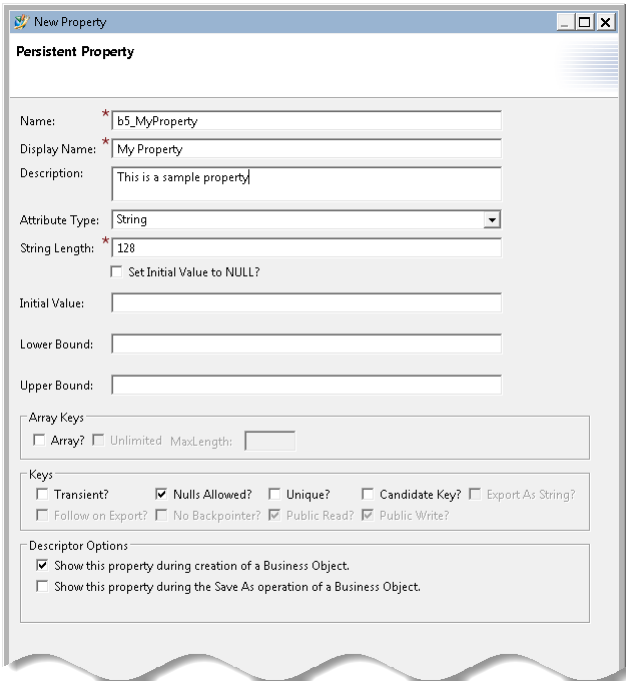
You can add persistent properties to COTS and custom business objects. Adding a persistent property to a business object is effectively the same process as adding attributes to classes.

1. Open the **New Property** wizard as described in steps 1 through 5 of Adding properties to an object .

2. Under **Property Types** select **Persistent**. Click **Next**.

© 2025 Siemens

The **Persistent Property** dialog box is displayed.



Note:

Business objects typically have one directly corresponding storage class. However, Form business objects have two storage classes: a class corresponding to the business object that the form attaches to (call it class X), and a second class corresponding to the form type (call it class Y). When you add a persistent property to a Form business object, the BMIDE adds a runtime property to the Form business object, and an attribute to class Y.

3. Complete the options in the **Persistent Property** dialog box.

For this option	Do this
Name	Type the property name as you want it to appear in the database. The name must be USASCII7 characters only and cannot contain spaces. When you name a new data model object, a prefix from the template is automatically affixed to the name to designate the object as belonging to your organization, for example, **a4_**. The prefix for properties begins with a lowercase letter.
Display Name	Type the name as you want it to appear in the user interface.
Description	Type a description of the persistent property.
Attribute Type	Select the storage type for the attribute. Choose from the following attribute types: **Boolean**

© 2025 Siemens

For this option	Do this
	Allows two choices to the user (**True** or **False**). **Character** A single character, such as **A**, **B**, **Z**. Note: Attributes with this type only store single-byte characters. To store a multibyte character, use the **String** type. **Date** A calendar date. A form using this format displays a shortcut date selector. For the date attribute type, the earliest date supported is January 2, 1900. **Double** A double-precision floating point decimal number. (For Oracle, the limit for the double property value is 1E-130 to 9E125. For SQL Server the limit is 2.3E-308 to 1.7E308.) **ExternalReference** Points to data outside of Teamcenter. **Integer** An integer without decimals. **LongString** A string of unlimited length. Before Teamcenter 2007.1, the **Note** type was used for unlimited strings. Use the **LongString** type instead. **String** A string of characters. Type the byte length of the string attribute in the associated **String Length** box. For the English alphabet, one character requires one byte. Special characters in Western languages such as French and Portuguese, for example, may require two bytes. For such special characters, and for most characters in multibyte languages such as Chinese, Japanese, and Korean, a field with a string length of 128 can accommodate a maximum of 64 double-byte characters. **TypedReference**

© 2025 Siemens

For this option	Do this
	Points to a Teamcenter class. Next to the **Reference Business Object** box, click **Browse** and select the class. **UntypedReference** Points to any class of data.
Set Initial Value to Null	If you do not want a beginning value on the property, select the check box. For Boolean properties, this means that neither true or false are selected.
Initial Value	Type the value to populate the attribute.
Lower Bound	Type the lower numerical limit for a numerical or alphanumerical attribute.
Upper Bound	Type the upper numerical limit for a numerical or alphanumerical attribute.
Array Keys	Check the properties that apply: **Array?** Specifies that the attribute is an array of the data of the data type (for example, string). **Unlimited** Indicates that there is no limit on the number of array elements used for the attribute. Siemens Digital Industries Software recommends using **Unlimited** for a variable length array (VLA) especially when you expect a large number of elements. **MaxLength** Specifies the maximum number of array elements allowed for the attribute.
Keys	Check the properties that apply: **Transient?** Does not persist the attribute in the database. **Nulls Allowed?** Allows an empty value for the attribute. If both the **Nulls Allowed?** check box and the **Set Initial Value to NULL?** check box are cleared, the system automatically defines a **CreateInput operation** on the **Operation Descriptor** tab of the business object.

© 2025 Siemens

For this option	Do this
	Unique? Specifies that the attribute value cannot be duplicated. Note that if the database contains existing occurrences of the object, those occurrences effectively have null values for the new property being added. Null values are incompatible with **Unique?**. To deploy your modified custom object, you can do either of the following: • First delete any existing occurrences of the object. • Deploy your modified custom object without **Unique?**, fill existing occurrences with unique and not null values, and then set **Unique?** and deploy again. **Candidate Key?** Specifies that when exporting an object, send this attribute and rely on the receiving site to look up this string in the local database. This is typically used for classes such as **user** where the **user_id** attribute must be located on the importing site. **Export As String?** Specifies that when exporting an object, export this attribute as a string. **Follow on Export?** For typed and untyped references, exports the referenced object when the attribute is exported. **No Backpointer?** Does not record the attribute in the POM backpointer table. Each time a forward pointer is created for an attribute, an inverse record is stored in the POM backpointer table. Therefore, the backpointer table keeps a record of where-referenced attributes and is used for where-referenced calls and for a check that things are not referenced when an item is deleted. To help system optimization, you may want to use the **No Backpointer** key for those attributes that are unlikely ever to be deleted (such as **owning-user**, **owning-group**, or **dataset-type**). Selecting the **No Backpointer** key saves the POM system from having to check every reference column in the table whenever a where-referenced or delete action is called. **Public Read?**

© 2025 Siemens

For this option	Do this
	Grants everyone read access to the attribute. **Public Write?** Grants everyone write privileges on the attribute.
Descriptor Options	Select the options that apply: **Show this property during creation of a Business Object** Adds this property to the dialog box displayed when creating the business object on which the property resides. **Show this property during the Save As operation of a Business Object** Adds this property to the dialog box displayed when performing a **Save As** operation on the business object on which the property resides. If these boxes are checked, the property appears on the Operation Descriptor tab. The section does not appear when adding properties to the **Form** business object or one of its children. Properties added to forms are automatically added to the **Operation Descriptor** tab. You cannot delete these properties from the **Operation Descriptor** tab, but you can modify them using the **Visible** and **Required** property constants.

4. Click **Next**.

   The Business Modeler IDE displays the **Property Operation** dialog box. A property operation is a function on a property. You can publish getter and setter methods on the operation.

   The following procedure is the same as you can perform when you select a custom property on the **Operations** tab and click **Add**.

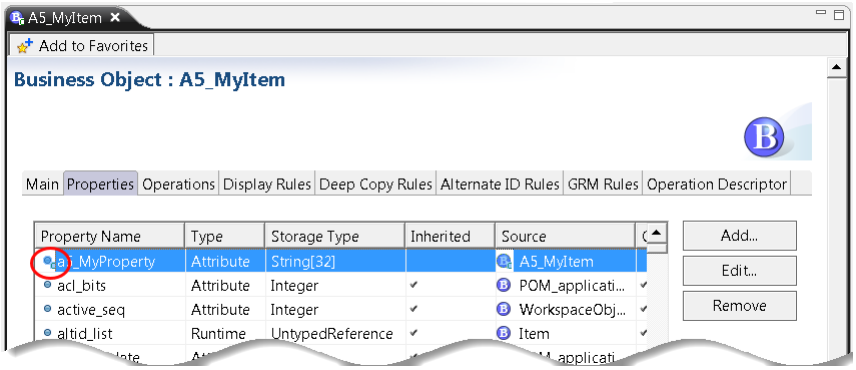
5. If you want to publish getter and setter methods on the property, in the **Property Operation** dialog box, select the **Getter** check box to enable getting the value of the property, and select the **Setter** check box to enable setting the value of the property. Select the following check boxes that apply:

Checking this box	Does this
Overridable?	Allows child business objects to override this method.
Published?	Generates the getter or setter method.
PreCondition?	Places limits before an action.
PreAction?	Executes code before an action.
PostAction?	Executes code after an action.

© 2025 Siemens

When done making changes, click **Finish**.

The new property appears in the properties table. The property name is marked with a **c** to identify it as a custom property.



6. Continue as in *Adding properties to an object* **Step 6**.

Change persistent property attribute definitions

To edit the definition of persistent property attributes, on the **Properties** tab for a business object, select the persistent property and then click **Edit**.

In the dialog box that opens, editable property attribute definitions are shown in white boxes.

Change a non-editable persistent property attribute definition

Certain persistent property attribute definitions cannot be edited once the property has been added. Settings for these attribute definitions are shown in gray boxes. To change these non-editable persistent property attribute definitions, use one of the following procedures, depending on whether the property has been deployed or not.

Has the property been deployed?	Do this
No	Delete the original persistent property and then add a new persistent property with the desired attribute definitions.
Yes	1. From your Teamcenter database, capture all data that you want to retain for the persistent property that you want to edit. 2. In the Business Modeler IDE, delete the persistent property that you want to edit. 3. Deploy the change.

© 2025 Siemens

Has the property been deployed?	Do this
	Warning: Deploying the change can result in loss of data that was contained in the property. Before you deploy your change, ensure that you have captured all data that you want to retain for the persistent property. 4. In the Business Modeler IDE, add a new persistent property with the desired attribute definitions. 5. In the database, re-populate the property with the data you captured in Step 1.

Add a runtime property

Adding a runtime property to a custom business object is similar to adding a persistent property, except that Teamcenter controls the behavior of the property at run time before displaying it to the user. The displayed value of the property is derived each time the property is displayed. Use runtime properties to derive data from one or more pieces of system information (for example, date or time) that are not stored in the Teamcenter database.

To **display custom properties in the end-user interface**, you must use XML rendering style sheets.

1. Open the **New Property** wizard as described in steps 1 through 5 of **Adding properties to an object**.

2. Under **Property Types**, select **Runtime**. Click **Next**.

   The **Runtime Property** dialog box is displayed.

3. Complete the options in the **Runtime Property** dialog box.

For this option	Do this
Name	Type the property name as you want it to appear in the database. The name must be USASCII7 characters only and cannot contain spaces. When you name a new data model object, a prefix from the template is automatically affixed to the name to designate the object as belonging to your organization, for example, **a4_**. The prefix for properties begins with a lowercase letter.
Display Name	Type the name as you want it to appear in the user interface.
Attribute Type	Select the storage type for the attribute. Choose from the following attribute types:

© 2025 Siemens

For this option	Do this
	Boolean Allows two choices to the user (**True** or **False**). **Character** A single character, such as **A**, **B**, **Z**. Note: Attributes with this type only store single-byte characters. To store a multibyte character, use the **String** type. **Date** A calendar date. A form using this format displays a shortcut date selector. For the date attribute type, the earliest date supported is January 2, 1900. **Double** A double-precision floating point decimal number. (For Oracle, the limit for the double property value is 1E-130 to 9E125. For SQL Server the limit is 2.3E-308 to 1.7E308.) **ExternalReference** Points to data outside of Teamcenter. **Integer** An integer without decimals. **LongString** A string of unlimited length. Before Teamcenter 2007.1, the **Note** type was used for unlimited strings. Use the **LongString** type instead. **String** A string of characters. Type the byte length of the string attribute in the associated **String Length** box. For the English alphabet, one character requires one byte. Special characters in Western languages such as French and Portuguese, for example, may require two bytes. For such special characters, and for most characters in multibyte languages such as Chinese, Japanese, and Korean, a field with a string length of 128 can accommodate a maximum of 64 double-byte characters.

© 2025 Siemens

For this option	Do this
	TypedReference Points to a Teamcenter class. Next to the **Reference Business Object** box, click **Browse** and select the class. **UntypedReference** Points to any class of data.
Description	Type a description of the runtime property.
Array Keys	Check the properties that apply: **Array?** Specifies that the attribute is an array of the data of the data type (for example, string). **Unlimited** Indicates that there is no limit on the number of array elements used for the attribute. Siemens Digital Industries Software recommends using **Unlimited** for a variable length array (VLA) especially when you expect a large number of elements. **MaxLength** Specifies the maximum number of array elements allowed for the attribute.
Descriptor Options	Select the options that apply: **Show this property during creation of a Business Object** Adds this property to the dialog box displayed when creating the business object on which the property resides. **Show this property during the Save As operation of a Business Object** Adds this property to the dialog box displayed when performing a **Save As** operation on the business object on which the property resides. If these boxes are checked, the property appears on the **Operation Descriptor tab**. The section does not appear when adding properties to the **Form** business object or one of its children. Properties added to forms are automatically added to the **Operation Descriptor** tab. You cannot delete these properties from the **Operation Descriptor** tab, but you can modify them using the **Visible** and **Required** property constants.

4. Click **Next**.

© 2025 Siemens

The Business Modeler IDE displays the **Runtime Property Operation** dialog box. A property operation is a function on a property. You can publish getter and setter methods on the operation. The following steps are the same as when you select a custom property on the **Operations** tab and click **Add**.

5. If you want to publish getter and setter methods on the property, in the **Property Operation** dialog box, select the **Getter** check box to enable getting the value of the property, and select the **Setter** check box to enable setting the value of the property. Select the following check boxes that apply:

Checking this box	Does this
Overridable?	Allows child business objects to override this method.
Published?	Generates the getter or setter method.
PreCondition?	Places limits before an action.
PreAction?	Executes code before an action.
PostAction?	Executes code after an action.

When done making changes, click **Finish**.

The new property appears in the properties table. The property name is marked with a **c** to identify it as a custom property.

6. Continue as in *Adding properties to an object* **Step 6**.

Add a compound property

You can add a *compound property* to a business object (the display or target object) so that some property defined and residing on another object (the source object) behaves as if it is defined on the display object class.

> Note:
>
> If your intent for the compound property is to display an in-context value or incremental change value on a BOM line object in Structure Manager, follow the procedure **Set a compound property on a BOM line**.

1. Open the **New Property** wizard as described in steps 1 through 5 of **Adding properties to an object**.

2. Under **Property Types**, select **Compound**. Click **Next**.

   The **Compound Property Page** dialog box is displayed.

3. Complete the options in the **Compound Property Page** dialog box.

© 2025 Siemens

For this option	Do this
Name	Type the compound property name. A prefix is automatically affixed to the name to identify the project where the compound property is defined. The name must be USASCII7 characters only and cannot contain spaces.
Display Name	Enter the display name for the compound property. Entering a display name identical to the display name of the referenced property on the source object may assist the end user in recognizing that the same property is displayed on different business objects.
Description	Type a description of the compound property.
ReadOnly	If you do not want users to be able to change the property value on instances of the display business object in Teamcenter, select the **ReadOnly** check box. Warning: During checkout of the display object, the relations **TC_Attaches** and **IMAN_specification** from an ItemRevision to a dataset are deep-copied. Deep-copy is performed for the purpose of restoring changes upon **Cancel Checkout**. If the source attribute of the compound property is from secondary objects that are not deep-copied, and the user saves the changes without checking in the display object and then cancels checkout, the original compound property value is not restored. For this reason, it may be prudent to make the compound property read only.

4. In the **Path** box, define the route from the display business object and property to the source business object and property.

   a. Click the **Add Segment** button.

      The Add Compound Property Segment wizard runs, displaying the **Compound Property Segment** dialog box and the message **Choose a property**.

   b. Select the property on the display business object to hold the compound property, such as **item_master_tag**.

© 2025 Siemens

To filter out the types of properties to display, clear the **Reference**, **Relation**, and **Runtime** check boxes. To limit the list of properties by name, type a string in the box below the property type check boxes.

> Note:
>
> If you have added a relation business object to the **ImanRelation** business object, it does not appear in this list until you add the new business object as a relation property. To add the relation business object as a relation property, do the following.
>
> A. Go back to the **Properties** tab.
> B. Click **Add**.
> C. In the **Property Definition** dialog box, select the **Relation** check box and click **Next**.
> D. In the **Relation Property** dialog box, to the right of the **Relation Business Object** box, click **Browse** and choose the relation business object.
>
> Now you can go back and add this new relation property as a segment on the compound property.

c. Click **Next**.

The **Compound Property Segment** dialog box displays the message **Choose a business object**.

d. Select the business object that is the source for the property.

e. Click **Finish**.

The **Compound Property Page** dialog box appears, showing the path thus far.

To add more segments to the path, in the **Path** box select the last segment and click **Add Segment**.

To replace a segment, select the segment and click **Replace Segment**.

5. To finish the path, click **Add Final Segment**. In the **Compound Property Segment** dialog box, select the property you want to use for the final segment of the compound property and click **Finish**.

The **Compound Property Page** dialog box appears, showing the completed compound property path.

6. Click **Finish**.

The new compound property appears in the property table.

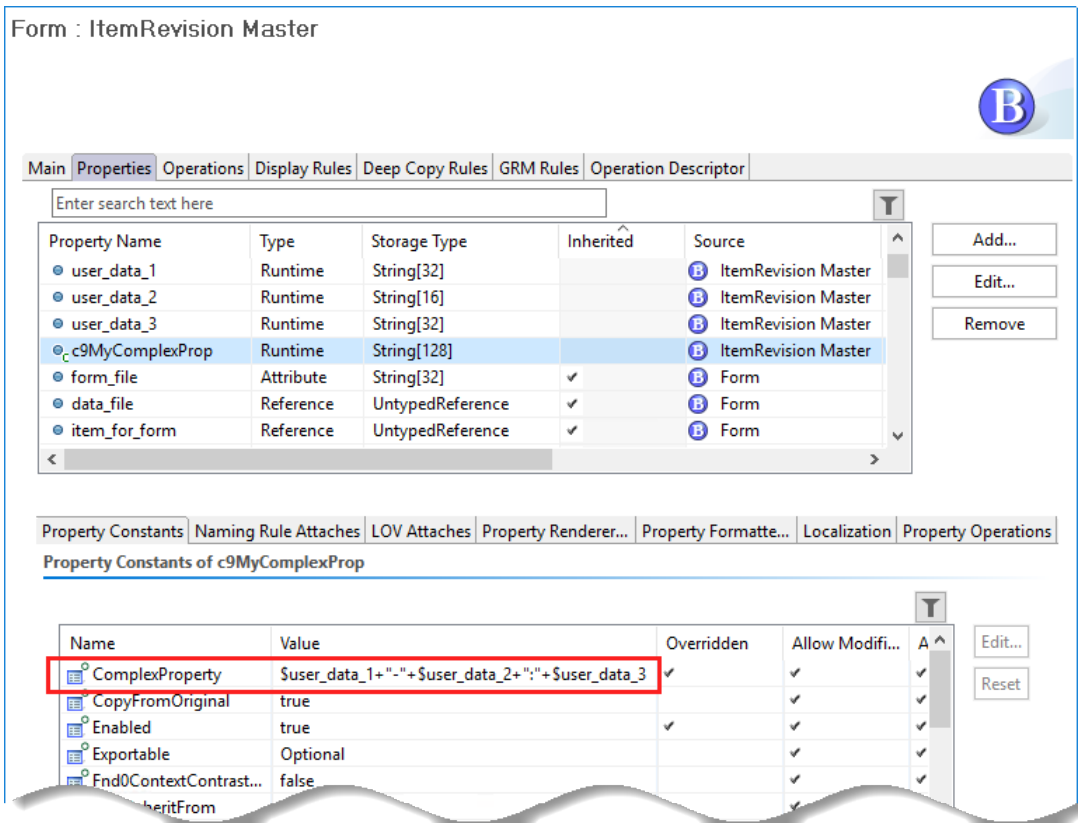
© 2025 Siemens

7. Continue as in *Adding properties to an object* **Step 6**.

To review an example of creating a compound property, see *Workshop 13: Add a compound property*.

Add a complex property

A *complex property* at runtime concatenates string values from other properties on the same business object. For example, a complex property **c9MyComplexProp** that concatenates the values from the properties user_data_1, user_data_2, and user_data_3, separated by a dash and a colon, would look like this on the ItemRevision Master form object in the BMIDE:



1. Open the **New Property** wizard as described in steps 1 through 5 of **Adding properties to an object** .

2. Add a new property with type **Runtime** and storage type **String**.

3. Select the new property and in the **Property Constants** tab for the property, define a concatenated string using the **ComplexProperty** constant.

   - Use the dollar sign '$' preceding the name of other properties.

     Properties must come from the business object.

© 2025 Siemens

- Enclose static strings in double quotes "".
- To separate concatenation clauses, use a plus sign '+'.

For example,

```
$user_data_1+"-"+$user_data_2+":"+$user_data_3
```

4. Continue as in *Adding properties to an object* **Step 6**.

Add a relation property

Relation properties define a relationship between objects. For example, a dataset can be attached to a custom item revision with relations such as a specification, requirement, or reference. You can also **create your own custom relation business objects**.

1. Open the **New Property** wizard as described in steps 1 through 5 of **Adding properties to an object**.

2. Under **Property Types**, select **Relation**. Click **Next**.

   The **Relation Property** dialog box is displayed.

3. Complete the options in the **Relation Property** dialog box.

For this option	Do this
Relation Business Object	Click **Browse** and select the relation business object you want to use. The available business objects are children of the **ImanRelation** business object.
Description	Type a description of the new relation property.
Descriptor Options	Select **Show this property during creation of a Business Object** to add this property to the dialog box displayed when creating the business object on which the property resides. This also adds the property to the **Operation Descriptor tab**.

4. Click **Finish**.

   The new relation property is added to the property table.

5. After you add a property, change characteristics of the property by using **property constants**.

   a. On the **Properties** tab, select the new property in the properties table.

   b. In the **Property Constants** table, select the constant you want to change and click the **Edit** button.

© 2025 Siemens

In addition to making properties enabled, you can use constants to make properties modifiable, exportable, required, a stub property, visible, or to set the initial value.

6. To save the changes to the data model, choose **BMIDE**>**Save Data Model**, or on the main toolbar click **Save Data Model** .

7. Deploy your changes to a test server. On the menu bar, choose **BMIDE**>**Deploy Template**, or select the project and on the main toolbar click **Deploy Template** .

8. After deployment, test your new properties in the Teamcenter rich client.

   For example, if you added a relation property to a custom child of the **ItemRevision** business object, in the My Teamcenter application, select an instance of that kind of item revision and attach a dataset to it using the new relation property by using the **Edit**>**Copy** and **Edit**>**Paste...** options.

Note:

If you want to make attached objects of the relation property display under the target business object, you must modify the *<Type_Name>***_DefaultChildProperties** preference to include your new relation property. For example, if you added a relation property to the **ItemRevision** business object, include the new relation property in the **ItemRevision_DefaultChildProperties** preference.

To set the relation property as a default paste relation, you must add it to the *business-object-name***_default_relation** preference. For example, add it to the **ItemRevision_default_relation** preference. To change preferences, in the rich client, choose **Edit**>**Options**, and click the **Search** link at the bottom of the **Options** dialog box.

The **NoChildrenAllowed** registry entry in the **com.teamcenter.rac.common\common.properties** file controls which type shows children below it (that is, supports expansion). If you want to see relations for any of the following types, you must override the registry entry and remove the type for which you want to show relations. By default, this registry entry prevents showing relations for the following types:

```
NoChildrenAllowed=com.teamcenter.rac.kernel.TCComponentForm,
com.teamcenter.rac.kernel.TCComponentBOMView,
com.teamcenter.rac.kernel.TCComponentQuery,
com.teamcenter.rac.kernel.TCComponentProcess,
com.teamcenter.rac.kernel.TCComponentReleaseStatus,
com.teamcenter.rac.kernel.TCComponentRevisionRule,
com.teamcenter.rac.kernel.TCComponentSite,
com.teamcenter.rac.kernel.TCComponentPerson,
com.teamcenter.rac.kernel.TCComponentUser,
com.teamcenter.rac.kernel.TCComponentGroupMember,
com.teamcenter.rac.kernel.TCCcomponentListOfValues,
com.teamcenter.rac.kernel.TCComponentReportDesign,
com.teamcenter.rac.kernel.TCComponentAssemblyArrangement,
com.teamcenter.rac.kernel.TCComponentReportDefinition
```

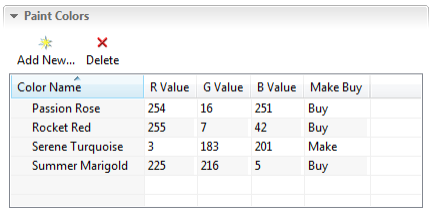

© 2025 Siemens

Add a table property

A table property displays properties in a table format. Each column in the table is defined as a property. You can include persistent and runtime properties in such a table, but not compound or relation properties. You can attach lists of values (LOVs) to properties in a table as well as use property constants. You can add a table property to a WorkspaceObject, or to any of its sub business objects.

A table property can only be displayed on the **Summary** tab of the business object. To display the table, you must add the property to the summary **XML rendering style sheet** of the business object.

Suppose you want to display the paint colors that are available for use with a custom item revision. The table may appear like this in the rich client:



Using this as an example, perform the following steps to add the table property:

1. Open the **New Property** wizard as described in steps 1 through 5 of **Adding properties to an object** .

2. Under **Property Types**, select **Table**. Click **Next**.

   The **Runtime Property** dialog box is displayed.

3. Complete the options in the **Table Property** dialog box.

For this option	Do this
Name	Type the property name as you want it to appear in the database. The name must be USASCII7 characters only and cannot contain spaces. When you name a new data model object, a prefix from the template is automatically affixed to the name to designate the object as belonging to your organization, for example, **a4_**. The prefix for properties begins with a lowercase letter.
Display Name	Type the name as you want it to appear in the user interface.
Description	Type a description of the table property.
Table Row Business Object	Click **Browse** to use an existing table row type, or click **New** to create a new table row type.

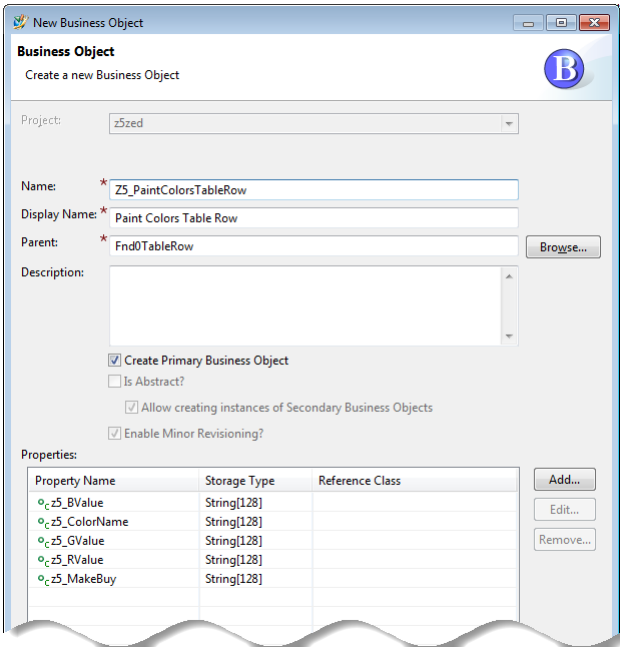
© 2025 Siemens

4. For our example, in the **Table Row Business Object** box click **New**.

   The **New Business Object** dialog box is displayed.

   a. Complete the options for the new table row business object.

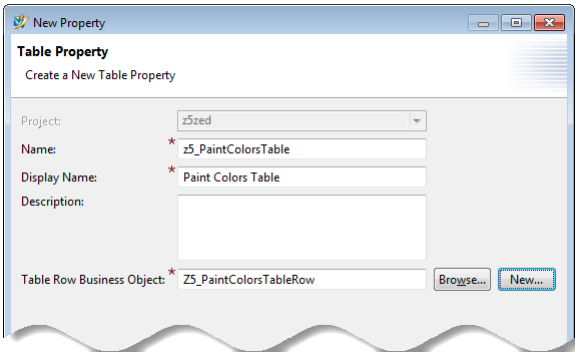
For this option	Do this
Name	Type the property name as you want it to appear in the database. The name must be USASCII7 characters only and cannot contain spaces. When you name a new data model object, a prefix from the template is automatically affixed to the name to designate the object as belonging to your organization, for example, **a4_**. The prefix for properties begins with a lowercase letter.
Display Name	Type the name as you want it to appear in the user interface.
Parent	If needed for the type of row data you want to use, click **Browse** to select a different parent type.
Description	Type a description of the runtime property.
Properties	To add properties to appear on the table row, click **Add**. This is similar to **adding persistent properties**.



   b. Review the entries and click **Finish**.

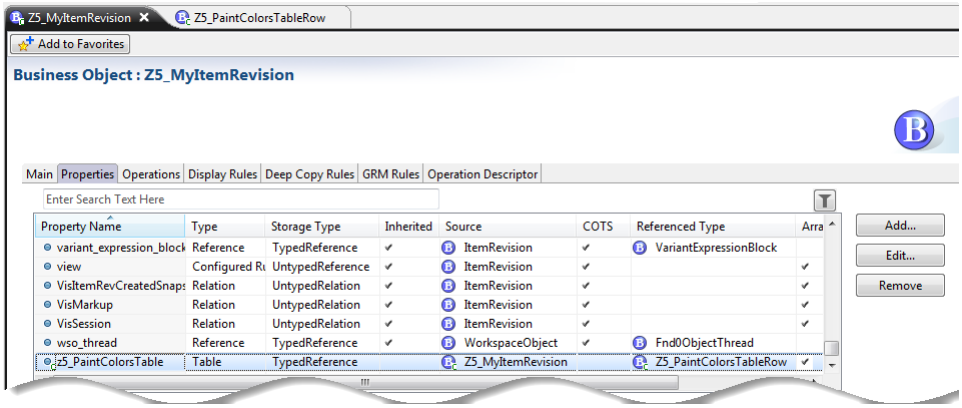
   The completed table property dialog box is displayed.

© 2025 Siemens



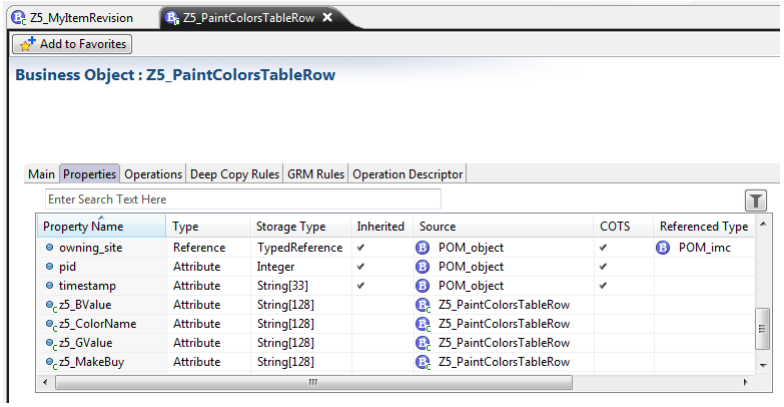
5. Click **Finish** in the **Table Property** dialog box.

   The new table property is displayed on the business object.



   At this point, you can open the new table business object to see the properties that are displayed in the new table, select properties to attach LOVs, or to change property constants values.

6. In our example, attach the **Make Buy** classic LOV to the **z5_MakeBuy** property. This allows the end user to indicate whether it is necessary to make or buy the paint color.



© 2025 Siemens

7. To save the changes to the data model, choose **BMIDE**>**Save Data Model**, or on the main toolbar click **Save Data Model** .

8. Deploy your changes to a test server. On the menu bar, choose **BMIDE**>**Deploy Template**, or select the project and on the main toolbar click **Deploy Template** .

9. After deployment, test your new table property in the rich client by adding the table property to the **Summary** tab of the business object that holds the table property:

   a. Create a custom summary style sheet for the business object that holds the table property.

   b. Add the table row properties to the custom summary style sheet.

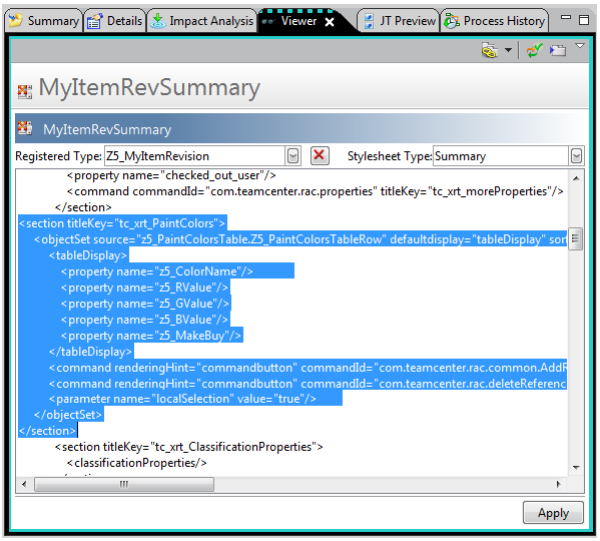
      For our example, add the following text to the style sheet and click **Apply**:

```
<section titleKey="tc_xrt_PaintColors">
     <objectSet source="z5_PaintColorsTable.Z5_PaintColorsTableRow"
defaultdisplay="tableDisplay"
 sortby="z5_ColorName" sortdirection="ascending">
          <tableDisplay>
              <property name="z5_ColorName"/>
              <property name="z5_RValue"/>
              <property name="z5_GValue"/>
              <property name="z5_BValue"/>
              <property name="z5_MakeBuy"/>
          </tableDisplay>
          <command renderingHint="commandbutton"
commandId="com.teamcenter.rac.common.AddReference"
actionKey="newBusinessObjectContextualAction"/>
          <command renderingHint="commandbutton"
commandId="com.teamcenter.rac.deleteReference"
actionKey="deleteAction"/>
          <parameter name="localSelection" value="true"/>
     </objectSet>
</section>
```

      Note that the **objectSet source** is defined as the *table-property.table-row-business-object* (for example, **z5_PaintColorsTable.Z5_PaintColorsTableRow**).

      The style sheet looks like the following:

© 2025 Siemens



Tip:

To display the table property in Active Workspace, use the **tableProperty** tag (instead of the **objectSet** tag) in your custom Active Workspace summary style sheet, for example:

```
<tableProperty name="z5_PaintColorsTable" sortby="z5_ColorName"
sortdirection="ascending">
  <property name="z5_ColorName"/>
  <property name="z5_RValue"/>
  <property name="z5_GValue"/>
  <property name="z5_BValue"/>
  <property name="z5_MakeBuy"/>
</tableProperty>
```

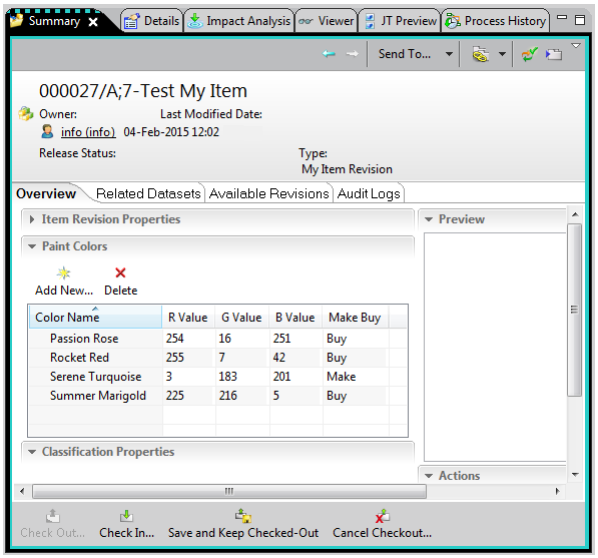
10. To display the tab title correctly, place the localized value of the table title (for example, **"tc_xrt_PaintColors"**) in the following file and restart your system for the change to take effect:

    *TC_ROOT***lang\textserver*****lang_locale***tc_xrt_text_locale.xml**

11. Select the business object that contains the table property. For our example, select the custom item revision and click the **Summary** tab.

    The table property is displayed in the **Summary** tab.

© 2025 Siemens



To enable the **Add New** button and **Delete** button, check out the object. After the object is checked out, you can double-click a row to edit its properties.

Supporting the table in PLM XML import and export

To support the table property in PLM XML import and export, add a property set clause to export the table. In addition, add a property set clause for each column.

Example:

```
CLASS.MyCustomType:PROPERTY.z5_PaintColorsTable:DO
CLASS.Z5_PaintColorsTableRow:PROPERTY.z5_ColorName:DO
CLASS.Z5_PaintColorsTableRow:PROPERTY.z5_RValue:DO
CLASS.Z5_PaintColorsTableRow:PROPERTY.z5_GValue:DO
CLASS.Z5_PaintColorsTableRow:PROPERTY.z5_BValue:DO
CLASS.Z5_PaintColorsTableRow:PROPERTY.z5_MakeBuy:DO
```

Supporting the table in the rich client.

If you intend to print your table property using the rich client's property viewer and there are more than two properties defined on the table row object, you must add a value to the **DisplayName** business object property for that object. The value must use the following syntax:

```
$prop1+":"+$prop2+…..+$propn
```

Example:

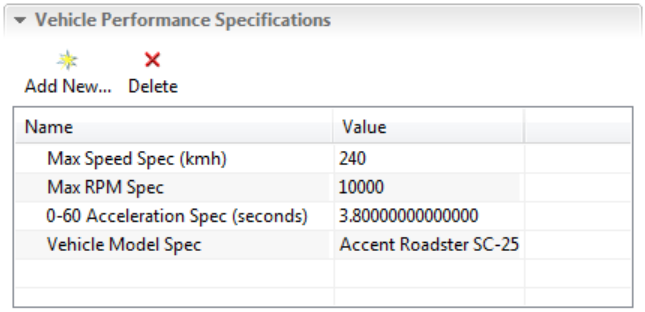
To print the entire paint colors table property, you must set the **DisplayName** constant on the **Z5_PaintColorsTableRow** object as follows.

© 2025 Siemens

```
$z5_ColorName+":"+$z5_RValue+":"+$z5_GValue+":"+$z5_BValue+":"+
$z5_MakeBuy
```

Add a name-value property

Name-value properties display name-value pairs in a tabular format to represent ad hoc characteristics not defined in the persistent properties for the business object. For example, performance specifications for a sports car might appear like this in the rich client:



Name-value properties only exist in the scope of a specific object and are not meant to substitute for a persistent property in any way. The following chart shows the known limitations of name-value properties.

	Persistent properties	Name-value properties
Scope	Global	Local to object
Inheritable	Yes	No
Supports default values	Yes	No
Supports upper and lower bound	Yes	No
Supports property constants	Yes	No
Supports compound properties	Yes	No
Supports localization	Yes	No

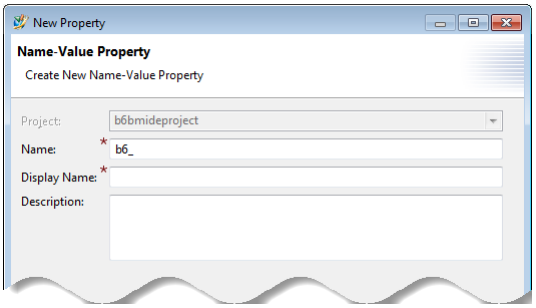
End users can add, edit, or remove rows of names and values in the table. Each row in the table is unique by name and can contain different kinds of primitive data, such as Boolean, date, double, integer, and string. In other words, a name-value property table can show rows of heterogeneous data. (In a **table property**, the type of data in each row is of the same type.) A name-value property can only be displayed on the **Summary** tab of the business object. To display the table, you must add the property to the summary **XML rendering style sheet** of the business object.

1. Open the **New Property** wizard as described in steps 1 through 5 of **Adding properties to an object** .

2. Under **Property Types**, select **Name-Value**. Click **Next**.

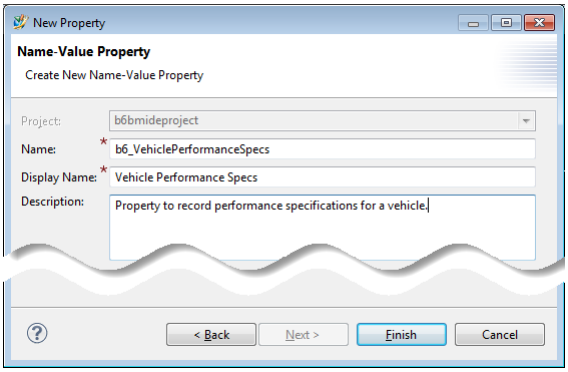
© 2025 Siemens

The **Name-Value Property** dialog box is displayed.



3. Complete the options in the **Name-Value Property** dialog box.

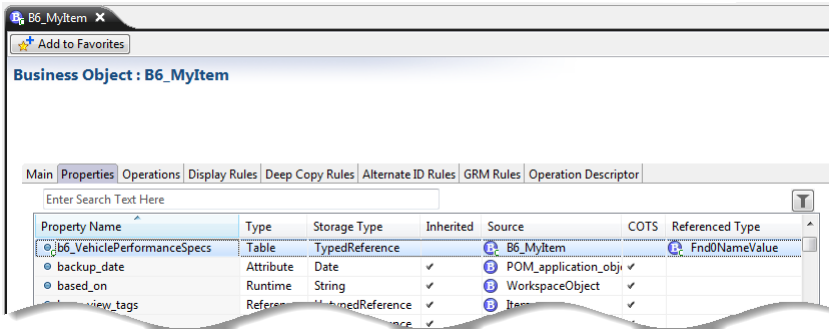
For this option	Do this
Name	Type the property name as you want it to appear in the database. The name must be USASCII7 characters only and cannot contain spaces. When you name a new data model object, a prefix from the template is automatically affixed to the name to designate the object as belonging to your organization, for example, **a4_**. The prefix for properties begins with a lowercase letter.
Display Name	Type the name as you want it to appear in the user interface.
Description	Type a description of the table property.
Table Row Business Object	Click **Browse** to use an existing table row type, or click **New** to create a new table row type.



4. Click **Finish**.

The new name-value property is displayed on the business object.

© 2025 Siemens



5. Save your changes to the data model and deploy your changes to a test server.

6. After deployment, test your new name-value property in the Teamcenter rich client by adding the name-value property to the **Summary** tab of the business object that holds the name-value property:

   a. Create a custom summary style sheet for the business object that holds the name-value property.

   b. Add the name-value property to the custom summary style sheet.

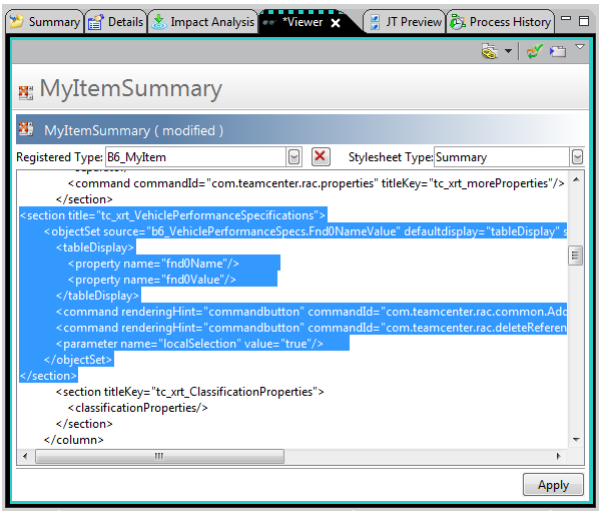
      For our example, add the following text to the style sheet and click **Apply**:

```
<section title="tc_xrt_VehiclePerformanceSpecifications">
  <objectSet source="b6_VehiclePerformanceSpecs.Fnd0NameValue"
defaultdisplay="tableDisplay"
sortby=" fnd0Name" sortdirection="ascending">
      <tableDisplay>
          <property name="fnd0Name"/>
          <property name="fnd0Value"/>
      </tableDisplay>
      <command renderingHint="commandbutton"
commandId="com.teamcenter.rac.common.AddReference"/>
      <command renderingHint="commandbutton"
commandId="com.teamcenter.rac.deleteReference"/>
      <parameter name="localSelection" value="true"/>
  </objectSet>
</section>
```

      Note that the **objectSet source** is defined as the *name-value-property*.**Fnd0NameValue** (for example, **b6_VehiclePerformanceSpecs.Fnd0NameValue**). **Fnd0NameValue** is the business object that defines the table, and the **fnd0Name** and **fnd0Value** properties define the columns on the table.

      The style sheet looks like the following:

© 2025 Siemens

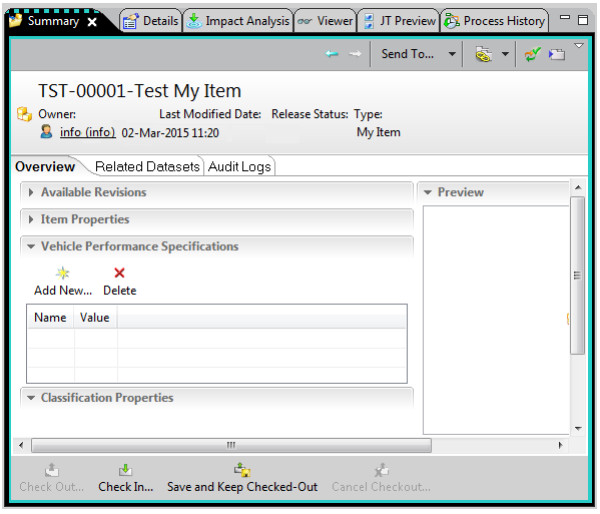


c. To display the tab title correctly, place the localized value of the table title (for example, **"tc_xrt_VehiclePerformanceSpecifications"**) in the following file and restart your system for the change to take effect:

*TC_ROOT***lang\textserver***lang_locale****tc_xrt_text_locale.xml**

d. Select the business object that contains the name-value property. For our example, select the custom item and click the **Summary** tab.

The name-value property is displayed in the **Summary** tab.



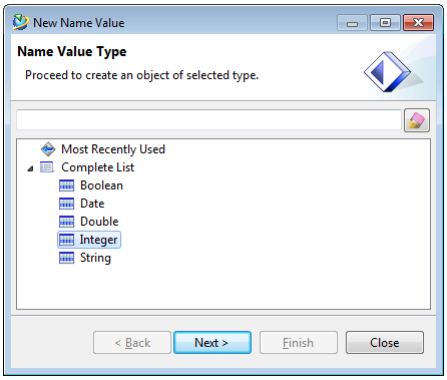
e. Check out the object to enable the **Add New** and **Delete** buttons.

f. Click **Add New** to add a new table row.

The **Name-Value Type** dialog box is displayed.

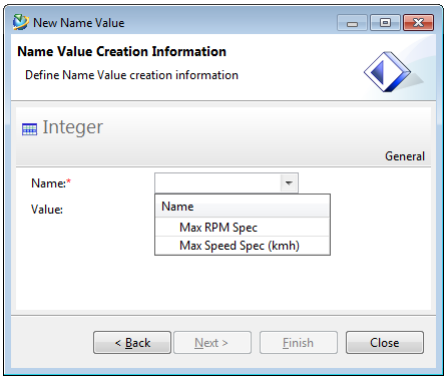
© 2025 Siemens

g. Select the type of data to add. For example, to add data whose values are whole numbers, select **Integer**.



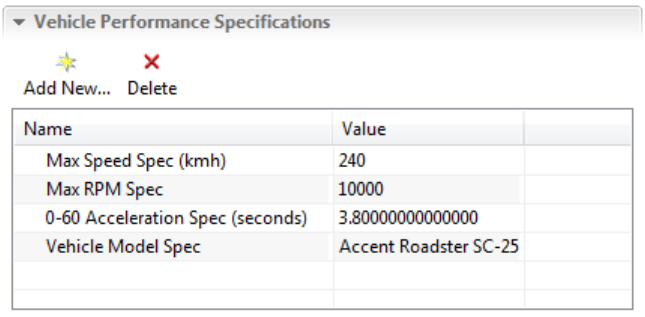
h. Type a new name in the **Name** box and type a value in the **Value** box.

> Tip:
>
> If names of that data type (for example, integer) have already been entered for this property, you can click the arrow in the **Name** box to select from a list.



i. Continue to enter names and values to the table. You can enter any of the available data types to the table: Boolean, date, double, integer, or string. When done, click **Finish**.

The name-value data is displayed in the table.



© 2025 Siemens

j. Check in the object to commit the name-value data to the database. When the object is checked in, the **Add New** button and **Delete** button are unavailable.

Property constants

Setting property behavior with property constants

You can use property constants to change the characteristics of business object properties. You can attach property constants to make properties enabled, modifiable, exportable, required, a stub property, visible, or to set the initial value. You can also create your own property constants to apply your own characteristics.

1. In the **Business Objects** folder, locate the business object that has the property whose characteristics you want to change.

2. Right-click the business object, choose **Open**, and click the **Properties** tab in the resulting view.

   The properties of the business object appear in a table.

3. In the **Properties** table, select the property whose characteristics you want to change.

4. On the **Property Constants** tab, select the property constant whose value you want to change for the property.

5. Click the **Edit** button.

   The system displays the **Modify Property Constant** dialog box.

6. Type an entry in the **Value** box, or select a value using the dropdown menu, or select the check box. Valid values are dependent on the property constant.

7. Click **Finish**.

   The new value displays in the **Value** column on the **Property Constants** table.

   > Note:
   >
   > If you want to set the value back to its previous value, click the **Reset** button. This resets the value and reloads the data model.

8. To save the changes to the data model, choose **BMIDE** > **Save Data Model**, or on the main toolbar click **Save Data Model**  .

© 2025 Siemens

Property constant details

Details of a property constant are displayed in the **BMIDE** edit view for the constant. You can include property constant details in BMIDE data model reports.

Create a property constant

Property constants provide default values to business object properties. Because these constants are attached to properties, they are inherited and can be overridden in the hierarchy.

You can define a property constant to have a specific scope so that it is available only on certain properties on certain business objects. This ensures that server API can retrieve the value properly on just those properties. You can also define a property constant to be available on all business objects or properties by using an asterisk (*) when you set the scope. In this way, you can set the scope to be as wide or narrow as you want.

You can create property constants for a number of situations. Some examples are:

- Set whether the property is required.
- Set the initial value of the property.

When you create a new constant, you must also add the code on the server to return the constant's value to the caller, so the caller can branch the business logic based on the returned value. The server side code can use the following published ITK to retrieve a property constant value:

```
int CONSTANTS_get_property_constant_value(
       const char*     constant_name,      /* <I> */
       const char*     type_name,          /* <I> */
       const char*     property_name,      /* <I> */
       char**          value               /* <OF> */
       );
```

Once a constant is set, it can be overridden by another template. The rule is that the last template that gets installed determines the constant value that is used.

1. Choose one of these methods:

   - On the menu bar, choose **BMIDE→New Model Element**, type **Property Constants** in the **Wizards** box, and click **Next**.
   - Open the **Extensions\Constants** folders, right-click the **Property Constants** folder, and choose **New Property Constants**.

   The New Property Constants wizard runs.

2. Perform the following steps in the **Create Property Constant** dialog box:

© 2025 Siemens

a. The **Project** box defaults to the already-selected project.

b. In the **Name** box, type the name you want to assign to the new constant in the database.

When you name a new data model object, a prefix from the template is automatically affixed to the name to designate the object as belonging to your organization, for example, **A4_**.

c. In the **Description** box, type an explanation of how the constant is to be used.

d. Click the **Add** button to the right of the **Scope** box.

Perform the following steps in the **Business Object and Property Scope** dialog box:

A. Click the **Browse** button to the right of the **Business Object Scope** box and choose the business object to apply the constant to. Remember that property constants are inherited by children business objects. If you want the constant to apply to all business objects, type an asterisk (*****).

B. Click the **Browse** button to the right of the **Property Scope** box and choose the property to apply the constant to. If you want the constant to apply to all properties, type an asterisk (*****).

C. Click **Finish**.

The business object and property appear in the **Scope** box separated by a period (**.**). An asterisk indicates the constant applies to all business objects or properties. For example:

- ***.***

  The constant can be applied to all properties on all business objects.

- ***.object_desc**

  The constant can be applied to the **object_desc** property on all business objects

- **Item.***

  The constant can be applied to all properties on the **Item** business object.

- **Item.object_des**

  The constant can be applied only to the **object_desc** property on the **Item** business object and its children.

e. If desired, click the **Add** button to the right of the **Scope** box to narrow the scope further.

© 2025 Siemens

f. Click the arrow in the **Data Type** box to select one of the following:

- **Boolean**

  Allows two choices to the user (**True** or **False**).

- **String**

  Indicates that the value is a text string.

- **List**

  Contains a list of values.

g. If you selected the **List** data type, a **Values** table appears. Click the **Add** button to add values to the list:

A. In the **Value** box, type a value for the list.

B. Select **Secured** to prevent the selected value from being overridden by another template.

C. Click **Finish**.

h. In the **Default Value** box, enter the initial value of the constant. Entry differs depending on the data type you previously chose:

- If you selected the **String** data type, type in the default value.
- If you selected the **Boolean** data type, click the arrow to select **True** or **False** for the default value.
- If you selected the **List** data type, click **Browse** to select a value from the available ones on the list.

> Note:
>
> If the selected default value is marked as **Secured** then this value cannot be overridden by any other template.

i. Use the following check boxes to enable the constant for **live updates**:

- **Allow Live Updates?**

  Select this check box to specify that the constant definition itself can be updated live.

- **Allow Live Updates to the Constant Override?**

© 2025 Siemens

Select this check box to specify that the constant attachment (override) can be updated. The constant attachment contains the value set for the constant.

In a live updates environment, you can deploy changes for custom constants, but you cannot deploy changes for COTS constants.

j. Click **Finish**.

The new constant appears under the **Property Constants** folder.

> Note:
>
> You can also see the property constant on the business object it is applied to. Right-click the business object, choose **Open**, and click the **Properties** tab. The constant appears on the **Property Constants** tab.

3. To save the changes to the data model, choose **BMIDE→Save Data Model**, or click the **Save Data Model** button on the main toolbar.

4. Deploy your changes to the server. Choose **BMIDE→Deploy Template** on the menu bar, or select the project and click the **Deploy Template** button on the main toolbar.

5. To verify the constant on the server, run the **getpropertyconstantvalue** utility. This utility helps test the value of a property constant on a particular business object and property in the database. The utility accepts the name of a property constant, business object, and property, and outputs the value of the constant if present.

Change the value of a custom property constant

Property constants provide default values to business object properties. Because these constants are attached to properties, they are inherited and can be overridden in the hierarchy. You can change the value of a custom property constant and the properties it applies to (its properties scope).

1. In the **Extensions** folder, open the **Constants\Property Constants** folders.

2. Right-click the custom property constant and choose **Open**. (You can only change custom property constants.)

   The **Property Constant** editor displays the constant.

3. To change the properties that the custom constant applies to, click the **Add** or **Remove** buttons to the right of the **Property Scope** box.

4. Use the **Default Value** box to change the value of the constant. How you change the value depends on its type:

© 2025 Siemens

- String

  Type a new value.

- Boolean

  Select **True** or **False**.

- List

  Click the **Browse** button to the right of the **Default Value** box to select another value from the list. Click the **Add** and **Remove** buttons to the right of the **Default Value** box to add values to the list, or to remove values.

5. To save the changes to the data model, choose **BMIDE→Save Data Model**, or click the **Save Data Model** button on the main toolbar.

   The constant change is saved in the active extension file.

Compound properties

Introduction to compound properties

A *compound property* on a business object (the *display* or *target* object) effectively adds some property defined on another object (the *source* object) to the display object so that the source property behaves on the display object as if it is defined on the display object class.

A compound property defines a path between the display object and source object consisting of one or more reference relations, GRM relations, or a combination of the two. The path is used to traverse between the display object and the source object. You can add compound properties to COTS and custom business objects.

> Note:
>
> While a compound property enables you to add a property to a business object without writing custom code or using runtime properties, a compound property is not a replacement for runtime property functionality.

Compound property characteristics

A compound property on an object displays the following characteristics:

- Default **SET** and **ASK** methods.
- The property inherits access rules from the source object.

© 2025 Siemens

- The protection (read/write permissions) of a compound property on the display object is the same as that of the source object. You can modify the protection of the compound property from the display object.
- If the source property is modifiable, the compound property is also modifiable. However, if a **ReadOnly** flag is set on the compound property, the compound property is a non-modifiable property even if the source property is modifiable.
- The compound property appears disabled on the display object if the property on the source object is not modifiable.
- The name of a compound property and the UI display name can be different from the property name and property UI display name on the source object.
- The value type of a compound property is the same as that of its source property. For example, if the value type of the source property is **PROP_string**, the value type of the compound property is also **PROP_string**.
- If the source property is a variable length array (VLA) the compound property is also a VLA.
- If a list of values (LOV) is attached to the source property, the same LOV is attached to the compound property.
- If the user does not have write privileges to the object on which the compound property exists, the compound property is not modifiable.
- If the user does not have write privileges to the object on which the source property exists, the user cannot modify the value of the compound property.
- To obtain the values of the compound property, the system traverses the object hierarchy specified in the compound property. If the system fails to reach the source property while traversing the object hierarchy, the value of the compound property cannot be retrieved and an error message is displayed. The value is displayed as a blank box in the Teamcenter rich client.

Modify legacy compound properties

Some of the compound properties created prior to Engineering Process Management 9.1.2.8 are no longer supported because they do not have the proper structure of *business_object.property* in their path. These compound properties must be modified to have the proper structure.

Compound properties were formerly created and maintained by the Business Modeler application in Engineering Process Management, and are now maintained in the Business Modeler IDE in Teamcenter. The current run-time model assumes that the last token in the object hierarchy is the source property, and the business object in the last token is the source type. If a customer has the compound properties defined prior to Engineering Process Management 9.1.2.8 that have reference properties in the object hierarchy (path to source), the Business Modeler IDE loads those compound properties and displays a warning to replace the object hierarchy.

© 2025 Siemens

Perform the following steps to correct the problem:

1. In the **Business Objects** folder, double-click a business object that has legacy compound properties.

2. Click the **Properties** tab.

3. Find a legacy compound property.

   You can locate compound properties because are identified by **Compound** in the **Type** column of the table.

4. Select a legacy compound property on the **Properties** tab and choose **Edit**.

   The Modify Property wizard runs.

5. A warning message appears stating that the property is not structured properly. Select the segment that is incorrect and click the **Replace Segment** button. Because incorrectly structured compound properties are missing the business object in the path, select the correct business object to place on the segment.

6. When you are done fixing the segments, click **Finish**.

Set a compound property on a BOM line

If you want to set a compound property on a BOM line object in Structure Manager that displays an in-context value or incremental change value, you must use the **BOMLineAbsOccCompProperties** global constant. You can also create a compound property directly on a BOM line object, but it does not display the value of in-context edits (that is, the absolute occurrence value) or incremental change edits of the source property.

1. Set a value on the **BOMLineAbsOccCompProperties** global constant to create a compound property for use on a BOM line, for example:

   ```
   FORM::IMAN_specification::BVRSyncInfo::PROPERTY::BVRSyncInfo::
   last_sync_date::Absocc Last Sync Date
   ```

   This sample compound property displays the value of the **last_sync_date** property; **Absocc Last Sync Date** is the display name of the property.

2. Save the data model and deploy to the rich client.

3. To verify the compound property, perform the following steps in the rich client:

   a. Create a simple structure and send it to Structure Manager.

© 2025 Siemens

b. Right-click the top line and select **Set In Context**.

c. Click the **Show/Hide Data Panel** button in the toolbar.

d. Select the child object in the structure and choose **File→New→Form**. Do this in context mode because the expectation is to display it only when the parent is loaded.

   The form is created as an attachment to item revision with the specification relation. The **Context Line** column in the **Attachments** tab of the data pane shows the context of the created form.

e. Open the form and edit the attribute values.

f. In Structure Manager, right-click the column headers, choose **Insert Columns**, and select the property you added using the global constant. The display name is the last item in the constant, for example, **Absocc Last Sync Date**.

   The column is added, and the value for that property is displayed in the column.

Display real-time RAC property values

When Teamcenter properties do not update real-time in the rich client, you must add the property in the object property policy.

For example, add the property in the *TC_DATA***soa\policies\RACBase.xml** property policy file. If you have many properties to update the object property policy, you should add your properties in a new file. Include your filename in the **RACBase.xml** file, for example:

The top portion of the `RACBase.xml` file:

```
<?xml version="1.0" encoding="UTF-8"?>
<ObjectPropertyPolicy xmlns="http://teamcenter.com/Schemas/Soa/
ObjectPropertyPolicy" >
   <Include file="site/RACSite.xml" />
   <Include file="site/myRACSite.xml" />       <!-- Include my policy
file  -->
 .
 .
 .
</ObjectPropertyPolicy>
```

The `site\myRACSite.xml` file:

```
<?xml version="1.0" encoding="UTF-8"?>
<ObjectPropertyPolicy xmlns="http://teamcenter.com/Schemas/Soa/
ObjectPropertyPolicy" >
```


© 2025 Siemens

```
<!-- Object Property Policy for DocumentRevision.  -->

   <ObjectType name="DocumentRevision">
        <Property name="b5_dp1"/>
        <Property name="b5_dp2"/>
   </ObjectType>

</ObjectPropertyPolicy>
```

This example adds two compound properties to the **DocumentRevision** business object. To display the table, you must add the property to the summary **XML rendering style sheet** of the business object.

Atrribute groups

Introduction to attribute groups

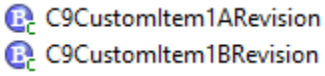
An attribute group is a collection of persistent database attributes stored in a class independent from item and item revision types. By itself, this class can not be used by your users to store information, and its associated business object cannot be instantiated nor displayed in the UI. However, when you attach an attribute group to a business object like an item or an item revision, the properties of the attribute group are displayed on that business object as if they were defined there.

Attribute groups share the limitations and benefits of **compound properties**.

> Note:
>
> For brevity, when this topic mentions *revisions* it means *item revision types*. For example, you cannot create an item revision type directly, so when something like "create a revision" is mentioned, you must **create an item type** instead, and the corresponding item revision type will be created automatically. The principles of property creation discussed here also apply to items, but it is less common to create properties on items than on item revisions.

Traditional properties

The traditional method of adding properties is to create them directly on the revision. This allows you to customize your revisions with the exact properties you want on each. If you create the following revisions, you can create a completely different set of properties on each of them without any issues.

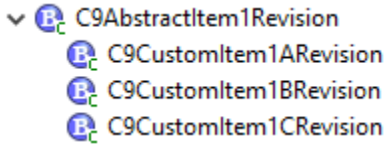


The problem with this is when you want both revision to share a property.

© 2025 Siemens

Inherited properties

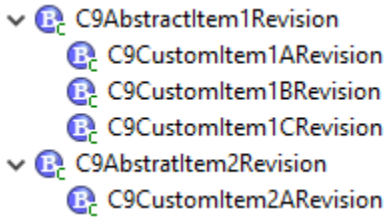
One feature of the data model is that properties exhibit inheritance. If you have created an abstract revision to hold your instantiable revisions as follows, then creating a property on the abstract revision causes the three child revisions to each inherit that property. You can still create separate properties on each individual revision, giving you the flexibility to have the same properties on all three as well as each revision having separate properties.



The problem with this is when you want only two of the three revision to share a property.

Attribute groups

Properties can be created on a separate business object, unrelated to items or item revisions. This group of properties can then be associated with an item or revision, giving it those properties as if they were **compound properties**. You can associate this group with as many items or item revision types as you want.



> Note:
>
> If you assign an attribute group to an item, those properties will also be displayed on the item revision.

Attribute group process

Create a group of properties

To create an attribute group, you must create a subtype of one of two business objects.

Subtype this business object...	to associate your properties with...
Fnd0AbsExtdItemAttrs	Item types.
Fnd0AbsExtdItemRevAttrs	Item revision types.

© 2025 Siemens

Associate the group to an item or item revision

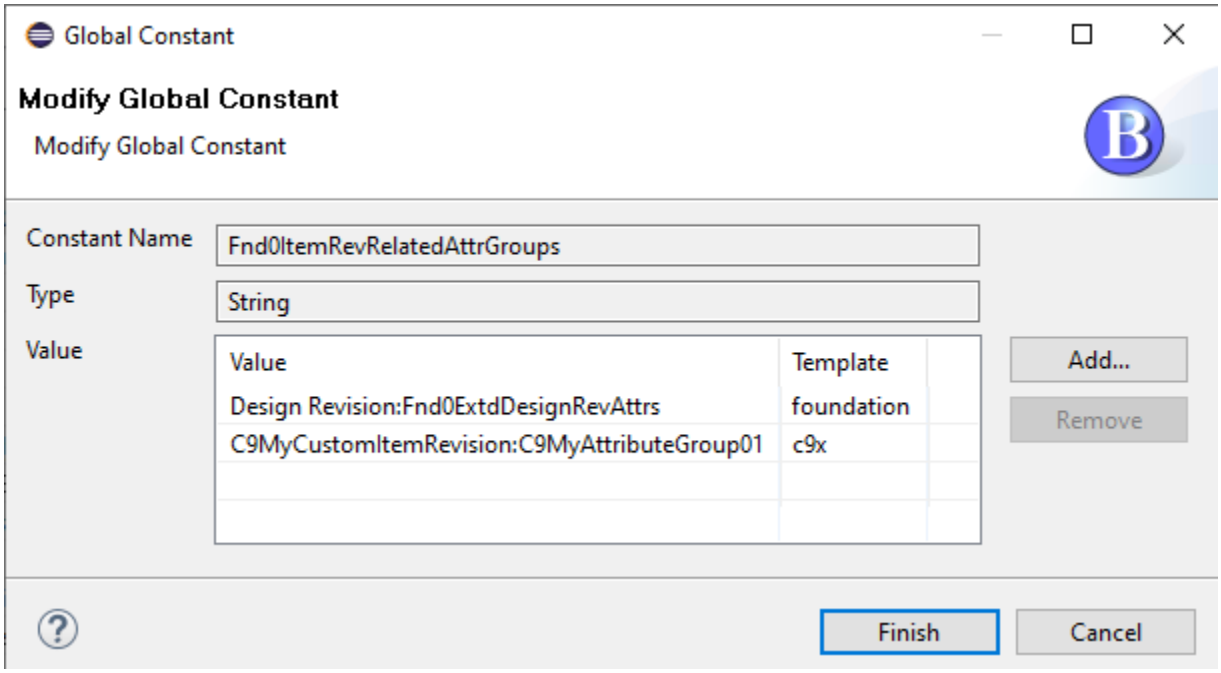
Use the **Global Constants Editor** to add a value to the global constant **Fnd0ItemRelatedAttrGroups** or **Fnd0ItemRevRelatedAttrGroups** depending on whether the attributes will associate with an item type or an item revision type. This value must consist of the name of the item or item revision type followed by the name of your attribute class you just created. These are separated by a colon.

```
{ItemRevision type name}:{Fnd0AbsExtdItemRevAttrs subtype name}

{Item type name}:{Fnd0AbsExtdItemAttrs subtype name}
```

Example:

In this example, the item revision type to receive the attributes is **C9MyCustomItemRevision**. Since this is an item revision, the group attribute **C9MyAttributeGroup01** is a subtype of **Fnd0AbsExtdItemRevAttrs**, and the global constant modified is **Fnd0ItemRevRelatedAttrGroups**.



Deploy and test

Once you have created the attribute group and assigned it to an item or item revision, then you must deploy your BMIDE template. Your new attributes will be available on item or item revision you have assigned the attribute group to. Following are some other things you may need to know:

- To make your new properties visible you must modify style sheets.
- These properties have all of the compound property characteristics.

© 2025 Siemens

- To display your properties in a structure, use following dynamic compound property syntax.

  REF(awb0Archetype,ItemRevision).REF(items_tag, Item).prop_name.

Create an attribute group

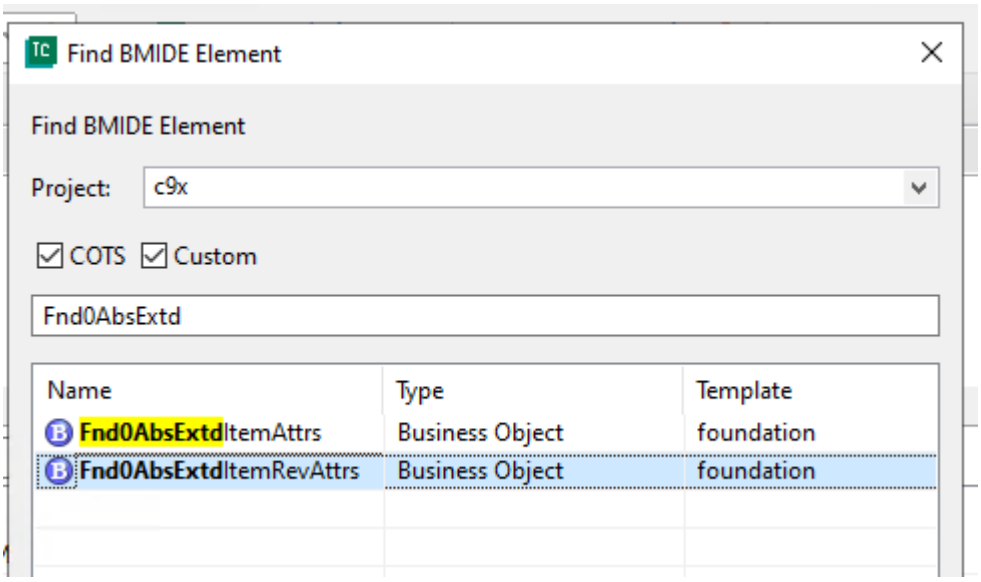
Create a new class of attributes which you can assigned to various business objects.

Prerequisites

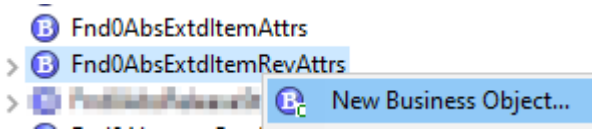
You must have a BMIDE template project open.

Procedure

1. Locate the **Fnd0AbsExtdItemRevAttrs** business object.



2. Right-click on the **Fnd0AbsExtdItemRevAttrs** business object and select **New Business Object...**.

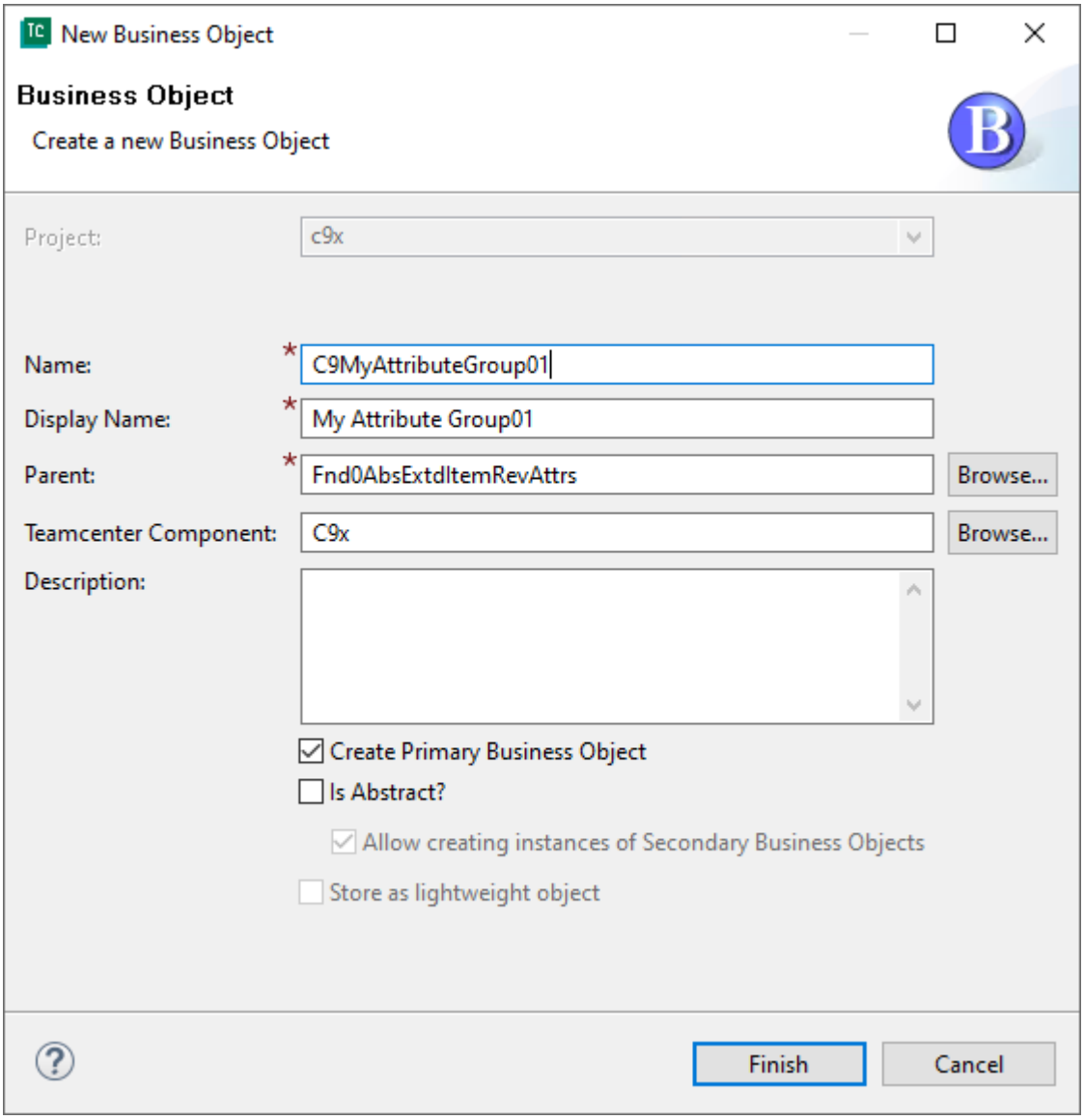


3. Give your new business object a name.

   Ensure that the **Create Primary Business Object** is checked. This will create a class of the same name as your business object.

   Name: C9MyAttributeGroup01

© 2025 Siemens



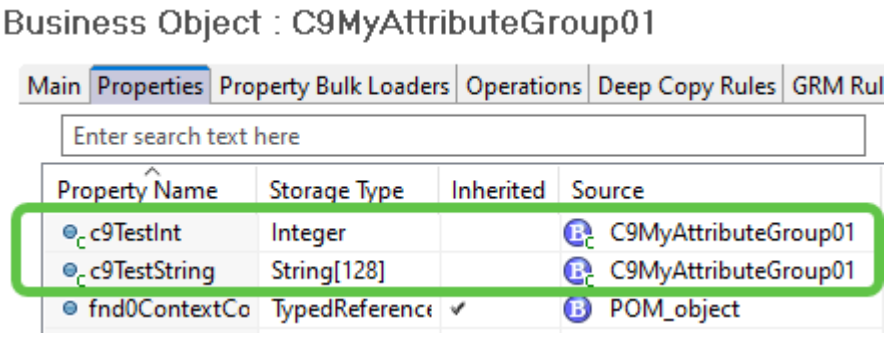
4. Click **Finish**.

5. Double-click your new business object to open it.

6. On the **Properties** tab, click **Add...** to add two **Persistent** properties.

Name	Attribute Type
c9TestInt	Integer
c9TestString	String

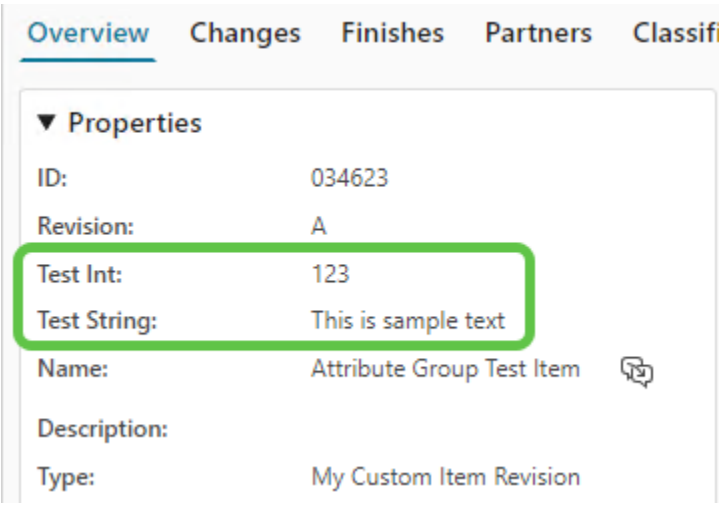
© 2025 Siemens



Results

You have a class of attributes stored as a direct descendant of the **Fnd0AbsExtdItemRevAttrs** class.

Following is an example of a **C9MyCustomItemRevision** with the attribute group properties populated.



Attach an attribute group to a business object

Attach an existing attribute group to an item revision.

Prerequisites

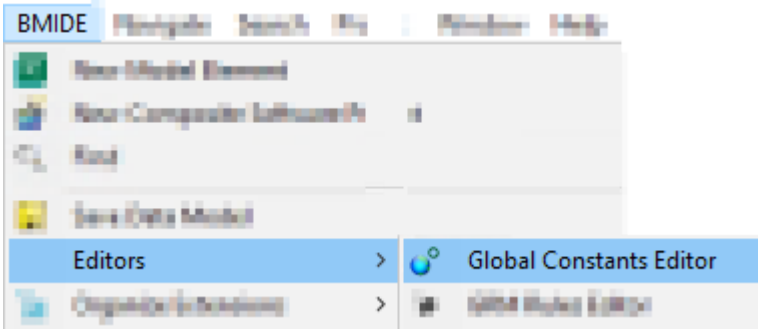
You must have an attribute group in your data model, and an item revision to which you will attach it.

Procedure

This example assumes the attribute group class is named **C9MyAttributeGroup01** and the item revision type is named **C9MyCustomItemRevision**.

1. On the menu bar, choose **BMIDE→Editors→Global Constant Editor**.

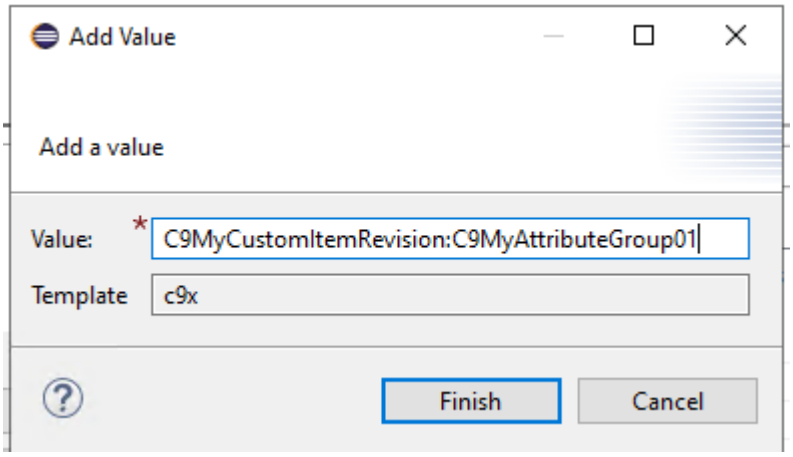
© 2025 Siemens



2. In the **Global Constant** view, double-click the **Fnd0ItemRevRelatedAttrGroups** row or select it and then click **Edit...**.

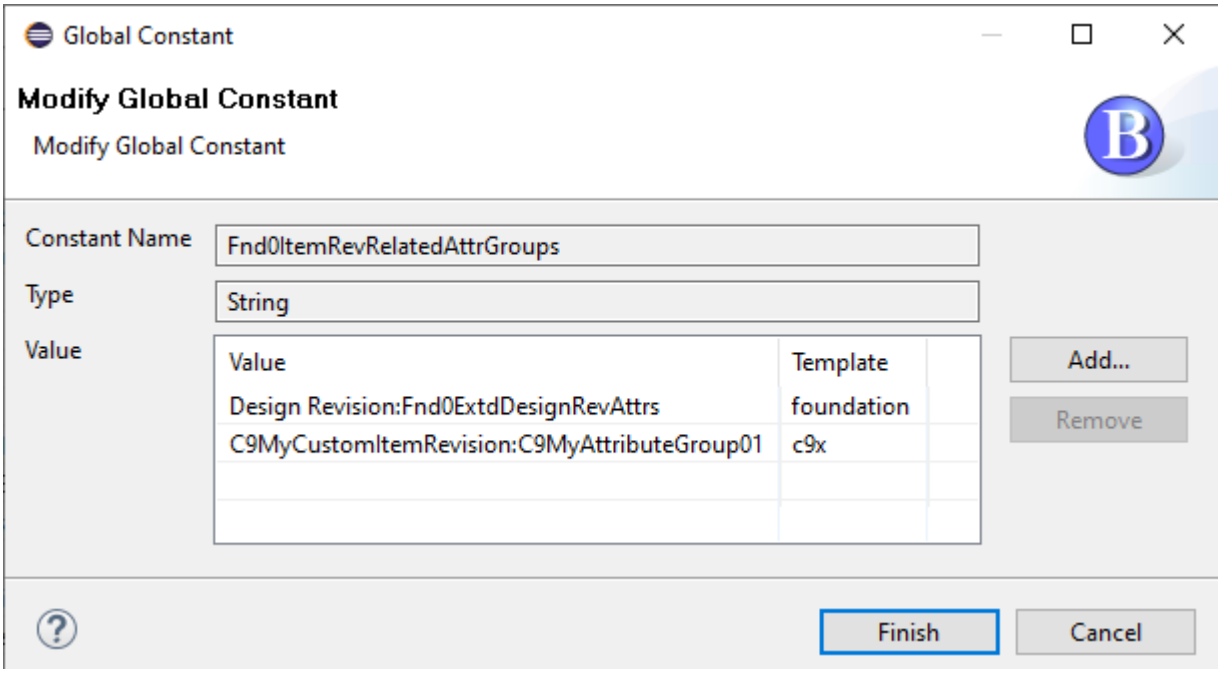
3. In the **Global Constant** dialog, click **Add...**.

4. In the **Add Value** dialog, enter **C9MyCustomItemRevision:C9MyAttributeGroup01** as the **Value** and click **Finish** to close the **Add Value** dialog.



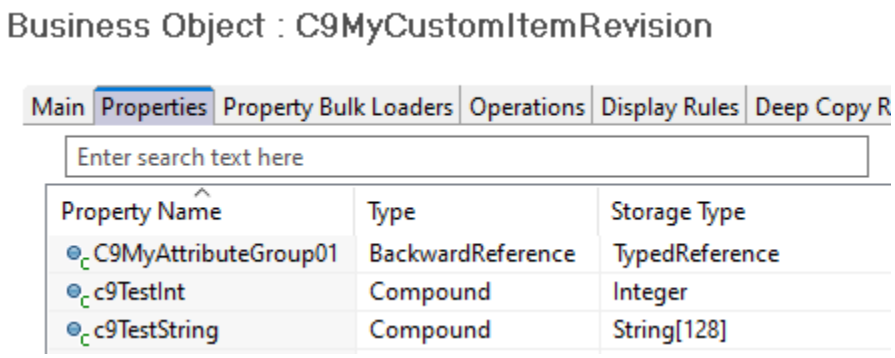
5. Click **Finish** to close the **Global Constant** dialog.

© 2025 Siemens



Results

The item revision shows the properties from the attribute group.



Caution:

A **BackwardReference** property is also created for internal tracking. Do not modify or remove this property.

Postrequisites

Deploy your BMIDE template project to see your attributes on your item revision.

© 2025 Siemens

Using property formatters to configure display format in the Active Workspace

Attach property formatters

To alter the format of a business object property value when it is displayed in Active Workspace, you can attach property formatters to the property.

> Example:
>
> The internal database value (actual value) for a property is **25.693850392**. A property formatter attached to the property specifies showing only two decimal spaces, so the property value displayed is **25.69**.

Characteristics of property formmatters:

- Property formatters do not affect the property value stored in the database.
- Property formatters do not affect the display of the property value in the rich client.
- You can attach property formatters to COTS or custom persistent, runtime, or configured runtime properties, but property formatters are applicable only to properties with the following storage types:

  String
  Integer
  Double
  Boolean

- For successful attachment of a property formatter to a selected business object property, the property constant **Fnd0IsFormattable** for the property must be set to **true**.
- You can attach multiple property formatters with mutually exclusive conditions to a single property. For example, you may want a particular property formatter to be used when one condition is met, and another to be used when another condition is met.

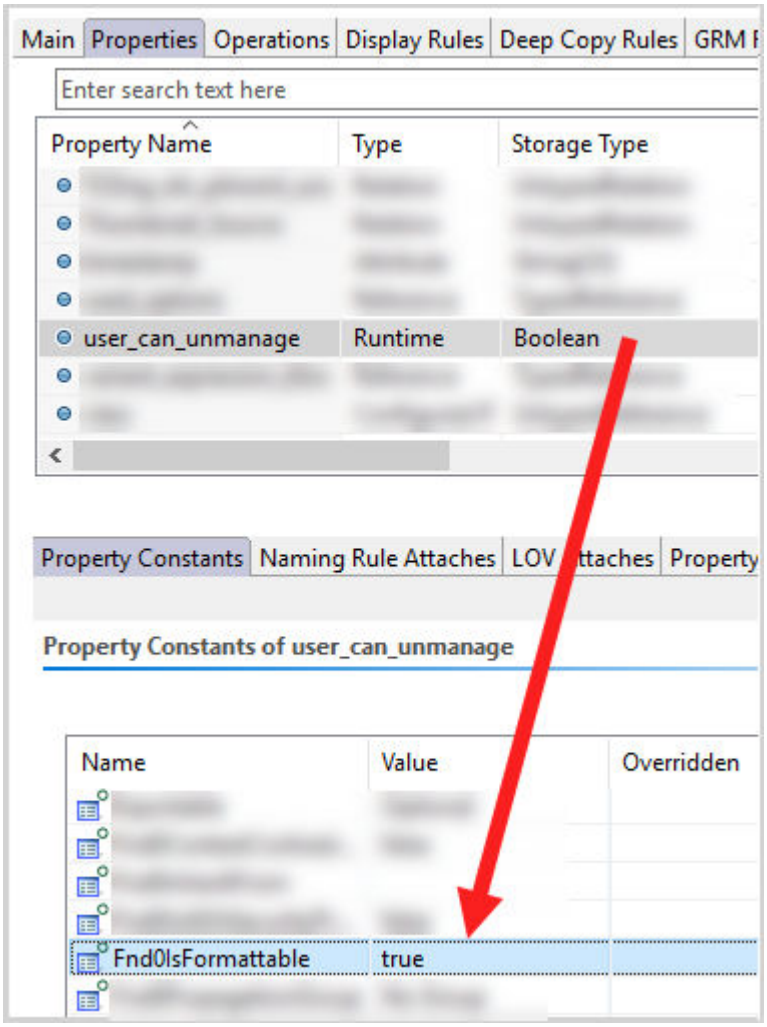
In the Business Modeler IDE, use either of these approaches to attach a property formatter to a property:

In a business object, attach a property formatter
In a property formatter, attach to a business object

In a business object, attach a property formatter

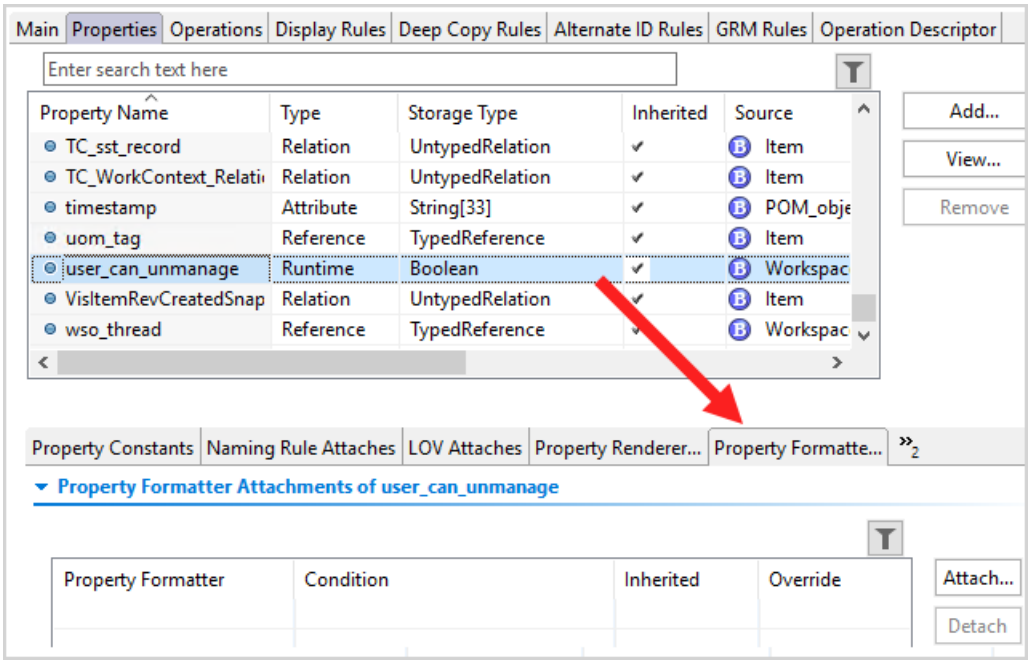
1. Open a business object, select the **Properties** tab, and select the property.
2. Click the **Property Constants** tab and ensure that **Fnd0IsFormattable** is set to **true**.

© 2025 Siemens



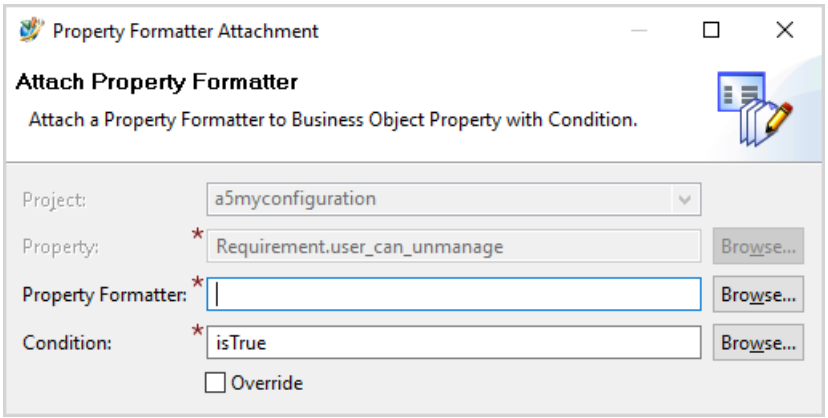
3. Click the **Property Formatter Attachments** tab.

© 2025 Siemens



4. Click **Attach**.

   The **Attach Property Formatter** dialog box is displayed.



5. Next to **Property Formatter**, click **Browse** and select a property formatter.

   The only property formatters available are those that match the data type of the property. For example, if the property is a double data type property, only double property formatters are shown. For successful attachment of a formatter, the property constant **Fnd0IsFormattable** for the property must be set to **true**.

6. Next to **Condition**, click **Browse** and select a conditional state when the formatter applies. The condition **isTrue** means that the property formatter always applies.

7. To override a property formatter inherited from a parent business object, select **Override** ☑.

© 2025 Siemens

Override can be set only with the **isTrue** condition.

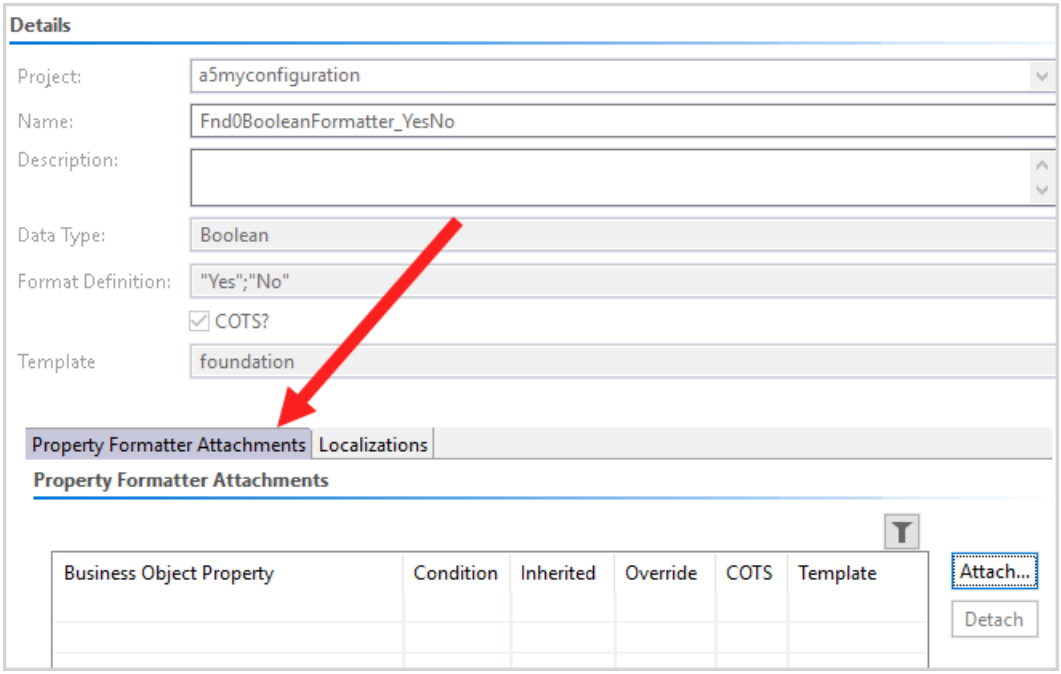
8. Click **Finish**.

   The attachment is displayed on the **Property Formatter Attachments** tab.

In a property formatter, attach to a business object

1. In the **BMIDE** view, expand the **Extensions** and **Property Formatters** folders.

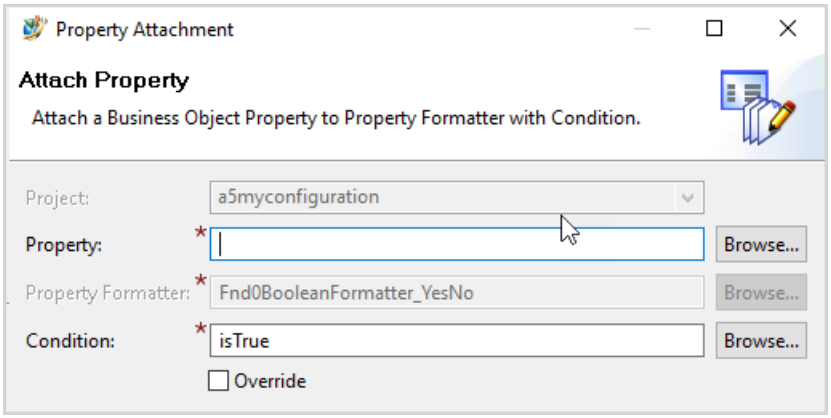
2. Open the property formatter you want to attach to a property and click the **Property Formatter Attachments** tab.



3. Click **Attach**.

   The **Attach Property** dialog box is displayed.

© 2025 Siemens



4. Next to **Property**, click **Browse** and select the business object property to which you want to attach the property formatter.

   The only property formatters available are those that match the data type of the property. For example, if the property is a double data type property, only double property formatters are shown. For successful attachment of a formatter, the property constant **Fnd0IsFormattable** for the property must be set to **true**.

5. Next to **Condition**, click **Browse** and select a conditional state when the formatter applies. Selecting **isTrue** means that the property is always formatted using the property formatter.

6. To override a property formatter inherited from a parent business object, select **Override** ☑.

   Override can be set only with the **isTrue** condition.

7. Click **Finish**.

Verifying property formatter behavior

To verify the behavior, save the changes to the data model and deploy your changes to the test server. After deployment, create an instance of the business object and check the display of the property in the user interface.

For example, if you attached a property formatter that limits display to two decimal places to a double property on an item revision, check out the item revision and type a number that extends more than two decimal spaces. Check in the item revision and observe the display of the value on the property. It should display only two decimal places despite having more decimal places stored in the database.

Create a new property formatter

Many property formatters are supplied with Teamcenter. However, if a property formatter you need does not already exist, you can create your own property formatter.

1. Open the **New Property Formatter** dialog box in one of these ways:

© 2025 Siemens

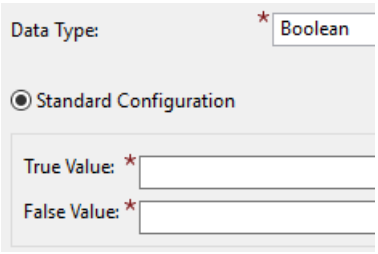
- On the menu bar, choose **BMIDE**>**New Model Element**, type **Property Formatter** in the **Wizards** box, and click **Next**.
- Open the **Extensions** folder, right-click the **Property Formatters** folder, and choose **New Property Formatter**.

2. Enter the parameters for the new property formatter.

For this parameter	Enter this
Name	The name for the new property formatter. Make the name descriptive of its behavior so that others can tell at a glance the format it specifies.
Description	The purpose of the property formatter.
Data Type	The property data type that the formatter can be used for: • **Boolean** - property data evaluates to a binary value, either true or false. • **Double** - property data is a double-precision, floating-point decimal number (sometimes called a real). • **Integer** - property data is a whole number without decimals. • **String** - property data is a string of characters.

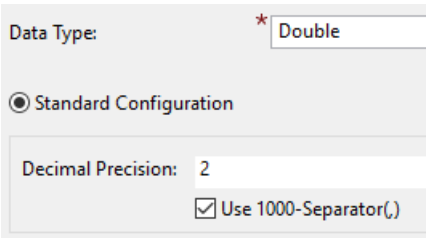
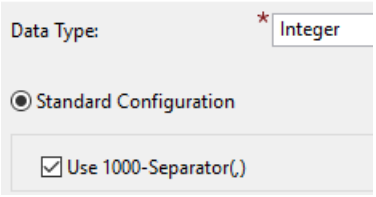
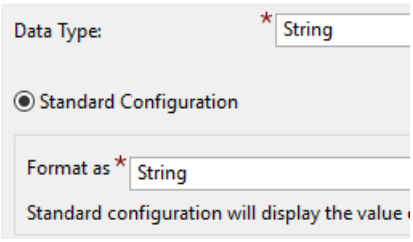
3. Choose a configuration type and configure the formatting.

For this configuration type	Do this
Standard Configuration	Specify a standard formatting for the selected data type.

For this Data Type	Do this
Boolean	

© 2025 Siemens

For this configuration type	Do this

For this Data Type	Do this
	Type a value to display when the Boolean property resolves to **true** and a value to display when it resolves to **false**. Examples of likely value pairs include **True**/**False**, **Yes**/**No**, **On**/**Off**, **Open**/**Closed**, or **1**/**0**.
Double	 In **Decimal Precision**, enter the number of decimal places to display in the user interface. Select the **Use 1000-Separator(,)** if you want the system to insert a comma (**,**) at every thousandth place. For example, when this option is selected, the commas are placed as follows: **1,000.00**, **10,000.00**, **100,000.00**, **100,000,000.00**.
Integer	 Select the **Use 1000-Separator(,)** if you want the system to insert a comma (**,**) at every thousandth place. For example, when this option is selected, the commas are placed as follows: **1,000**, **10,000**, **100,000**, **100,000,000**.
String	

© 2025 Siemens

<table>
<tr><th>For this configuration type</th><th>Do this</th></tr>
<tr><td></td><td>
<table>
<tr><th>For this Data Type</th><th>Do this</th></tr>
<tr><td></td><td>In Format as, select one of Double, Integer, or String.</td></tr>
</table>
</td></tr>
<tr><td>Advanced Configuration</td><td>When a Standard Configuration option does not produce the formatting you want, choose Advanced Configuration and in Format Definition enter a formatter definition.

Details and examples of formatter definitions by data type are described in these reference topics:

Boolean property formatter definition
Double property formatter definition
Integer property formatter definition
String property formatter definition</td></tr>
</table>

4. Use the **Example** area to see the effect of the property formatter definition you have created. To see the formatted value for a different actual value, in **Actual Value** type a value.

5. Click **Finish**.

The property formatter is added to the list of available formatters.

To apply the property formatter in the Active Workspace user interface, you must **attach it to a property** on a business object.

© 2025 Siemens

Boolean property formatter definition

The Boolean property formatter allows two choices to the user (for example, **True** or **False**).

PATTERN DEFINITION

true-value;*false-value*

PATTERN DETAILS

Y;

True value pattern	False value pattern
Y	

PATTERN EXAMPLES

You can type any representative string for the Boolean values.

Values are entered to the **True Value** and **False Value** boxes when creating a Boolean property formatter.

Pattern	Real value	Formatted value
Y;	true	Y
Yes;No	true	Yes
1;0	true	1
True;False	true	True
T;F	true	T
Right;Wrong	false	Wrong
Good;Bad	false	Bad
OK;Not OK	false	Not OK

© 2025 Siemens

Double property formatter definition

The double property formatter specifies a double-precision, floating-point decimal number (sometimes called a real).

PATTERN DEFINITION

positive;*negative*;*zero*

PATTERN DETAILS

#,##0.###; (#,##0.###);"Nil"

Positive value pattern	Negative value pattern	Zero value pattern
#,##0.###	**(#,##0.###)**	**"Nil"**

PATTERN CHARACTERS

Character	Description
0	Prints a digit. For example, if the value is **8.9**, and it is to be displayed as **8.90**, enter the format as **#.00** in the **Format Definition** box.
#	Prints a digit. Zero is not printed. For example, if the custom format is **#.##**, and the value is **8.9**, the value **8.9** is displayed.
.	Decimal separator.
,	Digit grouping character.
;	Pattern separator.
""	Pre-fix and post-fix string.

PATTERN EXAMPLES

Following are examples that you can type in the **Format Definition** box when creating a double property formatter.

© 2025 Siemens

Pattern	Real value	Formatted value
#,##0.###	.56	0.56
#,##0.###	-1234.56	-1,234.56
#,###.000	-1234.56	-1,234.560
#,##0.###;(#,##0.###)	-1234.56	(1,234.56)
#,##0;#,##0	-1234.56	-1,234
#,###.###;(#);"Nil"	-1234.56	(1234)
#,##0.###;(#);"Nil"	0	Nil
#.### " dia."	10.23	10.23 dia.
"~ " #.### " ft."	10.23	~ 10.23 ft.
"[" #.### "]"	10.23	[10.23]
"$" #.## " ea."	10.25	$10.25 ea.
##.## " GHz"	54.76	54.76 GHz

© 2025 Siemens

Integer property formatter definition

The integer property formatter specifies a whole number without decimals. Following are examples that you can type in the **Format Definition** box when creating an integer property formatter.

PATTERN DEFINITION

positive;*negative*;*zero*

PATTERN DETAILS

#,##0; (#,##0); "Nil"

Positive value pattern	Negative value pattern	Zero value pattern
#,##0	**(#,##0)**	**"Nil"**

PATTERN CHARACTERS

Character	Description
0	Prints a digit. For example, if the value is **12**, and it is to be displayed as **012**, enter the format as **000** in the **Format Definition** box.
#	Prints a digit. Zero is not printed. For example, if the custom format is **###**, and the value is **12**, the value **12** is displayed.
,	Digit grouping character.
;	Pattern separator.
""	Pre-fix and post-fix string.

PATTERN EXAMPLES

Following are examples that you can type in the **Format Definition** box when creating an integer property formatter.

Pattern	Real value	Formatted value
#,##0	**-1234**	**-1,234**
#,###	**-1234**	**-1,234**
#,##0;(#,##0)	**-1234**	**(1,234)**
"$" #,###	**-1234**	**$ -1,234**

© 2025 Siemens

Pattern	Real value	Formatted value
#,##0;(#);"Nil"	0	Nil
#,##0;(#)	0	0
# " MB"	1024	1024 MB

© 2025 Siemens

String property formatter definition

The string property formatter specifies a string of characters. Following are examples that you can type in the **Format Definition** box when creating a string property formatter.

PATTERN DEFINITION

{d:*Positive*; *Negative*; *Zero*}{i:*Positive*; *Negative*; *Zero*}{s:`String`}

PATTERN DETAILS

{d:#,##0.##; (#,##0.##);"-"}{s: "pre " @ " post"}

Double pattern			Integer pattern			String pattern
Positive value pattern	**Negative value pattern**	**Zero value pattern**	**Positive value pattern**	**Negative value pattern**	**Zero value pattern**	**String value pattern**
#,##0.##	**(#,##0.##)**	**"-"**				**"pre" @ "post"**

PATTERN CHARACTERS

Character	Description
0	Prints a digit. For example, if the value is **12**, and it is to be displayed as **012**, enter the format as **000** in the **Format Definition** box. This character is applicable for positive, negative, and zero value patterns.
#	Prints a digit. Zero is not printed. For example, if the custom format is **###**, and the value is **12**, the value **12** is displayed. This character is applicable for positive, negative, and zero value patterns.
{d: }	Double pattern specifier.
{i: }	Integer pattern specifier.
{s: }	String pattern specifier.
@	Placeholder for actual string value. This character is applicable for string value patterns.
.	Decimal separator. This character is applicable for positive, negative, and zero value patterns.
,	Digit grouping character. This character is applicable for positive, negative, and zero value patterns.

© 2025 Siemens

Character	Description
;	Pattern separator.
""	Pre-fix and post-fix string.

PATTERN EXAMPLES

Following are examples that you can type in the **Format Definition** box when creating a string property formatter.

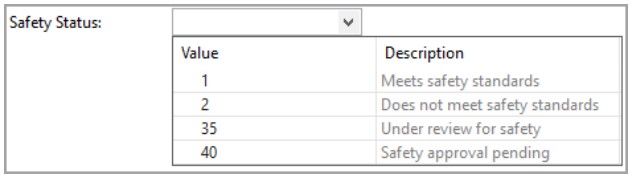
Pattern	Real value	Formatted value
{d:#,##0.##;(#,##0.##)}	1234.567	1,234.56
{d:#,##0.##;(#,##0.##)}	-1234.567	(1,234.56)
{d:#,##0.##;(#,##0.##);"Nil"}	0	Nil
{d:#,##0.##;(#,##0.##);"Nil"}{s: @ }	000018/A	000018/A
{s:"$ " @ }	1000	$ 1000
{d:#,##0}{s:"$ " @ }	1000	1,000
{d:#,##0.##;(#)}	-1234.6738	(1234)
{d:# " Dia";# " Dia";” 0 Dia”}	1000.24	1000 Dia
{d:#;#;"NIL"}{s: @ " KB"}	4 x 256	4 x 256 KB
{d:#;#;"NIL"}{s: @ " KB"}	1000.24	1000
{d:#,##0.## " ft/sec"}	1000.2345	1,000.23 ft/sec
{d: "$ " #,###.00 }	1000000.2000000	$ 1,000,000.20
{d: # " GT/s"}	5	5 GT/s

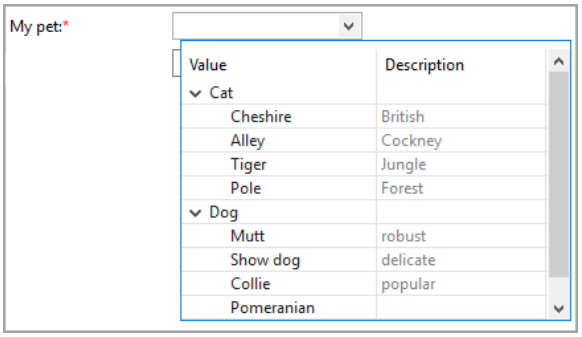
Lists of values

What are lists of values (LOVs)?

Lists of values (LOVs) are definitions that provide the end user a choice of values for object instance properties. The values typically appear in a table that shows a flat list of values and their descriptions, or as an hierarchical expanding list. An LOV can also be defined to allow a user to enter a value within a specified range.

© 2025 Siemens





A developer implements LOVs throughout the interface by attaching an LOV to a property on one or more business objects. The use of LOVs can greatly improve productivity at your site and help prevent incorrect user entries.

Every LOV has at least one level of choices, and when **attached to a business object property** enables an end user to choose a value for the property on an object instance. For example, a simple list of country abbreviation values with the full country name in the corresponding descriptions would allow a user to select an appropriate abbreviation.

A *filtered* list is a list that references another list, but includes only selected values from the original list. For example, you might create a region list by filtering a referenced country list.

Cascading lists nest LOV definitions to present a hierarchy of choices that enable a user to navigate the hierarchy and finally add a value for a property in an object instance. The hierarchy for the list does not necessarily need to be balanced. For example, consider a list of countries, only some of which have subordinate lists of states. A small country might be precise enough for an object property, while for a very large country a user might need to select from subordinate lists of regions, states, and cities.

Levels in a cascading LOV can be attached to *interdependent* properties of a business object so that in an object instance, selecting a value for a parent level property populates the available choices for the child level property. In such a case, the hierarchical lists must be balanced: every branch of the hierarchy must contain at least one value in every level.

In the Business Modeler IDE, a project's **Extensions** folder contains an **LOV** folder. All LOVs are initially defined in a BMIDE project. Values for **classic** LOVs are maintained within the project and published with the template. Classic LOVs can be extended to fetch values through a business logic defined in custom code. **Externally managed (batch)** LOVs are similar to classic LOVs, except that values are not stored in the template, but can be extracted from the database, modified, and uploaded without having to shut down the database. Values for **dynamic** LOVs are obtained by querying the database each time a user accesses the LOV.

Create classic lists of values

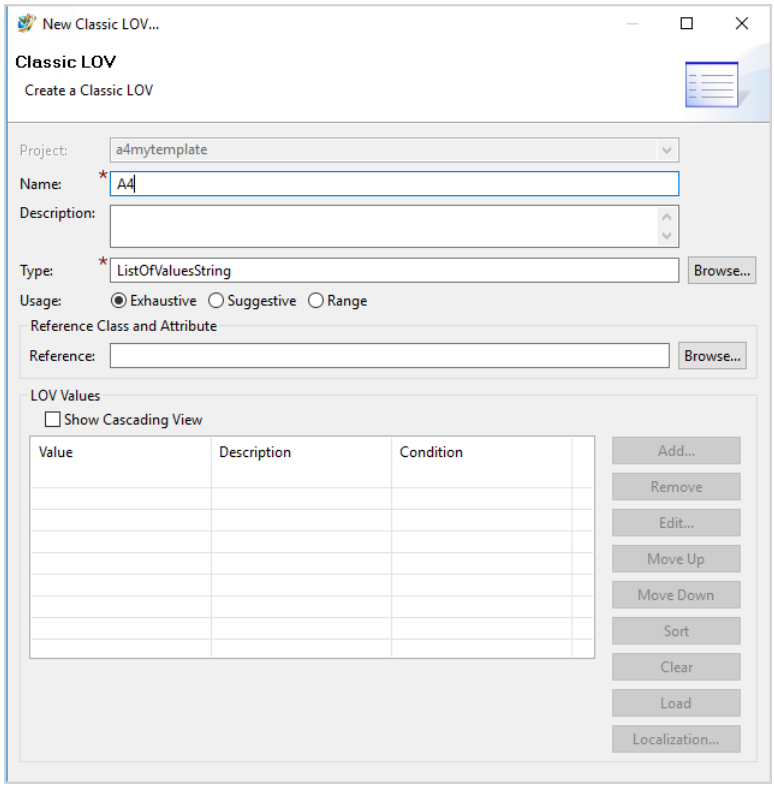
1. Start the Business Modeler IDE.

© 2025 Siemens

2. Open the **New Classic LOV** wizard in one of these ways:

   - On the menu bar, choose **BMIDE**>**New Model Element**, type **Classic LOV** in the **Wizards** box, and click **Next**.

   - Open the **Extensions**>**LOV** folder, right-click the **Classic LOV** folder, and choose **New Classic LOV**.



3. Enter the following information in the **Classic LOV** dialog box:

For this option	Do this
Name	Type the name you want to assign to the new LOV. The template project prefix is automatically supplied.
Description	Enter the purpose of the LOV. To control display of the description in the end-user interface, use **List of Values** preference (accessible in the rich client from the **Edit**>**Options** menu).
Type	Click **Browse** to select the type of LOV you want to create. The selection dialog box that appears lists available types with their descriptions. Mainstream LOV types that do not require custom code include the following:

© 2025 Siemens

For this option	Do this
	ListOfValuesChar – Single ASCII character. Only alphabetic characters can be used (for example, A-Z). You cannot use any other kinds of characters (numbers, punctuation, and so on). **ListOfValuesDate** – Date and time in the format used at your site. **ListOfValuesDouble** – Double-precision, floating-point decimal number (sometimes called a real). **ListOfValuesFilter** – Reference to another non-filtered LOV with the capability to filter the referenced LOV's values. **ListOfValuesInteger** – Whole number. **ListOfValuesString** – String of ASCII characters. The types are all children of the **ListOfValues** business object. Values for LOVs of the mainstream list types, except for ListOfValuesFilter, can be stored in the database as batch LOVs, separate from a BMIDE template.
Usage	Select an option for how a user can specify a value. • **Exhaustive** The list contains all possible choices. • **Suggestive** The list contains suggested choices. The user can enter their own value if they want. • **Range** The list allows a range of valid numeric values.
Reference (appears for applicable types)	Leave blank if you want to manually enter a list. If you want to add values to your list from object instances in the database, do the following: a. Click **Browse** and select the class and attribute to query for values. b. In the lower right of the dialog box, click **Load** to obtain values from the database and populate the table with items from the attribute.

4. To create cascading LOVs, select the **Show Cascading View** check box. Click the **Add Sub LOV** button to add existing LOVs to the list.

5. To manually populate the **LOV Values** list, click **Add**.

   In the **Add LOV Value** wizard, specify properties for a value.

© 2025 Siemens

For this property	Do this
Value	Enter a value as appropriate for the list type. Ensure that the LOV values entered adhere to the standard for the selected data type. • If you choose a double primitive data type, and the data you enter is more than the capacity of that data type, the data may be truncated or rounded per Institute of Electrical and Electronics Engineers (IEEE) standards; if you do not adhere to the capacity of the double data type, there may be a deployment issue for processing other elements like localization and sub-LOV attachments that are tied to the LOV values. • Characters are limited to the ASCII character set. Copying characters from Microsoft Word or other non-ASCII editor and pasting into Business Modeler IDE is an unsupported operation. • Because the comma (**,**) character is used as a separator for multivalue LOV selection, do not use the comma (**,**) character in LOV values when any primitive properties, such as character, integer, or string, are configured as arrays.
Value Display Name (for strings)	Type the value name as you want it to appear in the user interface.
Description	Type a description for the value.
Condition	This option value is fixed to **isTrue**. However, you can **apply conditions to LOVs**.

6. If the selected usage is **Range**, enter the high and low values for the range in **Upper** and **Lower**.

7. For usages other than **Range**, change the list as desired using **Remove**, **Edit**, **Move Up**, **Move Down**, **Sort**, and **Clear**.

   LOV values cannot be reordered in filter LOVs, extent types of LOVs, or tag extent types of LOVs.

8. If you want to **change the display text for the LOV values** in different language locales, click the **Localization** button.

9. Click **Finish**.

   The new LOV appears in the **Classic LOV** folder.

10. Save the changes to the data model.

To display the LOV for specifying an object instance property value in a Teamcenter client user interface, you must **attach the LOV** to a property on a business object.

© 2025 Siemens

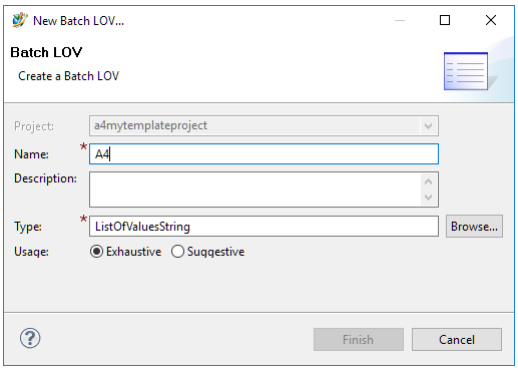
Batch LOVs

Create an externally managed (batch) list of values

You can define a list of values (LOV) as externally managed (batch). Once you deploy a batch LOV to the database, you can subsequently use the **bmide_manage_batch_lovs** utility to extract its values to an XML file, modify the file, and then use the **bmide_manage_batch_lovs** utility again to update from the file to the database without having to shut down the database.

1. Start the **New Batch LOV** in one of these ways:

   - On the menu bar, choose **BMIDE**>**New Model Element**, type **Batch LOV** in the **Wizards** box, and click **Next**.
   - Expand the **Extensions**>**LOV** folders, right-click the **Batch LOV** folder, and choose **New Batch LOV**.



2. In the **Batch LOV** dialog box, specify the following.

Parameter	Description
Name	Enter a name for the new LOV. When you name a new data model object, a prefix from the template is automatically affixed to the name to designate the object as belonging to your organization, for example, **A4**.
Description	Enter the purpose of the LOV. Descriptions can be displayed in the rich client for LOVs specified in the **List of Values** preference.
Type	Click **Browse** and select the LOV list type to create.

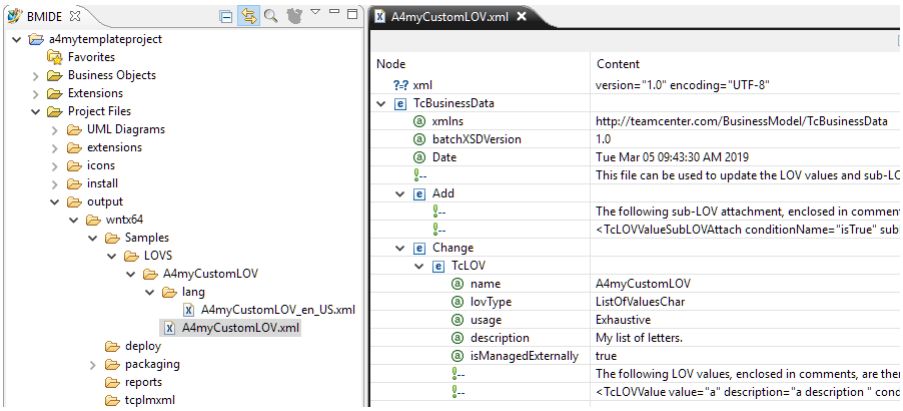
© 2025 Siemens

Parameter	Description
	The available LOV list type business objects are all children of the **ListOfValues** business object. The ability to store LOV values in the database as batch LOVs is available only on the following LOV list types: **ListOfValuesChar**, **ListOfValuesDate**, **ListofValuesDouble**, **ListOfValuesInteger**, and **ListOfValuesString**.
Usage	Select one of the following. • **Exhaustive** Indicates that the list contains all possible choices. • **Suggestive** Specifies that the list contains suggested choices. The user can enter their own value if they want.

3. Click **Finish**.

The project is reloaded, in the BMIDE view the new batch LOV appears in the **Batch LOV** folder, and a message is displayed in the **Console** view with a path to the new batch LOV file.

4. (Optional) In the **Project Files** folder, expand the **\output\Samples\LOVS** folders to see the sample LOV file.



5. To add list values, edit the files you just created.

   a. Modify the sample file by replacing the commented code with your values.

© 2025 Siemens

> Tip:
>
> Extract LOV values from your external ERP system and place the values in the XML file and the accompanying localization XML file. The XML files must confirm to the batch utility schema.
>
> For additional sample XML files, see the *TC_ROOT***bmide\client\samples\externallymanagedlovs** directory.

b. Modify the localization files in the **lang** folder by replacing the commented code with your values.

6. Package the template in the Business Modeler IDE and deploy it using Teamcenter Environment Manager (TEM) after shutting down the system.

Edit deployed batch LOVs

1. To extract batch LOVs from the database, run the **bmide_manage_batch_lovs** utility.

2. In the extracted LOV XML file, edit the list values.

> Tip:
>
> Extract LOV values from your external ERP system and place the values in the XML file and the accompanying localization XML file. The XML files must confirm to the batch utility schema.
>
> For additional sample XML files, see the *TC_ROOT***bmide\client\samples\externallymanagedlovs** directory.

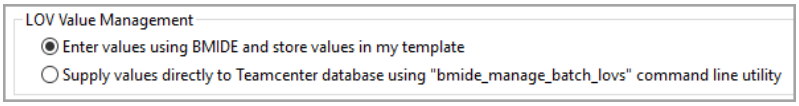
3. To update the externally managed LOVs in the database, run the **bmide_manage_batch_lovs** utility.

4. (If you are using client cache at your site) After updating externally managed LOVs, to update the LOV cache stored on the server, run the **generate_client_meta_cache** utility with the **generate lovs** command.

Convert an LOV managed in a template (classic LOV) to an externally managed LOV (batch LOV)

1. Open an LOV that is managed in a template (classic LOV).

2. In the **LOV Value Management** box of the LOV, select **Supply values directly to Teamcenter database using "bmide_manage_batch_lovs" command line utility**.



© 2025 Siemens

A dialog box describing the consequences of proceeding appears.

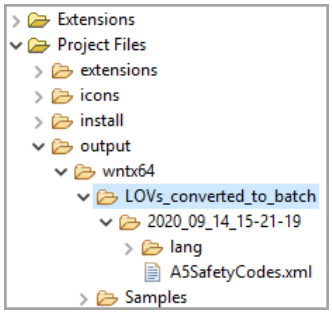
3. Click **OK**.

   The LOV is moved from the **Classic LOV** folder to the **Batch LOV** folder.

   The LOV values, sub-LOV attachments, and localizations are saved in an XML file.

4. In the **Project Files** folder, open the **\output\LOVs_converted_to_batch***date-and-time* folders to find the XML file. An XML file containing localizations for the LOV is created in a **lang** subfolder.

   If you convert multiple LOVs, separate XML files are created for each LOV.



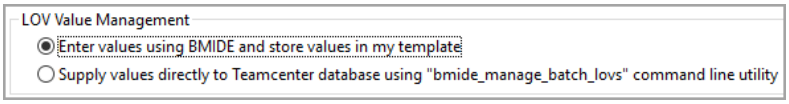
5. Choose **BMIDE**>**Save Data Model**, then **BMIDE**>**Generate Software Package**, and install the template to the server. This deletes all the LOV values, localizations, and sub-LOV attachments in the database for the LOVs that were converted.

   Now that the LOV values are stored in the database separate from the template, you can use the **bmide_manage_batch_lovs** utility to manage the values in the XML files to the database.

Convert an externally managed LOV (batch LOV) to an LOV managed in a template (classic LOV)

1. Open an LOV that is externally managed (batch LOV).

2. In the **LOV Value Management** box of the LOV, select **Enter values using BMIDE and store values in my template**.



   A dialog box describing the consequences of proceeding appears.

3. Click **OK** in the dialog box.

© 2025 Siemens

The LOV is moved from the **Batch LOV** folder to the **Classic LOV** folder.

The **Add** button in the user interface is enabled to allow you to add values to the LOV.

4. Add values to the LOV.

   Because the LOV values for this LOV are still in the database, you can extract them using the **bmide_manage_batch_lovs** utility. To extract LOV values, sub-LOV attachments, and localizations for specific LOVs in the database, enter a command like the following on a single line, for example:

   ```
   bmide_manage_batch_lovs -u=username -p=password -g=dba
   -option=extract -lov=BatchLOV_LOV1,BatchLOV_LOV2
   -file=BatchLOV_extracted.xml
   ```

   You can then use the extracted values as the basis for new values to enter in the template-managed LOV.

5. Choose **BMIDE**>**Save Data Model**, then **BMIDE**>**Generate Software Package**, and install the template to the server.

   After updating the template to the server, these changed LOVs are not available for extraction or update using the **bmide_manage_batch_lovs** utility.

Considerations for externally managing LOVs

Valid LOV list types for external (batch) LOVs include only the following:

ListOfValuesInteger
ListOfValuesDouble
ListOfValuesChar
ListOfValuesString
ListOfValuesDate

Valid usage types for external LOVs include only the following:

Exhaustive
Suggestive

For external LOVs:

- Interdependent LOV attachments are not supported.
- For **String** type LOVs, referencing a property of another class is not allowed.

Both internal and external LOVs can have either internal or external sub-LOVs. The tool used to attach sub-LOVs depends on the type of the parent LOV:

© 2025 Siemens

If the LOV is this type	Then use this tool to attach a sub-LOV
Classic (internal to a Business Modeler IDE template)	Business Modeler IDE
Batch (external)	**bmide_manage_batch_lovs** utility

Classic LOVs (managed in a Business Modeler IDE template) cannot be converted to external LOVs in the following cases:

- You do not own the LOV in your template.

  For example, COTS LOVs cannot be converted to externally managed LOVs.

- The LOV value is referenced in any other element, such as a verification rule or note type.

- The LOV is used as a based-on LOV in a filter LOV.

- In a cascading LOV, the LOV is not the leaf node.

  Lov1
  Lov2
  Lov3

  In this example, **Lov1**, **Lov2** and **Lov3** are LOVs managed in the Business Modeler IDE. Here, **Lov1** and **Lov2** cannot be converted to an externally managed LOV, but **Lov3** can be converted to an externally managed LOV.

Dynamic LOVs

Introduction to dynamic LOVs

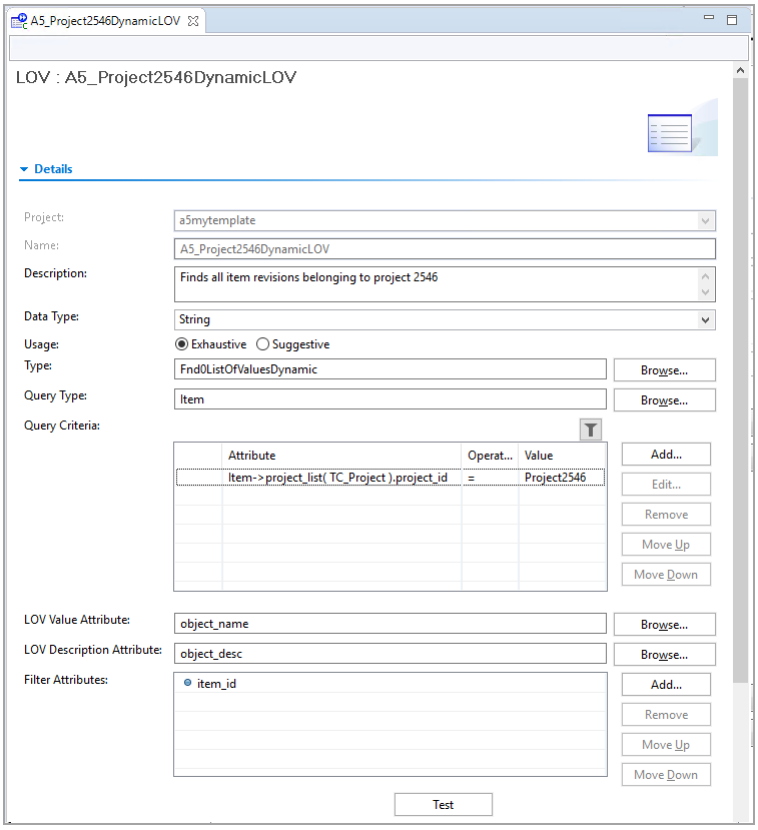
Lists of values (LOVs) are lists on property boxes in the user interface. Dynamic LOVs show a list of values obtained dynamically by querying the database.

For example, if you want a list of values of all aluminum widgets in the database that have a blue glossy finish, you can create a dynamic LOV to query the database for them. Or if you want a list of values of all the members of the widget development project, you can create a dynamic list of values to find them. Dynamic lists of values are not static but change as the data in the database changes. They allow end users to select from lists composed of data that is active in the database.

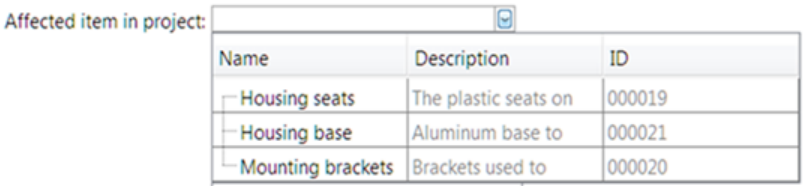
You can create a query as part of the dynamic LOV definition. You can query for product data from objects such as items, parts, and datasets, or query for administrative data from categories such as users, groups, and roles. You can also filter the LOV values from the queries and import LOV values from an external system using TC XML.

© 2025 Siemens

The following sample dynamic LOV queries for all items belonging to a certain project.



When you attach the dynamic LOV to a property and install the template to the server, the query results are displayed in the end-user interface as a list of values.



Create dynamic lists of values

Dynamic lists of values (LOVs) show a list of values obtained dynamically by querying the database. You can query for product data from objects such as items, parts, and datasets, or query for administrative data from categories such as users, groups, and roles. For example, you could create a list of values that queries for all items supplied by a particular vendor, or all users in a certain group.

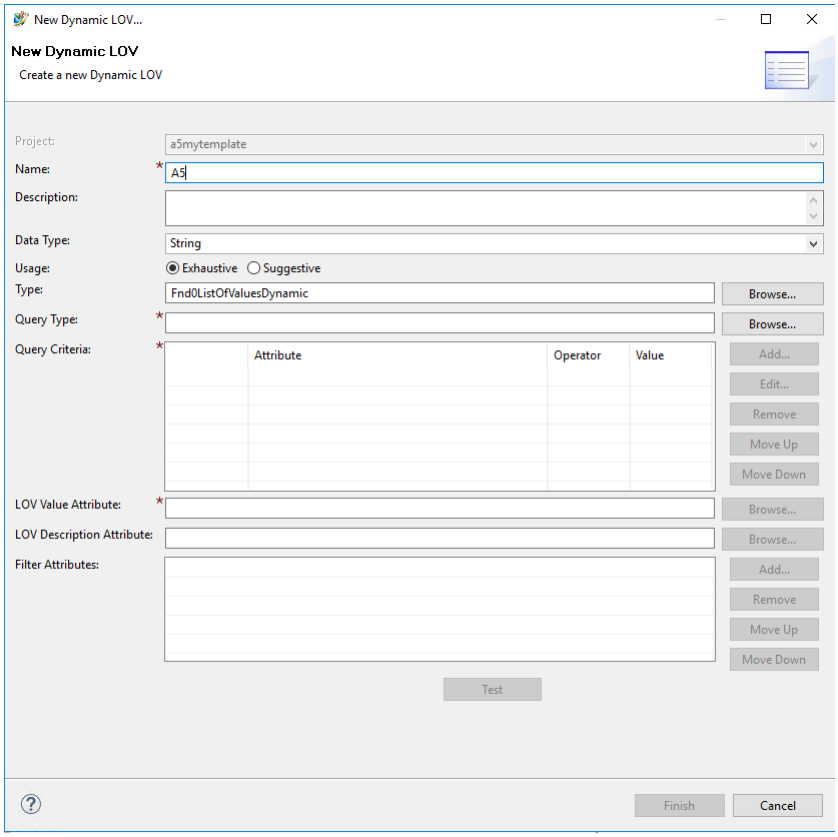
1. Choose one of these methods:

   - On the menu bar, choose **BMIDE**>**New Model Element**, type **Dynamic LOV** in the **Wizards** box, and click **Next**.

© 2025 Siemens

- Open the **Extensions\LOV** folders, right-click the **Dynamic LOV** folder, and choose **New Dynamic LOV**.

The New Dynamic LOV wizard runs.



2. Specify parameters for the query.

For this parameter	Do this
Name	Type the name you want to assign to the new LOV in the database.
Description	State the purpose of the LOV.
Data Type	Select the data type of the LOV values: • **String** String of ASCII characters. • **Integer** Whole number.

© 2025 Siemens

For this parameter	Do this
	• **Date** Date and time in the format used at your site. Tip: For the date attribute type, the earliest date supported is January 2, 1900. • **Double** Double-precision floating point decimal number (sometimes called a *real*). • **Tag** Reference to a unique tag in the database (for example, item ID).
Usage	Select one of the following buttons: • **Exhaustive** Indicates that the list contains all possible choices. • **Suggestive** Specifies that the list contains suggested choices. The user can enter their own value if they want.
Type	The box defaults to the **Fnd0ListOfValuesDynamic** business object, which is the business object used to define dynamic lists of values.
Query Type	Click **Browse** to select the first business object type to query. You can also press Alt+C or start typing to see a list of suggestions. For example, if you want to query for properties on item business objects, select **Item**. If you want to query for system users, select **User**.
Query Criteria	Click **Add** to select the attributes to include in the query. a. Select one of the **Filter** options: • **Attributes and References** Displays attributes, typed references, untyped references, and relations on the query class.

© 2025 Siemens

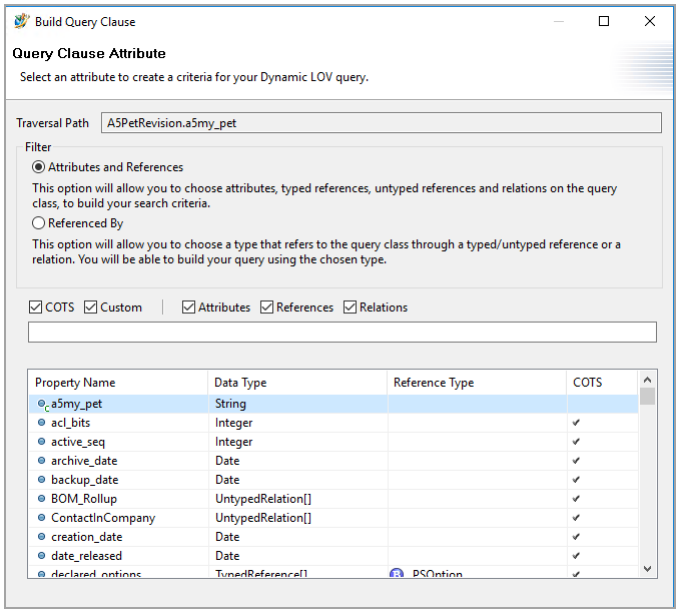
For this parameter	Do this

- **Referenced By**

  Displays typed and untyped references on a relation. When you select this and then select an attribute in the table below, you must click **Next** to select the referenced attribute.

b. Select an attribute in the table.

c. Click **Finish**.



d. After you add an attribute to the **Query Criteria** table, add the following to complete the criteria:

   A. Click in the **Operator** box and select the operator to evaluate the attribute.

Operator	Description
=	Equal to
>	Greater than
<	Less than

© 2025 Siemens

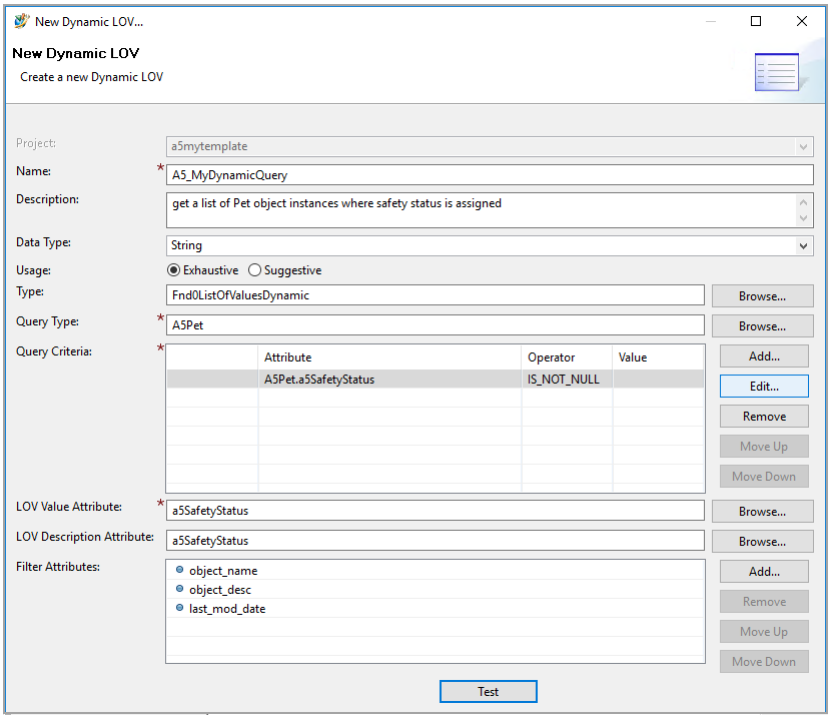
<table>
<tr><th>For this parameter</th><th>Do this</th></tr>
<tr><td></td><td>
<table>
<tr><th>Operator</th><th>Description</th></tr>
<tr><td>!=</td><td>Not equal to</td></tr>
<tr><td>>=</td><td>Greater than or equal to</td></tr>
<tr><td><=</td><td>Less than or equal to</td></tr>
<tr><td>IS_NULL</td><td>Empty</td></tr>
<tr><td>IS_NOT_NULL</td><td>Not empty</td></tr>
</table>
B. Click in Value and type the value against which the selected attribute has to be evaluated.

C. If you add another attribute to the table, AND automatically is added to the first column to indicate that this next criteria attribute to be added to the query. Click the arrow in the cell containing AND to change it to OR if desired.
</td></tr>
<tr><td>LOV Value Attribute</td><td>Click Browse and select the property to be used as the value of the LOV displayed in the end-user interface. You can also press Alt+C or start typing to see a list of suggestions.

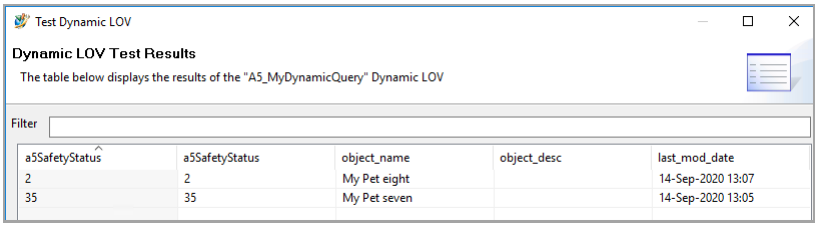
The property only can be a simple attribute and not a reference attribute, and it must be the same property type as in the Data Type box.</td></tr>
<tr><td>LOV Description Attribute</td><td>Click Browse and select the property to be used as the description of the LOV displayed in the end-user interface. You can also press Alt+C or start typing to see a list of suggestions.</td></tr>
<tr><td>Filter Attributes</td><td>Click Add to add properties to view as columns in the list of values table in the end-user interface. These provide more information about each value to help the end user. Also, the end user can click the column headings to sort (filter) the lists of values.

These properties cannot be reference or relation properties.</td></tr>
</table>

© 2025 Siemens



3. Click **Test** to send the query to the database and view the returned results. The results of the query are displayed as the LOV table that appears in the end-user interface.



4. Click **Finish** to complete the new dynamic list of values.

5. **Attach** the new dynamic LOV to a property.

   When an end user clicks the arrow on the property in the user interface, the LOV table displays the query results.

6. To save the changes to the data model, choose **BMIDE**>**Save Data Model**, or on the main toolbar click **Save Data Model** .

7. To deploy your changes to the server, choose **BMIDE**>**Deploy Template** on the menu bar, or select the project and on the main toolbar click **Deploy Template** .

8. After you complete this work, the LOV value and description pairs are displayed on the business object property in the Teamcenter rich client UI.

© 2025 Siemens

Adding support for dynamic LOV language or searchability

Add the name of a dynamic LOV to the **LOVLookupSupport** global constant to display the text of the dynamic LOV in different languages or to search for the text in the dynamic LOV.

The **LOVLookupSupport** global constant looks into the following methods to do the necessary mapping from display-to-internal or internal-to-display values:

LOV_ask_num_of_values_msg
LOV_ask_values_msg
LOV_ask_disp_values_msg

Considerations for creating dynamic LOVs

Following are considerations to keep in mind when you create dynamic lists of values.

If this is the case	Then this applies
The business object in the **Query Type** box is revisable (such as **ItemRevision**)	When you click the **Test** button, the values for all queried instances are displayed in separate rows. If you want only the values from the active sequence revisions to be displayed, then add the following line in the **Query Criteria** table: `AND` *query-type*`.active_seq != 0`
The properties in the **LOV Value Attribute**, **LOV Description Attribute**, or **Filter Attributes** boxes are variable length arrays (VLAs)	When you click the **Test** button, all the values for the properties in the queried instances are displayed in separate rows. In addition, the results cannot be sorted by clicking the column header; the results are displayed unsorted as they are retrieved from the server.
The property in the **LOV Description Attribute** box is a variable length array (VLA)	When you attempt to create an interdependent LOV using that dynamic LOV, the **Attach Description** button in the **Interdependent LOV** dialog box is disabled. Attaching a description in interdependent LOVs is not supported for VLA properties.
A variable length array (VLA) property is selected as an LOV value	All the values from the array of the selected instance are displayed. If any of the values from the array are invalid per the query condition, and that value is selected in the rich client, then an error message is displayed.

© 2025 Siemens

If this is the case	Then this applies
	Example: `The Value "80.5" is not valid for the property "a2AttachDouble" of type "Double".`
You create a dynamic LOV and select a date attribute in the **Query Criteria** table, and you use the **=** operator to find items with a particular date	When you click the **Test** button, you do not get the expected results. That is because hours and minutes are saved on date attributes, and using the **=** operator you must enter the exact hour and minute to query data attributes. Instead, use the **>=** or the **<=** operators. Also, keep in mind that the Business Modeler IDE displays date and time in Greenwich Mean Time (GMT), and the rich client displays the date and time in the end user's local time. As a result, there can be a mismatch between the date attribute values displayed in the Business Modeler IDE and those retrieved from the database.
You want to hide the LOV descriptions in the client user interface.	You cannot hide the descriptions for dynamic LOVs. However, you can hide descriptions for classic and batch LOVs in the Teamcenter rich client using the **List of Values** preference.
In a Teamcenter client, you want to display a localized value for a dynamic LOV.	A dynamic LOV end-user interface displays only the master locale value for the **LOV Value Attribute** property. Use the following procedure to display localized values for the **LOV Description Attribute** property. 1. When you create the dynamic LOV, select the properties to display in the end-user interface using the **LOV Value Attribute** and **LOV Description Attribute** boxes. 2. To add localized values to the properties, set the **localizable** property constant to **true** for the properties. 3. Use the rich client to add the localized values.

Examples of dynamic LOVs

Dynamic LOV example 1: Parts owned by a group

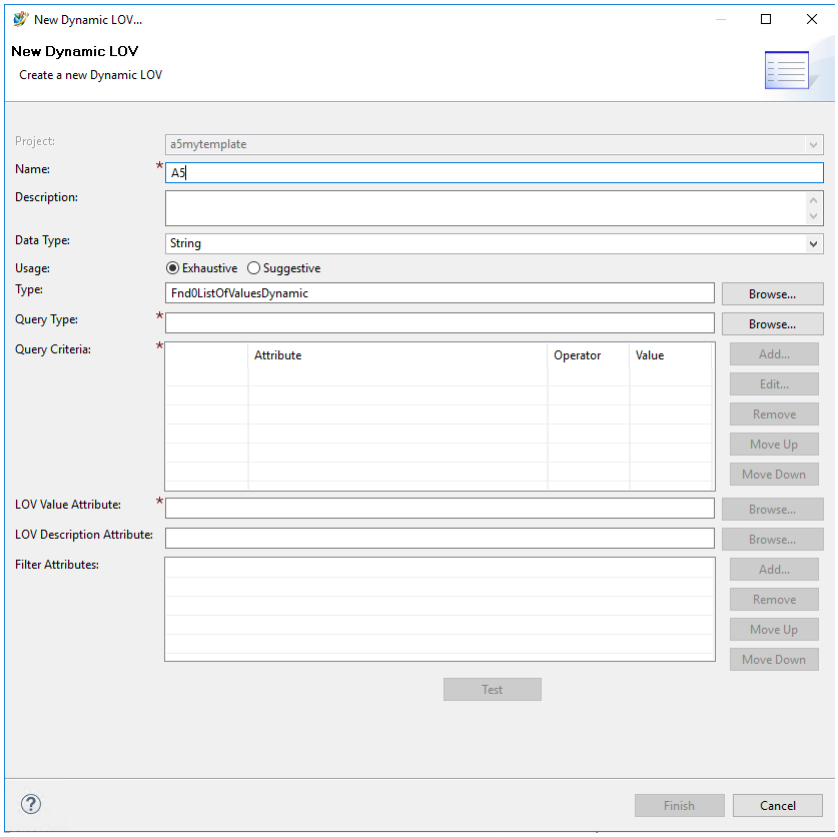
This dynamic LOV example queries for all instances of a custom **Part** business object owned by the **Engineering** group.

© 2025 Siemens

To perform this query, first use the **owning_group** type reference property on the custom **Part** business object (for example, **A5_MyPart**), and then navigate to the **Group** business object and form a query on the **name** property.

1. To create a dynamic LOV, right-click the **Dynamic LOV** folder and choose **New Dynamic LOV**.

   The **New Dynamic LOV** dialog box is displayed.

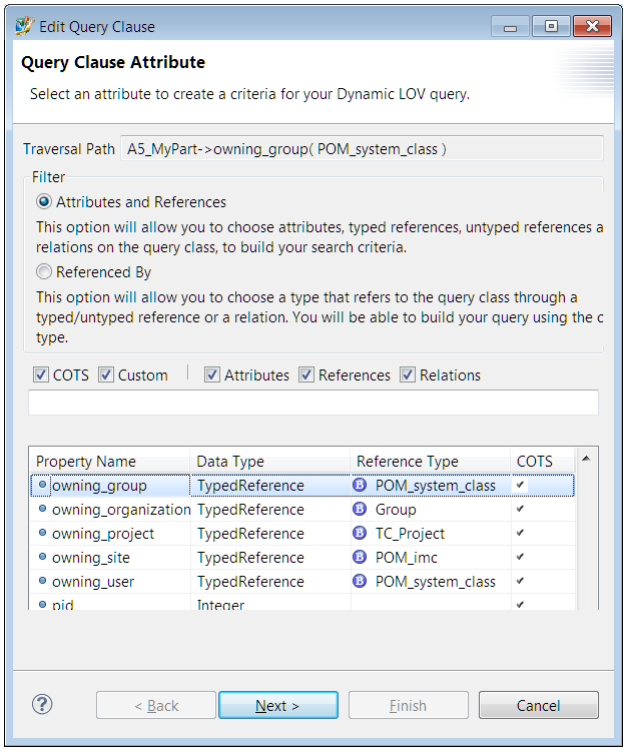


2. In the **Query Type** box, select the custom **Part** business object (for example, **A5_MyPart**).

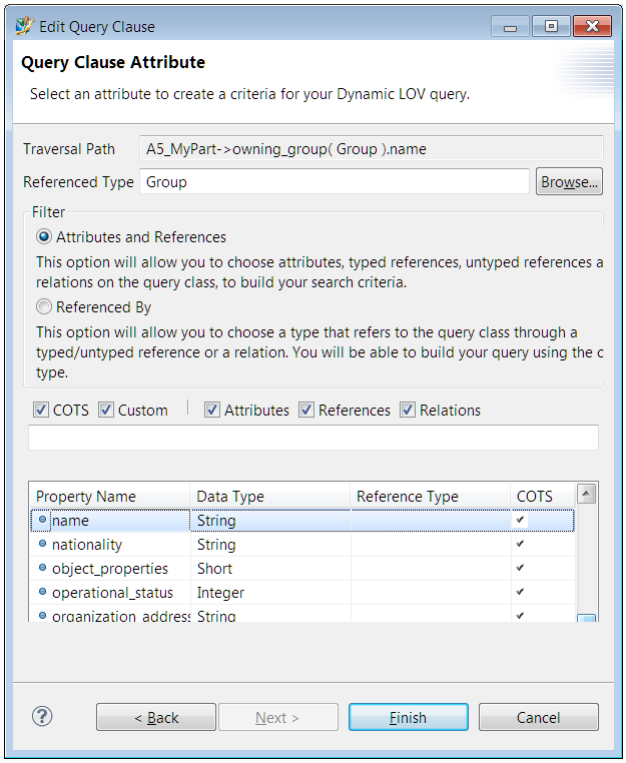
3. Click the **Add** button to the right of the **Query Criteria** box.

4. In the **Query Clause Attribute** dialog box, select the **owning_group** property and click **Next**.

© 2025 Siemens



5. In the next **Query Clause Attribute** dialog box, click the **Browse** button to the right of the **Referenced Type** box and select the **Group** business object. Then select the **name** property from the table. Click **Finish**.



© 2025 Siemens

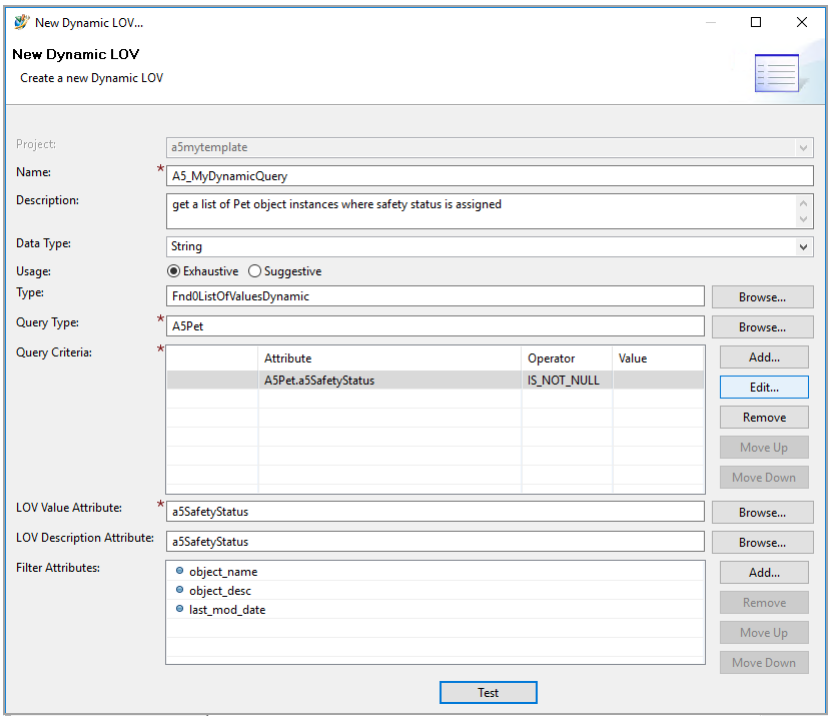
6. In the **Value** column of the **Query Criteria** table, type **Engineering**. The query now looks like this:

```
A5_MyPart->owning_group(Group).name = Engineering
```

7. Add properties to define the columns that appear on the LOV table in the end-user interface. Following is a suggestion for the properties to include.

Box	Properties
LOV Value Attribute	**object_name**
LOV Description Attribute	**object_desc**
Filter Attributes	**item_id** **object_type** **last_mod_date**

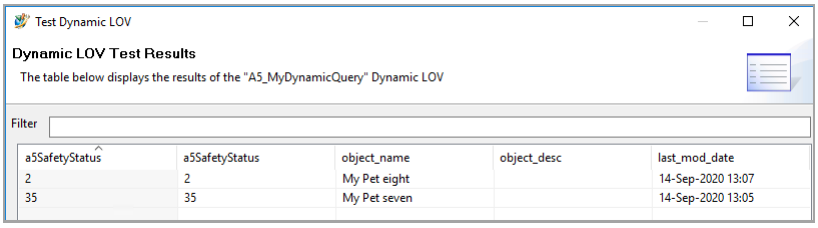
The dynamic LOV looks like the following figure.



8. Click the **Test** button.

The results of the query are displayed. This table shows the LOV values as they appear in the end-user interface.

© 2025 Siemens



Dynamic LOV example 2: Datasets with Rendering relation to an object

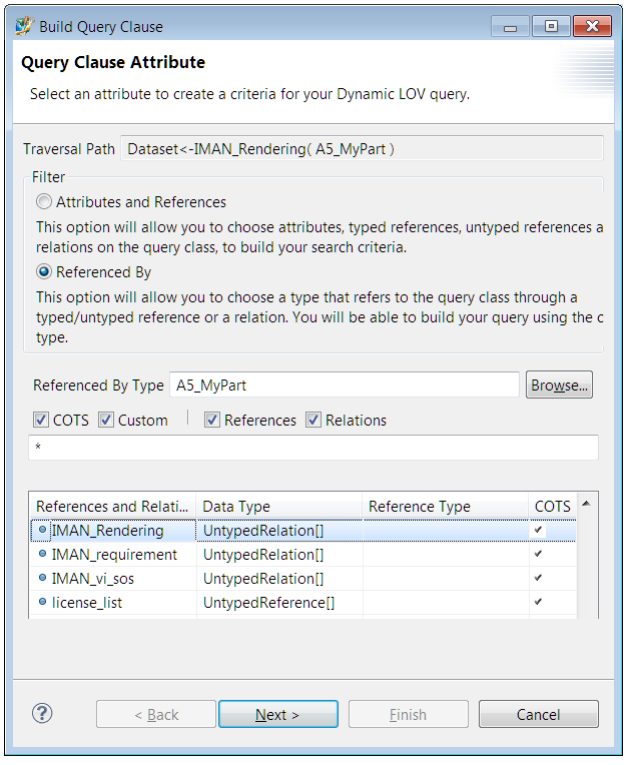
This dynamic LOV example shows all datasets related with the **Rendering** relation to an instance of the custom **A5_MyPart** business object named **Engine**.

To perform this query, choose the query type as **Dataset**, traverse to the **A5_MyPart** business object using the **IMAN_Rendering** relation, and set the **obect_name** value as **Engine**.

1. In the **Query Type** box, select the **Dataset** business object.

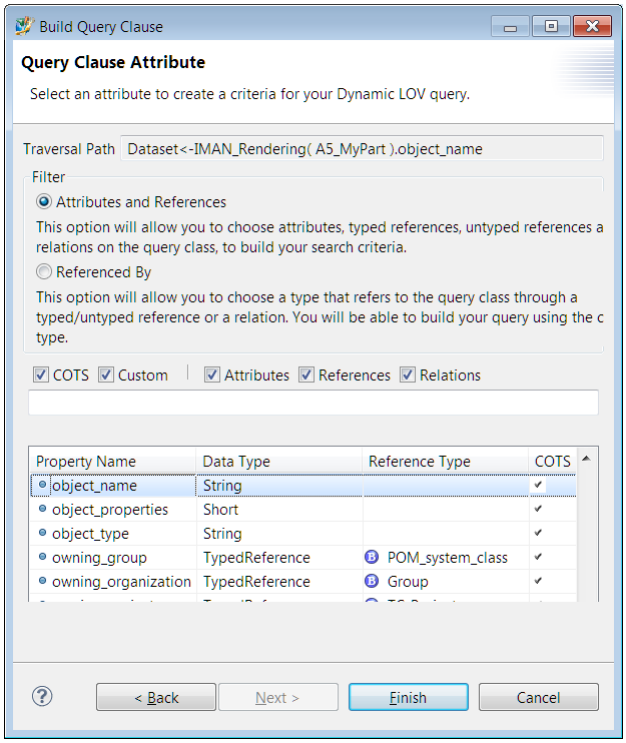
2. Click the **Add** button to the right of the **Query Criteria** box.

3. In the **Query Clause Attribute** dialog box, select **Referenced By**. Then in the **Referenced By Type** box, select the **A5_MyPart** business object and select **IMAN_Rendering** in the table. Click **Next**.



4. In the next **Query Clause Attribute** dialog box, select the **object_name** property in the table. Click **Finish**.

© 2025 Siemens



5. In the **Value** column of the **Query Criteria** table, type **Engine**. The query now looks like this:

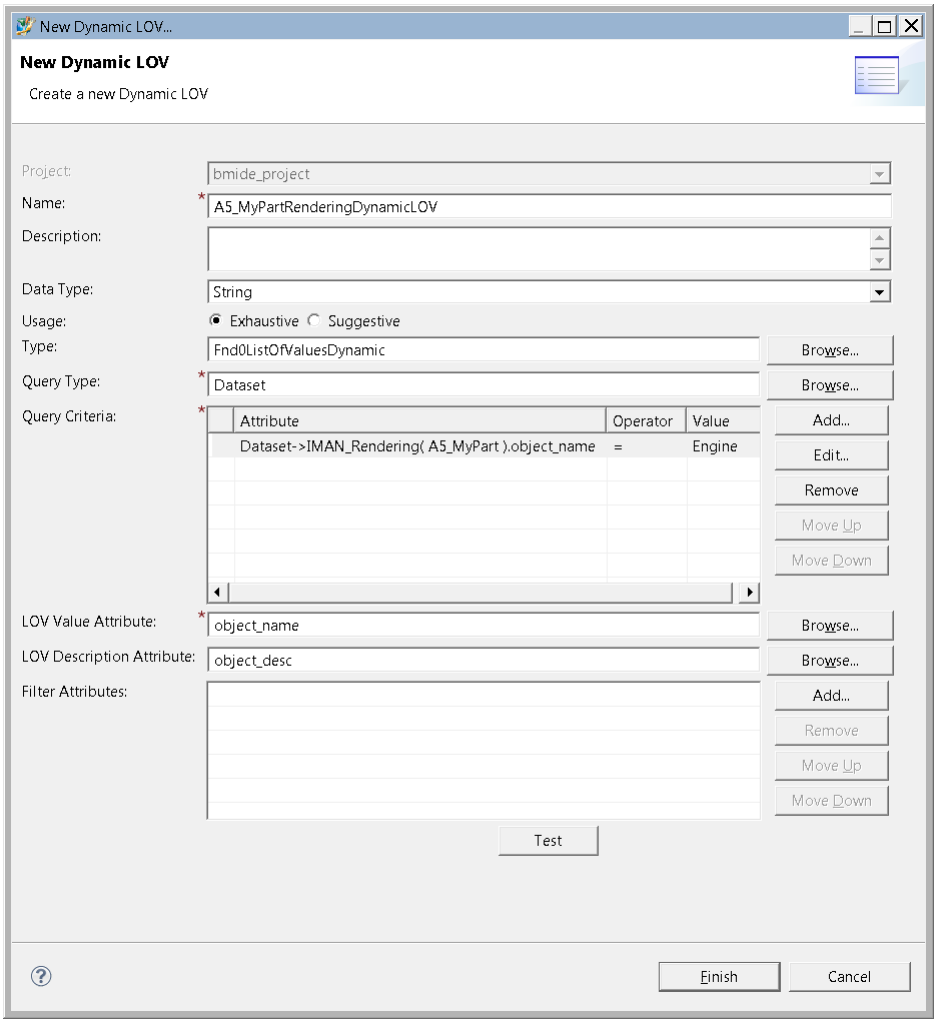
```
Dataset<-IMAN_Rendering(A5_MyPart).object_name = Engine
```

6. Add properties to define the columns that appear on the LOV table in the end-user interface. Following is a suggestion for the properties to include.

Box	Properties
LOV Value Attribute	**object_name**
LOV Description Attribute	**object_desc**

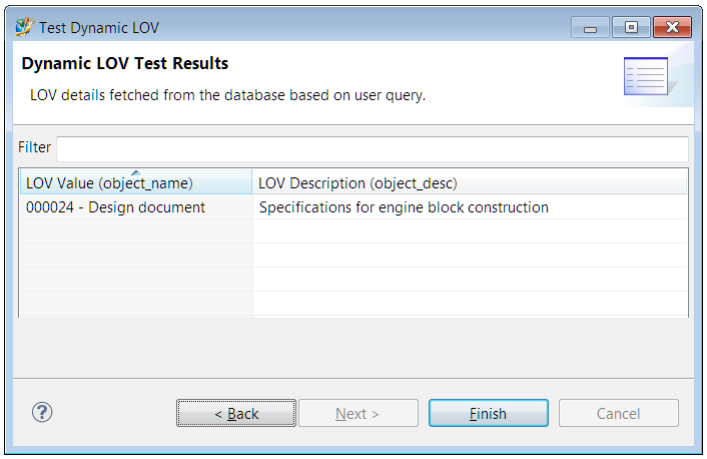
The dynamic LOV looks like the following figure.

© 2025 Siemens



7. Click the **Test** button.

   The results of the query are displayed. This table shows the LOV values as they appear in the end-user interface.



© 2025 Siemens

Dynamic LOV example 3: Datasets with Rendering relation to parts owned by a group

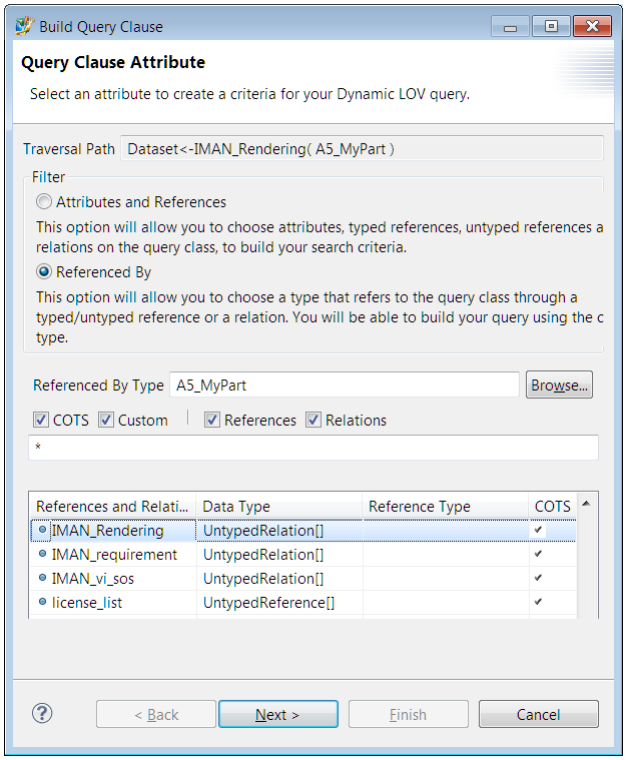
This dynamic LOV example shows all datasets related with the **Rendering** relation to **A5_MyPart** business object instances owned by the **Engineering** group.

This query is similar to **Example 2**. Instead of a simple attribute on the **A5_MyPart** business object, use the **owning_group** property to traverse to the **Group** business object on which you define the **name** value as **Engineering**.

1. In the **Query Type** box, select the **Dataset** business object.

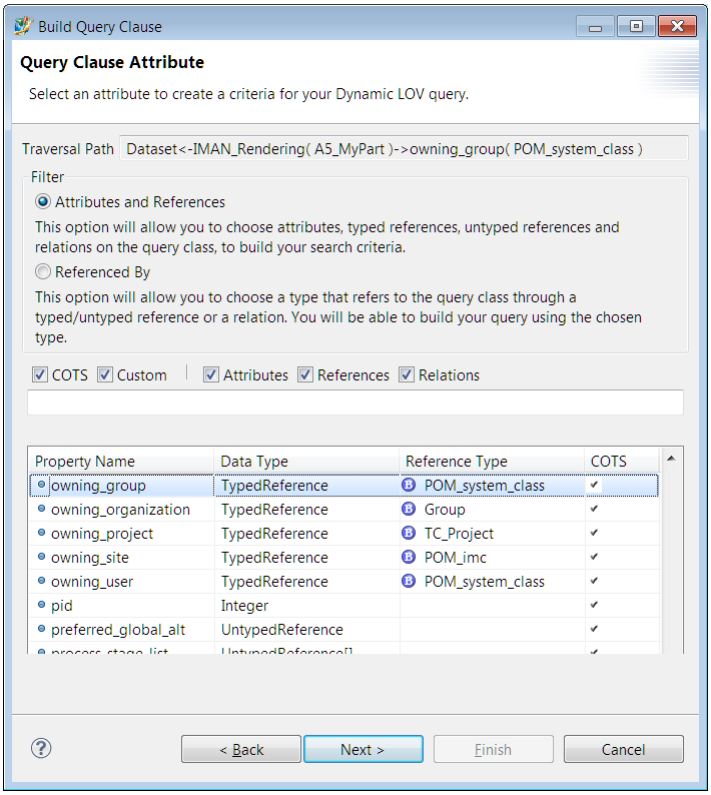
2. Click the **Add** button to the right of the **Query Criteria** box.

3. In the **Query Clause Attribute** dialog box, select **Referenced By**. Then in the **Referenced By Type** box select the **A5_MyPart** business object and select **IMAN_Rendering** in the table. Click **Next**.



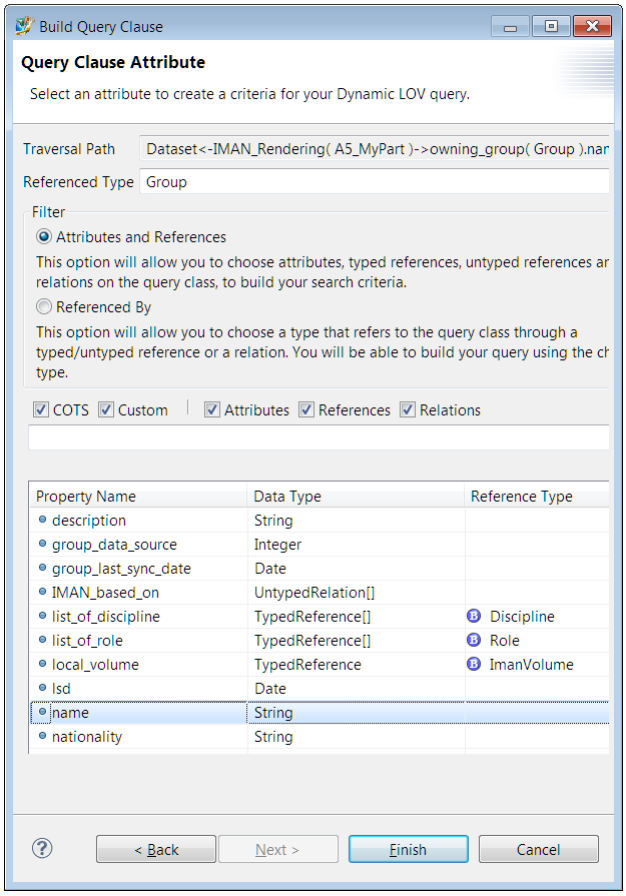
4. In the next **Query Clause Attribute** dialog box, select the **owning_group** property and click **Next**.

© 2025 Siemens



5. In the next **Query Clause Attribute** dialog box, click the **Browse** button to the right of the **Referenced Type** box and select the **Group** business object. Then select the **name** property from the table. Click **Finish**.

© 2025 Siemens



6. In the **Value** column of the **Query Criteria** table, type **Engine**. The query now looks like this:

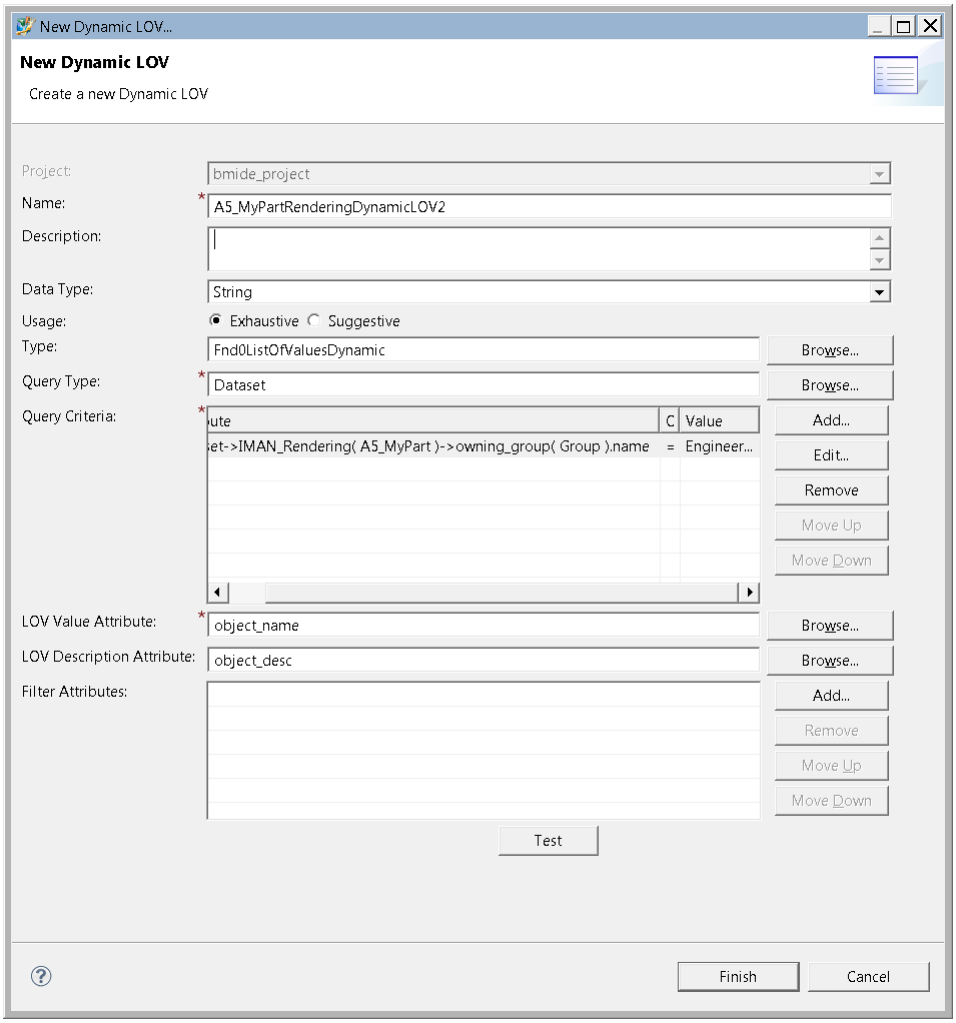
```
Dataset<-IMAN_Rendering(A5_MyPart)->owning_group(Group).name =
Engineering
```

7. Add properties to define the columns that appear on the LOV table in the end-user interface. Following is a suggestion for the properties to include.

Box	Properties
LOV Value Attribute	**object_name**
LOV Description Attribute	**object_desc**

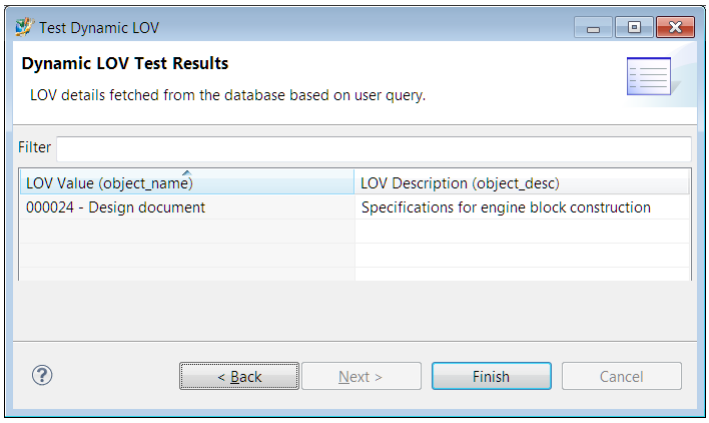
The dynamic LOV looks like the following figure.

© 2025 Siemens



8. Click the **Test** button.

   The results of the query are displayed. This table shows the LOV values as they appear in the end-user interface.



© 2025 Siemens

Dynamic LOV example 4: Name and ID of group role members

This example shows how to create a dynamic LOV of group member names, and their user IDs, that match the LOV's query for a specified group and role.

The user_name value of this dynamic LOV could be attached to an object property display attribute to provide an end user with an easy pick list of names, and simultaneously the user_id value could be attached to an object property functional attribute to provide a needed user ID.

1. To create a dynamic LOV, right-click the **Dynamic LOV** folder and choose **New Dynamic LOV**.

2. In the **New Dynamic LOV** dialog box, enter parameter values.

For this Parameter	Do this
Name	Enter a name for the list. The name must start with your project prefix.
Description	Enter a description for the list.
Data Type	Accept the default value, **String**.
Usage	Accept the default value, **Exhaustive**.
Type	Accept the default value, **Fnd0ListOfValuesDynamic**.
Query Type	Choose **User**.
LOV Value Attribute	Choose **user_name**.
LOV Description Attribute	Choose **user_id**.

3. In **Query Criteria**, add a query term that finds all users who are members of a specified group.

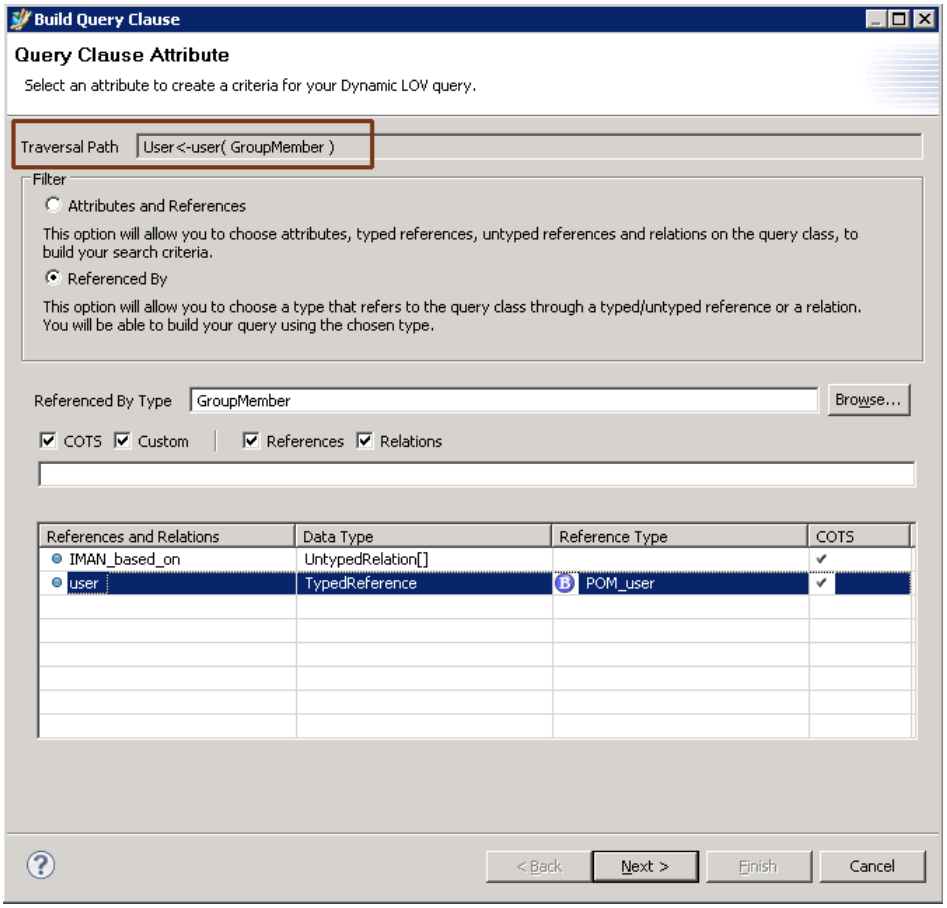
   a. Click **Add**.

   b. In the **Build Query Clause** dialog box, under **Filter**, choose **Referenced By**.

   c. In **Referenced by Type**, choose the **GroupMember** business object.

   d. In the **References and Relations** list, select **user** and then click **Next**.

© 2025 Siemens

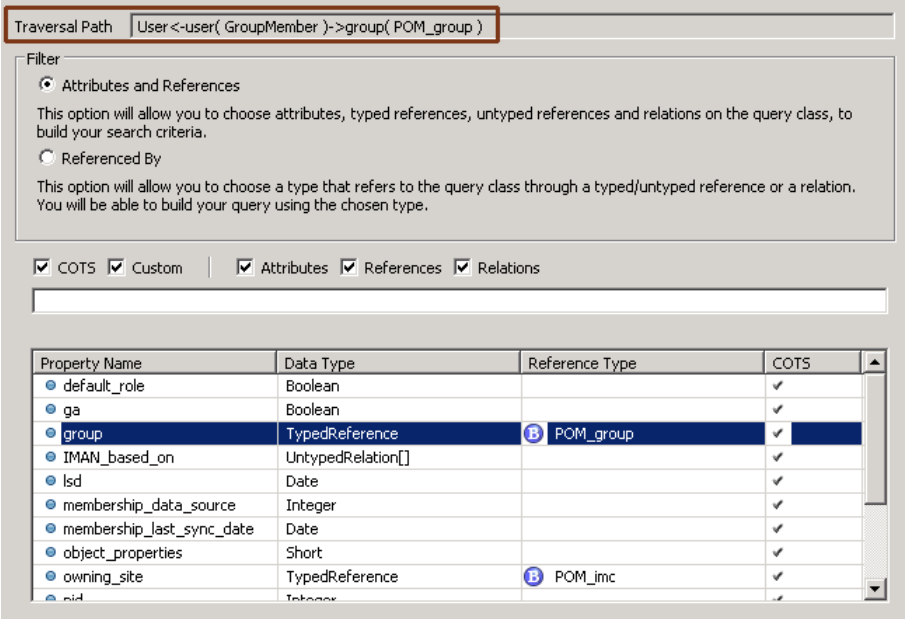


You have now established the first part of the **Traversal Path**.

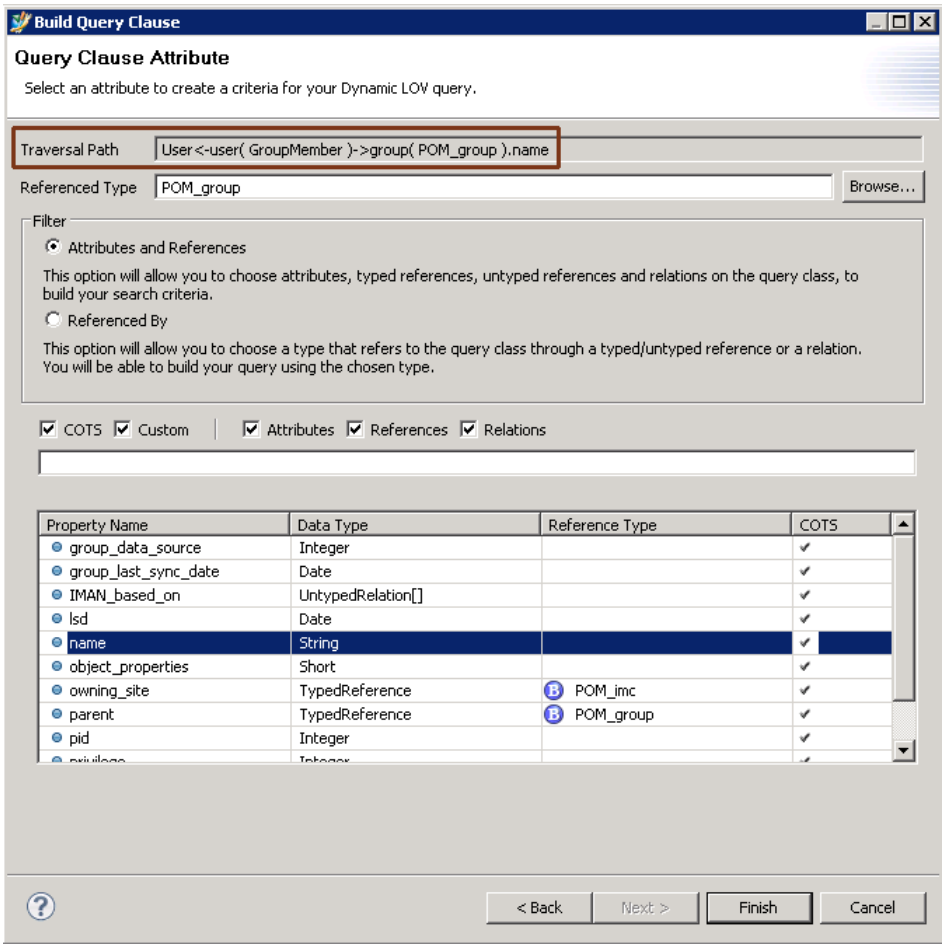
e. To establish the second part of the traversal path, in the **Build Query Clause** dialog box, under **Filter**, accept the default value **Attributes and References**.

f. In **Property Name** select **Group** and then click **Next**.

© 2025 Siemens

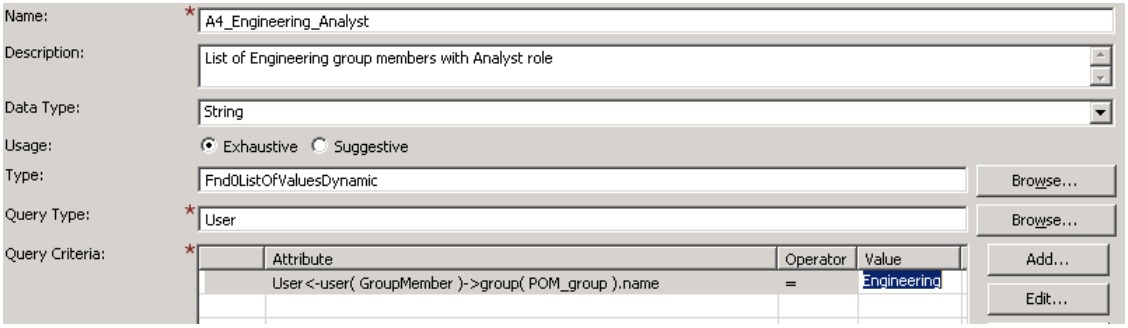


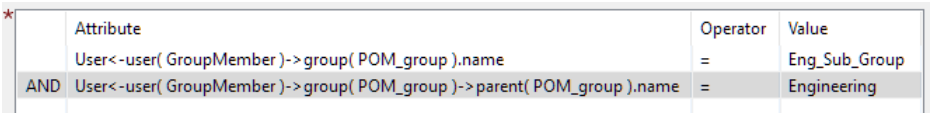
g. In the **Property Name** list, select **name** and then click **Finish**.



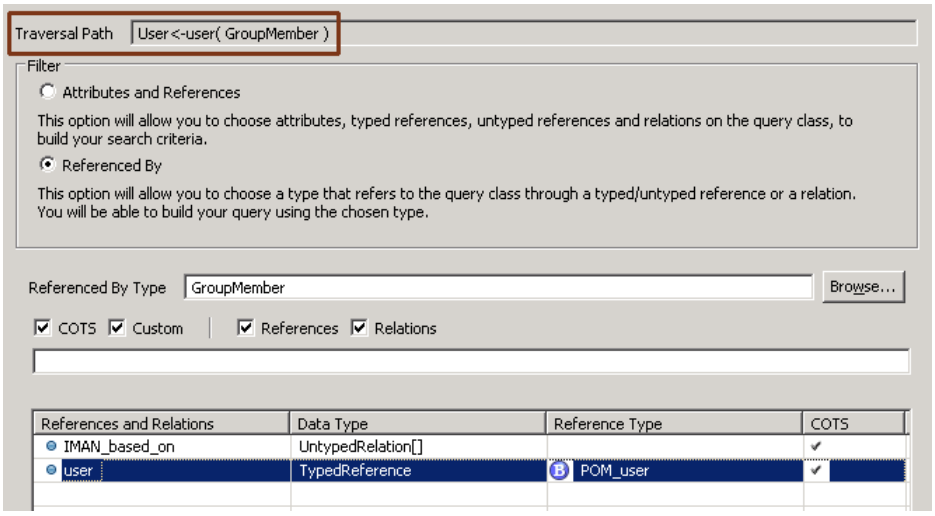
© 2025 Siemens

h. In the **New Dynamic LOV** dialog box, for the new **Query Criteria** attribute you just added, in the **Value** column, type **Engineering**.

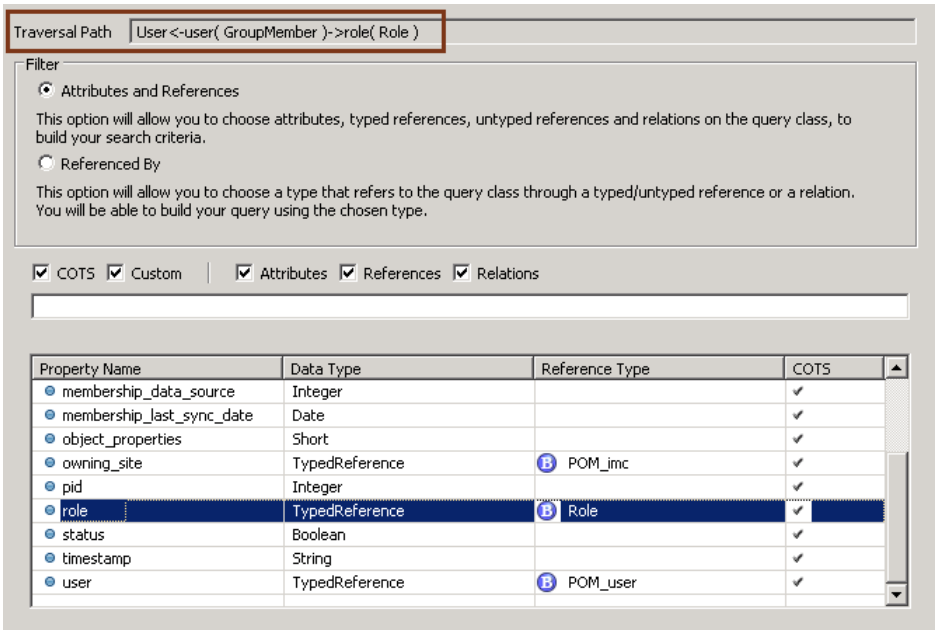


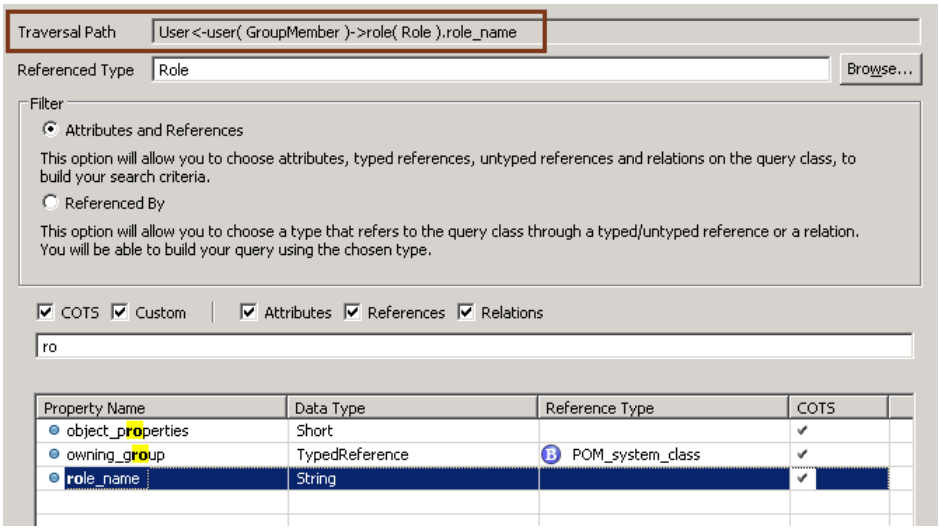
> Tip:
>
> If your organization structure includes sub groups, you could use additional query criteria for each level of the multilevel group structure. The following example checks for the first and second levels:
>
> 
>
>
> For each additional level, add one more `->parent(POM_group)` to the criteria. For example, to check the third level group, use the following line:
>
> ```
> User<-user(GroupMember)->group(POM_group)->parent(POM_group)-
> >parent(POM_group).name
> ```

i. Click **Add** and add another query attribute. The second query term finds all group members with a specified role.



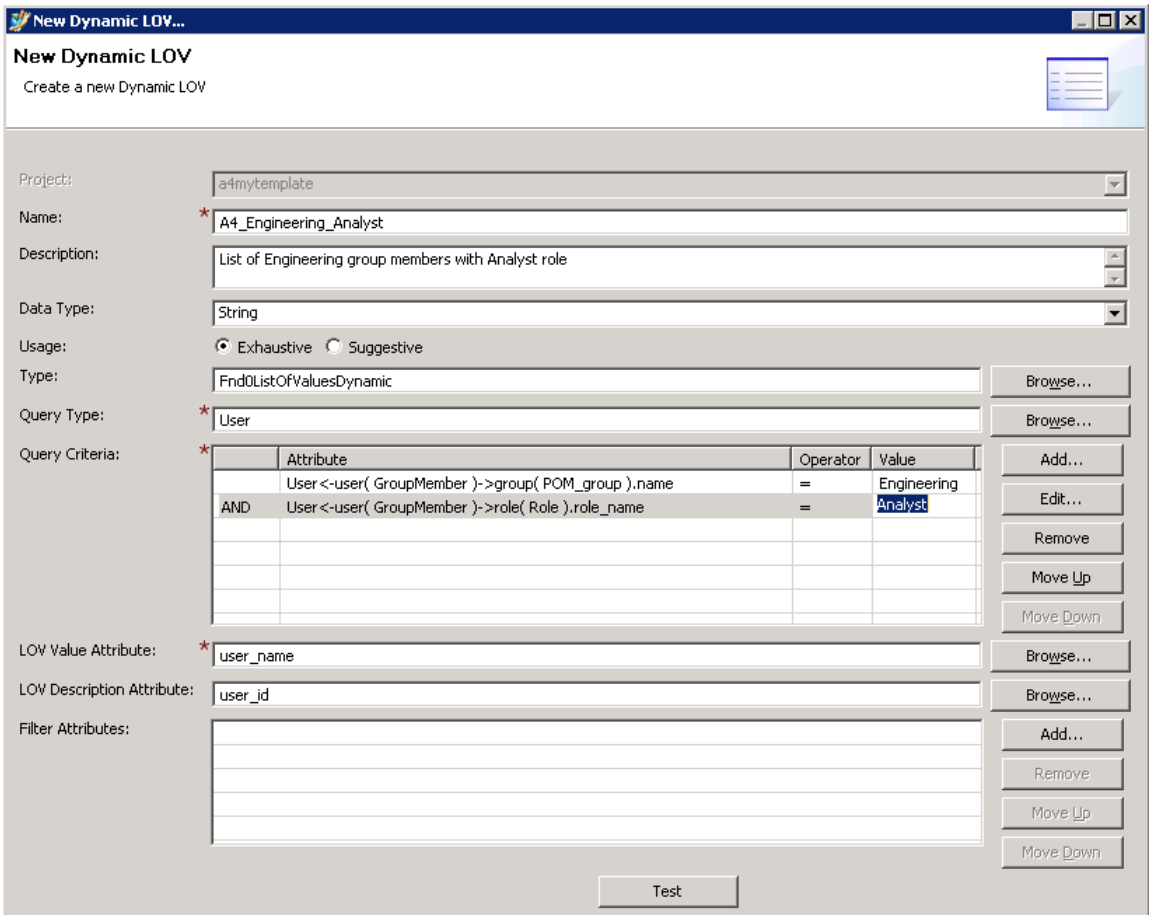
© 2025 Siemens





j. For the role_name value, type **Analyst**.

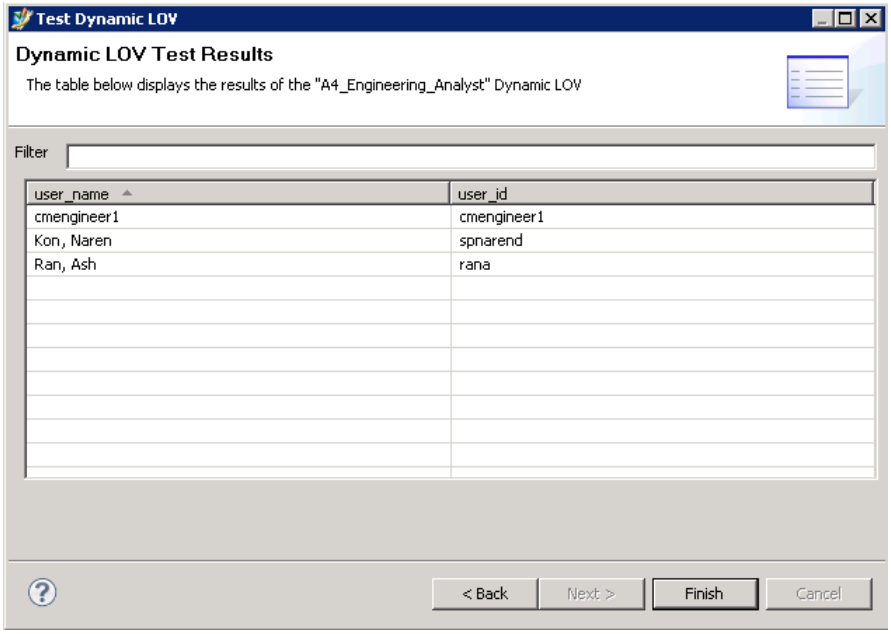
© 2025 Siemens



Test the dynamic LOV query

1. To test your new dynamic LOV definition, click **Test**.
2. If prompted, enter login credentials and click **Connect** to login to Teamcenter.
3. In **Server Profile**, ensure the correct server is selected, and then click **Next**.

© 2025 Siemens



Dynamic LOV example 5: Property of data imported from external source

Sometimes you may have instances of objects stored in a system other than Teamcenter, and you want to make the properties of those objects available for queries in dynamic lists of values. To solve this problem, you can use TC XML to import the properties information for these objects into lists of values.

This method uses the **Administrative List Of Values** (**Fnd0AdminLOVValue**) business object in Teamcenter to hold the lists of values. By default, this business object type has the following properties:

fnd0lov_category
fnd0lov_description
fnd0lov_value

You can also extend this business object with your own custom properties.

> Tip:
>
> Rather than importing lists of values, you can also create instances of the **Fnd0AdminLOVValue** business object in clients by choosing **File→New→Other** and selecting **Administrative List Of Values**. You can also use Teamcenter services to create **Fnd0AdminLOVValue** objects using your own client application.

Perform the following steps to configure and perform the import:

1. To import instances of the **Fnd0AdminLOVValue** business object, create a new XML file using the following sample TC XML file.

© 2025 Siemens

For example, you would like to store LOV values for a **Colors** LOV and a **Materials** LOV. The **Colors** LOV contains values **Red**, **Green**, and **Blue**. The **Materials** LOV contains values **Aluminum**, **Copper**, and **Zinc**.

In the following TC XML example, the **fnd0lov_*** properties are used to hold LOV data. Each entry uses the same **object_name**, which is the name of the list of values that the values belong to.

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<TCXML xmlns="http://www.tcxml.org/Schemas/TCXMLSchema" format="high_level">

<Fnd0AdminLOVValue creation_date="2013-01-18T05:32:47Z" date_released=""
ead_paragraph="" elemId="id1"
fnd0lov_value="Red"
fnd0lov_description="This is the color red"
fnd0lov_category="ColorsLOV"
gov_classification="" ip_classification="" last_mod_date="2013-01-18T05:32:47Z"
license_list="" object_desc=""
object_name="colorslov_red" owning_group="#id14" owning_organization=""
owning_project="" owning_site="#id16" owning_user="#id18" project_list=""
release_status_list="" revision_number="0">
  <GSIdentity elemId="id2" label="B1J9Wu4Q6oxYUD"/>
</Fnd0AdminLOVValue>

<Fnd0AdminLOVValue creation_date="2013-01-18T05:32:47Z" date_released=""
ead_paragraph="" elemId="id3"
fnd0lov_value="Green"
fnd0lov_description="This is the color green"
fnd0lov_category="ColorsLOV"
gov_classification="" ip_classification=""
last_mod_date="2013-01-18T05:32:47Z" license_list="" object_desc=""
object_name="colorslov_green" owning_group="#id14" owning_organization=""
owning_project="" owning_site="#id16" owning_user="id18" project_list=""
release_status_list="" revision_number="0">
  <GSIdentity elemId="id4" label="B1J9Wu4Q6oxYUE"/>
</Fnd0AdminLOVValue>

<Fnd0AdminLOVValue creation_date="2013-01-18T05:32:47Z" date_released=""
ead_paragraph="" elemId="id5"
fnd0lov_value="Blue"
fnd0lov_description="This is the color blue"
fnd0lov_category="ColorsLOV"
gov_classification="" ip_classification=""
last_mod_date="2013-01-18T05:32:47Z" license_list="" object_desc=""
object_name="colorslov_blue" owning_group="#id14" owning_organization=""
owning_project="" owning_site="#id16" owning_user="id18" project_list=""
release_status_list="" revision_number="0">
  <GSIdentity elemId="id6" label="A1J9Wu4Q6oxYUF"/>
</Fnd0AdminLOVValue>

<Fnd0AdminLOVValue creation_date="2013-01-18T05:32:47Z" date_released=""
ead_paragraph="" elemId="id7"
fnd0lov_value="Aluminum"
fnd0lov_description="This is aluminum material"
fnd0lov_category="MaterialsLOV"
gov_classification="" ip_classification="" last_mod_date="2013-01-18T05:32:47Z"
license_list="" object_desc="" object_name="materialslov_aluminum"
```


© 2025 Siemens

```
owning_group="#id14"
owning_organization="" owning_project="" owning_site="#id16" owning_user="id18"
project_list="" release_status_list="" revision_number="0">
  <GSIdentity elemId="id8" label="A1J9Wu4Q6oxYUG"/>
</Fnd0AdminLOVValue>

<Fnd0AdminLOVValue creation_date="2013-01-18T05:32:47Z" date_released=""
ead_paragraph="" elemId="id9"
fnd0lov_value="Copper"
fnd0lov_description="This is copper material"
fnd0lov_category="MaterialsLOV"
gov_classification="" ip_classification="" last_mod_date="2013-01-18T05:32:47Z"
license_list="" object_desc="" object_name="materialslov_copper"
owning_group="#id14"
owning_organization="" owning_project="" owning_site="#id16" owning_user="#id18"
project_list="" release_status_list="" revision_number="0">
  <GSIdentity elemId="id10" label="A1J9Wu4Q6oxYUH"/>
</Fnd0AdminLOVValue>

<Fnd0AdminLOVValue creation_date="2013-01-18T05:32:47Z" date_released=""
ead_paragraph="" elemId="id11"
fnd0lov_value="Zinc"
fnd0lov_description="This is zinc material"
fnd0lov_category="MaterialsLOV"
gov_classification="" ip_classification=""
last_mod_date="2013-01-18T05:32:47Z" license_list="" object_desc=""
object_name="materialslov_zinc" owning_group="#id14" owning_organization=""
owning_project="" owning_site="#id16" owning_user="#id18" project_list=""
release_status_list="" revision_number="0">
  <GSIdentity elemId="id12" label="A1J9Wu4Q6oxYUI"/>
</Fnd0AdminLOVValue>

<Group elemId="id13" list_of_role="#id18" name="dba" object_full_name="dba"
owning_site="#id16" transient_island_id="1">
      <GSIdentity elemId="id14" label="AcZBPS6IYtcuYD"/>
</Group>
<POM_imc elemId="id15" owning_site="#id16" site_id="-2049080462"
transient_island_id="0">
      <GSIdentity elemId="id16" label="w9TBPSppYtcuYD"/>
</POM_imc>
<User elemId="id17" owning_site="#id16" transient_island_id="1" user_id="tc_admin_1">
      <GSIdentity elemId="id18" label="AkVBPS6IYtcuYD"/>
</User>
<Role creation_date="2013-01-18T05:32:47Z" elemId="id19"
last_mod_date="2013-01-18T05:32:47Z" owning_group="#id14" owning_site="#id16"
owning_user="#id18" role_name="DBA" transient_island_id="0">
      <GSIdentity elemId="id20" label="AYaBPS6IYtcuYD"/>
</Role>

<Header author="tc_admin_1" date="2013-02-26" elemId="id21"
originatingSite="-2049080462"
targetSite="-2046041493" time="15:55:18"
version="Teamcenter P10000.1.0.20130206.00">
  <TransferFormula elemId="id22">
  <TransferMode elemId="id23" gov_classification="" ip_classification=""
license_list=""
object_name="TIEExportDefaultTM"
owning_site="id16" project_list=""/>
```


© 2025 Siemens

```
  <Reason elemId="id24"></Reason>
  </TransferFormula>
  <TraverseRootRefs>#id1 #id3 #id5 #id7 #id9 #id11</TraverseRootRefs>
  </Header>
</TCXML>
```

> Note:
>
> In the preceding XML file, the value of the **elemId** attribute (for example, **elemId="id1234"**) and the **label** attribute (for example, **label="abcd"**) should be unique throughout the file. (The **GSID.label** parameter holds the UID of the object. If the external system cannot provide valid UIDs, you can provide a unique string for the label. The label string must be at least 15 characters long. This enables the TC XML import to generate UIDs as part of the import.)
>
> The value of the **site_id**, **originatingSite**, and **targetSite** attributes must be updated according to the current database. The value of the site attributes is the value of the **site_id** attribute of the **POM_imc** class. (Use the **site_util** utility with the **-f=list** argument to obtain the ID of the database.)

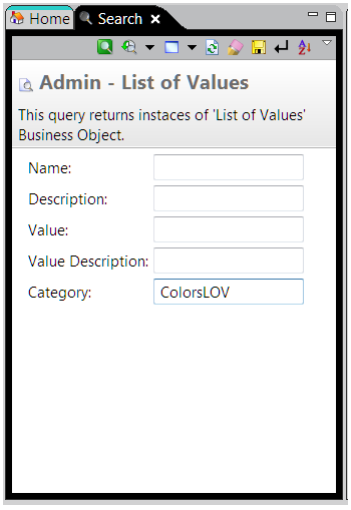
2. Import the custom XML file to Teamcenter using the **tcxml_import** utility, for example:

   ```
   tcxml_import -u=admin-username -p=admin-password -g=dba
   -file=custom-XML-file
   ```

   The following message is displayed:

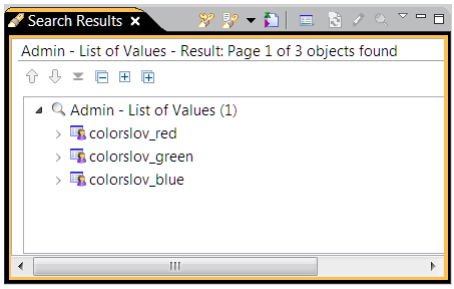
   ```
   Executing high level mode import
   ```

3. To verify that the objects are imported to Teamcenter, in the rich client, use the **Admin - List of Values** saved search, which searches for **Fnd0AdminLOVValue** business object instances.



Using that saved query and providing the **Category** as **ColorsLOV** displays all the imported instances of the **Fnd0AdminLOVValue** business object instances matching that category.

© 2025 Siemens

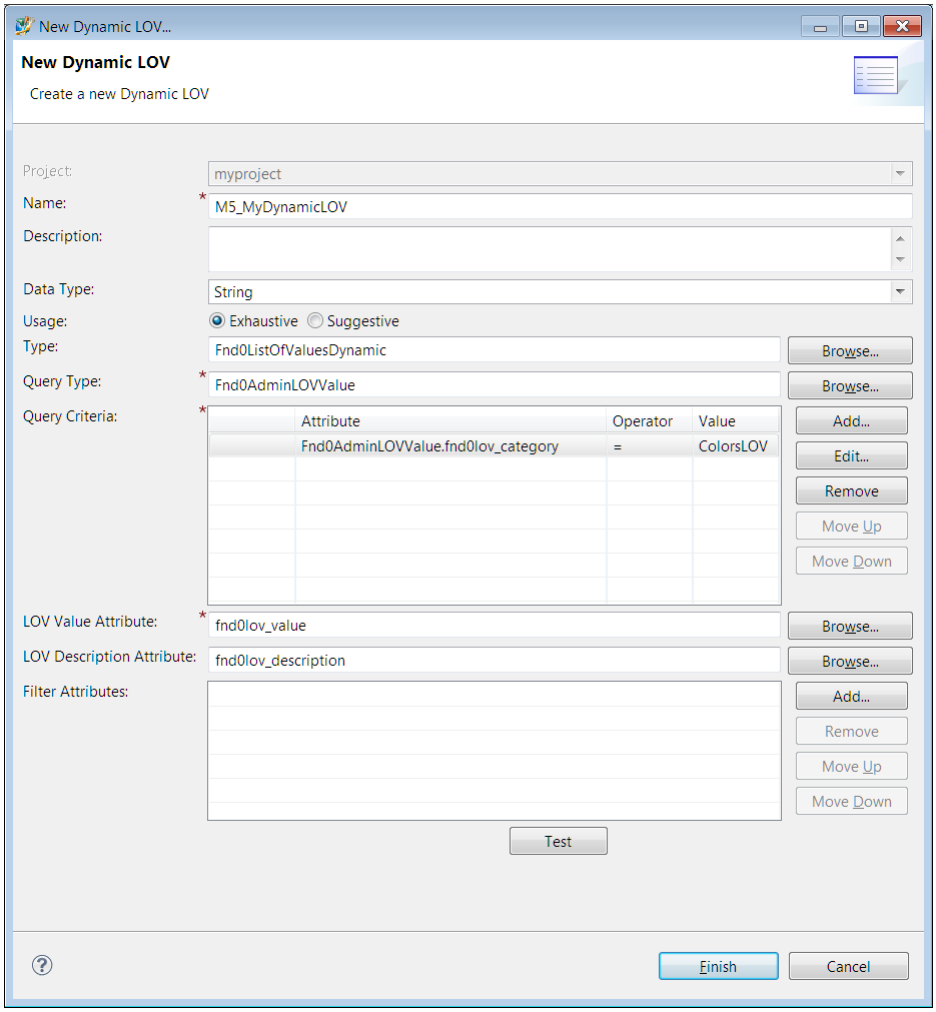


4. Now that instances of the **Fnd0AdminLOVValue** business object are created in Teamcenter, you can use the **Fnd0AdminLOVValue** business object as part of your dynamic LOV definition.

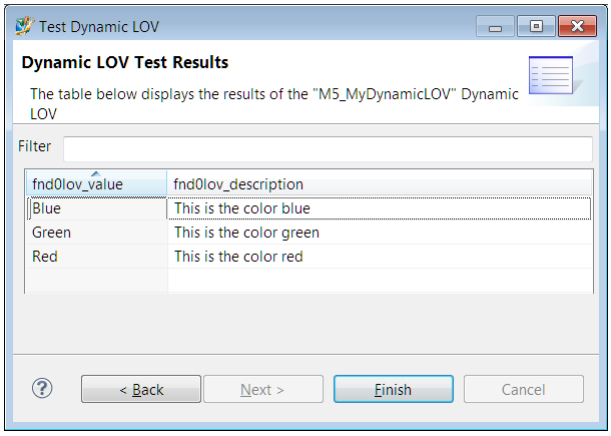
   a. Create the dynamic list of values using the **Fnd0AdminLOVValue** business object in the **Query Type** box.

      For example, the following figure shows a dynamic list of values that uses the instances of the **Fnd0AdminLOVValue** business object imported to the database using TC XML. (Notice how the query criteria asks for a **Fnd0AdminLOVValue** instance having the **fnd0lov_category="Plastic"** value.)

© 2025 Siemens



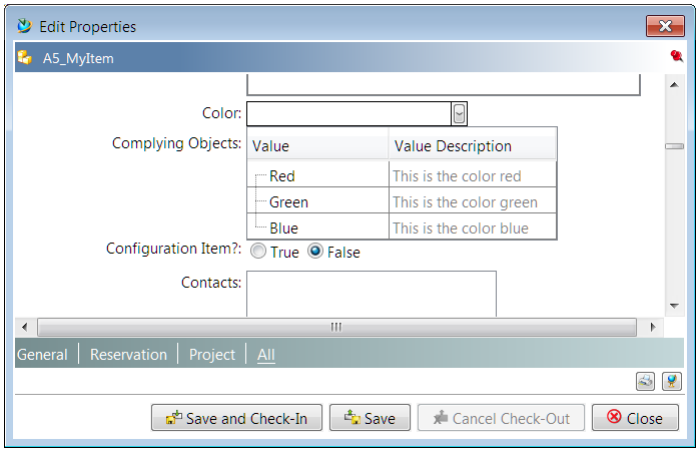
Click the **Test** button to see the imported list of values.



b. Attach the dynamic list of values to a property and deploy it.

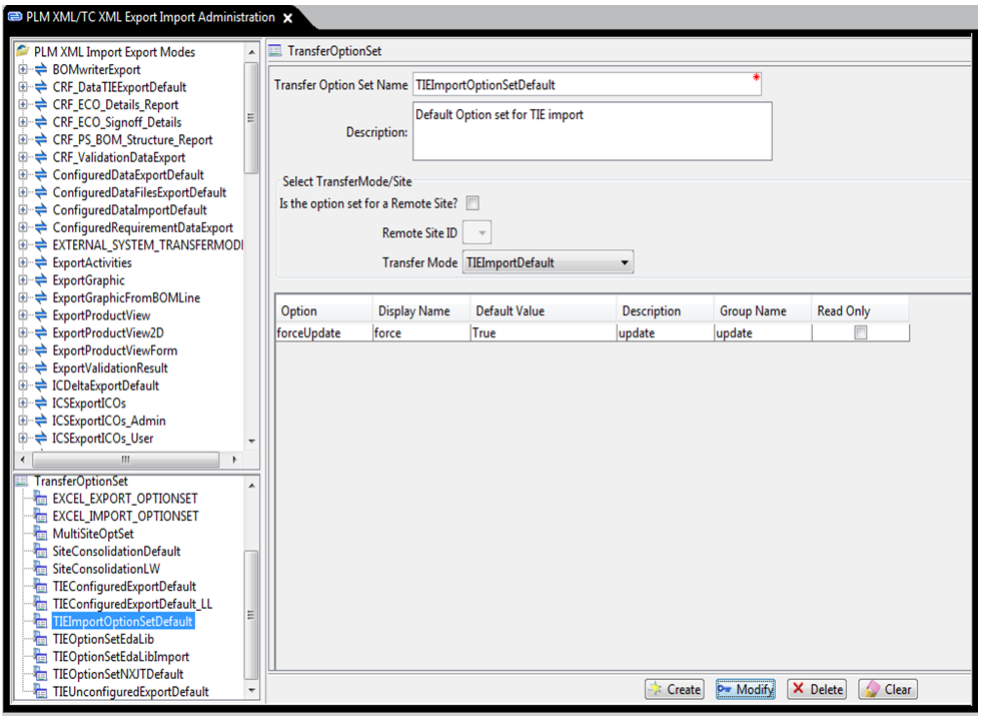
In the following example, the dynamic LOV is attached to a property named **Material**.

© 2025 Siemens



5. Property value of the instances imported to the database in the previous step can also be updated through the **tcxml_import** utility:

   a. In the rich client, open PLM XML/TC XML Export Import Administration and configure it as shown:



   b. After creating the new **forceUpdate** option in the **TIEImportOptionSetDefault** transfer option set, run the following command to update the values in the database:

```
tcxml_import -u=admin-username -p=admin-password -g=dba
-optionset=TIEImportOptionSetDefault  -file=modified-XML-file-path
```


© 2025 Siemens

Tip:

If you add your own custom properties on the **Fnd0AdminLOVValue** business object, you must add these properties to the *TC_ROOT***data\defaultTcxmlScopeRules.xml** file in the **TIEExportDefaultPS** property set, similar to how the **fnd0lov_category**, **fnd0lov_value**, and **fnd0lov_description** properties are added.

Execute the **tcxml_import** utility as follows:

```
tcxml_import -u=admin-username -p=admin-password -g=dba
 -scope_rules -scope_rules_mode=overwrite -file=defaultTcxmlScopeRules.xml
```

The following message is displayed:

```
Executing high level mode import
```

Thereafter, you can run the **tcxml_import** utility to load instances specified in your TC XML file.

Attach an LOV to a property

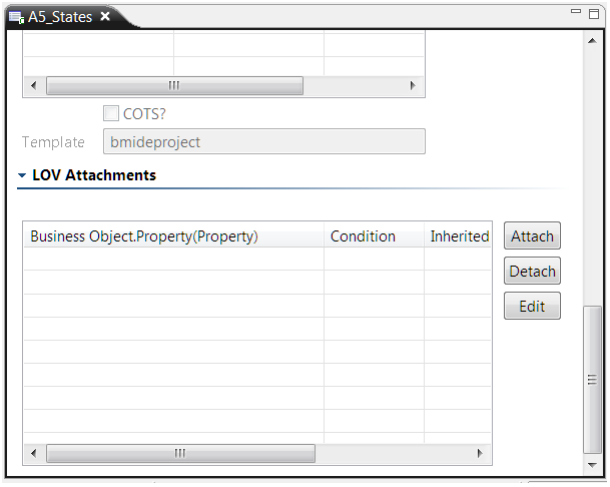
To display an LOV in the user interface, you must attach it to a property on a business object. For example, if you want to use an LOV to list possible descriptions for an item, attach the LOV to the **object_desc** property of the **Item** business object.

Note:

- To attach an LOV to a property on a form business object, attach the LOV to the same property on the form's storage business object. The form's storage business object serves as the storage class for the properties. For example, if you create a custom form named **A5_MyForm**, attach the LOV to the same property on the **A5_MyFormStorage** business object. Unless you do this, the LOV is not attached to the property and does not display in the end-user interface.
- Attaching lists of values to the properties of the **ListofValues** business object or to any of its children business objects is not supported. If you have attached lists of values to the properties of the **ListofValues** business object or to any of its children business objects in your custom template, you must delete the attachments prior to using the custom template in the upgrade process.

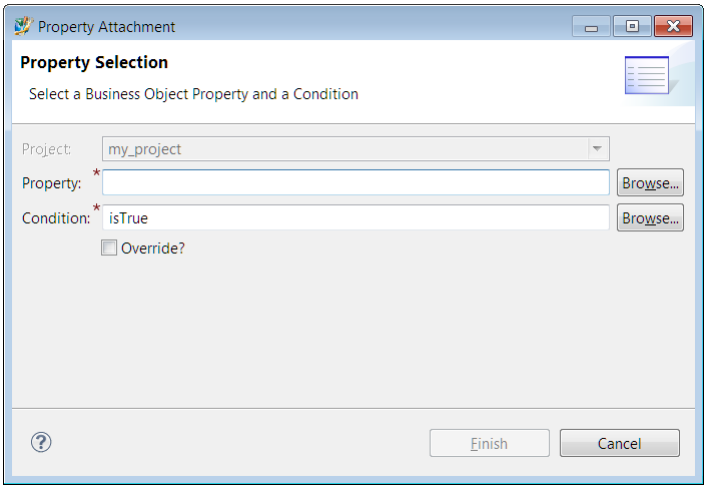
1. Expand the **Extensions** folder until you see the **LOV** folder is displayed.
2. In the **Batch LOV**, **Classic LOV**, or **Dynamic LOV** folder, right-click the LOV you want to attach and choose **Open**. The LOV details appear in a new view.
3. Scroll to the bottom of the view to the **LOV Attachments** table.

© 2025 Siemens



4. Click the **Attach** button under the **LOV Attachments** heading.

   The **Property Attachment** dialog box is displayed.



5. Perform the following steps in the **Property Attachment** dialog box:

   a. Click the **Browse** button to the right of the **Property** box and choose the property to which you want to attach the LOV.

   b. Click the **Browse** button to the right of the **Condition** box if you want the value to be available only if a certain condition applies. If you select **isTrue** as the condition, the value always applies.

   > Note:
   >
   > Only those conditions appear that have valid signatures. For LOV attachments, the valid condition signature is as follows:

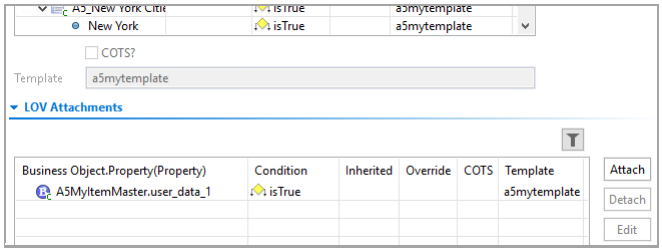
© 2025 Siemens

condition-name(**UserSession**)

c. Select the **Override** check box if you want to override attachment of the parent business object. Override can be set only with **isTrue** condition.

d. Click **Finish**.

The LOV is attached to the property on all the business objects that use that property. The business objects and properties appear in the **Property Attachments** table for the LOV.



6. Click the **Attach** button to attach the LOV to another property.

Click the **Detach** button to detach the LOV from a business object.

Click the **Edit** button to attach an LOV value description and sub-LOV values to a cascading LOV. This is known as an *interdependent LOV*.

7. To save the changes to the data model, choose **BMIDE→Save Data Model**, or click the **Save Data Model** button on the main toolbar.

8. To see the LOV attachment on the business object property, right-click the business object, choose **Open**, click the **Properties** tab, and locate the property on the properties table.

The LOV attachment appears in the **LOV** column.

9. After you attach the LOV, you can deploy your changes to the server. Choose **BMIDE→Deploy Template** on the menu bar, or select the project and click the **Deploy Template** button on the main toolbar.

10. After deployment, test your newly attached LOV in the Teamcenter rich client by creating an instance of the business object and clicking the arrow in the property box where the LOV is attached.

The list of values appears.

For example, if you attached an LOV to the **object_desc** property of the **Item** business object, in the My Teamcenter application, choose **File→New→Item** to create an instance of the **Item** business object. After you have created the business object, notice that in the **Summary** or **Viewer**

© 2025 Siemens

tab that the **Description** box has an arrow. When you click the arrow in the **Description** box, your new list of values appears.

Add values to an existing classic LOV

1. In the **Extensions** folder, open the **LOV** folder.
2. In the **Classic LOV** folder, right-click an LOV and choose **Open**.
3. In the LOV editor, click the **Add** button to the right of the value table.

> Caution:
>
> Special characters are limited to those supported by the ASCII character set only. Copy and pasting characters from Microsoft Word into Business Modeler IDE boxes is an unsupported operation.

> Note:
>
> The LOV values can be reordered in the template, and the new order is stored in the database and displayed in the Teamcenter clients that use the LOV. The LOV values cannot be reordered in filter LOVs, extent types of LOVs, or tag extent types of LOVs.

Create a filter LOV

You can create a list of values (LOV) based on another LOV by using the **Filter** LOV type.

1. Right-click the **LOV→Classic LOV** folder and choose **New Classic LOV**.

   The New Classic LOV wizard runs.
2. In the **Name** box, type the name you want to assign to the new LOV.
3. In the **Type** box, select **ListOfValuesFilter**.
4. Click the **Browse** button to the right of the **Based on LOV** box to choose the LOV you want to base the filter LOV on.

   The **Find LOV** dialog box appears.
5. Perform the following steps in the **Find LOV** dialog box:

   a. Select the LOV you want to base the new LOV on.

      The values for the LOV display in the **Preview Values** pane.

© 2025 Siemens

> Note:
>
> If no values display, you cannot select values to filter on.

b. Click **OK**.

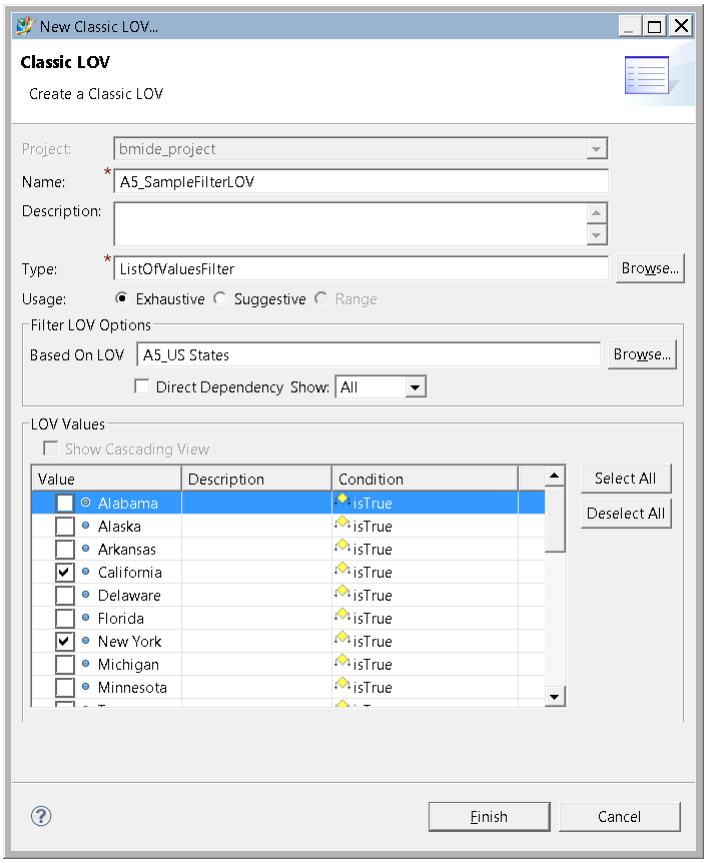
The values of the original LOV are displayed in the **LOV Values** table.

6. Select the **Direct Dependency** check box if you want all the new LOV values to be derived directly from the original LOV.

7. In the **Show** box, select **All** to show all the values of the original LOV or **Selected** to show only selected values.

8. In the **LOV Values** pane, select only the values you want to use.

> Note:
>
> If you had previously selected **Direct Dependency**, it is cleared when you clear any values in the **LOV Values** pane.



© 2025 Siemens

9. Click **Finish**.

   The new LOV appears in the **Classic LOV** folder.

10. To save the changes to the data model, choose **BMIDE→Save Data Model**, or click the **Save Data Model** button on the main toolbar.

    To display the LOV in the Teamcenter rich client user interface, you must **attach it to a property** on a business object.

Create a cascading LOV

A cascading LOV (also known as a hierarchical LOV) is an LOV to whose values sub-LOVs are attached. For example, each state in a list of states might have an attached list of cities.

1. Create classic LOVs that will form the hierarchy. For example, create:

   - An LOV that contains a list of states. This is the first level of the hierarchy, or the main LOV.
   - For each state in the main LOV, an LOV that contains a list of cities. These are the second level of the hierarchy.

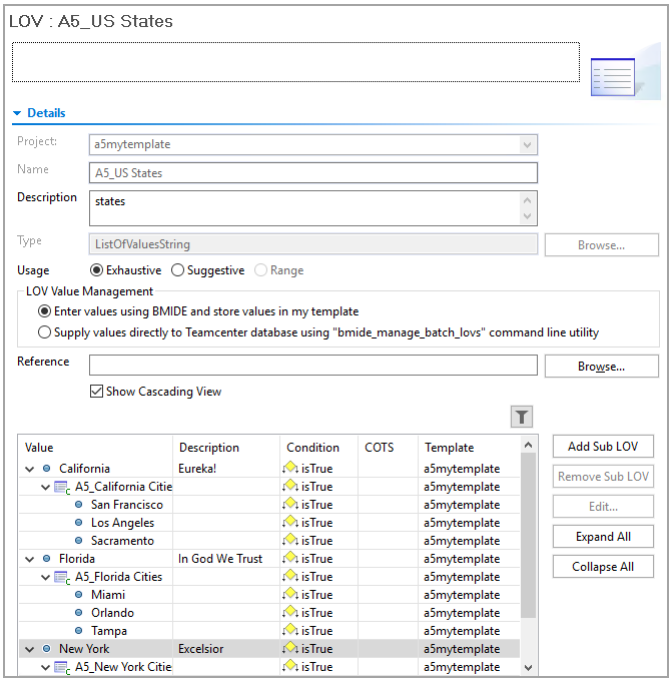
2. In the **Classic LOV** folder, right-click the main LOV, choose **Open**, and in the LOV edit window, select **Show Cascading View**.

3. Select a value in the list and choose **Add Sub LOV**.

   In the **LOV Selection** dialog box, choose the sub-LOV for that value.

   For example, in a **States** LOV, select the **California** value and add the **California Cities** sub-LOV. Then select the **Florida** value and add the **Florida Cities** sub-LOV.

© 2025 Siemens



Attaching a cascading LOV to a property

To use the cascading LOV to populate a single property of a business object instance, you **attach it to a property** as usual. For example, attach the main LOV to the **Item Master** business object on the **user_data_1** property. Then when you create an **Item** in the Teamcenter rich client, the cascading LOV appears in the **User Data 1** box in the second dialog box of the New Item wizard.

To use the cascading LOV to populate several properties of a business object instance, **create an interdependent attachment**.

Create an interdependent LOV attachment

You can attach levels of a cascading LOV to a business object's properties so that, as a user selects values for properties of a business object instance in a Teamcenter client, choices for parent level properties determine the choices that become available for the child level properties. This arrangement is called an interdependent LOV. You can also use the functionality to automatically populate a property with the description of the root level value choice.

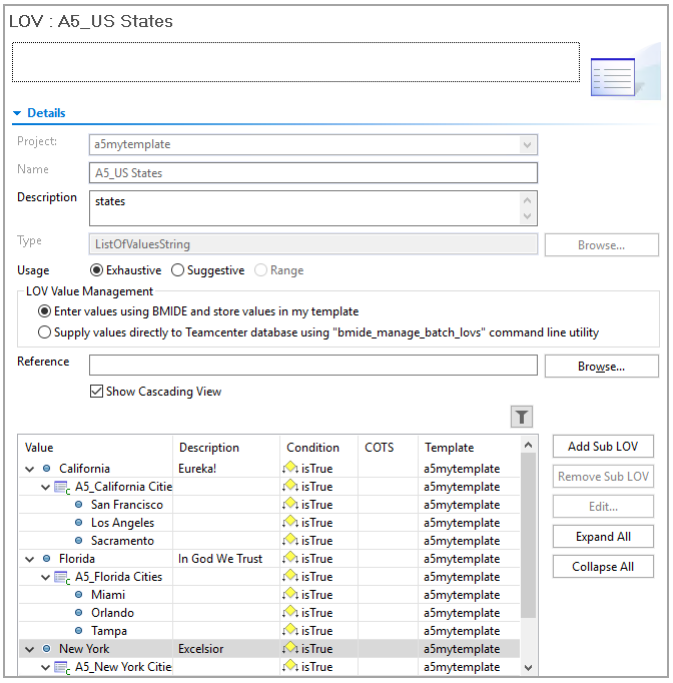
1. If an appropriate hierarchical LOV does not already exist, **create a cascading LOV**.

   For use as interdependent lists, the cascading LOV must be balanced. That is, each branch on a level must have a sub-LOV.

2. In the **BMIDE** view, expand **Extensions**>**LOV**>**Classic LOV**, then right-click a cascading LOV and choose **Open**.

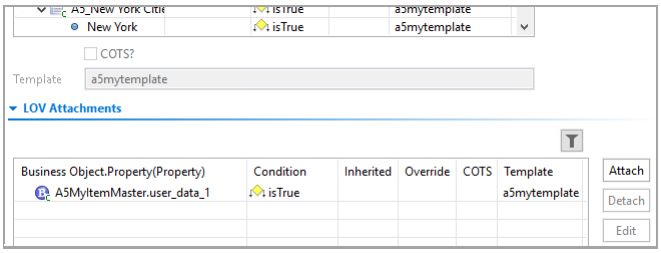
© 2025 Siemens

3. In the LOV edit window, select the **Show Cascading View** check box. Ensure that the LOV is balanced.



4. In the **LOV Attachments** table, attach the cascading LOV to a business object property that is appropriate for the first level of the hierarchy.

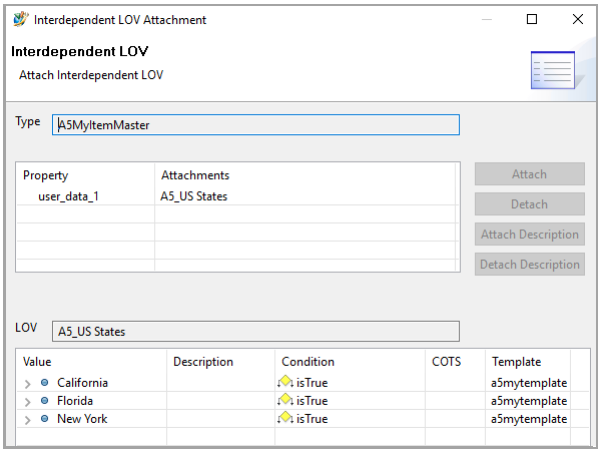
   For example, consider a cascading LOV of states and cities. Attach the cascading LOV to the **Item Master** business object **a5user_data_1** property.



5. In the **LOV Attachments** table, select the attachment and click **Edit**.

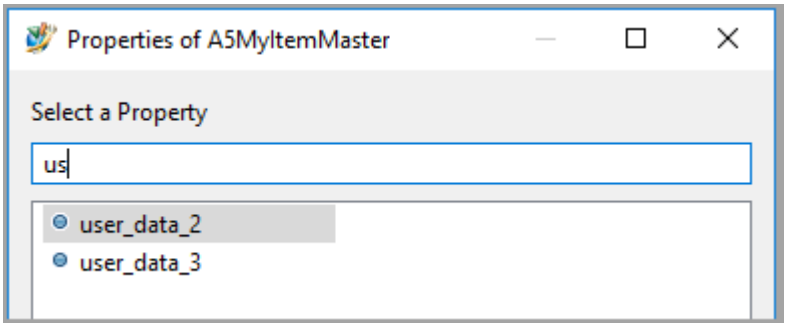
   The **Interdependent LOV** dialog box appears. In this example, the first level of the hierarchy of the **A5_US States** LOV is already attached to the **user_data_1** property.

© 2025 Siemens



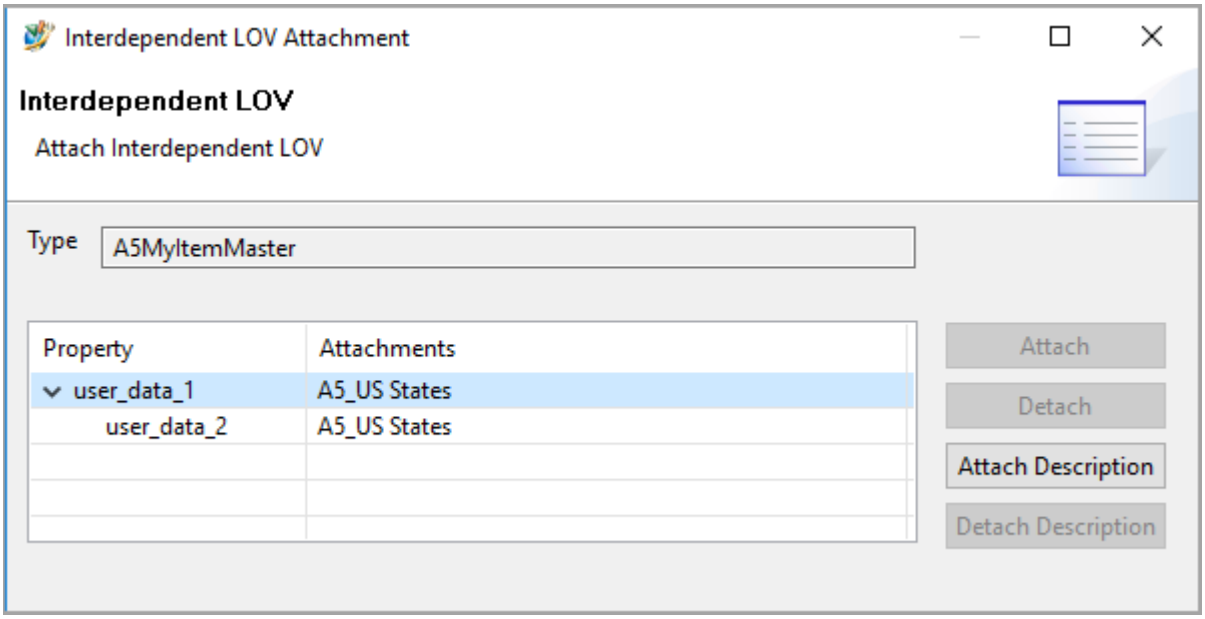
6. In the **Property** table, select the root level property (in this case **user_data_1**) and then click **Attach**.

7. In the property selection dialog box, select a property for the second level of the LOV. For this example, choose the **user_data_2** property.



Expand the list to see that user_data_2 has been added.

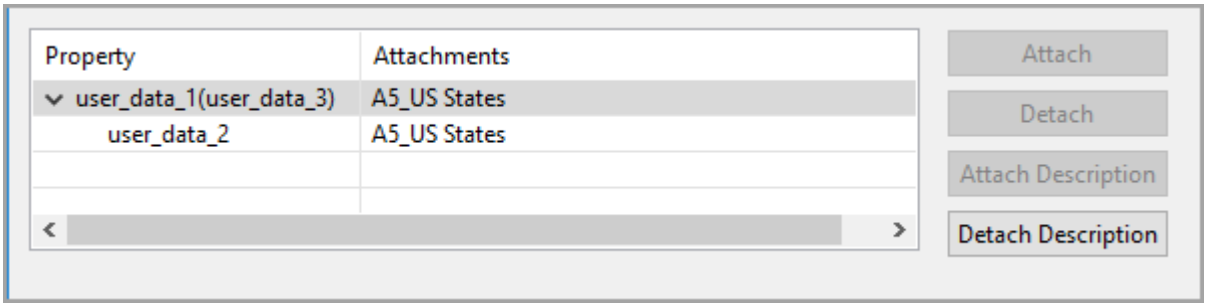
© 2025 Siemens



8. Attach additional properties for additional levels as needed.

9. (Optional) To use the description of the root level value that is selected at run time as the value displayed for another property, select the root level of the tree, click **Attach Description**, and select the target property.

   The **Attach Description** capability is valid only for the root level property of the LOV. Attaching the description of sub-levels is not supported.



10. Click **Finish**.

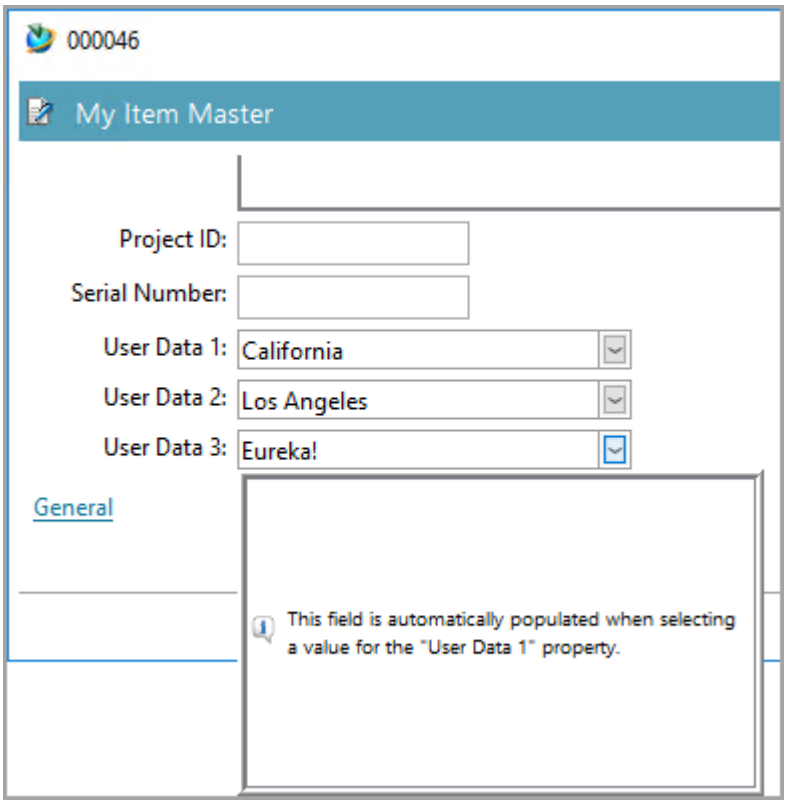
11. Save the data model and deploy it to the server.

Test the interdependent LOV

1. In My Teamcenter, while the selection is at an appropriate location such as the **Home** folder, choose **File**>**New**>**Other**>**[custom object type]** and on the New Item wizard click **Next**.

© 2025 Siemens

2. In the **Define business object create information** dialog box, on the page for the business object to which the interdependent list attachment was made (in this example, the item master form) click the button next to **User Data 1** to select the state, and click the button next to **User Data 2** to select the city.



Attaching LOVs with conditions

Rather than applying conditions directly to LOVs, you apply conditions when attaching the LOVs to properties.

> Note:
>
> LOVs using conditions on LOV and sub-LOV values continue to work. However, evaluating every condition directly on the LOV or sub-LOV slows down processing. You can migrate those LOV definitions to the newer method using LOV attachment.

The following examples demonstrate how to apply conditions with LOV and sub-LOV attachments:

- Example 1: Use conditions on LOVs

  In this example, you want to display different LOV values based on whether a user logs on to Teamcenter as a member of certain programs.

© 2025 Siemens

If the end user logs on as a member of the **Program A** program, you want to show the **Green** and **Red** values for the **color** property on the **MyCarObject** business object. But if the user logs on as a member of the **Program B** program, you want to show the **Gold** and **Silver** values on the same property.

1. Create the classic LOVs.

    a. Create a **MyColorLOV1** LOV and add **Green** and **Red** values.

    > Note:
    >
    > When you add LOV values, all values default to the **isTrue** condition; you cannot change the condition.

    b. Create a **MyColorLOV2** LOV with **Gold** and **Silver** values.

2. Create the conditions.

    a. Create an **isUserBelongToProgramA** condition with an **isUserBelongToProgramA (UserSession o)** signature and an **o.project_name = "Program A"** expression.

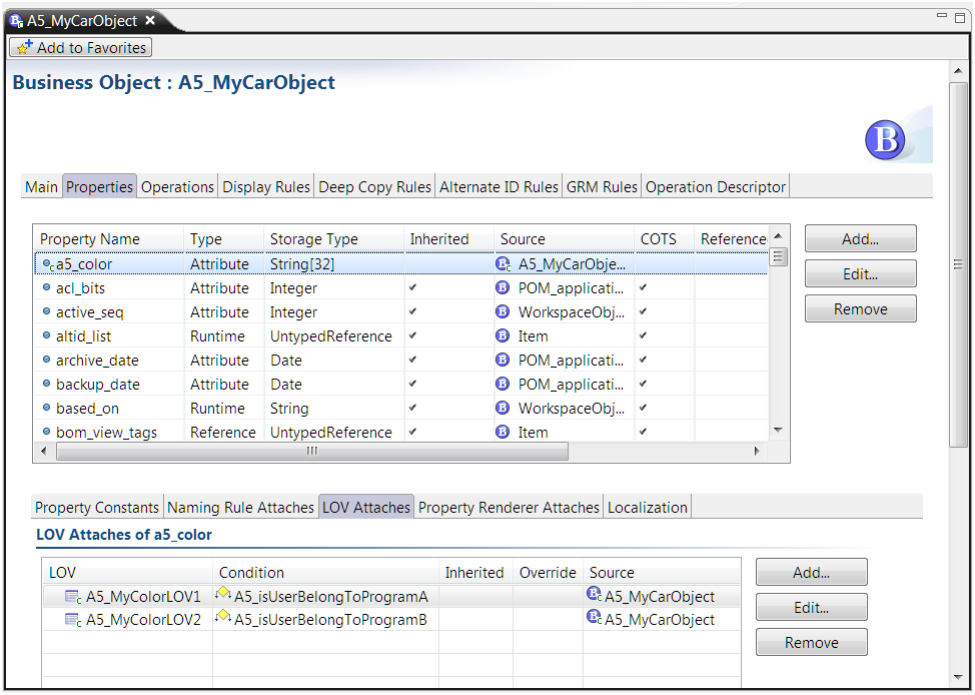
    b. Create an **isUserBelongToProgramB** condition with an **isUserBelongToProgramB (UserSession o)** signature and an **o.project_name = "Program B"** expression.

3. Attach the LOVs.

    a. Attach the **MyColorLOV1** LOV to the **color** property on the **MyCarObject** business object with the **isUserBelongToProgramA** condition.

    b. Attach the **MyColorLOV2** LOV to the **color** property on the **MyCarObject** business object with the **isUserBelongToProgramB** condition.

© 2025 Siemens



4. Verify the behavior.

   a. Log on to Teamcenter as a member of the **Program A** program and check the LOV for the **color** property on an instance of the **MyCarObject** business object. Because the **isUserBelongToProgramA** condition is satisfied, the **MyColorLOV1** LOV is applied and the **Green** and **Red** values are displayed.

   b. Log on to Teamcenter as a member of the **ProgramB** program and check the LOV for the **color** property on an instance of the **MyCarObject** business object. Because the **isUserBelongToProgramB** condition is satisfied, the **MyColorLOV2** LOV is applied and the **Gold** and **Silver** values are displayed.

> Note:
>
> This example is similar to **applying conditions based on which project an object belongs to**.

- Example 2: Use conditions on sub-LOVs

  In this example, you want to display different LOV values based on whether a user logs on to Teamcenter as a member of certain groups.

  Your corporation is operating sales and R&D operations in India and the United States. In India, sales operations are based in Mumbai and Chennai, and R&D operations are based in Pune and Hyderabad. In the United States, sales operations are based in Boston and Chicago, and R&D operations are based in Cincinnati and Los Angeles.

© 2025 Siemens

When a user logs on as a member of the sales group and creates an instance of the **CostItem** business object and clicks the **cost_center** property, you want the cities with sales operations to be displayed. When a user logs on as a member of the R&D group and clicks the same property, you want the cities with R&D operations to be displayed.

1. Create a **CostItem** business object with a **cost_center** string property.

2. Create the classic LOVs.

    a. Create the sales group LOVs.

        A. Create a **CostcenterSales** LOV and add **India** and **US** values.

        > Note:
        >
        > When you add LOV values, all values default to the **isTrue** condition; you cannot change the condition.

        B. Create an **IndiaSalesCities** LOV and add **Mumbai** and **Chennai** values.

        C. Add the **IndiaSalesCities** sub-LOV to the **India** value of the **CostcenterSales** LOV.

        D. Create a **USSalesCities** LOV and add **Boston** and **Chicago** values.

        E. Add the **USSalesCities** LOV to the **US** value of the **CostcenterSales** LOV.

        F. Verify that the **CostcenterSales** LOV structure is as follows:

        India
        IndiaSalesCities
        Mumbai
        Chennai
        US
        USSalesCities
        Boston
        Chicago

    b. Create the R&D group LOVs.

        A. Create a **CostcenterR&D** LOV and add **India** and **US** values.

        > Note:
        >
        > To avoid duplicate data entries, you could also create a filter LOV based on another LOV where all countries are defined.

© 2025 Siemens

B. Create an **IndiaR&DCities** LOV and add **Pune** and **Hyderabad** values.

C. Add the **IndiaR&DCities** sub-LOV to the **India** value of the **CostcenterR&D** LOV.

D. Create a **USR&DCities** LOV and add **Cincinnati** and **Los Angeles** values.

E. Add the **USR&DCities** LOV to the **US** value of the **CostcenterR&D** LOV.

F. Verify that the **CostcenterR&D** LOV structure is as follows:

India
IndiaR&DCities
Pune
Hyderabad
US
USR&DCities
Cincinnati
Los Angeles

3. Create the conditions.

a. Create an **isSalesUser** condition with an **isSalesUser (UserSession o)** signature and an **o.group_name = "SalesGroup"** expression.

b. Create an **isR&DUser** condition with an **isR&DUser (UserSession o)** signature and an **o.group_name = "R&DGroup"** condition.

4. Attach the LOVs.

a. Attach the **CostcenterSales** LOV to the **cost_center** property of the **CostItem** business object using the **isSalesUser** condition.

b. Attach the **CostcenterR&D** LOV to the **cost_center** property of the **CostItem** business object using the **isR&DUser** condition.

5. Verify the behavior.

a. Log on as a member of the **SalesGroup** group and check the LOV shown for the **cost_center** property on an instance of the **CostItem** business object. Because the **isSalesUser** condition is satisfied, the **CostcenterSales** LOV is applied, and the LOV is displayed as a cascading LOV with the following values:

India
Mumbai
Chennai
US

© 2025 Siemens

Boston
Chicago

b. Log on as a member of the **R&DGroup** group and check the LOV shown for the **cost_center** property on an instance of the **CostItem** business object. Because the **isR&DUser** condition is satisfied, the **CostcenterR&D** LOV is applied, and the LOV is displayed as a cascading LOV with following values:

India
Pune
Hyderabad
US
Cincinnati
Los Angeles

Attaching LOVs conditionally based on object properties

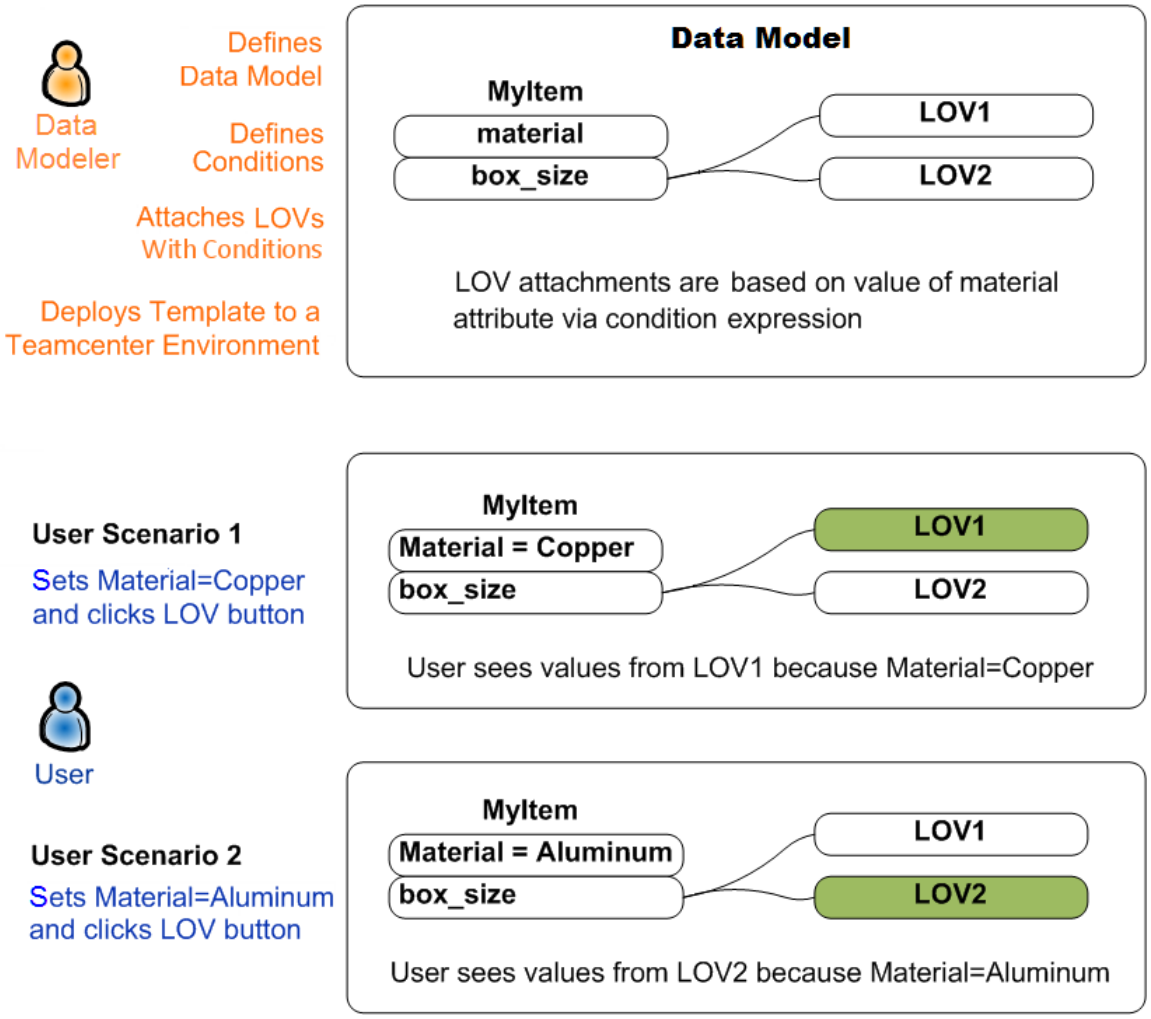
In the Business Modeler IDE, you apply a condition when attaching an LOV to object properties. During object creation, Teamcenter evaluates the condition using property values as set in the user interface. This characteristic is valuable when an LOV may not always be applicable.

> Example:
>
> Suppose that your business ships a product that is made either of copper or of aluminum, and that appropriate box sizes differ depending on the product material. You want to offer users an appropriate list of box sizes based on the material.

1. Define the **MyItem** business object with a **Material** property and a **box_size** property.
2. Create two conditions:
   - **Condition_1**: The **Material** property is **Copper**. For example, `o.A4_Material="Copper"`.
   - **Condition_2**: The **Material** property is **Aluminum**
3. Create your two lists of box sizes, **LOV1** and **LOV2**.
4. In the **MyItem** business object, to the **box_size** property, attach **LOV1** using **Condition_1**. Attach **LOV2** using **Condition_2**.

The following illustration shows the process and effect of conditions with LOV attachments.

© 2025 Siemens



Examples of displayed LOV depending on user selection sequence

Sequence 1: material value is changed before box_size is set

The user creates a **MyItem** business object and sets the **Material** property to **Copper**. Next, the user edits the **MyItem** properties. The user changes the value of the **Material** property to **Aluminum** and clicks the LOV button for **box_size** property. The **LOV2** value is displayed to the user based on the current value of the **Material** property, which is **Aluminum**.

Sequence 2: material value is not set before box_size is set

The user creates a **MyItem** business object and no value is set for the **Material** property. Next, the user edits the **MyItem** properties and clicks the LOV button for **box_size** property. The **LOV1** and **LOV2** values are displayed to the user because the **Material** property has no value. Additionally, Teamcenter displays a message to the user that a list of values could not be determined.

Sequence 3: material value is changed after box_size is set

The user creates a **MyItem** business object and sets the **Material** property to **Copper** and the **box_size** property to **LOV1**. Next, the user edits the item properties and changes the value of the

© 2025 Siemens

Material property to **Aluminum**. The value for the **box_size** property is automatically cleared. The user clicks the LOV button for **box_size** property and the **LOV2** is displayed.

Properties for conditions

The object-based condition can refer to any property in the hierarchy of the input business object and can be used in a condition expression.

The condition expression must not use child objects. For example, do not use item master or item revision master.

Supported condition signatures

condition-name(**UserSession u**)
condition-name(**POM_object o**)
condition-name(**POM_object o, UserSession u**)

Guidelines when using object-based conditions

Follow these guidelines when using object-based conditions:

- Ensure that the attributes used in condition expression are configured in the **Operation Descriptor** tab of the business object.
- For properties with manual user input, use one of these techniques:
  - Configure the necessary properties as **Required** properties to avoid invalid entry for a property attached with an object-based condition.
  - Add an empty **Exhaustive** LOV with the **isTrue** condition to the property to ensure restricting invalid user input.

Display LOVs based on a project

Using conditions to display LOVs

Administrators can set up Teamcenter to display different lists of values (LOVs) on an object's property depending on the project the object is assigned to.

Administrators use the Business Modeler IDE to create conditions that define which list of values to display for a project. The conditions are limited to using the following project properties in their definitions: **owning_project**, **project_list**, **owning_user**, and **owning_group**.

Project LOV scenarios

Following are scenarios you may encounter when creating conditions to display LOVs:

© 2025 Siemens

- Scenario 1: An object is assigned to only one project at a time.

  Mutually exclusive conditions are designed so that each time an object is assigned to only one project, one LOV is selected for display. In this use case, define one condition per project.

  For example, define **isProjectA**, **isProjectB**, **IsProjectC** conditions as mutually exclusive conditions. Assuming that these conditions use the **project_list** property, they can be designed as follows. For example, the **isProjectA** condition can have the following signature:

  ```
  isProjectA(WorkspaceObject o, UserSession u )
  ```

  The condition has the following expression:

  ```
  ((o!=null AND o.fnd0IsShadowObjectForEdit () = true) AND
  Function::INLIST("ProjectA", o.project_list, "project_name")) OR
  ((o=null OR  o.fnd0IsShadowObjectForEdit () = false) AND u.project_name="ProjectA")
  ```

  Create one condition for each project following this pattern.

  > Note:
  >
  > In this scenario, if the **isTrue** condition is attached in addition to the project-based conditions, the **isTrue** condition is used if none of the project-based conditions evaluate to true.

- Scenario 2: An object is assigned to more than one project.

  If an object is assigned to more than one project, and the conditions are designed as in the previous use case, the system may display an LOV randomly from one of the projects. One solution is to create conditions by grouping. Assuming that conditions use the **project_list** property, you can create a group condition for each possible combination of projects that an object could belong to. For example, the **projectGroup1** condition could group **ProjectA** project and **ProjectB** project:

  ```
  ((o!=null AND o.fnd0IsShadowObjectForEdit () = true) AND
  ( Function::INLIST("ProjectA", o.project_list, "project_name")OR
  Function::INLIST("ProjectB", o.project_list, "project_name"))) OR
  ((o=null OR  o.fnd0IsShadowObjectForEdit () = false) AND
  ( u.project_name="ProjectA" OR u.project_name="ProjectB" ))
  ```

  Or simply:

  ```
  u.fnd0isInProjListORSessionProject(o,"ProjectA")
  OR u.fnd0isInProjListORSessionProject(o,"ProjectB")
  ```

  Then create two LOVs such that either the **ProjectA** project or the **ProjectB** project is assigned to display the LOV. This ensures predictable LOV display.

- Scenario 3: Display LOVs based on owning project.

© 2025 Siemens

Another way to deal with the situation of an object that is assigned to more than one project is to use the **owning_project** property instead of the **project_list** property. Because an object can only be owned by one project at a time, the condition resolves to display only the LOV from the owning project. For example, if an object is assigned to both **ProjectA** and **ProjectB**, but is only owned by one of them, create an **isOwningProjectA** condition and an **isOwningProjectB** condition to check for the owning project instead of the project list. For example, for the **isOwningProjectA** condition, use the following signature:

```
isProjectA(WorkspaceObject o, UserSession u )
```

The condition has the following expression:

```
(o!=null AND o.fnd0IsShadowObjectForEdit () = true) AND o.owning_project
!= null AND o.owning_project.project_name = "ProjectA" OR u.project_name = "ProjectA"
```

Or:

```
u. fnd0isOwningORSessionProject(o, “ProjectA”)
```

Project LOV use case

In this use case, if an item is assigned to project A, the **Color** property shows one set of colors to choose from in the LOV; if the item is assigned to project B, the **Color** property shows another set of colors.

1. Create a custom business object and property.

   a. Create a child of the **Item** business object, for example, **B5_MyPart**.

   > Note:
   >
   > Replace **B5_** with your project’s unique naming prefix. For all the objects you create for this example (properties, LOVs, and conditions) replace **B5_** with your project’s unique naming prefix.

   b. Add a property to designate the object’s colors, for example, **b5_MyColors**.

2. Create classic LOVs.

   a. Create an LOV to hold color values, for example, **B5_Colors1**.

   Add the following colors:

   Red
   Blue

   b. Create another LOV to hold color values, for example, **B5_Colors2**.

© 2025 Siemens

Add the following colors:

Green
Yellow

3. Create project-based conditions.

   - **B5_isProjectA**

     This condition is evaluated to the **True** value when working with an object in Teamcenter whose **project_list** property has a project named **ProjectA**.

     Signature:

     ```
     B5_isProjectA ( WorkspaceObject o , UserSession u)
     ```

     Expression:

     ```
     ((o!=null AND o.fnd0IsShadowObjectForEdit () = true) AND
     Function::INLIST("ProjectA", o.project_list, "project_name")) OR
     ((o=null OR  o.fnd0IsShadowObjectForEdit () = false) AND
     u.project_name="ProjectA")
     ```

     This expression says that if the workspace object exists (**o!=null**), and the **projectA** project exists, the condition is true.

   - **B5_isProjectB**

     This condition is evaluated to the **True** value when working with an object in Teamcenter whose **project_list** property has a project named **ProjectB**.

     Signature:

     ```
     B5_isProjectB ( WorkspaceObject o , UserSession u)
     ```

     Expression:

     ```
     ((o!=null AND o.fnd0IsShadowObjectForEdit () = true) AND
     Function::INLIST("ProjectB", o.project_list, "project_name")) OR
     ((o=null OR  o.fnd0IsShadowObjectForEdit () = false) AND
     u.project_name="ProjectB")
     ```

     This expression says that if the workspace object exists (**o!=null**), and the **projectB** project exists, the condition is true.

4. Attach the LOVs to the property

© 2025 Siemens

a. Attach **B5_Colors1** LOV to the **b5_MyColors** property and specify the condition as **B5_isProjectA**.

This means that when a **B5_MyPart** object is assigned to the **ProjectA** project, the **b5_MyColors** property displays the values on the **B5_Colors1** LOV.

b. Attach **B5_Colors2** LOV to the **b5_MyColors** property and specify the condition as **B5_isProjectB**.

This means that when a **B5_MyPart** object is assigned to the **ProjectB** project, the **b5_MyColors** property displays the values on the **B5_Colors2** LOV.

5. Test in the rich client.

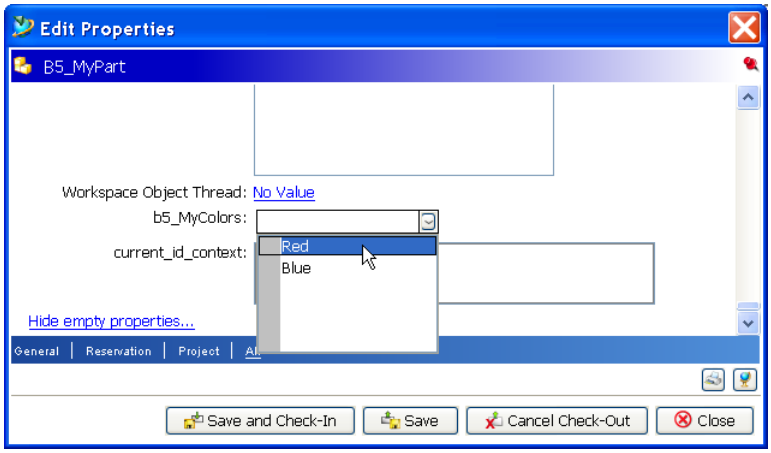
a. Deploy the template from the Business Modeler IDE to the server.

b. Create a project whose name is **ProjectA** and another whose name is **ProjectB**.

c. Create an instance of the **B5_MyPart** business object.

d. Set the project on the business object instance to **ProjectA** by right-clicking the object and choosing **Project→Assign**.

On the **b5_MyColors** property, the available values are **Red** and **Blue**.

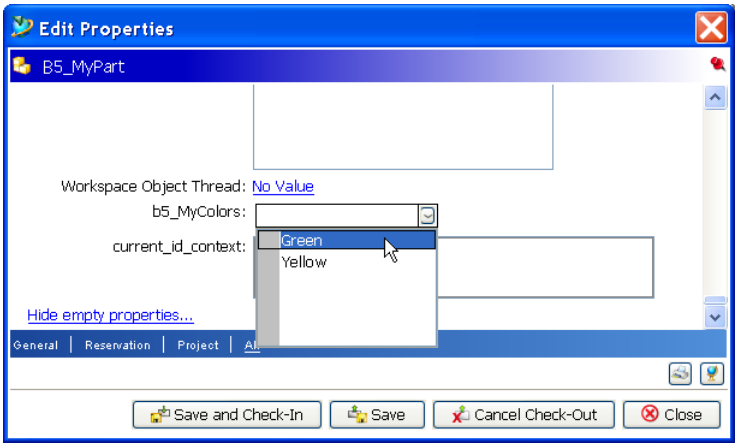


e. Unassign **ProjectA** by right-clicking the object and choosing **Project→Remove**.

f. Set the project on the business object instance to **ProjectB** by right-clicking the object and choosing **Project→Assign**.

On the **b5_MyColors** property, the available values are **Green** and **Yellow**.

© 2025 Siemens



Note:

If you view a compound property created from a property with a project-based conditional LOV attachment, the incorrect LOV may initially appear on the compound property. Manually refresh the object to see the correct LOV displayed on the compound property.

LOV value types

There are six valid value types used to construct LOVs. Each LOV can contain only one of these value types (that is, value types cannot be mixed in the same LOV).

Value	Definition
Integer	Whole number.
Double	Double-precision floating point decimal number (sometimes called a *real*).
Char	Single ASCII character.
String	String of ASCII characters.
Date	Date and time in the format used at your site.

© 2025 Siemens

Value	Definition
	Tip: For the date attribute type, the earliest date supported is January 2, 1900.
Reference	Reference to a list of unique tags in the database (for example, item ID).

LOV usage types

Each LOV must also be assigned one of three usage types.

Usage	Definition
Exhaustive	Used to define all allowable entries. A user is prevented from specifying a value that is not contained in an exhaustive LOV.
Suggestive	Used to provide a suggested list of allowable values. For example, a suggestion LOV could be used to list commonly used description strings. Because description boxes typically accept any user-defined string, the user can select one of the suggested description strings from the LOV or enter another user-defined string.
Range	Used to provide the user with a constrained subset of allowable entries. For example, a range LOV could be used to construct a small consecutive list of serial numbers. A user is prevented from specifying a value not contained within a range LOV.

Special considerations for LOVs

There are some special considerations to keep in mind when creating LOVs:

- Cascading LOVs must be balanced. In a balanced cascading LOV, each level of the LOV consistently displays a related set of values.

© 2025 Siemens

Example:

Balanced LOV	The first level of the LOV shows **state** values, all second level LOVs show **city** values, and all third level LOVs show **zip** values.
Unbalanced LOV	The first level of the LOV shows **state** values. One second level LOV shows **city** values, but another second level LOV shows **zip** values.

- Large LOV values

  A list of values (LOV) with a value greater than 450 characters is not supported on UTF-8 based installations using a SQL Server database. This is due to an internal limitation in SQL Server. Attempting to deploy such LOV values in this environment results in an error when the data is saved to the database.

- The values in an LOV must necessarily be unique. If a **Value Display Name** or **LOV Value Localization** for a value is defined that differs from the internal value, care must be taken that the display name or localized display name defined for one internal value does not match the name of a different internal value.

  Queries for objects with a specific value from an attached LOV will return results for objects that have either a matching internal value or a matching display value.

Creating options

About the Options folder

The **Options** folder in the **Extensions** folder is for working with *options*. Whereas business objects represent parts, documents, and other design objects, options represent configurations for business objects. For example, a change item tracks a change to a business object, a status item designates the status of a business object in a workflow, a view item holds structure information for a business object, and so on.

Add an ID context

An *ID context* defines when you use unique item IDs. ID contexts are used when you create **alias** or **alternate** IDs.

1. Choose one of these methods:

   - On the menu bar, choose **BMIDE→New Model Element**, type **ID Context** in the **Wizards** box, and click **Next**.

   - Open the **Extensions\Options** folders, right-click the **ID Context** folder, and choose **New ID Context**.

© 2025 Siemens

The New ID Context wizard runs.

2. Perform the following steps in the **ID Context** dialog box:

   a. The **Project** box defaults to the already-selected project.

   b. In the **Name** box, type the name you want to assign to the new ID context object in the database.

      When you name a new data model object, a prefix from the template is automatically affixed to the name to designate the object as belonging to your organization, for example, **A4_**.

   c. In the **Display Name** box, type the name as you want it to appear in the user interface.

   d. In the **Description** box, type a description of the new ID context object.

   e. Click **Finish**.

3. To save the changes to the data model, choose **BMIDE→Save Data Model**, or click the **Save Data Model** button on the main toolbar.

Now that you have created the ID context, you can create an alias or alternate ID rule to use it.

Note types

Introduction to note types

A note type is an object associated with a product structure occurrence in a Structure Manager bill of material (BOM). Note types support list of values, but the LOV must be of type **String**.

Users can specify a value for any note type that has been defined for the site. The occurrence note objects that are defined are listed in the Structure Manager **Columns**, **BOMLine Properties**, and **Notes Editor** dialog boxes. The user can use any of these dialogs to enter a value for a particular occurrence.

The initial list of note types shown are standard note types supplied with the system that are required for Teamcenter manufacturing process management and for synchronizing object attributes from NX. These should not be deleted.

Add a note type

A note type is an object associated with a product structure occurrence in a Structure Manager bill of materials (BOM).

1. Choose one of these methods:

© 2025 Siemens

- On the menu bar, choose **BMIDE→New Model Element**, type **Note Type** in the **Wizards** box, and click **Next**.
- Open the **Extensions\Options** folders, right-click the **List of Note Types** folder, and choose **New Note Type**.

The New Note Type wizard runs.

2. Perform the following steps in the **Note Type** dialog box:

   a. The **Project** box defaults to the already-selected project.

   b. In the **Name** box, type the name you want to assign to the new note object in the database.

      When you name a new data model object, a prefix from the template is automatically affixed to the name to designate the object as belonging to your organization, for example, **A4_**.

      > Note:
      >
      > When you type the name of a new note type, do not use a period (**.**) in the name. The note type does not function property if a period is placed in the name.

   c. In the **Display Name** box, type the name as you want it to appear in the user interface.

   d. In the **Description** box, type a description of the new note object.

   e. Select the **Attach Value List** check box if you want to attach a value to the note from a list of values (LOV).

      LOV and **Default Value** boxes are displayed.

      A. Click the **Browse** button to the right of the **LOV** box to locate the list of values to attach to the note.

         > Note:
         >
         > The LOV must be a **String** type because notes are string types.

      B. Click the **Browse** button to the right of the **Default Value** box to choose the default value from the list of values that you want to attach to the note.

   f. Click **Finish**.

      The new note object appears under the **List of Note Types** folder.

© 2025 Siemens

In addition, a new property with the same name as the new note appears on the **BOMLine** run-time business object. If you specified an LOV with the note, the new property automatically has the LOV attached to it.

3. To save the changes to the data model, choose **BMIDE→Save Data Model**, or click the **Save Data Model** button on the main toolbar.

4. Deploy your changes to the test server. Choose **BMIDE→Deploy Template** on the menu bar, or select the project click the **Deploy Template** button on the main toolbar.

5. After deployment, test your note in the Teamcenter rich client by viewing **BOMLine** properties in Structure Manager:

   a. In the My Teamcenter application, right-click any item or item revision and choose **Send To→Structure Manager**.

   b. In the Structure Manager application, right-click the item and choose **Properties**.

   c. Scroll to the bottom of the **Properties** dialog box and click **Show empty properties**.

      All the **BOMLine** properties appear. Your new note appears near the bottom of the dialog box.

   You can add an instance of your new note to any child item in the Structure Manager application. Select the child item, choose **View→Notes**, click the arrow in the **Create** box and choose your new note option.

Add an occurrence type

An *occurrence type* is used to distinguish how items occur in a product structure. An occurrence consists of one component in an assembly including its relative position with respect to its parent assembly. Occurrence types are representations of the **PSOccurrence** business object.

1. Choose one of these methods:

   - On the menu bar, choose **BMIDE→New Model Element**, type **Occurrence Type** in the **Wizards** box, and click **Next**.

   - Open the **Extensions\Options** folders, right-click the **List of Occurrence Types** folder, and choose **New Occurrence Type**.

   The New Occurrence Type wizard runs.

2. Perform the following steps in the **Occurrence Type** dialog box:

   a. The **Project** box defaults to the already-selected project.

© 2025 Siemens

b. In the **Name** box, type the name you want to assign to the new occurrence type object in the database.

When you name a new data model object, a prefix from the template is automatically affixed to the name to designate the object as belonging to your organization, for example, **A4_**.

c. In the **Display Name** box, type the name as you want it to appear in the user interface.

d. In the **Description** box, type a description of the new occurrence type object.

e. Click **Finish**.

The new occurrence appears under the **List of Occurrence Types** folder.

3. To save the changes to the data model, choose **BMIDE→Save Data Model**, or click the **Save Data Model** button on the main toolbar.

4. Deploy your changes to the server. Choose **BMIDE→Deploy Template** on the menu bar, or select the project and click the **Deploy Template** button on the main toolbar.

5. After deployment, test your new change in the Teamcenter rich client:

a. In the My Teamcenter application, right-click an item with subitems and choose **Send to→Structure Manager**.

The item displays in the Structure Manager.

b. In the Structure Manager application, right-click a column heading, choose **Insert Column(s)**, and add the **Occurrence Type** column.

c. Click in the cell below the **Occurrence Type** column.

An arrow is displayed. Click the arrow in the cell to see your new occurrence type listed among the available types.

d. Select your new occurrence type. Doing this declares that the occurrence is of this type.

View types

Introduction to view types

A view type is a definition that controls the name of a product structure view object. The view object works with an **Item** and **Item Revision** business object to maintain product structure information in Teamcenter.

Keep in mind the following:

© 2025 Siemens

- View type display names

  The **view** property on the **ItemRevision** business object should have the same display name value as that of the view type display name.

  View type names that are used for creating run-time properties on item revisions derive their display name from the view type objects. Therefore, to maintain consistency, the display name of view types that are **ItemRevision** properties should have the same display name as that of view types in the Business Modeler IDE. The item revision properties dealing with view types should show the same display names that are seen for **PSView** types in Structure Manager.

- Multiple view types

  Consider a site that uses Structure Manager primarily for modeling engineering data. If transitioning to multiple views, this site should consider renaming views to something that are more meaningful in the context of multiple views that appropriately describe the legacy product structure data.

  In this case, **engineering** may be a good choice. This allows another view such as **shipping** to be defined. From that point forward, different groups and users could use the most appropriate view to model their data.

- View type preferences

  In the Teamcenter rich client **Edit→Options→Product Structure** menu, you can examine the default view object, which is set for Structure Manager with the **PSE_default_view_type** preference. Initially, this preference is set to the following:

  ```
  PSE_default_view_type=view
  ```

  If you rename or add view types, this preference may need to be changed to another value. To change preferences, choose **Edit→Options** and click **Search** at the bottom left of the dialog box.

The default configuration is a single **PSView** object defined for the entire site called **view**.

Add a view type

A *view type* is a BOM view revision (BVR) category. View types control the name of a product structure view object. The product structure view types work with the item and item revision business objects to maintain product structure information.

1. Choose one of these methods:

   - On the menu bar, choose **BMIDE→New Model Element**, type **View Type** in the **Wizards** box, and click **Next**.

   - Open the **Extensions\Options** folders, right-click the **List of View Types** folder, and choose **New View Type**.

© 2025 Siemens

The New View Type wizard runs.

2. Perform the following steps in the **View Type** dialog box:

   a. The **Project** box defaults to the already-selected project.

   b. In the **Name** box, type the name you want to assign to the new view type in the database.

      When you name a new data model object, a prefix from the template is automatically affixed to the name to designate the object as belonging to your organization, for example, **A4_**.

   c. In the **Display Name** box, type the name as you want it to appear in the user interface.

   d. In the **Description** box, type a description of the new view object.

   e. Click **Finish**.

      The new view type appears under the **List of View Types** folder.

3. To save the changes to the data model, choose **BMIDE→Save Data Model**, or click the **Save Data Model** button on the main toolbar.

4. Deploy your changes to the server. Choose **BMIDE→Deploy Template** on the menu bar, or select the project and click the **Deploy Template** button on the main toolbar.

5. After deployment, test your new view type in the Teamcenter rich client.

   For example, in the My Teamcenter application, select an item or item revision and choose **File→New→BOM View Revision**. In the **New BOMView Revision** dialog box, click the **More...** button in the lower left and select the new view object from the list of available view objects. Create an instance of the new view object and click **OK**.

Add a status type

A status type is applied to an object after it goes through a workflow. Typical status types are **Pending** and **Approved**. You can add your own status type.

> Note:
>
> - **Schedule Manager statuses** are set using the **Status Types** LOV.
> - Do not subclass the **ReleaseStatus** business object to create new status types.

1. In the BMIDE, start the New Status wizard in one of these ways:

© 2025 Siemens

- On the menu bar, choose **BMIDE**>**New Model Element**, type **Status** in the **Wizards** box, and click **Next**.
- Open the **Extensions>Options** folder, right-click the **Status** folder, and choose **New Status**.

2. In the **Status** dialog box, enter properties for the new status.

For this property	Do this
Name	Type the name you want to assign to the new status object in the database. When you name a new data model object, a prefix from the template is automatically affixed to the name to designate the object as belonging to your organization, for example, **A4_**.
Display Name	Type the name as you want it to appear in the user interface.
Description	Type a description of the new status object.

3. Click **Finish**.

   The new status object appears under the **Status** folder.

4. To save the changes to the data model, choose **BMIDE**>**Save Data Model**, or on the main toolbar click **Save Data Model** .

5. Deploy your changes to the test server in one of these ways:

   - On the menu bar, choose **BMIDE**>**Deploy Template**.
   - Select the project and on the main toolbar click **Deploy Template** .

Test your new status in the Teamcenter rich client

Use the following procedure to see the new status in **Workflow Designer**.

1. In the Workflow Designer application, choose **File**>**New Root Template**.

2. In the **New Root Template** dialog box, click the arrow in the **Based on Root Template** box, select **TCM Release Process**, and click **OK**.

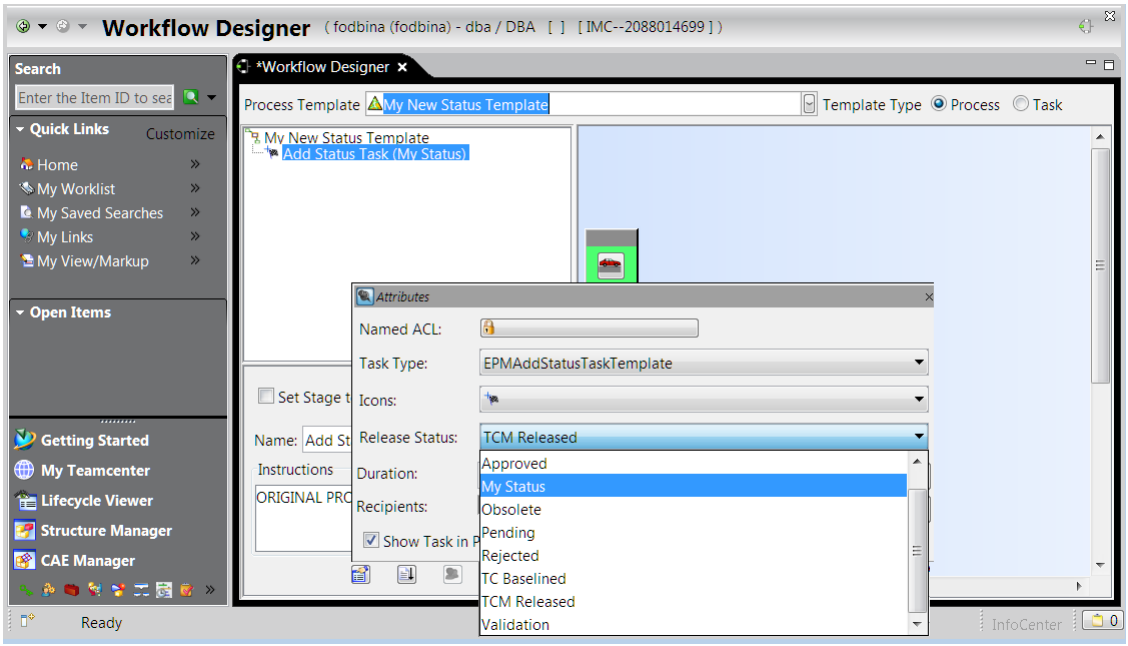
   The new process appears.

3. In the upper left pane, select **Add Status Task (TCM Released)**.

4. In the lower left pane, click **Display the Task Attributes Panel** .

© 2025 Siemens

5. In the **Attributes** dialog box, in the **Release Status** box, click the arrow.



6. Select your new status and close the **Attributes** dialog box.

7. In the **Name** box in the lower-left pane, change **Add Status Task (TCM Released)** to something that describes your new status. For example, **Add Status Task (My Status)**.

Use the following procedure to see the new status in the **Modify Revision Rule** dialog box.

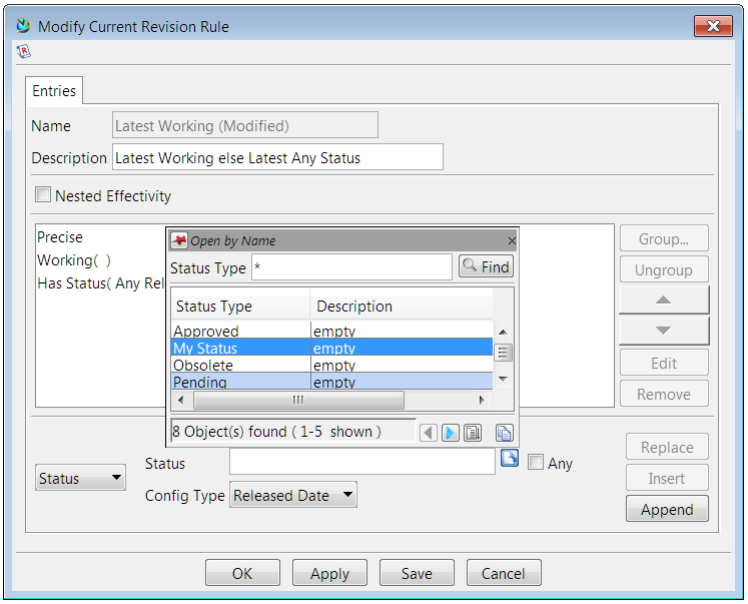
1. In the Structure Manager application, choose **Tools**>**Revision Rule**>**Modify Current**.

2. In the **Modify Revision Rule** dialog box, in the lower left corner click **Working** and select **Status**.

3. To the right of the **Status** box, click  .

4. Search for the new status.

© 2025 Siemens



Add a storage media option

A *storage media* is a storage device category such as a hard disk or optical device. It is used by third-party content-storage systems.

Storage media devices are used for archiving, restoring, importing, and exporting objects. Teamcenter supports content storage, hard disk, and 4mm digital audio tape (DAT) storage media. It does not support 1/4-inch cartridge tape. Use content storage media for integration with third-party content storage systems such as EMC Centera. The integration is implemented via neutral APIs. Hard disks can be locally or NFS-mounted. DAT devices must be locally mounted. Remote DAT devices are not supported.

1. Choose one of these methods:

   - On the menu bar, choose **BMIDE→New Model Element**, type **Storage Media** in the **Wizards** box, and click **Next**.

   - Open the **Extensions\Options** folders, right-click the **Storage Media** folder, and choose **New Storage Media**.

   The New Storage Media wizard runs.

2. Perform the following steps in the **Storage Media** dialog box:

   a. The **Project** box defaults to the already-selected project.

   b. In the **Name** box, type the name you want to assign to the new storage media object in the database.

© 2025 Siemens

When you name a new data model object, a prefix from the template is automatically affixed to the name to designate the object as belonging to your organization, for example, **A4_**.

c. Click the arrow in the **Media Type** box and choose one of the following:

- **Disk**

  Stores data on a hard disk.

- **Tape**

  Stores data on magnetic tape.

d. In the **Logical Device** box, type the logical name to assign to the device.

e. In the **Description** box, type a description of the new storage media object.

f. Click **Finish**.

The new storage media object appears under the **Storage Media** folder.

3. To save the changes to the data model, choose **BMIDE→Save Data Model**, or click the **Save Data Model** button on the main toolbar.

4. Deploy your changes to the server. Choose **BMIDE→Deploy Template** on the menu bar, or select the project and click the **Deploy Template** button on the main toolbar.

Add a tool

A *tool* represents a software application, such as Microsoft Word or Adobe Acrobat. You associate a tool with a type of dataset business object so that, for an instance of the business object, from the Teamcenter rich client you can launch the external application and load the dataset file.

1. Start the New Tool wizard in one of these ways:

   - On the menu bar, choose **BMIDE→New Model Element**, type **Tool** in the **Wizards** box, and click **Next**.

   - Open the **Extensions\Options** folders, right-click the **Tool** folder, and choose **New Tool**.

2. In the **Tool** dialog box, specify parameters for the tool:

© 2025 Siemens

For this parameter	Do this
Name	Type the name that you want to assign to the new tool object in the database.
MIME/TYPE	Enter the media (MIME) application association. For example, for Acrobat, enter **application/acrobat**. This parameter is not required by Windows systems because Windows systems use the file extension to determine the MIME type. Tip: Users can choose **Edit→Options→Dataset** in the rich client to find the default tool used to open the dataset file.
Shell/Symbol	Type the full path and program name to be run in a shell on Linux systems. For example, for Adobe Acrobat, type **$TC_BIN/start_acrobat.bat**. Tip: If you have a Windows-only enterprise, you can type any string to this field. In this case, best practice is to type a string that indicates the application name.
Vendor Name	Type the name of the application vendor. For example, Adobe is the vendor for Acrobat, and Microsoft is the vendor for Word.
Revision	Type the version number of the application. For example, **6.0**, **2014**, or something similar.
Release Date	Click the calendar button to the right of the box and select the release date of the application.
Description	Type a description of the new tool object.

3. Click **Next**.
4. In the **New Tool Input/Output** dialog box, add the types of input and output data that the tool accepts and produces. For example, **ASCII** or **Binary**.
5. Click **Next**.
6. In the **New Tool Markup Information Page** dialog box, define view and markup capabilities.

© 2025 Siemens

For this parameter	Do this
Mac Launch Command	Type the command to launch the tool in the Macintosh operating system. For example, **sample.app**.
Win Launch Command	Type the command to launch the tool in the Windows operating system. For example, **sample.exe**.
Download Required?	Select the check box if the application launcher must download any files before launching the application.
Callback Required?	Select the check box to enable callback through the application launcher for a non-Teamcenter application.
View Capable?	Select the check box if the defined application can view files.
Markup Capable?	Select the check box if the defined application can perform markups.
Embed Application?	Select the check box if the defined application is an embedded rich client tool. For these tools, the **Shell/Symbol** box contains the command to launch the embedded tool.
VVI Required?	Select the check box if the defined application accepts Velocity Vector Imaging (VVI).
Digital Signature Capable?	Select the check box if the defined application can do digital signing.

7. Click **Finish**.

   The new tool object appears under the **Tool** folder. The new tool is available in the Business Modeler IDE for addition to the **Tools for Edit** or **Tools for View** lists when you create a new dataset business object.

8. To save the changes to the data model, choose **BMIDE→Save Data Model**, or on the main toolbar click **Save Data Model**  .

Linked Data Framework

Introduction to Linked Data Framework

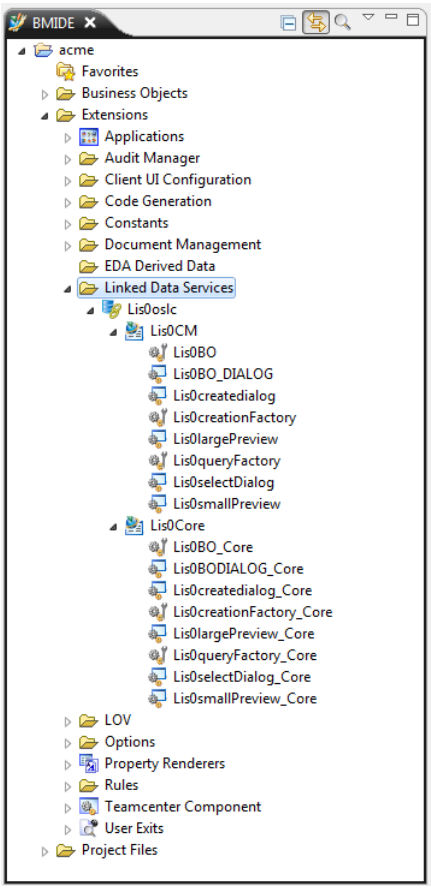
The **Linked Data Services** folder in the Business Modeler IDE allows you to create objects needed to link external life cycle collaboration tools (such as IBM RTC) to Teamcenter.

You must first install the Linked Data Services feature to use these objects.

© 2025 Siemens

The Linked Data Services feature uses the Linked Data Framework (LDF).

This framework allows you to create, retrieve, update, and delete data shared with external life cycle management systems. Linked Data Framework is based on open technologies like Linked Data, Resource Description Framework (RDF), and REpresentational State Transfer (REST). Using the **Linked Data Services** folder in the Business Modeler IDE, you can create an application interface to be a Linked Data Framework consumer or provider. A consumer consumes data from other applications while a provider provides data access to other applications.



To set up Teamcenter as a Linked Data Framework provider, add the following components in the Business Modeler IDE:

- Service catalog

  Specifies a domain for a protocol. A domain typically reflects specific functionality in Teamcenter, for example, requirements management. The change management domain (**Lis1CM**) is available by default.

- Linked Data Framework service

© 2025 Siemens

Specifies what functionality in a domain is available, for example, requirements management preview functionality. If you create a Linked Data Framework service, you must perform additional customizations.

The Linked Data Framework service can be of the following types:

- Factory service

  Provides programmatic interfaces for using Linked Data Framework HTTP interfaces. The factory services use the REST methods **DELETE**, **GET**, **POST**, and **PUT**.

- Delegated user interface (UI) service

  Provides a UI-based service. This service is defined with a URI pointing to an Active Workspace UI designed for that service.

In the change management domain, the following functionality is available by default:

- Change request query capability (**Lis1queryFactory** service)
- Change request creation factory (**Lis1creationFactory** service)
- Change request creation dialog (**Lis1createdialog** service)
- Change request selection dialog (**Lis1selectDialog** service)

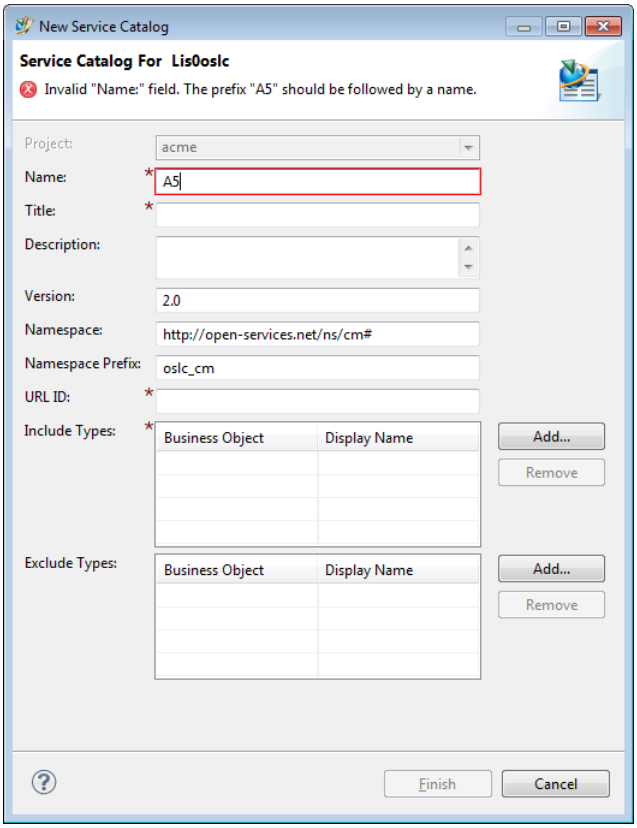
Create a new service catalog

The Linked Data Framework provider catalog typically reflects specific functionality in Teamcenter, and it is referred to as a service catalog in the Business Modeler IDE Linked Data Framework registry. The change management domain is provided in Linked Data Framework by the **Lis0CM** catalog. You can create your own catalog to represent other areas of functionality in Teamcenter.

1. Open the **Extensions\Linked Data Services** folders, right-click an already-created protocol, and choose **New Service Catalog**.

   The **New Service Catalog** dialog box is displayed.

© 2025 Siemens



2. Enter the following information in the **Service Catalog** dialog box:

   a. In the **Name** box, type the name you want to assign to the new catalog.

   b. In the **Title** box, type the display name to the new catalog.

   c. In the **Description** box, state the purpose of the catalog.

   d. In the **Version** box, type the protocol version supported by the catalog.

   e. In the **Namespace** box, type the namespace path.

   f. In the **Namespace Prefix** box, type the prefix you want to assign.

   g. In the **URL ID** box, type the uniform resource identifier to form the URL for this resource.

   h. In the **Include Types** table, select the list of business objects that the catalog supports.

   i. In the **Exclude Types** table, select the list of business objects that the catalog does not support.

   j. Click **Finish**.

© 2025 Siemens

The new catalog appears under the selected protocol instance.

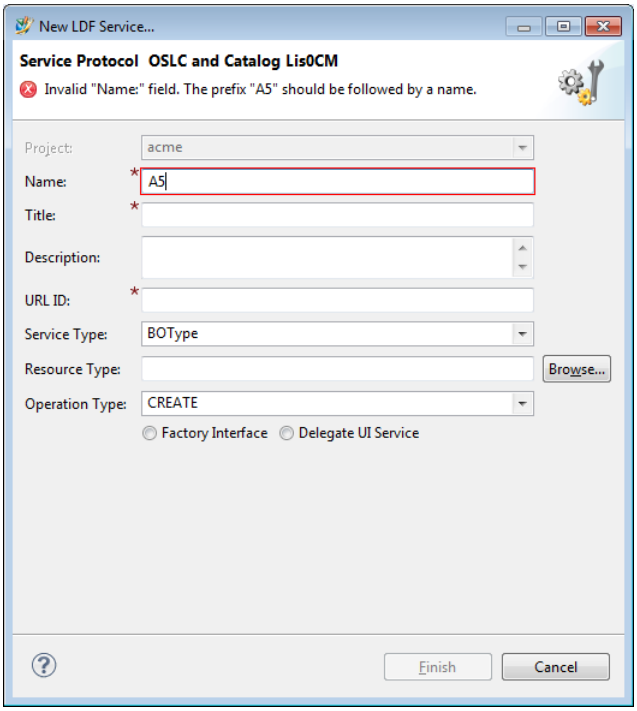
3. To save the changes to the data model, choose **BMIDE→Save Data Model** on the menu bar or click the **Save Data Model** button on the main toolbar.

Create a new Linked Data Framework service

The Linked Data Framework services are used to create factory APIs that are used programmatically to link to external applications. These services are defined for a given catalog. If you **create a catalog**, you must create the services comprising the catalog.

1. Open the **Extensions\Linked Data Services** folders, right-click the already-created catalog, and choose **New LDF Service**.

   The **New LDF Service** dialog box is launched.



2. Enter the following information in the **New LDF Service** dialog box:

   a. In the **Name** box, type the name you want to assign to the new service.

   b. In the **Title** box, type the display name to the new service. This appears in the service catalog.

   c. In the **Description** box, state the purpose of the service.

   d. In the **URL ID** box, type the uniform resource identifier. This forms the URL for the resource.

© 2025 Siemens

e. Click the arrow in the **Service Type** box to select the kind of service type, either a business object type, generic, or an instance type.

- **BOType**

  Specifies that the service being defined is based on a business object type.

- **Generic**

  Specifies that the service being defined does not apply to a specific business object type.

- **Instance**

  Specifies services for update, delete, and preview for a specified business object instance.

f. Click the **Browse** button in the **Resource Type** box, and select the business object type, such as **ChangeItem**.

g. Click one of the following buttons:

- **Factory Interface**

  Creates the service factory. When you click **Factory Interface**, the following boxes are populated:

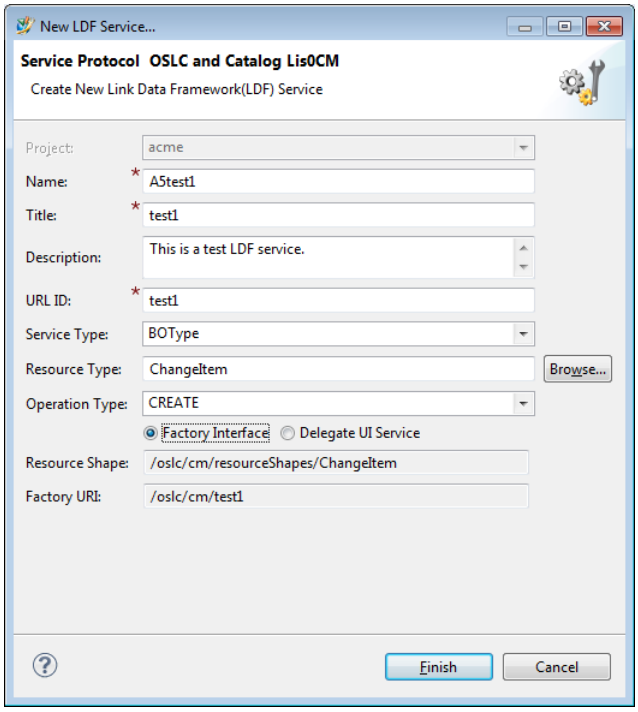
  - **Resource Shape**

    Holds the URI required to get the resource shape (a data dictionary in RDF/JSON output format) for a given business object type.

  - **Factory URI**

    Holds the URI for invoking this factory service.

© 2025 Siemens



- **Delegate UI Service**

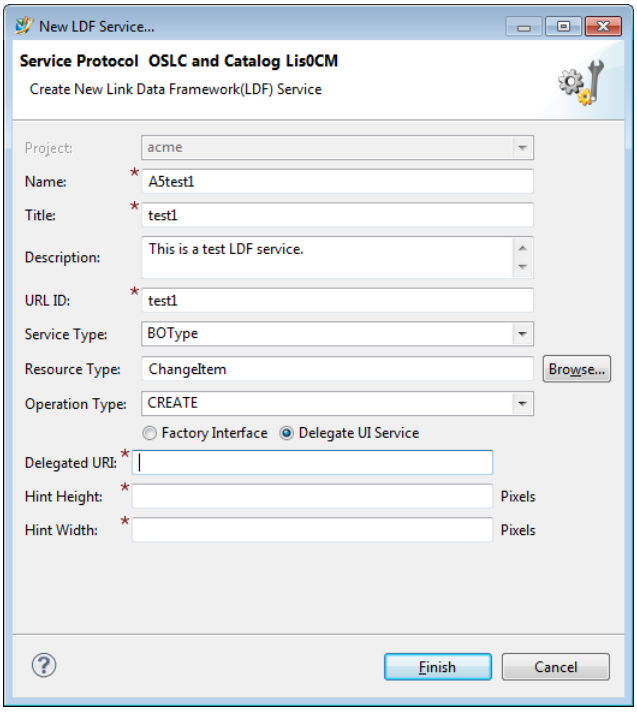
  Creates the service dialog box. Perform the following steps:

  A. In the **Delegated URI** box, specify the delegated user interface URI for this service implementation.

  B. In the **Hint Height** box, specify the value for the height of the **Delegated UI** dialog box in pixels.

  C. In the **Hint Width** box, specify the value for the width of the **Delegated UI** dialog box in pixels.

© 2025 Siemens



3. To save the changes to the data model, choose **BMIDE→Save Data Model** on the menu bar, or click the **Save Data Model** button on the toolbar.

Create a Linked Data Framework service operation

For every service, you can associate four possible different REpresentational State Transfer (REST) operation implementations: **GET**, **POST**, **PUT**, and **DELETE**.

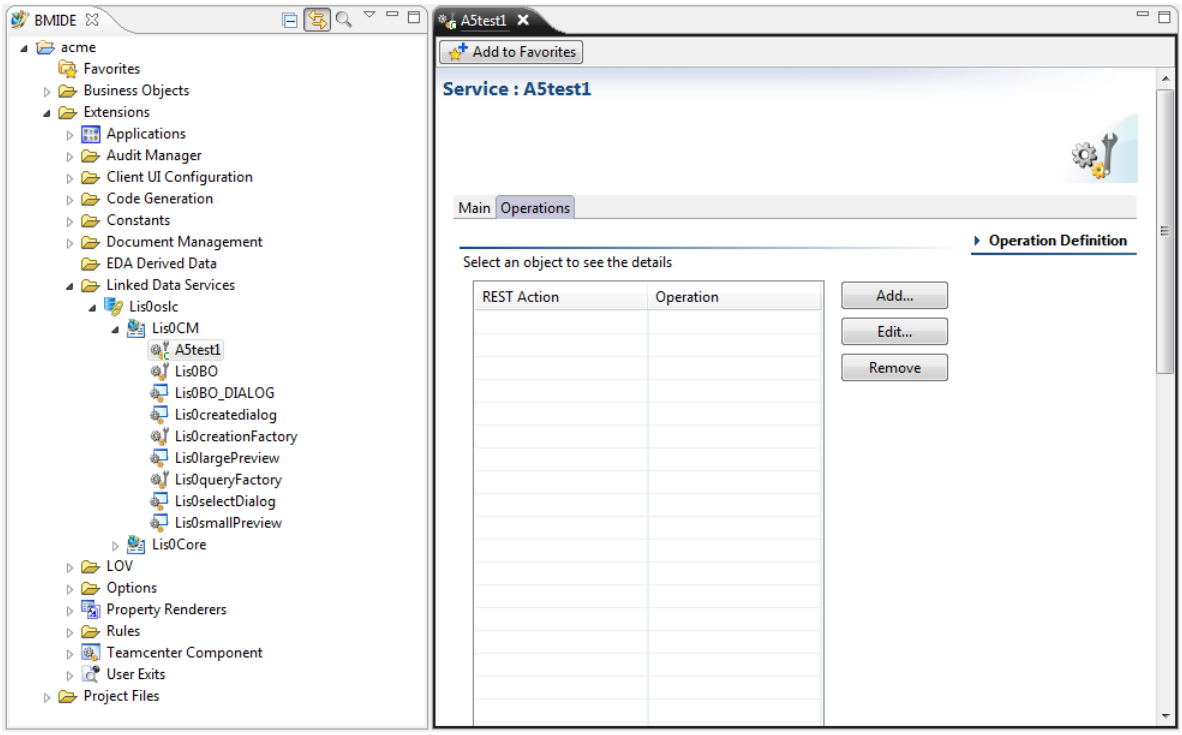
1. Expand the **Extensions\Linked Data Services** folders.

2. Open the service factory to hold the new operation.

> Tip:
>
> You must create a service factory before you create a service operation.

3. Right-click the service factory in which you want to create the new operation, choose **Open**, and click the **Operations** tab.

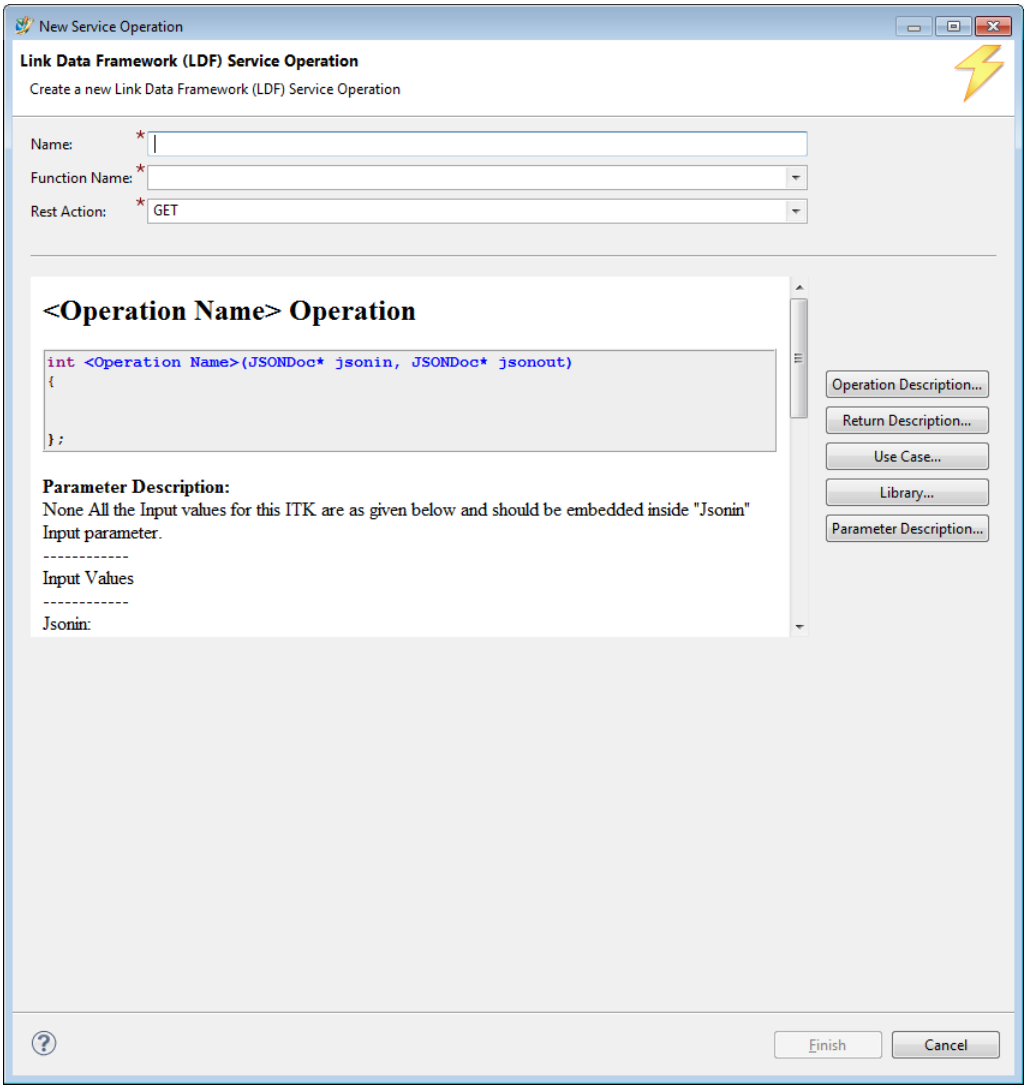
© 2025 Siemens



4. Click the **Add** button on the **Operations** tab.

   The **New Service Operation** wizard is launched.

© 2025 Siemens



5. Perform the following steps in the **Link Data Framework (LDF) Service Operation** dialog box:

   a. In the **Name** box, type a unique name for that operation.

   b. In the **Function name** box, type the API you want to assign for executing the service operation.

   c. Click the arrow in the **REST Action** box to select the REST operation type: **GET**, **PUT**, **POST**, or **DELETE**. Only one operation with the selected REST action can be active.

   d. Click the **Operation Description** button to type a complete description of the functionality exposed through the service operation.

      The **Description Editor** is displayed.

      Follow these best practices:

© 2025 Siemens

- Describe what this operation does. Explain more than simply stating the method name.
- Make the description complete in its usefulness. Keep in mind the client application developer while writing the content.
- Whenever appropriate, describe how each argument interrelates when this operation completes.
- Use correct formatting with fixed and bold text where appropriate.
- Use correct Teamcenter terminology.
- Define acronyms before using them.

e. Click the **Return Description** button to type a complete description of what the service operation returns. Follow these best practices:

- Describe what the output represents and provide high-level details of the output data. Do not specify only the type of service data returned.
- Describe any partial errors returned.
- Specify returned objects that are created, updated, or deleted as part of service data.
- Use complete sentences.
- Use correct formatting with fixed and bold text where appropriate.
- Use correct Teamcenter terminology.
- Define acronyms before using them.

f. Click the **Use Case** button to describe how the user interacts with this operation to accomplish the operation's goal. Follow these best practices:

- Document when and why this operation can be consumed.
- Describe how operations interrelate to satisfy the use case (if there is interrelation between operations).
- Use complete sentences.
- Specify all possible use cases.
- Use correct formatting with fixed and bold text where appropriate.

© 2025 Siemens

- Use correct Teamcenter terminology.
- Define acronyms before using them.

g. Click the **Library** button to select the library where the API is defined, for example, **Lis0lisfmwrk**.

h. Examine the contents of the preview pane.

i. Click **Finish**.

The new service operation displays on the **Operations** tab. To see the characteristics of the operation, select it and click **Operation Definition** on the right side of the editor.

6. To save the changes to the data model, choose **BMIDE→Save Data Model** on the menu bar, or click the **Save Data Model** button on the main toolbar.

7. You must also implement the operation in C++. A sample implementation of a GET operation is as follows:

```
#include <fclasses/tc_string.h>
#include <tc/tc.h>
#include <fclasses/tc_string.h>
#include <base_utils/ScopedPtr.hxx>
#include <base_utils/ScopedSmPtr.hxx>
#include <base_utils/SharedPtr.hxx>
#include <base_utils/Mem.h>
#include <tc/tc_util.h>
#include <fclasses/tc_types.h>
#include <tccore/aom_prop.h>
#include <Lis0lisfmwrk/LisfmwrkUtils.hxx>
#include <Lis0lisfmwrk/lis_defines.h>
#include <Lis0lisfmwrk/Lis0lisfmwrk_errors.h>
#include <Lis0lisfmwrk/toolkit/AbstractResource.hxx>
#include <Lis0lisfmwrk/lis_factory.h>
#include <Lis0lisfmwrk/Lis0JsonWriter.hxx>
#include <metaframework/BusinessObject.hxx>
#include <metaframework/BusinessObjectRef.hxx>


using namespace Teamcenter;
using namespace std;
using namespace lis0lisfmwrk;

/**
Retrieves an object factory Lifecycle Interoperability Services (LIS) service
<br/>It will retrieve the object information (attributes names and values) based on
resourceshape.
<br/>Open Services for Lifecycle Collaboration (OSLC) defines the ResourceShape
resource
 to allow the specification of a list of properties with allowed values.
In teamcenter, OSLC resource will be the object type and its properties.
<br/>This object information is returnnd in JSON format.
```


© 2025 Siemens

```
<br/>This method is invoked in case of get object factory request given from REST
client
 for GET rest action.
<br/>e.g. REST client request.
<br/>Rest action - GET.
<br/>URL http://localhost:8080/lis/oslc/cm/BO?uid=g0bRF0$RI_JGwA

@returns
<ul>
<li>#ITK_ok on success.
<li>#LIS0LISFMWRK_valid_attr_helper_not_found if the LisAttributeHelper is not valid.
<br/>The LisAttributeHelper maps the Teamcenter properties with Open Services for
Lifecycle
 Collaboration (OSLC) core properties. It can be created through the utility
'create_botype_reader'
<li>#LIS0LISFMWRK_valid_writer_not_found if valid media writer not for generating
output JSON
 xml.
</ul>
*/
int LIS_get_object_factory( const char *request_body,   /**< (I) Request BODY in
JSON format. */
                            char **response             /**< (OF) Response in JSON
format. */
                          )
{
    int ifail = ITK_ok;
    ResultStatus rstat;
    PomResultStatus pomStat;
    std::string responseStr;
    try
    {
        std::string reqMediaType =  "application/json";
        tag_t mediaWriterTag =
lis0lisfmwrk::Lis0MediaWriterImpl::getMediaWriter( reqMediaType,
         true );
        if ( mediaWriterTag == NULLTAG )
        {
            rstat = LIS0LISFMWRK_media_writer_not_exist;
        }
        POMRef<lis0lisfmwrk::Lis0MediaWriterImpl> mediaWriterImpl( mediaWriterTag );
        std::string reqBody = request_body;
        lis0lisfmwrk::AbstractResource *resourceModel = new
lis0lisfmwrk::AbstractResource();
        scoped_ptr <lis0lisfmwrk::AbstractResource> freeResource1( resourceModel );
        if ( mediaWriterImpl != 0 )
        {
             // write the document
             rstat = mediaWriterImpl->lis0ReadFrom( &reqMediaType, &reqBody,
&resourceModel );
        }
        //Get the urlParameter attribute(UID=XXX) here from JSON
        std::string uidParam = resourceModel-
>getSimpleStringPropertyValue(URL_PARAMETER);
        std::string catalogTitle = resourceModel-
>getSimpleStringPropertyValue(CATALOG_TITLE);
        std::string queryParam = uidParam;
        std::string objUid = "";
        LisDataUtils::parseRequestParamaterToGetUid(queryParam, &objUid);
        tag_t objTag = NULLTAG;
```


© 2025 Siemens

```
        lis0lisfmwrk::AbstractResource *outResourceModel = 0;
        Lis0ResourceShapeUtils *resourceShapeUtils = new
lis0lisfmwrk::Lis0ResourceShapeUtils();
        scoped_ptr <lis0lisfmwrk::Lis0ResourceShapeUtils>
freeUtil( resourceShapeUtils );
        if(!objUid.empty())
        {
            bool tag_valid = false;
            rstat = LisDataUtils::checkObjectUIDIsValidOrNot(objUid, tag_valid,
objTag);
            if(tag_valid)
            {
                 rstat = resourceShapeUtils->getBoResourceShape( objTag, objUid,
catalogTitle,
                 &outResourceModel );
            }
        }
        else
        {
            throw IFail( LIS0LISFMWRK_input_invalid_request);
        }
        std::string requiredMediaType =  "application/json";
        // write the resource model to document
        rstat = LisDataUtils::addEtagToResourceModel(objTag, &outResourceModel);
        rstat = LisDataUtils::generateOutputFromResourceModel( outResourceModel,
        &requiredMediaType, &responseStr );
        scoped_ptr<lis0lisfmwrk::AbstractResource>
freeoutResourceModel( outResourceModel );
        *response = SM_string_copy(responseStr.c_str());
    }
    catch(const IFail& ex)
    {
        ifail = ex.ifail();
        Teamcenter::Lis0lisfmwrk::logger()->error( ERROR_line, ifail,
        "LIS_get_object_factory failed with error[%d]\n",
        ifail );
    }
    return ifail;
}
```

Working with applications

The **Applications** folder in the **Extensions** folder is for working with *applications*, tools that manage interaction with rich client applications.

Applications are used on classes to indicate what rich client application can add or edit attributes on that class. To see the application that can be used for a class, open the class in the **Classes** view in the **Advanced** perspective and look at the **Application Name** box. You cannot configure applications, but you may need to look at them to understand which classes you can edit with ITK code.

The **Applications** folder contains the following:

- **AM**

© 2025 Siemens

This application applies to classes used by the Access Manager. The AM rule tree, ACLs, and some supporting classes are stored in the database via POM. The application protection ensures that some operations (such as write and delete) to instances of these classes must go via the published AM ITK and cannot be circumvented by direct use of generic POM ITK.

- **AUDITMGR**

  This application applies to classes used by Audit Manager.

- **CFM**

  This application applies to classes used by the Configuration Management application.

- **CMMV**

  This application applies to classes used by the Configuration Management and Multiple Views application (in Engineering Process Management).

- **EIM**

  This application applies to classes used by the Engineering Information Management application (in Engineering Process Management).

- **INFOMANAGEV200**

  This application is used for general core. It is not effectively utilized because ITK logon enables the application, thus defeating the point of application protection.

- **PMM**

  This application applies to classes used by Product Master Manager.

- **POM**

  This application applies to classes used by the persistent object manager (POM). These classes (**POM_class**, **POM_attribute**, and **POM_application**) are owned by POM so that direct access using generic POM ITK is disallowed. The POM data dictionary is used to represent the class-to-database mapping and general reflection.

- **RLM**

  This application applies to classes used by the Workflow application.

- **SCHMGTV100**

© 2025 Siemens

This application is used to specify the classes used by the Schedule Manager application. Schedule Manager is used for planning and tracking activities.

- **USER**

  This application applies to classes used by the Organization application (**POM_user**, **POM_group**, and **POM_member** classes). It is used to ensure changes to instances in these classes go via the supported POM ITK (such as changes in user ID or deletion of a user or group) and not via the more general POM ITK (such as **POM_ask_attr_***type* or **POM_delete_instances**).

Teamcenter Component objects

What is a Teamcenter Component object?

A **Teamcenter Component** object is an object that creates a category. Each extension (business object, extension definition, operation, GRM Rule, and so on) in the Business Modeler IDE can be tagged with a **Teamcenter Component** name, thus identifying which category the extension belongs to. A category typically identifies a software module that has a specific business purpose. Examples of Teamcenter software modules include the Business Modeler IDE itself, Access Manager, preferences functionality, Project, Query Builder, Structure Manager, and Workflow. Examples of external modules include Solid Edge, NX, Wiring Harness Design Tools Integration, and SLM.

In Business Modeler IDE templates that supply data model behaviors and configurations for a single business purpose, such as **Fnd0Classification**, all extensions could be tagged as belonging to a single Teamcenter Component. Many Business Modeler IDE templates are for a single business purpose.

Templates such as the Foundation template include extensions for many different business purposes. The component tag value can identify the business purpose, or category, that the extension impacts. For example, if an operation is tagged with the BMIDE Client component, you can expect that modifying the operation impacts behavior of the Business Modeler IDE.

In addition to categorizing extensions, the **Teamcenter Component** folder and the **Verification Rule** editor can be used to set up rules that use conditions to evaluate how business objects are used in Teamcenter. Verification rules specify a set of conditions to be used in an application. To access the **Verification Rule** editor, choose **BMIDE**>**Editors**>**Verification Rules Editor**.

For example, in the Business Modeler IDE the Teamcenter Component object **Fnd0BOMGrading** categorizes conditions, objects, and verification rules to be used in BOM grading (verifying items in BOMs). The BOM application needs a set of conditions that a user can select to grade a BOM. Verification rules provide these conditions. In the rich client, BOM administrators can use the Structure Manager **Tools**>**BOM Grading** menu command to check parts to see if they are valid for use in selected BOMs. The BOM administrator can select from the list of all available conditions that apply based on the verification rules.

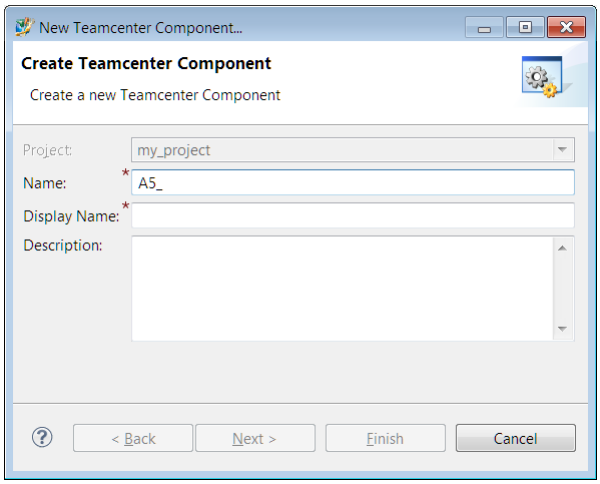
© 2025 Siemens

Create a Teamcenter Component object

1. Choose one of these methods:

   - On the menu bar, choose **BMIDE→New Model Element**, type **Teamcenter Component** in the **Wizards** box, and click **Next**.
   - In the **Extensions** folder, right-click the **Teamcenter Component** folder and choose **New Teamcenter Component**.

   The New Teamcenter Component wizard runs.



2. Perform the following steps in the **Create Teamcenter Component** dialog box:

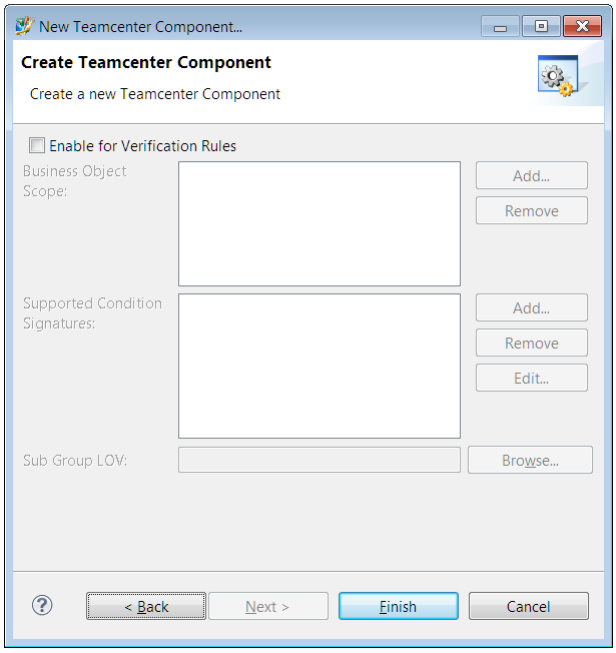
   a. In the **Name** box, type the name you want to assign to the new **Teamcenter Component** object.

   b. In the **Description** box, type a description for the new **Teamcenter Component** object.

   c. Click **Next**.

      The next dialog box in the wizard appears.

© 2025 Siemens



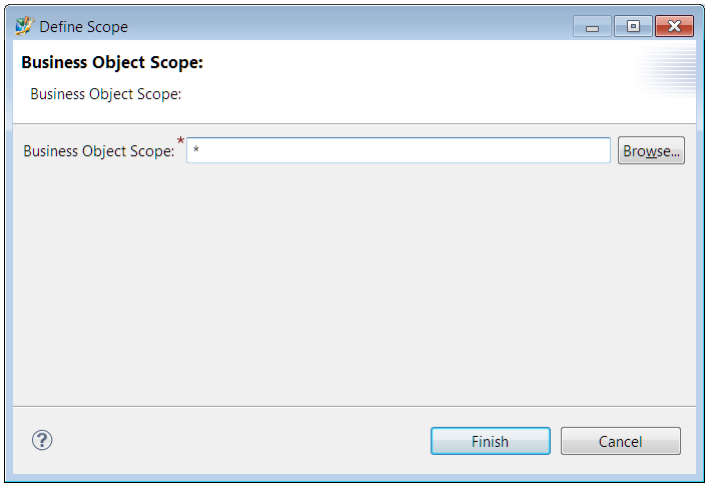
3. Perform the following steps in the next **Create Teamcenter Component** dialog box:

   a. Select the **Enable for Verification Rules** check box to make the new **Teamcenter Component** object available for use by verification rules.

   > Note:
   >
   > To see the verification rules, choose **BMIDE→Editors→Verification Rules Editor**.

   b. Click the **Add** button to the right of the **Business Object Scope** box to define the business object scope list for the **Teamcenter Component** object.

      The **Business Object Scope** dialog box is displayed.



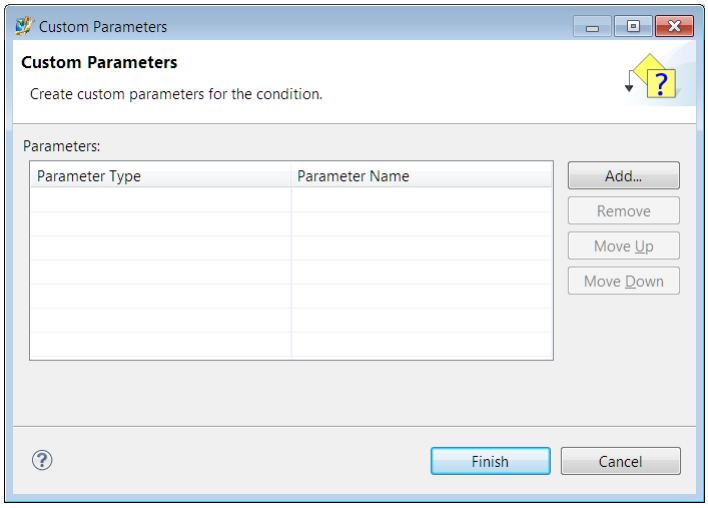
© 2025 Siemens

A. In the **Business Object Scope** dialog box, click the **Browse** button to the right of the **Business Object Scope** box to select the business objects to add to the list.

B. Click **Finish** to add the business object to the scope list.

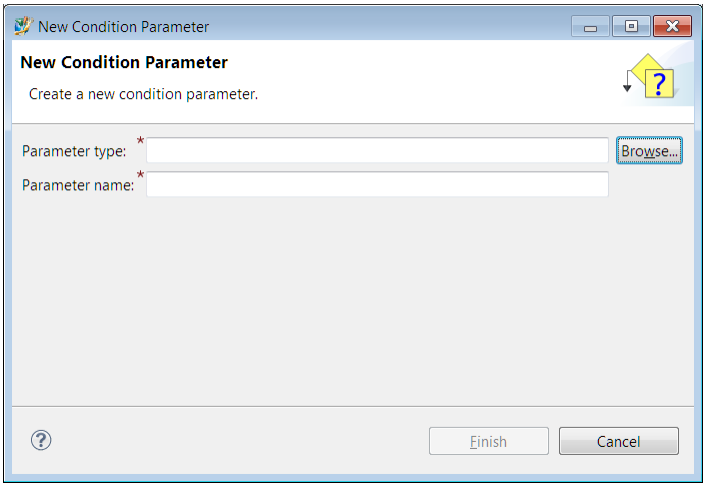
c. Click the **Add** button to the right of the **Supported Condition Signatures** box to build the condition signatures for the **Teamcenter Component** object.

The **Custom Parameters** dialog box is displayed.



A. In the **Custom Parameters** dialog box, click the **Add** button to the right of the **Parameters** table to add the parameters to evaluate for the condition.

The **New Condition Parameter** dialog box is displayed.



© 2025 Siemens

B. In the **New Condition Parameter** dialog box, click the **Browse** button to the right of the **Parameter type** box to select the business object or user session to evaluate, and type the parameter in the **Parameter name** box.

C. Click **Finish** in the **New Condition Parameter** dialog box and the **Custom Parameters** dialog box.

d. Click the **Browse** button to the right of the **Sub Group LOV** box to select the appropriate list of values (LOV) that is valid for the rule.

e. Click **Finish**.

The **Teamcenter Component** object is added under the **Teamcenter Component** folder.

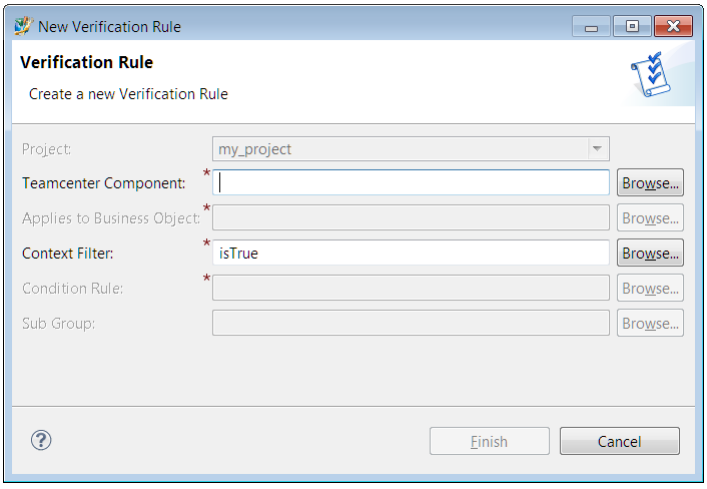
Create a verification rule

1. On the menu bar, choose **BMIDE→Editors→Verification Rules Editor**.

   The **Verification Rule** editor is displayed.

2. Click the **Add** button to the right of the table.

   The New Verification Rule wizard runs.



3. In the **Verification Rule** dialog box, click the **Browse** button to the right of the **Teamcenter Component** box to select the **Teamcenter Component** object the rule is applied to.

> Note:
>
> To see a **Teamcenter Component** object in the list, the **Enable for Verification Rule** check box must be selected on the **Teamcenter Component** object.

© 2025 Siemens

4. Click the **Browse** button to the right of the **Applies to Business Object** box to select the business object that the verification rule applies to. The available business objects are those in the business object scope of the **Teamcenter Component** object selected in step 3.

5. Click the **Browse** button to the right of the **Context Filter** box to select the context for when the rule applies.

6. Click the **Browse** button to the right of the **Condition Rule** box to select the condition for the rule.

   > Note:
   >
   > The only conditions that are available are those with supported condition signatures or their child types as defined in the selected **Teamcenter Component** object. The condition signature order must match the selected **Teamcenter Component** object supported signature order. If there is a selected applied business object, it must match the condition's first parameter or be a subtype of the first parameter.

7. Click the **Browse** button to the right of the **Sub Group** box to select the LOV value to use for the rule

8. Click **Finish**.

   The new verification rule is displayed on the **Verification Rule** table.

Global constants

What are global constants?

A *global constant* is an extension that defines a value, which by reference can be used throughout the system. If you change the value of a global constant, then the new value applies wherever the constant is used.

The data type for a global constant definition can be one of Boolean (true/false), string (single or multivalued), or list.

When you create a new constant, you must also add code on the server to return the constant's value to the caller, so the caller can branch the business logic based on the returned value.

You can create global constants for a number of situations. Some examples are:

- Set the folder names for the **Home** folder.
- Set how many seconds before the client application should time-out.
- Enable functionality.

© 2025 Siemens

For example, a global constant called **ActiveWindows** has a list of three possible choices, **1**, **2**, or **3**, with a default value of **1** in the Foundation template. Once a constant is set, it can be overridden by another template. Another template can set the default value to **2**, and yet another template can set it to **3**. The rule is that the last template to be installed determines the constant value that is used.

The server-side code can use the following published ITK to retrieve a global constant value:

```
int CONSTANTS_get_global_constant_value (
       const char*     constant_name,   /* <I> */
       char **         value            /* <OF> */
       );
```

Global constant details

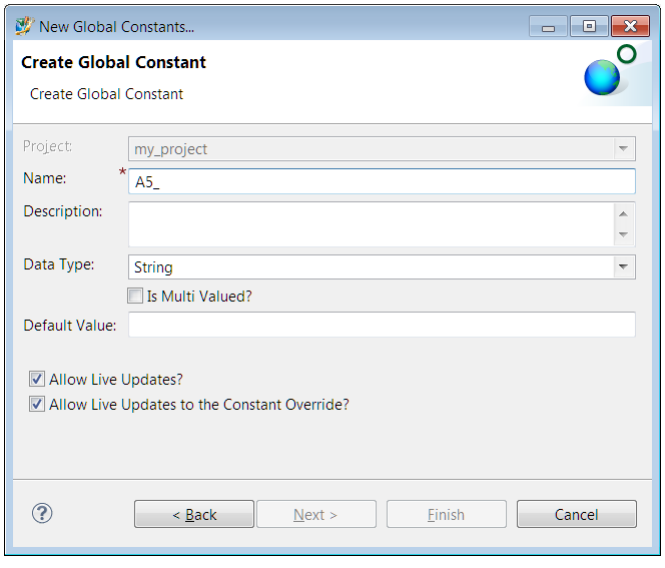
Details of a global constant are displayed in the **BMIDE** edit view for the constant. You can include global constant details in BMIDE data model reports.

Create a global constant

1. Choose one of these methods:

   - On the menu bar, choose **BMIDE→New Model Element**, type **Global Constants** in the **Wizards** box, and click **Next**.
   - Open the **Extensions\Constants** folders, right-click the **Global Constants** folder, and choose **New Global Constants**.

   The New Global Constants wizard runs.



2. Perform the following steps in the **Create Global Constant** dialog box:

© 2025 Siemens

a. The **Project** box defaults to the already-selected project.

b. In the **Name** box, type the name you want to assign to the new constant in the database.

When you name a new data model object, a prefix from the template is automatically affixed to the name to designate the object as belonging to your organization, for example, **A4_**.

c. In the **Description** box, type an explanation of how the constant is to be used.

d. Click the arrow in the **Data Type** box to select one of the following:

- **Boolean**

  Allows two choices to the user (**True** or **False**).

- **String**

  Indicates that the value is a text string.

- **List**

  Contains a list of values.

e. If you selected the **String** data type, the **Is Multi Valued?** check box appears. Select this check box if you want to enter multiple values for the constant.

f. If you selected the **List** data type, a **Values** table appears. Click the **Add** button to add values to the list:

A. In the **Value** box, type a value for the list.

B. Select **Secured** to prevent the selected value from being overridden by another template.

C. Click **Finish**.

g. In the **Default Value** box, enter the initial value of the constant. Entry differs depending on the data type you previously chose:

- If you selected the **String** data type, type in the default value. If you also selected the **Is Multi Valued?** check box, the **Default Value** box changes to a table with multiple rows that you can use to enter values.

- If you selected the **Boolean** data type, click the arrow to select **True** or **False** for the default value.

© 2025 Siemens

- If you selected the **List** data type, click **Browse** to select a value from the available ones on the list.

h. Use the following check boxes to enable the constant for **live updates**:

- **Allow Live Updates?**

  Select this check box to specify that the constant definition itself can be updated live.

- **Allow Live Updates to the Constant Override?**

  Select this check box to specify that the constant attachment (override) can be updated. The constant attachment contains the value set for the constant.

In a live updates environment, you can deploy changes for custom constants, but you cannot deploy changes for COTS constants.

i. Click **Finish**.

The new constant appears under the **Global Constants** folder.

> Note:
>
> You can also see the global constant by choosing **BMIDE→Editors→Global Constants Editor** on the menu bar. To modify the value of the constant, select the constant on the **Global Constants** table and click the **Edit** button.

3. To save the changes to the data model, choose **BMIDE→Save Data Model**, or click the **Save Data Model** button on the main toolbar.

4. Deploy your changes to the server. Choose **BMIDE→Deploy Template** on the menu bar, or select the project and click the **Deploy Template** button on the main toolbar.

5. To verify the constant on the server, run the **getglobalconstantvalue** utility. This utility tests the value of the constant in the database. The utility accepts the name of the global constant and outputs the value of the constant if present.

Change the value of a global constant

Global constants define values that can be used throughout the system. If you change the value of a global constant, then the new value applies wherever the constant is used.

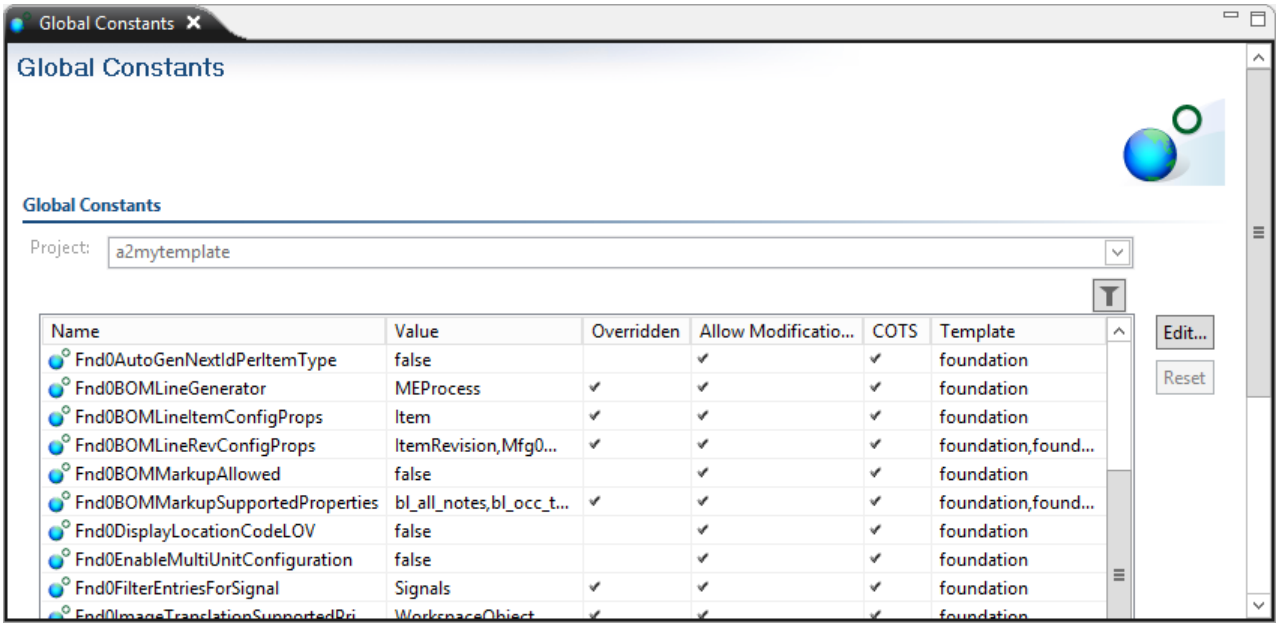
Once a constant is set, it can be overridden by another template. The rule is that the last template to be installed determines the constant value that is used.

1. Start the Business Modeler IDE.

© 2025 Siemens

2. On the menu bar, choose **BMIDE→Editors→Global Constants Editor**.

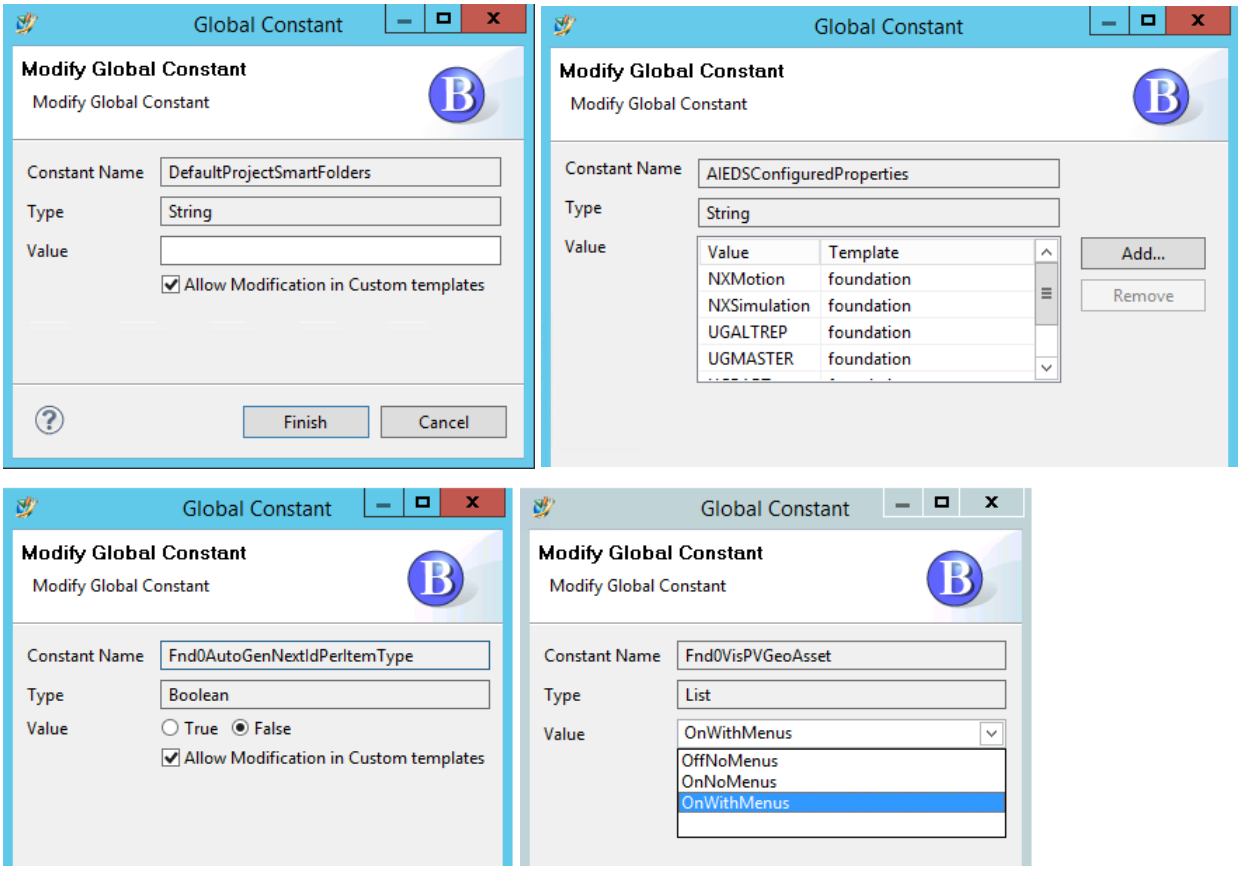
   The Global Constants Editor is displayed.



3. In the **Global Constants** table, select the constant and click **Edit**.

   The Business Modeler IDE displays the **Modify Global Constant** dialog box. The **Value** portion of the dialog box varies, depending on the data type of the constant.

© 2025 Siemens



4. In the **Value** area, specify a new value.

> Note:
>
> To see the constant details, including valid values, open the constant in an editor view: In the **BMIDE** view, under the project expand the **Extensions** > **Constants** > **Global Constants** folder and double-click the constant.

5. As applicable, to allow the constant attachment value to be changed by a custom template, select the **Allow Modification in Custom templates** ☑ check box.

6. Click **Finish**.

   The changed value displays in the **Global Constants** table. Note that a check mark appears in the **Overridden** column and the name of your project displays in the **Template** column, indicating that your project overrides the value of the constant.

7. To save the changes to the data model, choose **BMIDE→Save Data Model**, or click the **Save Data Model** button on the main toolbar.

   The constant change is saved in the active extension file.

© 2025 Siemens

12. Creating business rules

Introduction to business rules

System administrators define rules that determine system behavior at a site. Creating rules is also known as business behavior modeling. Most business rules are set on business object properties. You can create the following types of rules for business objects:

Use this rule type	To do this
Naming	Set the name automatically assigned when a business object is created.
Display	Limit the kinds of objects that can be created by particular individuals.
Generic Relationship Management (GRM)	Limit the kinds of business objects that can be pasted to other business objects.
Deep copy	Define whether objects belonging to item revisions are copied when the item is further revised.
Propagation	Automatically propagate changes in security-related property values from source business object instances to destination business object instances.
Extension	Add a predefined custom function or method that fires as a precondition, preaction, or postaction to a business object operation.
Alternate ID	Store information about the same part from different perspectives.
Alias ID	Store part numbers and other attribute information for similar parts.

Inheritance of business object rules

Because rules are attached to properties of business objects, like properties the rules are subject to the principles of inheritance. Inheritance of business rules by child business objects is as typically encountered for object-oriented programming:

- A sub-class acquires properties and behaviors from its parent class.
- A sub-class can introduce its own properties and behaviors, which override any conflicting inherited properties and behaviors.
- There is no limit to the number of recursive levels.

Naming rules

Introduction to naming rules

Naming rules define the data entry format for a business object property. Naming rules can be used to name items, item revisions, datasets, forms, projects, and work contexts. They can also be used

© 2025 Siemens

to name any persistent string property, except a property defined as an array. A naming rule consists of **rule patterns** and a counter. After you create a naming rule, you must attach it to the business object property. You can also attach the naming rule to a property on all business objects that use that property.

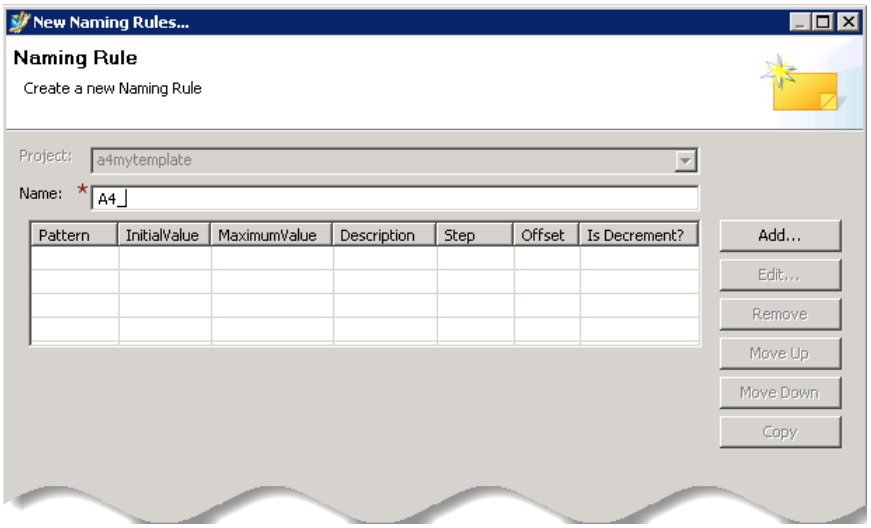
> Example:
>
> You can create a naming rule for an item ID that starts with **CCC** plus a six-digit number, so that it generates numbers from **CCC000001** to **CCC999999**.
>
> First enter a pattern of **"CCC"**. Then enter a second pattern of **NNNNNN** (for numbers) with an initial value of **CCC000001** and a maximum value of **CCC999999**. Then attach it to the **item_id** property of the item business object you want.

Add a naming rule

A naming rule consists of **rule patterns** and an optional counter. Before creating a naming rule, you should be familiar with the pattern and counter formats.

1. Start the **New Naming Rules** wizard in one of these ways:

   - On the menu bar, choose **BMIDE→New Model Element**, type **Naming Rules** in the **Wizards** box, and click **Next**.
   - In the **BMIDE** view, right-click the **Extensions \ Rules \ Naming Rules** folder and choose **New Naming Rules**.

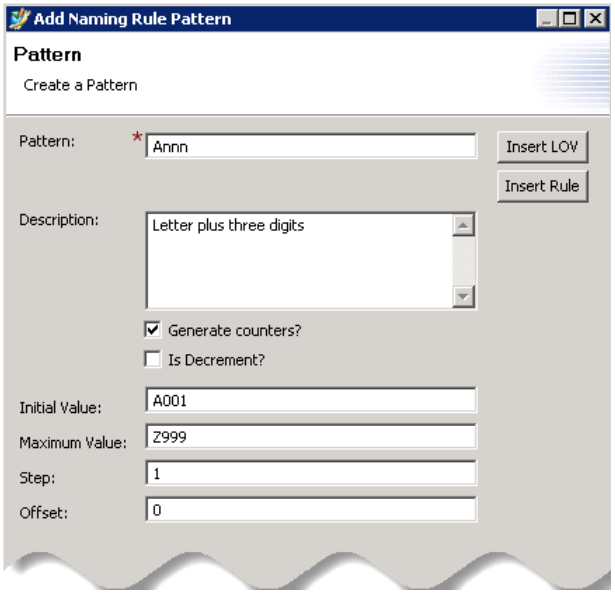


2. In the **Naming Rule** dialog box, in **Name**, type the name for the new naming rule. The name must begin with the project prefix. For example, **A4_**.

3. To add a naming rule pattern in the **Patterns** list, click **Add**.

© 2025 Siemens

The Add Naming Rule Pattern wizard runs.



4. In the **Pattern** dialog box, enter information for the new rule pattern.

For this parameter	Do this
Pattern	Enter a naming pattern. You can add pattern characters three ways: • Type characters using the keyboard. • Click **Insert LOV** and add an LOV. • Click **Insert Rule** and add an existing naming rule. Example: For a three-digit number pattern running 001 to 999, type **nnn**. For a two-character alphabetic pattern running from aa to zz, type **aa**. For a two-character pattern running AA to ZZ, type **AA**. For a mixture of letters and numbers, such as A001 to Z999, type **Annn**. The following dynamic characters can be used in naming rule patterns: **U** Uppercase dynamic character **u** Lowercase dynamic character **D** Mixed-case dynamic character When you use a dynamic character in a pattern and select the **Generate Counter** ☑ check box, corresponding characters typed in the **Initial Value** box are used in all subsequent IDs.

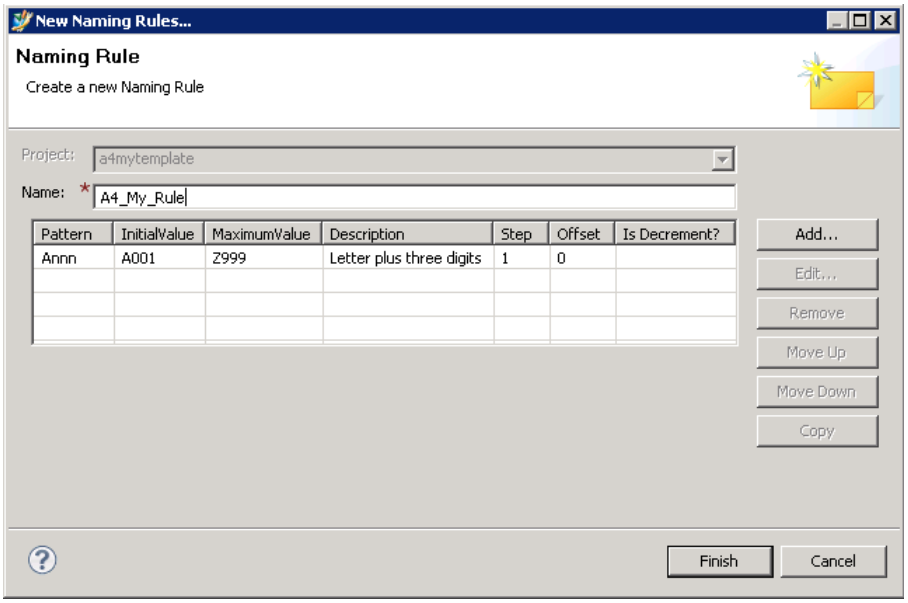
© 2025 Siemens

For this parameter	Do this
	Example: If the pattern is **UUU"-"NNNNN** and you type **REQ-00000** in the **Initial Value** box, then all IDs automatically generated using that pattern begin with **REQ** (**REQ-00000**, **REQ-00001**, **REQ-00002**, and so on). However, end users can override the generated text in the client user interface. For example, they can either click **Assign**, or they can replace the **REQ** in the ID with some other text. Therefore, the pattern is dynamic, allowing it to be changed by end users.
Description	Type a brief description of the naming rule.
Generate counters?	Select the check box if you want to generate counters from the pattern. When a naming rule includes multiple patterns that generate counters, a selection list of the patterns will be available in the Teamcenter client for use with **Assign**. Patterns that include an inserted LOV, an inserted naming rule, a system variable that is not enclosed in quotes, or a regular expression cannot be used to generate counters.
Is Decrement?	Select the check box if the counter is to be reduced by the Step amount, rather than increased.
Initial Value and **Maximum Value**	If you selected the **Generate counters?** check box, then type characters that match the pattern to set the initial and maximum values. For example, if you entered **nnn** for the pattern, then type a three-digit number in the **Initial Value** box and the **Maximum Value** box, such as **100** and **899**. Or, if you entered a pattern of **Annn**, then you might type **A001** and **Z999**.
Step	Type the amount by which the generated counters are to be incremented. The default is **1**, meaning that each additional number that is generated is to be changed by one.
Offset	Type a number by which the generated counter is increased the first time the rule is used. The default is **0**, meaning there is no offset.

5. Click **Finish**.

The pattern appears in the **Pattern** list in the **Naming Rule** dialog box.

© 2025 Siemens



Additional patterns can be added to the list to allow end users a choice of rules to follow when manually typing an ID. All the valid patterns are displayed in the end-user interface to provide guidance to the end user.

> Note:
>
> - You can add a pattern to the list that allows the end user to type any ID desired, for example: **%^{1,128}**.
>
> - If the rule will be attached to the **item_revision_id** property of an object revision, then to ensure that a valid option is generated, enable a counter for at least one pattern.

6. Change the patterns as desired by using the **Add**, **Edit**, **Remove**, **Copy**, **Move Up** and **Move Down** buttons.

7. Click **Finish**.

   The naming rule is added to the **Naming Rules** folder.

To apply the naming rule, **attach it to a property** on a business object, such as an item name.

Add a revision naming rule

A *revision naming rule* is a business rule that defines the naming convention and sequence for a revision property. This functionality is required for the strict naming requirements needed for some industries. If you have the Aerospace and Defense Foundation template installed, then you may need to use this functionality to conform to industry naming standards.

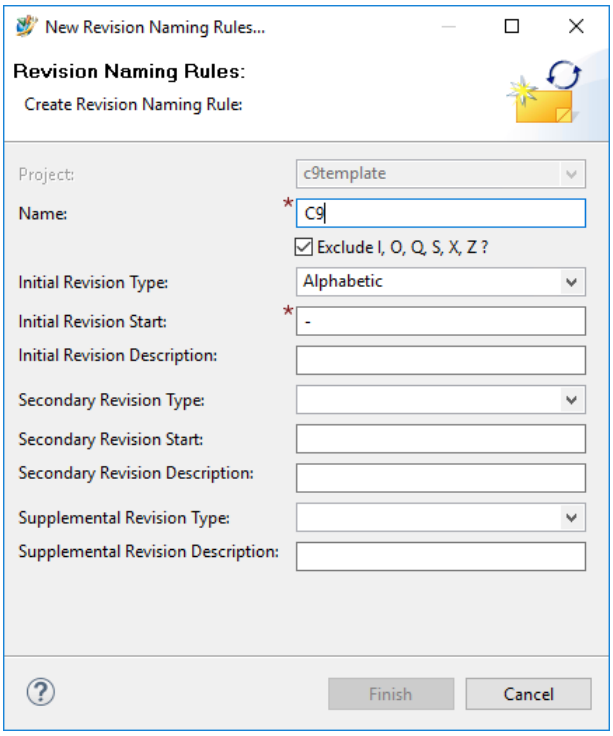
© 2025 Siemens

To define the scheme for the rule, you use a **valid combination** of initial, secondary, and supplemental patterns.

1. Start the New Revision Naming Rules wizard in one of these ways:

   - On the menu bar, choose **BMIDE**>**New Model Element**, type **Revision Naming Rules** in the **Wizards** box, and click **Next**.

   - In the **BMIDE** view, right-click the **Extensions \ Rules \ Revision Naming Rules** folder and choose **New Revision Naming Rules**.



2. In the **Revision Naming Rules** dialog box, enter the information as needed for your naming pattern.

For this parameter	Do this
Name	Type the name for the revision naming rule. The name must begin with the project prefix.
Exclude [letters] ?	To exclude the [letters] characters from use in the rule, select the check box. The characters are set in the **TcRevisionSkipLetters** LOV.
Initial Revision Type	Choose the kind of revision numbering system to use at the start of the revision name scheme, one of:

© 2025 Siemens

<table>
<tr><th>For this parameter</th><th>Do this</th></tr>
<tr><td></td><td>Alphabetic (only alphabet characters)
Numeric (only numbers)
Alphanumeric (a mix of letters and numbers)</td></tr>
<tr><td>Initial Revision Start</td><td>Type the characters to start the revision numbering. The characters must be of the Initial Revision Type.

For example, if the desired revision name scheme is A001 to Z999, type A.</td></tr>
<tr><td>Initial Revision Description</td><td>Type a description of the start of the revision name scheme.</td></tr>
<tr><td>Secondary Revision Type</td><td>Choose the kind of numbering system to use for the second portion of the revision name scheme. This must be a different type from the Initial Revision Type.</td></tr>
<tr><td>Secondary Revision Start</td><td>Type the characters to start the second portion of the revision name scheme numbering. The characters must be of the Secondary Revision Type.

For example, if the desired revision numbering scheme is A001 to Z999, type 001 in this box.</td></tr>
<tr><td>Secondary Revision Description</td><td>Type a description of the end portion of the revision naming scheme.</td></tr>
<tr><td>Supplemental Revision Type</td><td>Select a pattern for the end portion of the revision name scheme.
<table>
<tr><th>Type</th><th>Description</th><th>Example</th></tr>
<tr><td>NumericNoZeroFill</td><td>Assigns numbers with no zeros.</td><td>1, 2, 3, up to 9999</td></tr>
<tr><td>FixedTwoDigitsZeroFill</td><td>Assigns two-digit numbers using a zero fill-in.</td><td>01, 02, 03, up to 99</td></tr>
<tr><td>FixedThreeDigitsZeroFill</td><td>Assigns three-digit numbers using a zero fill-in.</td><td>001, 002, 003, up to 999</td></tr>
<tr><td>FixedFourDigitsZeroFill</td><td>Assigns four-digit numbers with a zero fill-in.</td><td>0001, 0002, 0003, up to 9999</td></tr>
<tr><td>CurrentRevLetterNumericNoZeroFill</td><td>Assigns numbers to the current revision letter with no zeros.</td><td>A1, A2, A3, up to A99</td></tr>
<tr><td>CurrentRevLetterFixedOneDigit</td><td>Assigns single-digit numbers to the current revision letter.</td><td>A1, A2, A3, up to A9</td></tr>
</table>
</td></tr>
</table>

© 2025 Siemens

<table>
<tr><th>For this parameter</th><th>Do this</th></tr>
<tr><td></td><td>
<table>
<tr><th>Type</th><th>Description</th><th>Example</th></tr>
<tr><td>CurrentRevLetterFixedTwoDigitsZeroFill</td><td>Assigns two-digit numbers to the current revision letter.</td><td>A01, A02, A03, up to A99</td></tr>
<tr><td>NextRevLetterNumericNoZeroFill</td><td>Assigns numbers to the next revision letter.</td><td>B1, B2, B3, up to B99</td></tr>
<tr><td>NextRevLetterNumericFixedOneDigit</td><td>Assigns single-digit numbers to the next revision letter.</td><td>B1, B2, B3, up to B9</td></tr>
<tr><td>NextRevLetterFixedTwoDigitsZeroFill</td><td>Assigns two-digit numbers to the next revision letter.</td><td>B01, B02, B03, up to B99</td></tr>
</table>

Note:

For the CurrentRevLetter and NextRevLetter types, the default supplemental revision name starts with current or next revision letter, but a user can type any letter except the ones controlled by the TcRevisionSkipLetters LOV.

Use the following preferences to prevent users from typing a letter for the revision:

• Treat_Curr_Rev_As_Any_Rev Set to False to prevent users from changing the current revision letter.

• Treat_Next_Rev_As_Any_Rev Set to False to prevent users from changing the next revision letter.
</td></tr>
<tr><td>Supplemental Revision Description</td><td>Type a description of the end portion of the revision naming scheme.</td></tr>
</table>

3. Click **Finish**.

   The naming rule is added to the **Revision Naming Rules** folder.

To apply the revision naming rule, **attach it to a property** on a business object, such as an item name.

© 2025 Siemens

Assign a baseline suffix naming rule

A *baseline* is an interim version of a design for future reference. When you create a baseline, you can assign a naming rule to it.

The **Default Baseline Suffix Rule** has the **"."NNN** pattern with an initial value of **.001** and a maximum value of **.999**. If you want to create your own baseline naming rule, you can define the rule using the same pattern (and ensure you select the **Generate Counters** check box).

1. In the **BMIDE** view, browse to the **ItemRevision** business object or a child of the **ItemRevision** business object. (Baseline suffix naming rules are only allowed for item revisions or children of item revisions.) To search for a business object, you can click the **Find** button at the top of the view.

2. Right-click the item revision business object and choose **Open**.

   The details of the object appear on the **Main** tab in a new view.

3. To assign a naming rule to the item revision business object, click the **Browse** button to the right of the **Baseline Suffix** box. Select a naming rule (such as **Default Baseline Suffix Rule**) and click **OK**.

4. To save the changes to the data model, choose **BMIDE→Save Data Model**, or click the **Save Data Model** button on the main toolbar.

5. Deploy your changes to the server. Choose **BMIDE→Deploy Template** on the menu bar, or select the project and click the **Deploy Template** button on the main toolbar.

6. After deployment, test your newly assigned baseline suffix naming rule in the Teamcenter rich client by creating a baseline and checking the name that is generated.

   To baseline an item, open an item revision in the Structure Manager and choose **Tools→Baseline**.

Attach a naming rule to a property

To put a naming rule into effect, you must attach it to a property of a business object. For example, if you want to use a naming rule to define how **Item** business objects items are named, attach the naming rule to the **item_id** property of the **Item** business object.

Administrators can attach naming rules to any persistent string property, except a property defined as an array. This allows organizations to enforce naming standards.

Use these notes when attaching naming rules:

- You cannot attach a naming rule to the **object_name** property of the **Form** business object or its children.

© 2025 Siemens

- You can attach naming rules to the Product Configurator option values; however, use these guidelines for the Boolean data type:

  - Attach the same naming rule to both the Boolean family and Boolean value. Only attaching the naming rule to the Boolean value generates an error message when only a Boolean value is created.

  - If you must attach different naming rules for the Boolean family and Boolean value, customize the Boolean family type and use it in Product Configurator.

- If you want to use the same naming pattern for two different business object types, Siemens Industry Software Inc. recommends creating a separate naming rule for each type. This ensures that each type has its own counter. If you attach the same naming rule to two different types, they share the same counter.

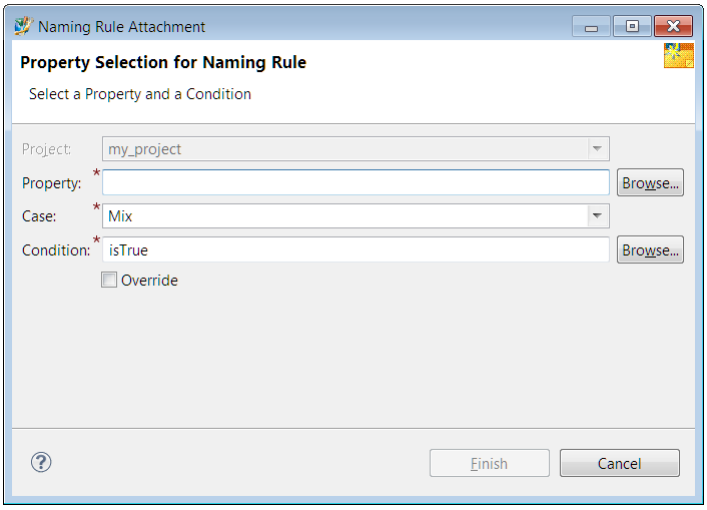
The following procedure describes how to open the naming rule and add property attachments in the **Naming Rule Attachments** table.

1. Open the **Extensions\Rules\Naming Rules** folders, right-click the naming rule you want to assign, and choose **Open**.

   The Naming Rule editor is displayed.

2. Click the **Attach** button to the right of the **Naming Rule Attachments** table.

   The Naming Rule Attachment wizard runs.



3. Perform the following steps in the **Property Selection for Naming Rule** dialog box:

   a. Click the **Browse** button to the right of the **Property** box to choose the business object and property to which you want to attach the naming rule. You can also press Alt+C or start typing to see a list of suggestions.

© 2025 Siemens

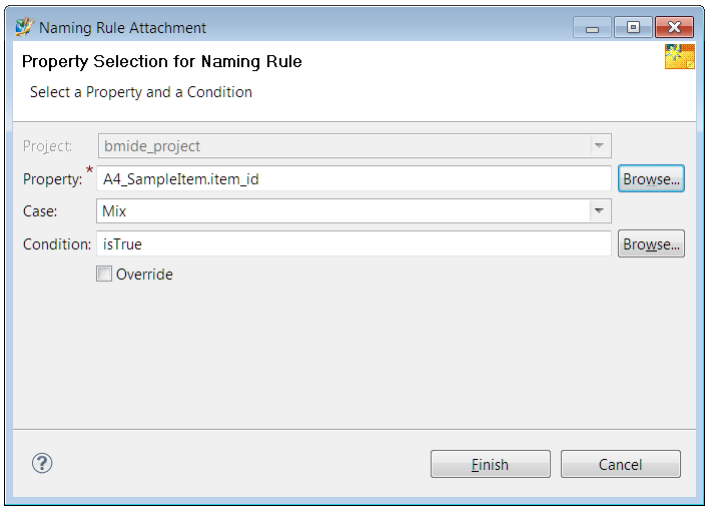
b. Click the arrow in the **Case** box to select the kind of user input allowed for the naming rule, **Mix** for mixed case, **Upper** for uppercase, or **Lower** for lowercase.

> Note:
>
> If a naming rule has uppercase characters in the pattern (for example, **Ann**), you must choose **Mix** or **Upper**. If you choose **Lower** in this situation, the end user encounters an error when the naming rule is generated in the client user interface. To correct the problem, replace the **A** in the pattern with **@**, for example, **@nn**.

c. Click the **Browse** button to the right of the **Condition** box to **select the condition that determines when the naming rule is applied**. If you select **isTrue** as the condition, the value always applies.

> Note:
>
> Only those conditions appear that have valid signatures. For naming rules, the valid condition signature is as follows:
>
> *condition-name***(UserSession)**

d. Select **Override** to **override the existing naming rule** attached to this property.

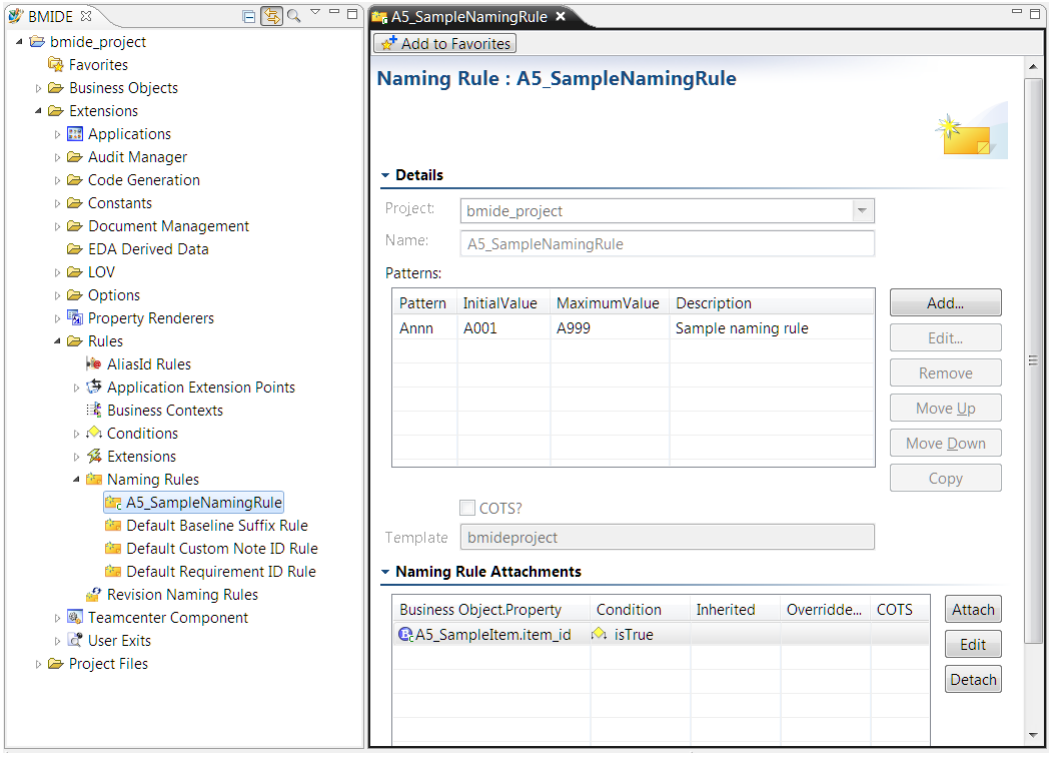


e. Click **Finish**.

The naming rule is attached to the property on the business object. The business objects and properties display in the **Naming Rule Attachments** table.

To detach the naming rule from any particular property, select the business object in the table and click the **Detach** button.

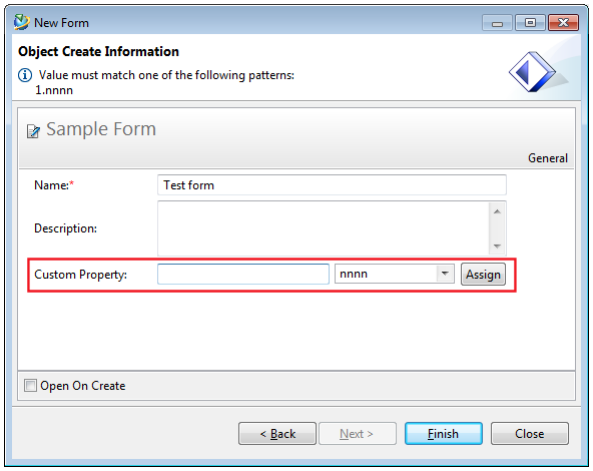
© 2025 Siemens



4. When you attach a naming rule to a persistent string property, use the **Operation Descriptor tab** to make the property both visible and required so that users are prompted to assign the naming rule when creating the object.

   For example, in the following figure, the custom property was set as visible so that the user is prompted to assign a name.

> Tip:
>
> After creation, the **Assign** button and hint box are *not* displayed with the property.



© 2025 Siemens

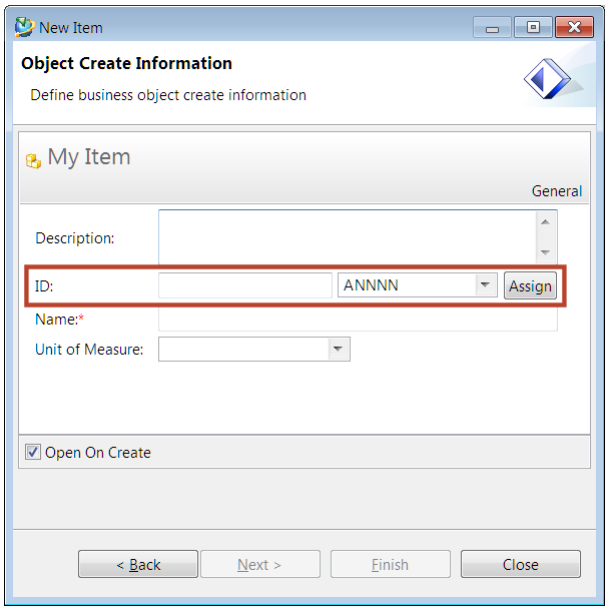
5. To save the changes to the data model, choose **BMIDE→Save Data Model**, or click the **Save Data Model** button on the main toolbar.

6. Deploy your changes to the server. Choose **BMIDE→Deploy Template** on the menu bar, or select the project and click the **Deploy Template** button on the main toolbar.

7. After deployment, test your newly attached naming rule in the Teamcenter rich client by creating an instance of the business object.

   For example, if you attached a naming rule to the **item_id** property on a custom item business object, in the My Teamcenter application, try creating some items to see if the rule is applied. For example, choose **File→New→Item** and choose the custom item as the type. When you click the **Assign** button, your new naming rule pattern appears in the **ID** box.

   > Tip:
   >
   > If a naming rule has any patterns that are not automatically assignable such as a regular expression, nested rules, and so on, then the **Assign** button is not displayed.



Attach naming rules with conditions

You can **use conditions to determine when naming rules are applied**. When you attach the naming rule, select the condition that determines when the naming rule should be used.

1. Open the business object with the property whose naming rule you want to set with a condition.

2. On the **Properties** tab, select the property for which you want to use the naming rule.

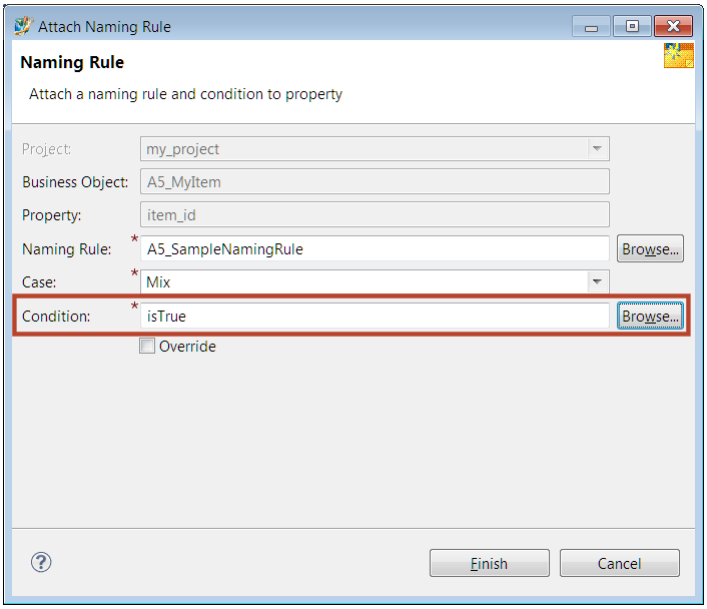
3. Click the **Add** button to the right of the **Naming Rule Attaches** table.

© 2025 Siemens

The Attach Naming Rule wizard runs.

4. Click the **Browse** button to the right of the **Property** box to select the property.

5. Click **Browse** to the right of the **Condition** box to select the condition that determines when the naming rule applies.



6. Select the **Override** check box to replace the existing naming rule attached to that property, if any.

7. Click **Finish**.

The condition when the naming rule applies is displayed in the **Naming Rule Attachments** table.

Override a COTS naming rule

You can override COTS naming rules with custom naming rules by selecting the **Override** button when you attach the naming rule.

1. Open the business object with the property whose naming rule you want to override, for example, **Item**.

2. On the **Properties** tab, select the property whose naming rule you want to override, for example, **item_id**.

3. Click the **Add** button to the right of the **Naming Rule Attaches** table.

The Attach Naming Rule wizard runs.

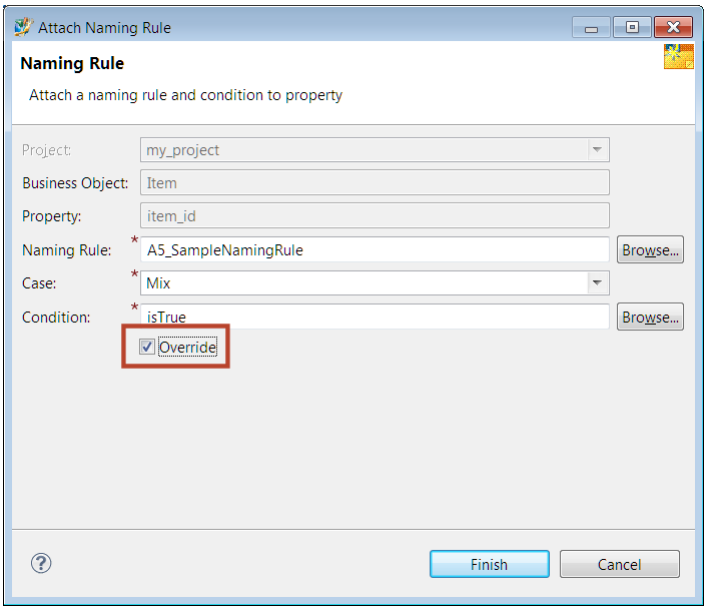
© 2025 Siemens

4. Click **Browse** to the right of the **Naming Rule** box to select the new naming rule that you want to override the COTS naming rule.

5. In the **Condition** box, ensure the **istrue** condition is selected if you are overriding using a custom naming rule. The override can be used only with **istrue** condition while using a custom naming rule.

6. Select the **Override** check box.

   Selecting this button overrides the COTS naming rule.



7. Click **Finish**.

Naming rule patterns

Defining naming rule patterns

Naming rules are considered pattern variables. Naming rule patterns can be composed of literal strings of characters, lists of patterns using lists of values (LOVs) as literal variables, and pattern variables composed of existing naming rules.

> Note:
>
> Lists of values are literal variables and must be enclosed in double quotation marks (**"**).

Each pattern can consist of combinations of the characters shown in the following table.

© 2025 Siemens

Character	Pattern match
&	Mixed case alphabetic value or numeric value
X	Uppercase alphabetic value or numeric value
x	Lowercase alphabetic value or numeric value
N	Numeric value
n	Numeric value
@	Alphabetic value
A	Uppercase alphabetic value
a	Lowercase alphabetic value
U	Uppercase dynamic character
u	Lowercase dynamic character
D	Mixed case dynamic character
"*string***"**	String delimiters. For example, if you want the word **Part** to appear in the naming rule, define it as **"Part"** in the naming rule pattern.
?	Any single character value
****	Escape the next character to have a literal meaning. This character is only required when using delimiters inside of other delimiters, for example, **"A\"Special\"Name"**, which matches **[A "Special" Name]**. If you do not use the back slash (****) it is not possible to include delimiters as part of a name. The same is true of braces (**{ }**) and brackets (**[ ]**). For example, the pattern `{RULE:Test\$Now\{a\}${ROLE}}`, applied to a user whose current role is **Designer**, would look for the following definition: `Test$Now{a}Designer`
${*system-variable***}**	Substitutes the value of a Teamcenter system variable.

© 2025 Siemens

Character	Pattern match

> Note:
>
> System variables are normally used in double quotation marks. However, it is possible that a preference value can be a pattern rather than a literal.

The following values are valid for system variable:

GROUP	User's current group.
ROLE	User's current role.
USERID	User's ID.
USERNAME	User's name.
SITENAME	Site name.
All:*pref*	Specifies a preference variable.
GROUP:*pref*	Specifies a group protection scope preference variable.
ROLE:*pref*	Specifies a role protection scope preference variable.
SITE:*pref*	Specifies a site protection scope preference variable.
USER:*pref*	Specifies a user protection scope preference.

{*variable-type* **:***variable-name***}**

Literal variable delimiters **{***Type***:***Name***}**

Values in the list are treated as quoted strings. The following values are valid for variable type:

LOV	Lists of values
RULE	*NAME_Rules*

[*variable-type***:** *pattern-name***]**

Pattern variable delimiters **[***Type***:***Name***]**

Values in the list are treated as patterns. The following values are valid for variable type:

LOV	Lists of values
RULE	*NAME_Rules*

Regular expressions

Regular expressions are valid inputs to the patterns. The regular expression delimiter **%** indicates that the rest of the pattern is a regular expression. The regular expression delimiter allows names to be validated, not generated, using standard regular expressions. This delimiter cannot be used in a pattern that automatically generates counters.

© 2025 Siemens

Character	Pattern match

For example, the pattern for an item ID with two uppercase alphabetic characters followed by one digit is **AAn**. However, if the digit was not allowed to be zero, the following regular expression can be used: **AA%^[1-9]$**.

You cannot use non-regular expression characters after the **%** delimiter. If you attempt to do so, you receive an error when you attempt to create the pattern.

> Caution:
>
> If you place the **%** expression before dynamic characters (**u**, **U**, or **D**), an error is not thrown, for example:
>
> ```
> %^[mejk]$DDD
> ```
>
> Although an error is not thrown, it is still an invalid expression.

The circumflex (**^**) at the beginning of the input constrains the input to match an initial segment of a line. The dollar sign (**$**) at the end of the input constrains the input to match a final segment of a line. Therefore, the construction **^[1-9]$** means that the expression must match the entire segment, not only part of it.

The following values are valid regular expressions:

?	0 or 1
*****	0 or more
+	1 or more
.	Any character
|	Or (as in **this OR that**)
( )	Define a section, for later reference in short form
[]	Define a set, one of the enclosed characters
-	Range

You must define sets of characters to represent the pattern of your naming convention. Some simple examples follow:

[a-z0-9]	One lowercase letter or number
[a-z0-9]+	One or more lowercase letters or numbers in any order
[a-z]+[0-9]+	One or more lowercase letters followed by one or more numbers
[a-hj-np-z]	Any one lowercase letter except **i** and **o**

© 2025 Siemens

Character	Pattern match
[mejx]	Any one of the letters **m**, **e**, **j**, or **x**

> Note:
>
> The Teamcenter regular expression parser is based on classic UNIX.
>
> The use of shorthand forms **\w** and **\d** is not supported. These metacharacters are regular expression extensions used in scripting languages like Perl, PHP, and Python.
>
> You can find more information about regular expressions on the Internet by searching for the phrase **regular expression** or **regex**.

To specify a special character as a literal character within a set, you must use the escape character, which on Windows systems is the backslash **** and on Linux systems is the slash **/**. For example:

[a-z]	One lowercase letter or number
[a\-z]+	One or more of the letters **a** or **z**, or the dash character, on Windows systems

You can use quantifiers to allow specific numbers of characters. Instead of using **[0-9][0-9][0-9]** to represent any three digits, you can write the regular expression with a quantifier as **[0-9]{3}**. For example:

{n,m}	At least **n** and no more than **m**
{n,}	At least **n**
{,m}	May have **0**, but no more than **m**
{m}	Exactly **m**

Using literal variables in patterns

To illustrate the use of literal variables, assume that the naming convention for a particular type of dataset requires a 3-character suffix, either **ENG** or **MFG**, followed by a 4-digit number ranging from **0000** to **9999**.

To establish this pattern, a list of values (LOV) named **Context** is created containing the values **ENG** and **MFG**. A naming rule is then created using the LOV and numeric characters. The pattern would appear as follows:

{*LOV*:*Context*}*nnnn*

© 2025 Siemens

The naming rule is then attached to the **name** property of the dataset business object.

> Note:
>
> Literal variables cannot be used in patterns used to autogenerate counters.

Using counters with naming rules

The first pattern in the naming rule is used with counters. If there are more patterns, the client prompts you to choose the pattern you want to use.

Any number of counters can be used in a pattern, for example **nnn"."a**. With counters activated for this pattern, the **Assign** button allocates IDs as follows:

```
000.a
000.b
000.c
:
000.z
```

The right-side counter range completes first and then moves to the next range from the right, as follows:

```
001.a
:
001.z
002.a
:
002.z
003.a
:
003.z
```

Using system variables in patterns

System variables can be used in naming rule patterns to retrieve information from Teamcenter. Some examples of system variables are **${USERID}**, **${SITENAME}**, and **${GROUP}**.

If the pattern contains a system variable inside quotation marks, for example, **"${GROUP}-AD-"NNN**, then you can select the **Generate counters?** check box when creating the naming rule.

When generating counters for the **"${GROUP}-AD-"NNN** example, the **Initial Value** and **Maximum Value** boxes can contain values similar to the following:

© 2025 Siemens

Pattern	**"${GROUP}-AD-"NNN**
Initial Value	**${GROUP}-AD-111**
Maximum Value	**${GROUP}-AD-999**

Another similar example is:

Pattern	**"AA-${GROUP}-BB-"NN**
Initial Value	**AA-${GROUP}-BB-11**
Maximum Value	**AA-${GROUP}-BB-99**

Using dynamic characters in naming rule patterns

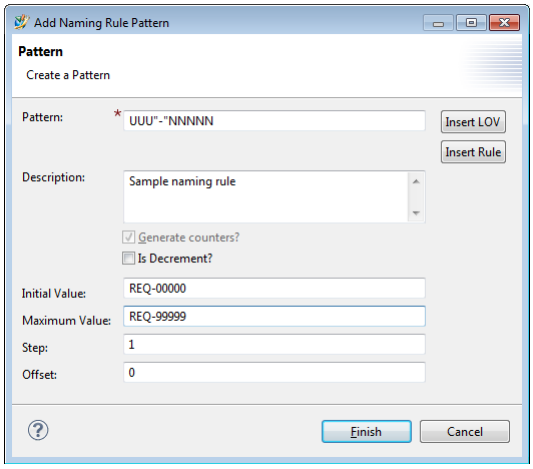
Dynamic characters in **naming rule patterns** allow end users to override the automatically generated text in the client user interface. The following dynamic characters can be used:

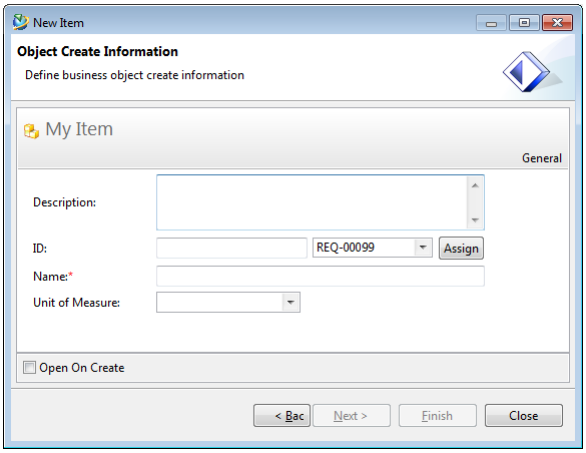
U - Uppercase dynamic character
u - Lowercase dynamic character
D - Mixed-case dynamic character

When you use a dynamic character in a pattern and select the **Generate Counter** ☑ check box, corresponding characters typed in the **Initial Value** box are used in all subsequent automatically generated IDs.

For example, if the pattern is **UUU"-"NNNNN** and you type **REQ-00000** in the **Initial Value** box, all IDs automatically generated using that pattern begin with **REQ** (**REQ-00000**, **REQ-00001**, **REQ-00002**, and so on).

© 2025 Siemens





In the client user interface, end users can override the text automatically generated for the dynamic characters. Instead of clicking the **Assign** button, they can replace the **REQ** in the ID with some other text.

Naming rules reference

Example: property inheritance and naming rule behavior

Assume that for the **Item** class a business object, **Document**, exists. A sub-business object of the **Document** business object, **MyDoc**, exists. Naming rules are defined for the properties of the object class/business object/sub-business object as shown in the following table.

Object	Property	Naming rule applied
Item	**item_id**	**RuleA**
	name	**RuleB**

© 2025 Siemens

Object	Property	Naming rule applied
Document	**item_revision_id**	**RuleC**
MyDoc	**name**	**RuleD**

With these rules defined, rules are applied accordingly when a user performs the following actions:

When creating an instance of this object	These rules are applied
Item	The item ID and name for the new **Item** are derived using naming rules **RuleA** and **RuleB**, which are defined in the top-level **Item** business object.
Document	The item ID and name are derived using naming rules **RuleA** and **RuleB**, which are inherited from the **Item** business object. The item revision ID is derived using naming rule **RuleC**, which is defined in the **Document** business object.
MyDoc	The item ID is derived using naming rule **RuleA**, which is inherited from the **Item** business object. The item revision ID is derived using naming rule **RuleC**, which is inherited from the **Document** item business object, and the name is derived using **RuleD**, which is defined in the **MyDoc** sub-business object.

Note:

Properties on a business object are inherited by all its sub-business objects. Properties that are removed from a business object are no longer inherited by all its sub-business objects. Because naming rules define valid patterns for values of properties associated with business objects, the rules are likewise subject to the principles of inheritance.

Applying naming rules to identifiers and identifier revisions

The **Identifier** class uses pairs of *identifier-name* and *identifier-name***Rev** business objects. Alternate identifiers of the *identifier-name* business object are item-level masters. Alternate identifiers of the *identifier-name***Rev** business object are revision levels supplements to the item-level master.

Naming rules on the ID property of a master identifier business object are automatically inherited by the ID property of the supplemental revision-level identifier business object. Therefore, unless a separate naming rule is defined for the revision-level identifier, the ID property of both the master and the revision-level business object use the same naming rule, which results in the sequence counter for the identifier revision-level IDs being increased each time a revision is created, as shown in the following example:

```
CAAA00011-CORP1048                          (Item)
CAAA00011                                   (Item Master)
ALT-AAA00009@Evaluat-Nozzle EVA             (Identifier)
```


© 2025 Siemens

```
ALT-AAA00017@Evaluat-jojojo                        (Identifier)
CAAA00011/A-CORP1048                               (Item Revision)
CAAA00011/A                                        (Item Revision Master)
ALT-AAA00009/ALT-AAA00010@Evaluat-Nozzle EVA       (IdentifierRev)
ALT-AAA00017/ALT-AAA00018@Evaluat-jojojo           (IdentifierRev)
```

To prevent this behavior, you must define a separate revision naming rule on the ID property of supplemental revision-level identifier business objects, as follows:

1. Define a naming rule, for example, **MyRule1**, that generates the counter and attach it to the ID property of the identifier.

2. Define another naming rule, for example, **MyRule2**, that is identical to **MyRule1** and attach it to the ID property of the identifier revision.

   The alternate ID is generated and the counter is incremented as follows:

   ```
   000/000@Master-1
   001/000@Master-1
   002/000@Master-1
   003/000@Master-1
   ```

Using conditions with naming rules

You can **attach naming rules using conditions** to determine when naming rules are applied. You can attach multiple naming rules for the same property and business object.

Consider the following use case:

Business object	Property	Naming rule	Condition	Description
MyItem	**item_id**	**naming_rule1**	**isTrue**	Sets the default naming rule for the property on the business object.
MyItem	**item_id**	**naming_rule2**	**isProject1**	Checks if the Teamcenter session current project is **Project1**.
Item	**item_id**	**naming_rule3**	**isProject2**	Checks if the Teamcenter session current project is **Project2**.

For the above use case, the following table shows the selected rule depending on the user's current project in the Teamcenter session and, when applicable, whether **Override** is selected for **naming_rule1**.

© 2025 Siemens

Current project	Condition for naming_rule1 evaluates to	Condition for naming_rule2 evaluates to	Override selected on naming_rule1	Condition for naming_rule3 evaluates to	Selected rule(s)
Project1	true	true	does not apply	does not apply	**naming_rule1 naming_rule2**
Project2	true	false	true	does not apply	**naming_rule1**
Project2	true	false	false	true	**naming_rule3**
Project3	true	false	true	does not apply	**naming_rule1**
Project3	true	false	false	false	**naming_rule1**

Impact of conditions on naming rule overrides

There are only two types of conditions allowed for naming rule attachments, **IsTrue** conditions and session-based conditions. In evaluating **naming rule attachments that use conditions**, the session-based condition takes precedence over the **IsTrue** condition.

At any level of the business object hierarchy, if any session-based condition evaluates to true, the corresponding naming rule is used. If the **Override** check box is selected on a naming rule attachment with an **IsTrue** condition, the parent traversal is stopped at that level and the first encountered naming rule attachment with an **IsTrue** condition is used. Otherwise, naming rules at the parent business object are evaluated in similar manner.

The following example shows how a naming rule is applied while creating an instance of a **MyCustomItem2** business object where the **Override** check box is selected on a naming rule attachment with an **IsTrue** condition at the **MyCustomItem** business object. All session-based conditions are evaluated to false so the traversal continues until the **NR6** naming rule where the override is set:

```
item.item_id : NR1.isTrue(): (override=true)
Item.item_id: NR2: IsSession2()
Item.item_id: NR3: IsSession3();
MyCustomItem.item_id: NR4:IsSession4()
MyCustomItem.item_id: NR5:IsSession5()
MyCustomItem.item_id: NR6:isTrue() (override = false )
MyCustomItem2.item_id: NR7:IsSession7()
MyCustomItem2.item_id: NR8:IsSession8()
MyCustomItem2.item_id: NR9:isTrue() (override = false )
```

When all the following session-based conditions are false, the naming rule attachment related to the **NR9** naming rule is returned for the **MyCustomItem2** business object:

```
IsSession7 = false
IsSession5 = false
IsSession4 = false
```


© 2025 Siemens

```
IsSession3 = false
IsSession2 = false
```

Revision naming rules reference

A *revision naming rule* is a business rule that defines the naming convention and sequence for a revision property.

The following table shows valid combinations of initial, secondary, and supplemental format types for revision naming rules.

Initial format	Initial examples	Secondary format	Secondary examples	Supplemental format	Supplemental examples
Alphabetic	**A, B...** **AA, AB...** **AAA, AAB...**	None		None	
Alphabetic	**A, B...** **AA, AB...** **AAA, AAB...**	None		Numeric	**1, 2... 999** **01, 02... 999** **001, 002... 999**
Alphabetic	**A, B...** **AA, AB...** **AAA, AAB...**	None		Alphanumeric	**A1, A2...** **A01, A02...** **AA1, AA2...**
Alphabetic	**A, B...** **AA, AB...** **AAA, AAB...**	Numeric	**1, 2... 999** **01, 02... 999** **001, 002... 999**	None	
Alphabetic	**A, B...** **AA, AB...** **AAA, AAB...**	Alphanumeric	**A1, A2...** **A01, A02...** **AA1, AA2...**	None	
Numeric	**1, 2... 999** **01, 02... 999** **001, 002... 999**	None		None	
Numeric	**1, 2... 999** **01, 02... 999** **001, 002... 999**	Alphabetic	**A, B...** **AA, AB...** **AAA, AAB...**	None	
Numeric	**1, 2... 999** **01, 02... 999** **001, 002... 999**	Alphabetic	**A, B...** **AA, AB...** **AAA, AAB...**	Alphanumeric	**A1, A2...** **A01, A02...** **AA1, AA2...**
Numeric	**1, 2... 999** **01, 02... 999** **001, 002... 999**	Alphanumeric	**A1, A2...** **A01, A02...** **AA1, AA2...**	None	
Alphanumeric	**A1, A2...** **A01, A02...** **AA1, AA2...**	None		None	

© 2025 Siemens

Initial format	Initial examples	Secondary format	Secondary examples	Supplemental format	Supplemental examples
Alphanumeric	**A1, A2...** **A01, A02...** **AA1, AA2...**	Alphabetic	**A, B...** **AA, AB...** **AAA, AAB...**	None	
Alphanumeric	**A1, A2...** **A01, A02...** **AA1, AA2...**	Alphabetic	**A, B...** **AA, AB...** **AAA, AAB...**	Numeric	**1, 2... 999** **01, 02... 999** **001, 002... 999**
Alphanumeric	**A1, A2...** **A01, A02...** **AA1, AA2...**	Numeric	**1, 2... 999** **01, 02... 999** **001, 002... 999**	None	

The following table shows the supported formats of the supplemental revision types.

Supplemental revision format	Current released revision	Example
NumericNoZeroFill	Any revision	**1**, **2**, **3**, up to **9999**
FixedTwoDigitsZeroFill	Any revision	**01**, **02**, **03**, up to **99**
FixedThreeDigitsZeroFill	Any revision	**001**, **002**, **003**, up to **999**
FixedFourDigitsZeroFill	Any revision	**0001**, **0002**, **0003**, up to **9999**
CurrentRevLetterNumericNoZeroFill	**A**	**A1**, **A2**, **A3**, up to **A99**
	1	Not applicable
	A01	**A02**, **A03** up to **A99**
CurrentRevLetterFixedOneDigit	**A**	**A1**, **A2**, **A3**, up to **A9**
	1	Not applicable
	A01	**A02**, **A03** up to **A99**
CurrentRevLetterFixedTwoDigitsZeroFill	**A**	**A01**, **A02**, **A03**, up to **A99**
	1	Not applicable
	A01	**A02**, **A03** up to **A99**
NextRevLetterNumericNoZeroFill	**A**	**B1**, **B2**, **B3**, up to **B99**
	1	Not applicable
	A01	**A02**, **A03** up to **A99**
NextRevLetterNumericFixedOneDigit	**A**	**B1**, **B2**, **B3**, up to **B9**
	1	Not applicable
	A01	**A02**, **A03** up to **A99**
NextRevLetterFixedTwoDigitsZeroFill	**A**	**B01**, **B02**, **B03**, up to **B99**
	1	Not applicable
	A01	**A02**, **A03** up to **A99**

© 2025 Siemens

Note:

For those formats in the table whose current released revision is **A01**, if the current revision is alphanumeric, that alphanumeric sequence is followed.

If the revision rule uses special characters (for example, **. , ' + !** and so on), priority is given to alphanumeric characters when generating the next revision ID in the sequence. If there is ambiguity in the format between the most recent ID and a previous ID, priority is given to the most recently created revision ID to serve as the basis of the new revision ID.

The following table lists the revisions shown for various revision ID situations.

For this current revision ID situation	These revisions are shown
Current rev id is initial and the only available rev id is revOptions[0] (initial).	Initial
Current rev id is initial, secondary is valid type.	Both
Current rev id is initial, secondary is invalid.	Initial and supplemental
Current rev id is secondary, supplementary is invalid.	Secondary
Current rev id is secondary, and supplementary is valid.	Both
Current rev id is supplemental, and secondary is valid.	Both
Current rev id is supplemental, secondary is invalid.	Supplemental and primary
Fallback.	Add initial and secondary

For example, consider the following application of these rules.

1. The following revision rule is attached to the property item_revision_id for the object MyItem.

   Initial revision type: Numeric
   Initial revision start: 00
   Secondary revision type: Alphanumeric
   Secondary Revision Start: E00

2. Create an instance of MyItem. 00 is assigned to the revision property.

3. Revise the instance. Because the current ID is initial type and secondary type is valid, choices of 01 and E00 are available for revision ID . Assign E00.

4. Revise the instance again. Because the current ID is secondary type and supplementary is invalid (undefined), only E01 is available for revision ID.

© 2025 Siemens

Renaming a naming rule

You cannot rename a naming rule. The only way to give a naming rule a new name is to detach the naming rule from the property and attach a new naming rule that has the same pattern as the old naming rule. However, if you do this, the counter does not start at the highest number assigned by the previous naming rule. Instead, it looks for unassigned numbers in the pattern and starts at the earliest one available.

For example, suppose that the pattern is **NNN** with an initial value of **001** and a maximum value of **999**. Suppose that the highest number assigned by the previous naming rule was **100**, and also suppose that items assigned the numbers **003**, **014**, and **098** have been deleted. You may expect that the first number assigned using the new naming rule would be **101**. Instead, the new naming rule looks for unassigned numbers in the pattern and assigns the unused numbers first (that is, **003**, **014**, and **098**) before assigning number **101**.

Therefore, if you detach a naming rule and attach a new naming rule that uses the same pattern, be aware that the new naming rule first assigns unused numbers in the pattern.

Business object display rules

Add a business object display rule

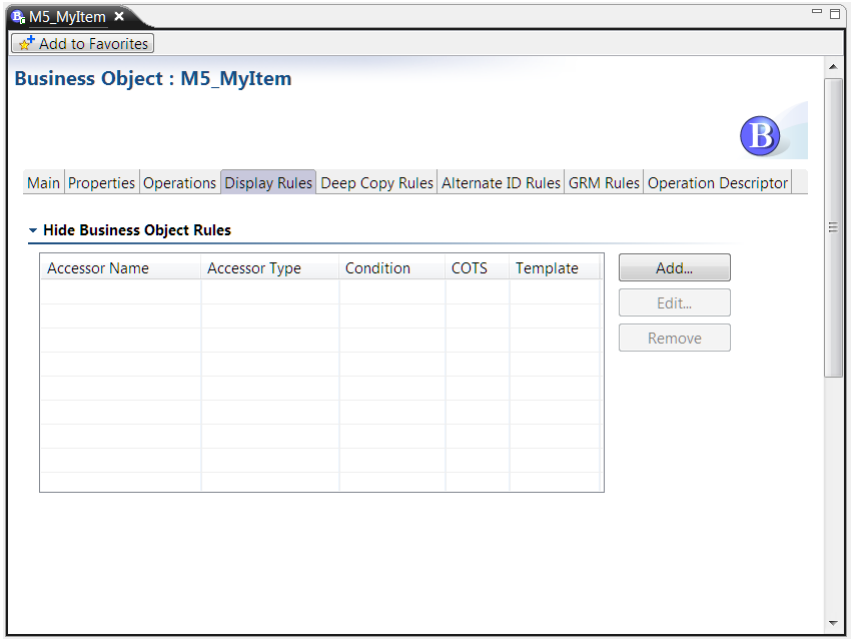
Business object display rules determine the members of the organization who cannot view a business object type in menus in the Teamcenter user interface. The **Display Rules** editor displays the groups and roles that are not allowed to see the selected type of business object in menus. This rule is primarily used to hide business objects from creation (**File→New**) menus, thereby restricting those who can create the business object type.

You can also use the Command Suppression application to suppress the display of menus and commands for certain groups.

1. Select a business object for which you want to create a new display rule. To search for a business object, you can click the **Find** button at the top of the **BMIDE** view. Display rules can be applied to the following business objects and their children: **Alias**, **Dataset**, **Folder**, **Form**, **Identifier**, and **Item**.

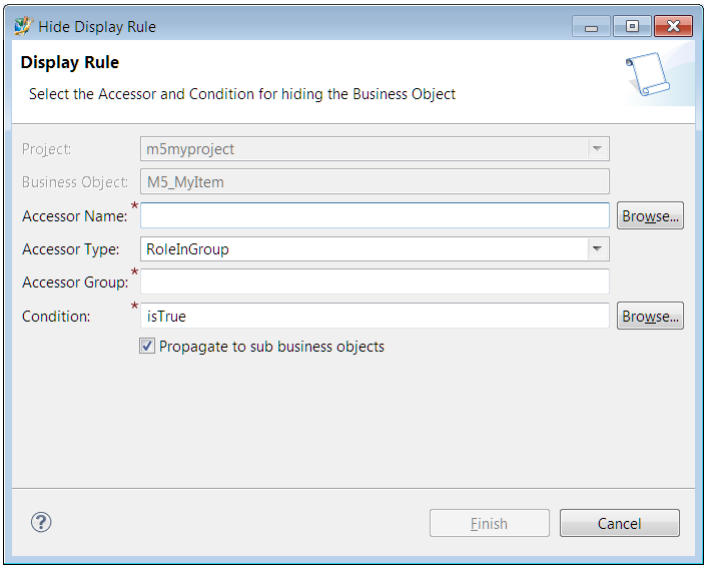
   In the **BMIDE** view, right-click the relevant business object or one of its children, choose **Open**, and click the **Display Rules** tab.

© 2025 Siemens



2. Click the **Add** button to the right of the **Hide Business Object Rules** table.

   The Hide Display Rule wizard runs.

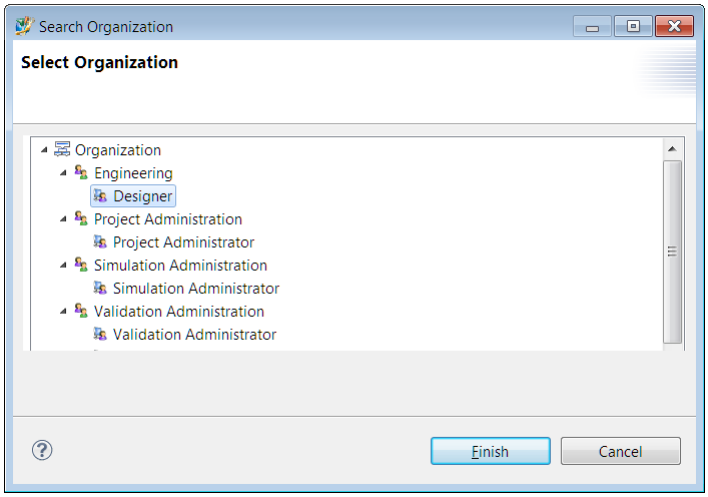


3. Perform the following steps in the **Display Rule** dialog box:

   a. In the **Accessor Name** box, type the name of group or role for which you are writing the rule. You can also click the **Browse** button to the right of the box to select the role or group.

© 2025 Siemens

The Teamcenter Repository Connection wizard prompts you to log on to a server to look up the available groups and roles in the organization. Select the role or group in the **Select Organization** dialog box.



b. Click the arrow in the **Accessor Type** box to select the kind of accessor you are creating the rule for (role, group, or organization). This box is already filled out if you made a selection in the **Select Organization** dialog box.

c. In the **Accessor Group** box, type the name of the group. This box is already filled in if you made a selection in the **Select Organization** dialog box.

d. Click the **Browse** button to the right of the **Condition** box to select the condition for which this display rule runs. If you select **isTrue** as the condition, the rule always applies.

> Note:
>
> Only those conditions appear that have valid signatures. For business object display rules, the valid condition signature is as follows:
>
> *condition-name***(UserSession)**

e. Select the **Propagate to sub business objects** check box to propagate the display rule to the existing child business objects.

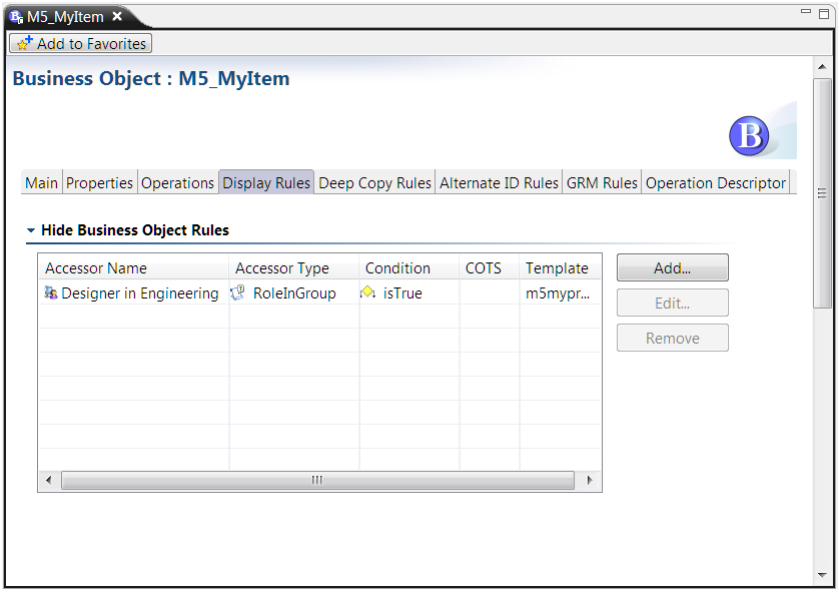
> Note:
>
> Child business objects do not *inherit* the display rule from parent business object.
>
> When you add a display rule, selecting the **Propagate to sub business objects** check box also adds the display rule to the existing child business objects. Editing a display rule and selecting the **Propagate to sub business objects** check box updates the

© 2025 Siemens

display rule on the child business objects that are similar, that is, with a matching accessor name, type, and group.

f. Click **Finish**.

The **Hide Business Object Rules** table displays the groups and roles that have the business object hidden from them in the menus in Teamcenter rich client applications.



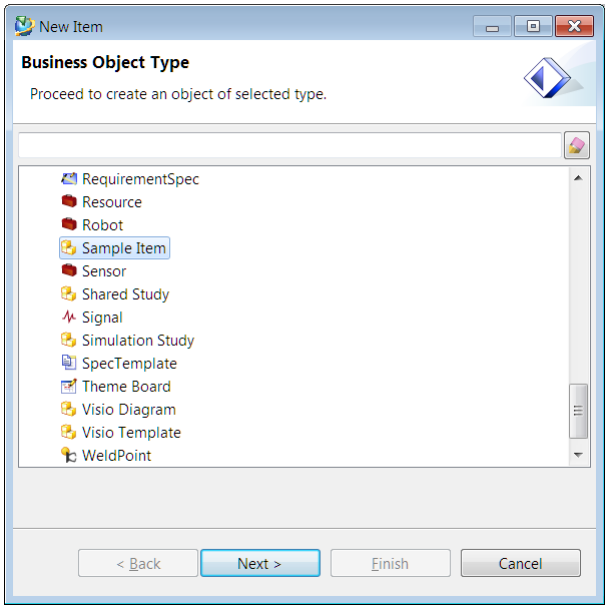
4. To save the changes to the data model, choose **BMIDE→Save Data Model**, or click the **Save Data Model** button on the main toolbar.

5. After you create a rule, you can deploy your changes to the server. Choose **BMIDE→Deploy Template** on the menu bar, or select the project and click the **Deploy Template** button on the main toolbar.

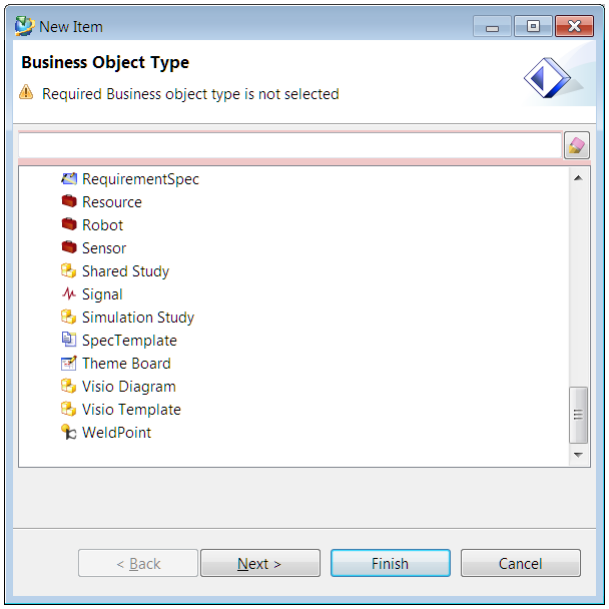
6. After deployment, test your new display rule in the Teamcenter rich client.

   For example, if you created a display rule on a custom business object, log on to the My Teamcenter application as a user in a group that has no display rule defined for that business object type (and therefore has rights to view that business object type in menus). Choose **File→New→Item**. You should be able to create new instances of the custom business object.

© 2025 Siemens



Then log on as a user in a group that has a display rule defined (and therefore does not have rights to see the custom business object in menus). You should not be able to create instances of the custom business object.



Business object display rules reference

Business object display rules suppress the display of business object types in the menus for certain groups of users. This rule is primarily used to hide business objects from creation (**File→New**) menus, thereby restricting those who can create the business object type.

Business object display rules can be applied to the following business objects and their children:

© 2025 Siemens

- **Dataset**
- **Folder**
- **Form**
- **Item**

The list of business objects on which you can create display rules is defined in the **TYPE_DISPLAY_RULES_list_of_primary_types** preference and the **TYPE_DISPLAY_RULES_list_types_of_subclasses** preference.

> Note:
>
> Although users cannot create objects associated with restricted object types, they can view them and perform other operations, such as copying, modifying, or deleting the objects.

Business object display rules are subject to the principles of group hierarchy and inheritance. Rules defined at the site level are inherited by all groups and roles within groups. Rules defined for a group are inherited by all subgroups, but rules applied to a subgroup do not affect the parent groups or other subgroups at the same level in the hierarchy (sibling groups). Rules applied to roles within groups apply to all users with that role, but do not affect other roles in the same group.

In addition to being subject to the principles of group hierarchy and inheritance, business object display rules are also subject to inheritance. Display rules set at the subobject level take precedence over rules set at the business object level.

> Note:
>
> You can also use the Command Suppression application to suppress the display of menus and commands for certain groups.

Generic Relationship Manager (GRM) rules

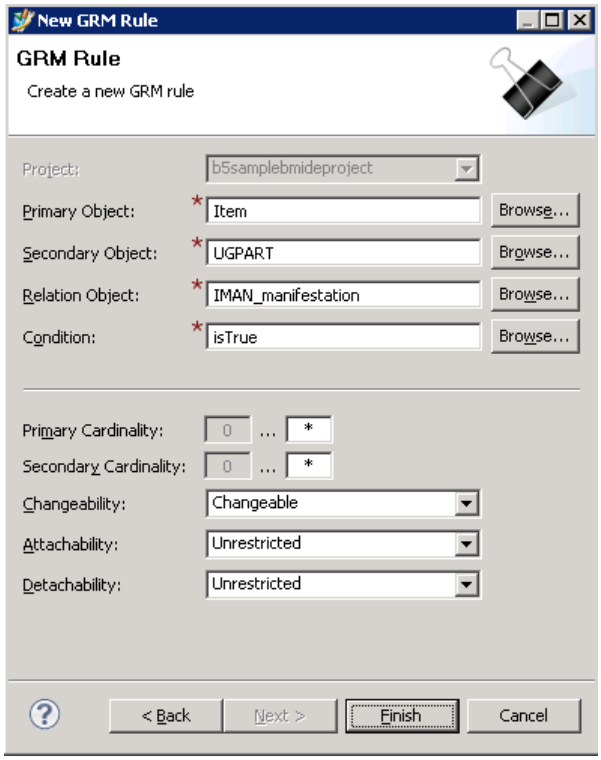
Introduction to GRM rules

Generic Relationship Management (GRM) rules apply constraints on the relationship between two business objects.

You can use Generic Relationship Management (GRM) rules to limit what objects can be pasted to other objects. For example, if you do not want a certain type of object to have a specification relation to another type, you can set the cardinality to **0** to deny pasting of one type of object to another with the specification relation.

© 2025 Siemens

You can create GRM rules with the same primary object, secondary object, and relation object, but with different conditions. This allows two GRM rules to apply different cardinality constraints based on the conditions. If you select **isTrue** as the condition, the GRM rule always applies.



When you create a GRM rule, you select the primary and secondary business objects for the relationship, the relationship they have to one another, and the constraints to be applied. Available relationships are children of the **ImanRelation** business object.

Constraint	Description
Cardinality	Determines the number of allowed occurrences of the primary object in relation to the secondary object, and of the secondary object in relation to the primary object. When you create a GRM rule, type a number in the **Primary Cardinality** or **Secondary Cardinality** boxes:

© 2025 Siemens

Constraint	Description	
	Number	**Action**
	-1 or *	Allow an unlimited number of relationships.
	0	Do not allow any relationships.
	1, or **2**, or **3**, and so on	Allow the specified number of relationships.
Changeability	Specifies whether the relationship links between objects can be added, deleted, or otherwise changed.	
Attachability	Specifies whether new relationship links can be made between objects.	
Detachability	Specifies whether the relationship links that exist between objects can be removed.	

You can see the relations between business objects in the UML editor by right-clicking and choosing **Show→Relations**.

Inheritance of GRM rules

The GRM rule applies for all children of the primary and secondary objects. However, rules defined for a sub-business object take precedence over the relation rules defined for a parent object.

> Example:
>
> If the **ItemRevision** business object is chosen as the primary object, and the **DirectModel** dataset is chosen as the secondary object, then all children of these objects that have the relationship inherit the rule.
>
> Therefore, all instances of the children of **ItemRevision** that are related by the relation specified in the rule to any instance of children of **DirectModel** are subject to the constraints defined by the rule.

Add a GRM rule

1. Choose one of these methods:

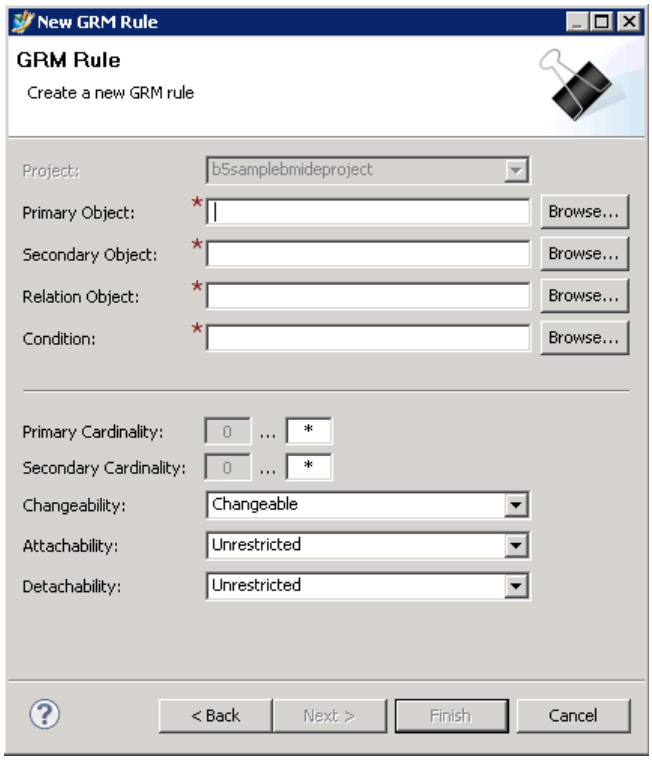
   - On the menu bar, choose **BMIDE→New Model Element**, type **GRM Rule** in the **Wizards** box, and click **Next**.

   - On the menu bar, choose **BMIDE→Editors→GRM Rules Editor**, and click the **Add** button to the right of the table.

© 2025 Siemens

- Open a business object, click the **GRM Rules** tab, and click the **Add** button to the right of the table.

The GRM Rule wizard runs.



2. Perform the following steps in the **GRM Rule** dialog box:

   a. Click the **Browse** button to the right of the **Primary Object** box to select the primary business object in the relationship.

   b. Click the **Browse** button to the right of the **Secondary Object** box to select the secondary business object in the relationship.

   c. Click the **Browse** button to the right of the **Relation Object** box to choose the relationship between the primary and secondary business objects. These relations are children of the **ImanRelation** business object.

   d. Click the **Browse** button to the right of the **Condition** box to select the condition for which this GRM rule runs. Only those conditions appear that have valid signatures. If you select **isTrue** as the condition, the GRM rule always applies. For GRM rules, the valid condition signatures are as follows:

      condition-name(**ImanGRM, POM_object, POM_object, UserSession**)
      condition-name(**ImanGRM, POM_object, POM_object**)
      condition-name(**UserSession**)

© 2025 Siemens

For the first two condition signatures, the first argument is the current GRM rule, the second argument is the primary business object used in the GRM rule, and the third argument is the secondary business object used in the GRM rule.

e. In the **Primary Cardinality** box, type the number of primary objects that can be related to a secondary object with the relationship. An asterisk (*****) means an unlimited number. (**-1** also means an unlimited number.)

f. In the **Secondary Cardinality** box, type the number of secondary objects that can be related to a primary object with the relationship. An asterisk (*****) means an unlimited number. (**-1** also means an unlimited number.)

g. In the **Changeability** box, select **Changeable** if the relationships using this rule can be deleted or otherwise changed, **Add Only** if only new relationships can be made between the objects, or **Frozen** if relationships cannot be changed in any way.

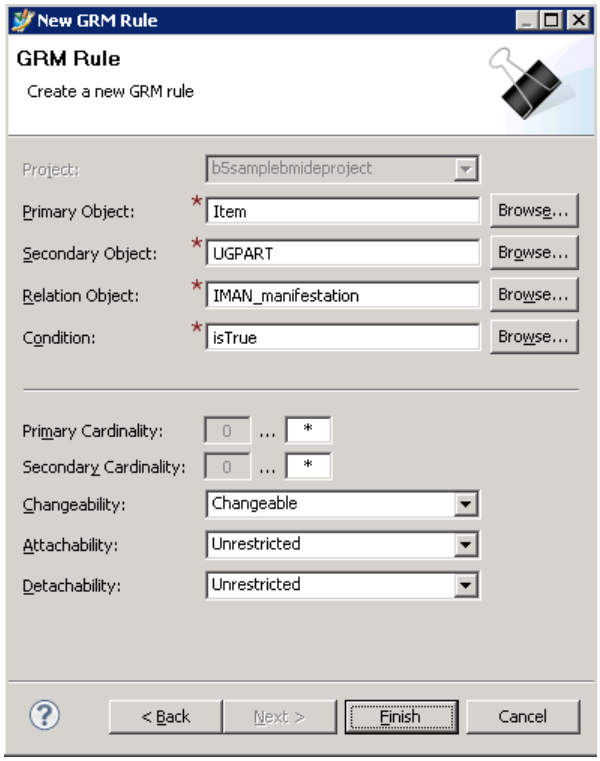
h. In the **Attachability** box, select **Unrestricted** if all users can relate new objects using the rule, or select **WriteAccessReq** if Access Manager rules should be used to evaluate if the relationship creation is allowed.

> Note:
>
> When using access control lists (ACLs), the **Attachability** setting must be set to **Unrestricted** or **Write Access required** on the parent object.

i. In the **Detachability** box, select **Unrestricted** if all users can remove the relationships between objects using the rule, or select **WriteAccessReq** if Access Manager rules should be used to evaluate if the relationship creation is allowed.

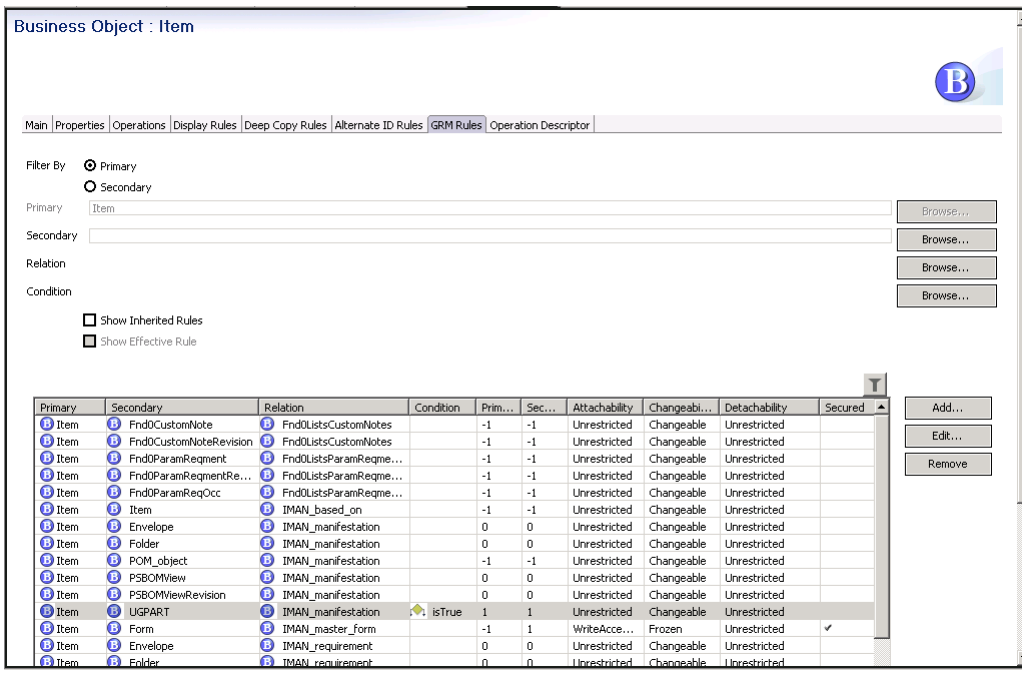
In the following example, the **Item** business object is restricted to having only one **UGPART** attached with the **IMAN_manifestation** relation. If you attempt to attach another of that type, the system displays an error.

© 2025 Siemens



j. Click **Finish**.

The rule is created and appears in the table on the **GRM Rules** editor.



© 2025 Siemens

k. To search for existing GRM rules by primary object, secondary object, or relation, type strings in the **Primary**, **Secondary**, or **Relation** boxes, or click the **Browse** buttons. The GRM rules table displays the results of the search.

   Use the **Add**, **Edit**, or **Remove** buttons to work with the GRM rules.

3. To save the changes to the data model, choose **BMIDE→Save Data Model**, or click the **Save Data Model** button on the main toolbar.

4. Deploy your changes to the server. Choose **BMIDE→Deploy Template** on the menu bar, or select the project and click the **Deploy Template** button on the main toolbar.

5. After deployment, verify the new relationship rule in the Teamcenter rich client.

   You can use the following example:

   a. Create a GRM rule with the following characteristics:

      Primary object: **Item**
      Secondary object: **UGPART**
      Relation object: **IMAN_manifestation**
      Primary cardinality: **0 ... 1**
      Secondary cardinality: **0 ... 1**
      Changeability: **Changeable**
      Attachability: **Unrestricted**
      Detachability: **Unrestricted**

   b. After you create a business object, deploy your changes to the test server. Choose **BMIDE→Deploy Template** on the menu bar, or select the project and click the **Deploy Template** button on the main toolbar.

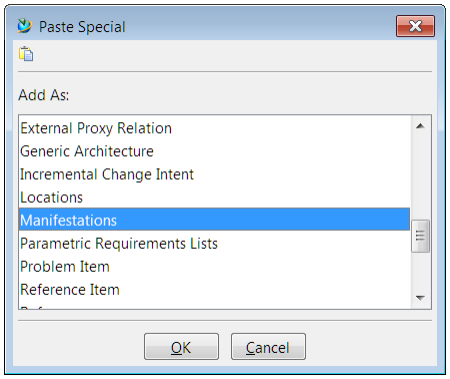
   c. To test the GRM rule, in the My Teamcenter application, perform the following steps to create an item and put a **UGPART** dataset on it with the **IMAN_manifestation** relationship:

      A. Create an **Item** business object by choosing **File→New→Item** and selecting the **Item** type.

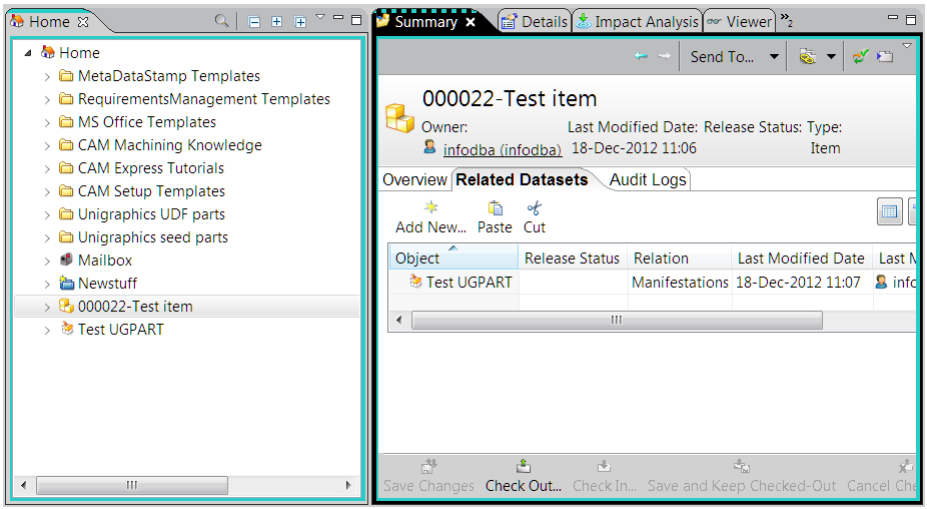
      B. Create a **UGPART** dataset by choosing **File→New→Dataset** and choosing the **UGPART** type.

      C. Copy the **UGPART** dataset by selecting it and choosing **Edit→Copy**, select the **Item** business object instance, choose **Edit→Paste Special**, and choose the **Manifestations** relationship.

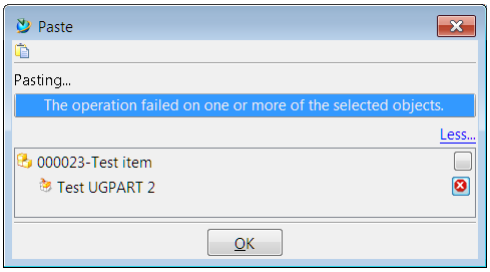
© 2025 Siemens



The **UGPART** dataset is pasted to the **Item** business object with the **Manifestations** relationship. To see the relationship, select the **Item** business object and click the **Related Datasets** tab.



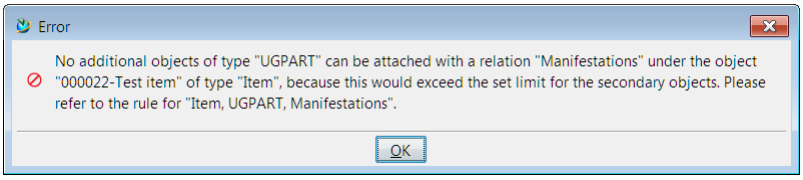
d. Try to use the **Edit→Paste Special** operation to paste another **UGPART** dataset on the same item with the **Manifestations** relationship. A paste error message states that you cannot do this because it violates GRM rule constraints that allow only one **UGPART** object to be related.



e. Click the red **X** button in the lower right corner of the paste error dialog box.

The following error dialog box shows that the GRM rule is the reason that you cannot attach another **UGPART** dataset to the **Item** business object with the **Manifestations** relationship.

© 2025 Siemens



Generic Relationship Manager

The Generic Relationship Manager (GRM) module of the Integration Toolkit provides a general way to enable association of two objects via a relationship.

> Example:
>
> A *specification* relationship is defined for use in associating an item revision with a dataset.

> Tip:
>
> You can use relationships, or attributes that are references, to navigate from object to object and in either direction.

In some cases, generic relationships are exposed as properties.

The GRM module of the Integration Toolkit supports the concept of explicit relationships.

You can use the GRM module to define and enforce specific rules pertaining to relationships, as well as separate the maintenance of relationships from the data itself.

Evaluation order for GRM rules

GRM rules are evaluated whenever a relation between a primary and secondary object is created, modified, or deleted. The order in which GRM rules are evaluated for a given primary business object (**PType**), relation business object (**RType**), and secondary business object (**SType**), is as follows:

1. Find a match for **PType-RType-SType**.
2. If not found, find a match for **PType-GRM_match_All-SType**.
3. If not found, repeat steps 1 and 2 for each super type of **PType** and **SType**.
4. If a GRM rule is found, apply the rule.

Impact of business object inheritance on GRM rules

GRM rules apply constraints on objects based on the relationship between primary and secondary objects. The primary and secondary business object trees in the GRM rule interface allow you to

© 2025 Siemens

configure relation rules between primary and secondary objects at any level of the business object hierarchy. For example, if the **ItemRevision** business object is chosen as the primary business object and the **DirectModel** dataset business object is chosen as the secondary business object, all sub-business objects of the **ItemRevision** and **DirectModel** business objects inherit the relation rule. Therefore, all instances of the sub-business objects of the **ItemRevision** primary business object that are related to all instances of the sub-business objects of the **DirectModel** secondary business object by the relation business object specified by the rule are subject to the constraints defined by the rule.

Relation rules defined for a specific sub-business object take precedence over relation rules defined for a parent business object. However, if a GRM rule on the parent business object is marked as **Secured**, you cannot override the rule by applying another GRM rule to it. To see if a GRM rule is secured, look at the **Secured** column in the table on the **GRM Rules** tab.

Inheritance also applies to relationships. If there is a GRM rule having **ItemRevision** as primary, **DirectModel** as secondary, with the **Iman_specification** relation, if you have a custom relation of **Iman_specification** that is called **my_specification**, then the rule should be inherited by **my_specification** relationship with the same primary and secondary.

> Note:
>
> GRM rules do not apply to folders. Whenever an object is pasted to a folder, there is no GRM relation created. Rather, the folder object itself stores a reference to the objects that are in it. Because anything pasted to a folder does not use GRM relations, GRM rules cannot be applied when the folder is a primary object (that is, when an object is pasted to a folder).

Maintain stable relation IDs

The **fnd0CopyStableId** property on the **ImanRelation** business object and its children can be used as a stable ID during revise and save as operations. When a relation is copied during a save as or revise operation, this ID is also copied so that the ID is always the same on each relation between the source object and the target object. When the primary, secondary, and relation objects are copied together, the identifier on the relation is maintained and the application data is not required to be updated simply because the objects have been copied.

This functionality can be used by integrations to Teamcenter such as the NX Integration to maintain a record of the original relationships between the source and target objects before entering the Teamcenter system.

There are additional properties on the relation object to determine when the primary and secondary objects are not synchronized. Following is the complete list of properties used for stable relation IDs:

- **fnd0CopyStableId**

  Holds a stable ID during revise and save as operations.

- **fnd0CopyStableDate**

© 2025 Siemens

Contains the date when stable ID originated. Its value is the last date when the secondary object was synchronized to the primary object.

- **fnd0IsPrimaryInSync**

  Tracks whether the primary object is in sync with the secondary object. This property uses the value of the **fnd0CopyStableDate** property to determine if the objects are in sync.

- **fnd0IsCsIdMigratedOnly**

  Tracks if the stable ID has been migrated. This property is set to true at upgrade when the value of the **fnd0CopyStableId** property is set to **null**.

This stable relation ID functionality was originally introduced in 4th Generation Design with the **Mdl0CopyStableRelation** business object. This relation is used to track the **Cpd0DesignElement** object that is connected with a **Cpd0DesignFeature** object. This previous method to track relations is deprecated and replaced by the newer method.

Deep copy rules

Add a deep copy rule

Deep copy rules define whether objects belonging to a business object instance are copied when a user performs a save as or revise operation on that instance. Deep copy rules can be applied to any business object type and are inherited by children business object types.

Deep copy rules inherited by child business objects are not editable on the children. When an inherited rule is selected, the **Edit** and **Remove** buttons are unavailable on the **Deep Copy Rules** tab.

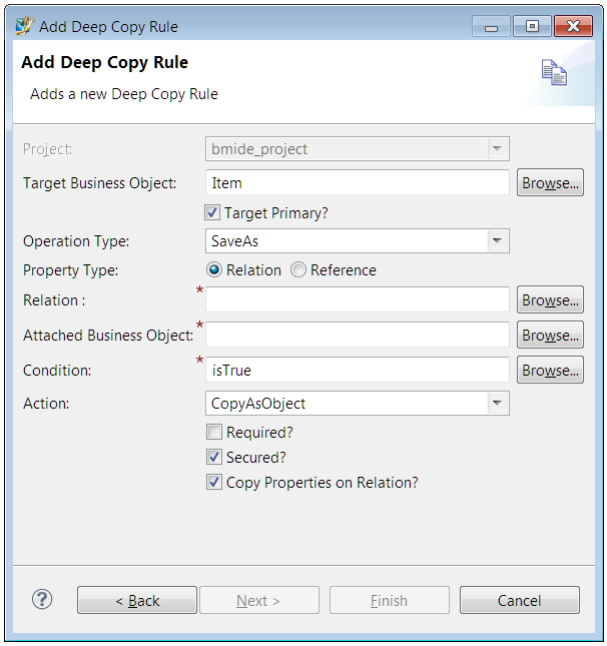
Perform the following steps to create a deep copy rule on a business object:

1. Start the **Add Deep Copy Rule** wizard in one of these ways:

   - On the menu bar, choose **BMIDE**>**New Model Element**, type **Deep Copy Rule** in the **Wizards** box, and click **Next**.

   - Open a business object, click the **Deep Copy Rules** tab, and click the **Add** button to the right of the deep copy rules table.

© 2025 Siemens



2. Specify parameters for the rule.

Parameter	Description
Target Business Object	Choose the business object that the deep copy rule is applied to. For example, **ItemRevision**.
Target Primary?	Mark or clear the check box as appropriate. ☑ **Target Business Object** is the primary object of the relationship specified in the **Relation Type** box. When the business object instance is revised or saved, the secondary objects are carried forward and related using the relation in the **Relation Type** box. ☐ **Target Business Object** is the secondary object of the relationship specified in the **Relation Type** box. When the business object instance is revised or saved, the primary objects are carried forward and related using the relation in the **Relation Type** box.
Operation Type	Select the operation that invokes the rule, one of **Save As** or **Revise**.
Property Type	If the **Operation Type** is **Save As**, then select one of the following: • **Relation** creates the deep copy relationship • **Reference** defines a deep copy rule between an object and a referenced object through the reference property.

© 2025 Siemens

Parameter	Description
Relation / Reference Property	The parameter label changes depending on the selected **Property Type**. **Relation** Select the relationship business object to use for the relationship between the copied object and its business object instance. Alternatively, press Alt-C or start typing to see a list of suggestions. Available relationships are children of the **ImanRelation** business object. Objects with a relationship that matches the selected relationship are only copied from the source business object instance to the destination business object instance. To match all relationships, select **Match All**. **Reference Property** Select the property. This is similar to defining deep copy rules for relation properties except that you are defining a deep copy rule between an object and a referenced object through the reference property. If the reference property is a typed reference, then the object on the other side is the type specified in the definition of the reference property or any of its subtypes. The **Browse** button allows selecting this type or its subtypes. Example: The **ItemRevision** business object has a **item_master_tag** reference property that is a typed reference property to the **Form** business object. When configuring a deep copy rule for an **ItemRevision** business object, if the reference property is selected, you can configure the deep copy rule on the **item_master_tag** property and the **Form** or any of its subtypes as the secondary object.
Attached Business Object	Click **Browse** to select the business object type to be copied. Alternatively, press Alt-C or start typing to see a list of suggestions. Select the **MatchAll** value if you want all objects to be copied forward no matter what type of business object they are.
Condition	Select the condition for which this deep copy rule runs. If you select **isTrue** as the condition, then the deep copy rule always applies. Only those conditions appear that have valid signatures. For deep copy rules, the valid condition signatures are as follows: *condition-name***(DeepCopyRule,ItemRevision,POM_object)** *condition-name***(DeepCopyRule,ItemRevision,POM_object, UserSession)** *condition-name***(DeepCopyRule,POM_object,POM_object)**

© 2025 Siemens

Parameter	Description
	*condition-name***(UserSession)** Any condition other than **isTrue** must use one of the following conventions: • Uses three input business object parameters in this order: **DeepCopyRule**, **ItemRevision** (or one of its children), **POM_object** (or one of its children). • Uses three input business object parameters in this order: **DeepCopyRule**, **POM_object** (or one of its children), **POM_object** (or one of its children). • Uses four input business object parameters in this order: **DeepCopyRule**, **ItemRevision** (or one of its children), **POM_object** (or one of its children), **UserSession**. • Uses the **UserSession** business object as its only input parameter.
Action	Choose the kind of copying to be allowed for the business object. The available options differ depending on the type of target business object. **CopyAsObject** Creates a new object of the same type as the related object and relates to the new revision. Objects created by this method are totally independent of the source object. Therefore, modifications to the new object are not reflected in the source object. **CopyAsReference** Creates a new relation between the new revision and the related object. Therefore, modifications performed on the copied object are propagated to the source object. **CopyAsObjectOrReferenceNewCopy** Creates a new object of the same type as the related object and relates to the new revision if the secondary object has not been copied or revised during the current operation. If a new copy of the secondary object is produced during the current operation, it relates the current primary object to the new copy of the secondary object. **CopyAsReferenceOrReferenceNewCopy** Relates the current primary object to the original secondary object if the secondary object has not been copied or revised during the current operation. If a new copy of the secondary object is produced during the current operation, it relates the current primary object to the new copy of the secondary object. **NoCopyOrReferenceNewCopy**

© 2025 Siemens

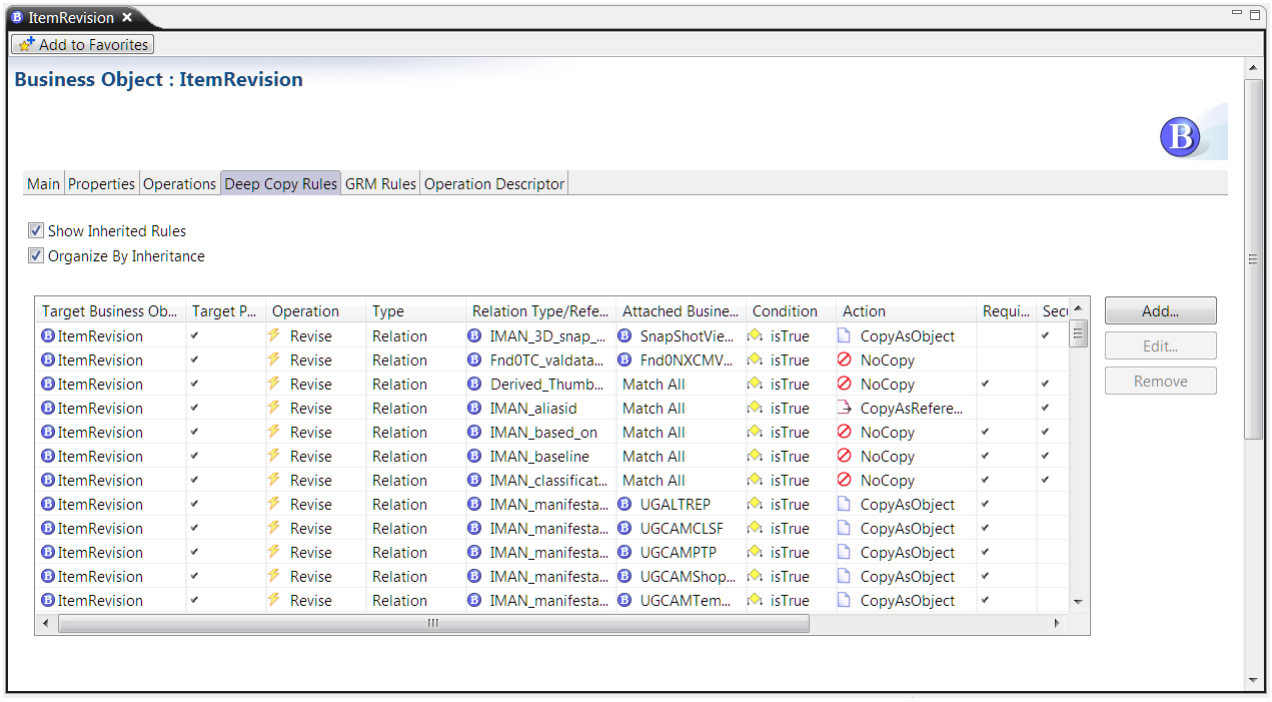
Parameter	Description
	• If the secondary object has not been copied or revised during the current operation, then a relationship is not created. • If a new copy of the secondary object is produced during the current operation, then a relationship is created to the new copy of the secondary object. **RelateToLatest** When the operation type is **Revise**, finds the latest revision of a related object and creates a relation to the new revision. Note: This action replaces the **AutoCopyRel** business object constant, which is now deprecated. **ReviseAndRelateToLatest** When the operation type is **Revise**, finds the latest revision of a related object, revises it, and creates the relation to the new revision. Note: This action replaces the **AutoRevise** business object constant, which is now deprecated. **ReviseObject** Directs the system to revise the secondary object during processing. You can use the **ReviseObject** action only with the **Revise** operation type. This action is shown only for the **Revise** operation for revisable types. Revisable business objects types are **ItemRevision**, **Identifier**, and **Mdl0ConditionalElement**, as well as their children. **Select** Indicates that the object to be copied is to be selected by the client. This is similar to the **CopyAsReference** action, but instead of the reference to original being implied, the end user can select a reference to any object in the user interface. Note: To use the **Select** option, you must create a custom user interface or a server-side customization. The COTS user interface does not automatically prompt for selection of an object. **SystemCopy** Copies the objects at the system level. The deep copy rule whose copy action is **SystemCopy** is configured to prevent you from adding a custom deep copy

© 2025 Siemens

Parameter	Description
	rule or changing the corresponding deep copy rule. The corresponding copy is handled by the system. If you want to add a deep copy rule with the **SystemCopy** action, then you must add the system handling code too. **NoCopy** Does not copy forward objects of the specified property type and relation.
Required	Select if you want to prevent users of the Teamcenter rich client from overriding the rule at run time.
Secured	Select if you want to prevent the deep copy rule from being modified or overridden by another template.
Copy Properties on Relation	Select if you want persistent properties on relation objects carried forward when the primary objects participating in relations are revised or saved as new objects. If not selected, only mandatory properties are carried forward.

3. Click **Finish**.

The rule is created and appears in the table in the **Deep Copy Rules** editor.



4. You can perform these additional actions in the **Deep Copy Rules** editor:

- Select the **Show Inherited Rules** check box to display all rules inherited from parent business objects.

© 2025 Siemens

- Select the **Organize by Inheritance** check box to sort the rules by parent business object names.
- Use the **Add**, **Edit**, or **Remove** buttons to work with the deep copy rules.

> Note:
>
> Deep copy rules with high specificity have higher precedence than those with more general applicability. Four levels of specificity, and thus precedence, are possible:
>
> Rule Level 1 {Target Business Object, Relation Type, Attached Business Object} - 1st highest precedence
> Rule Level 2 {Target Business Object, Relation Type, Match All} - 2nd highest precedence
> Rule Level 3 {Target Business Object, Match All, Attached Business Object} - 3rd highest precedence
> Rule Level 4 {Target Business Object, Match All, Match All} - 4th highest precedence
>
> At each level, a rule with a custom condition=true has higher precedence over a rule with isTrue condition.

5. To save the changes to the data model, choose **BMIDE→Save Data Model**, or click the **Save Data Model** button on the main toolbar.

6. Deploy your changes to the server. Choose **BMIDE→Deploy Template** on the menu bar, or select the project and click the **Deploy Template** button on the main toolbar.

7. Restart the Teamcenter server.

8. To verify the deep copy rule, open the My Teamcenter application in the Teamcenter rich client, select a business object instance of the type for which you created the rule, and choose **File→Save As** or **File→Revise**. Verify that the behavior works as expected.

Understanding the impact of inheritance on deep copy rule behavior

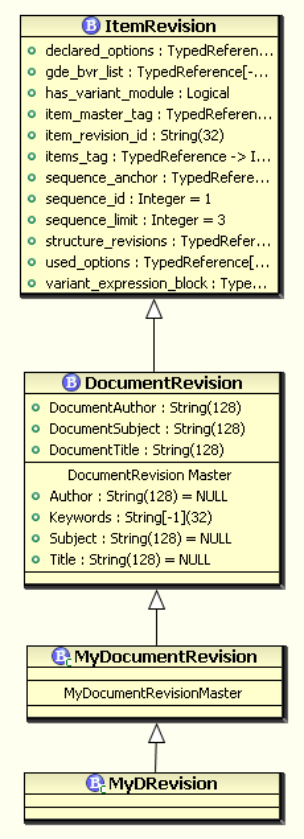
Deep copy rules defined for a parent business object are automatically inherited by all sub-business objects of the parent business object. Conversely, deep copy rules removed from a parent business object are automatically removed from all sub-business objects. In addition, inheritance is also based on relation type and object type. If there are multiple rules for a particular combination of target, relation, and object, the first rule whose condition evaluates to true is applied.

> Note:
>
> Although deep copy rules defined at a parent business object are inherited by the children, they are not editable at the child level. The **Edit** and **Remove** buttons are disabled on the **Deep Copy Rules** tab for the child business objects.

Deep copy rules are evaluated as follows: If a deep copy rule exists for the business object, it is applied. Otherwise, the hierarchy is ascended until a rule is located or the top-level parent is reached.

© 2025 Siemens

The following example assumes the existence of a hierarchy in which the **MyDRevision** business object is a sub-business object of the **MyDocRevision** business object, which in turn is a sub-business object of the **DocumentRevision** business object which is a child business object of the **ItemRevision** class.



Deep copy rules are applied to the **Revise** action of the object classes/business objects/sub-business objects, as shown in the following table.

Class/business object/sub-business object	Copy option	Deep copy rule applied
DocumentRevision	**CopyAsObject**	**RuleA**
	CopyAsReference	None
	NoCopy	**RuleB**
MyDocRevision	**CopyAsObject**	None
	CopyAsReference	**RuleC**

© 2025 Siemens

Class/business object/sub-business object	Copy option	Deep copy rule applied
MyDRevision	**NoCopy**	None
	CopyAsObject	None
	CopyAsReference	None
	NoCopy	**RuleD**

With these rules established, a user performs the following actions and the rules are applied accordingly:

- Revises a document item of the **DocumentRevision** business object.

  The related objects are copied forward as specified in the definition of **RuleA**, but the related objects as specified in the definition of **RuleB** are not copied, as **RuleB** uses the **Don't Copy** option. Both rules are defined for the **DocumentRevision** parent business object.

  > Note:
  >
  > There are no inherited rules in this example.

- Revises a document item of the **MyDocRevision** business object.

  The related objects are copied forward as specified in the definition of **RuleA** and **RuleC**, but the related objects as specified in the definition of **RuleB** are not copied. **RuleA** and **RuleB** are inherited from the **DocumentRevision** parent business object. **RuleC** is defined against the **MyDocRevision** sub-business object.

- Revises a document item of the **MyDRevision** business object.

  The related objects are copied forward as specified in the definitions of **RuleA** and **RuleC**, but the related objects as specified in the definition of **RuleD** are not copied. **RuleA** and **RuleC** are inherited from the **DocumentRevision** and **MyDocRevision** business objects, respectively. Although a **NoCopy** rule is defined for the **DocumentRevision** parent business object, it is not applied to this example, because a **NoCopy** rule is defined explicitly for the **MyDRevision** sub-business object.

Restrictions on the use of deep copy rules

The following general restrictions apply to the use of deep copy rules.

- By default, objects attached to the source business object by the **IMAN_reference** relation use the **CopyAsReference** action. They are also allowed to use the **NoCopy** and **CopyAsObject** actions.

© 2025 Siemens

- Once a rule is defined for a business object/operation combination, the combination cannot be duplicated with other copy options.

  > Example:
  >
  > If a rule is defined to copy forward **UGMASTER** datasets attached to a specific source business object by the **IMAN_specification** when the **Save As** operation is performed on the source business object, then you cannot define a rule on the same object business object/operation combination to use a different copy option.

- When deep copy rules are applied for a **Revise** operation and 1) the rule specifies the **Copy As Object** action and 2) the object to be copied is an **ItemRevision** object, then the action performed may differ depending on the location within the client where the operation initiates:

Client	location	action performed
rich client	4GD application	**Copy As Object**
rich client	other than 4GD application	**Copy As Reference**
Active Workspace	any	**Copy As Object**

- When you set deep copy rules for **Revise** and **Save As** operations on classification objects (**IMAN_classification**), observe the following:

  - Only the **CopyAsObject** or **NoCopy** actions are permissible. A **CopyAsReference** action would corrupt your data.

  - If the presentation hierarchy is installed and you do not want classification objects included in the copy action, then you must disable both the **icm0** and **cls0Object** objects.

- When you create copy rules for **Save As** or **Revise** operations on any workspace object child types, select **Required** so that the action cannot be changed in the user interface.

Deep copy API

A call to any one of the following C API methods automatically invokes the **ITEM_perform_deepcopy** method:

ITEM_copy_rev
ITEM_copy_item
ITEM_copy_item_with_masters
ITEM_copy_rev_with_master

This ensures that you always perform a deep copy after an **ItemRevision** business object is revised or saved.

© 2025 Siemens

The **ITEM_perform_deepcopy** method first checks if the object is already revised or saved. If yes, the deep copied objects are returned without performing deep copy again. If no, deep copy rules are applied. There is no requirement for customer code to invoke the **ITEM_perform_deepcopy** method after a call to any one of these methods.

When a **Save As** operation is performed on an **Item** object, the old relations of the associated **ItemRevision** object are carried forward based on the deep copy rules in the database. Therefore, the deep copy rules in the database are the deciding factor for copying relations even during **Save As** operations on an **Item** object.

There is one action that is hard coded. If there exists a generic deep copy rule to perform the **Copy As Object** action for all its related objects, and if the related object happens to be an **ItemRevision** object, then the system always performs the **Copy As Reference** action.

Deep copy rule conditions

© 2025 Siemens

checkOtherSideOneToMany

TEMPLATE

Foundation

SIGNATURE

(DeepCopyRule dr, ItemRevision target, POM_object otherSide)

- **dr**

  The deep copy rule business object.

- **target**

  The item revision on which the **Revise** or **Save As** operation is performed.

- **otherSide**

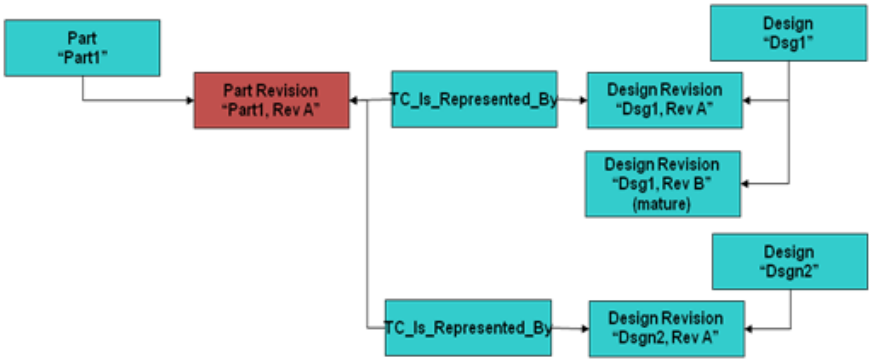
  Any **POM_object** object related to the target.

DESCRIPTION

This condition is to be used only in deep copy rule definitions.

The condition checks if the target and **otherSide** item revision object has a one-to-many relation.

For example, in the following figure, the target item revision **Part1, Rev A** is related using the **TC_Is_Represented_By** relation to the **otherSide** item revision **Dsg1, Rev A** and **Dsg2, Rev B**.



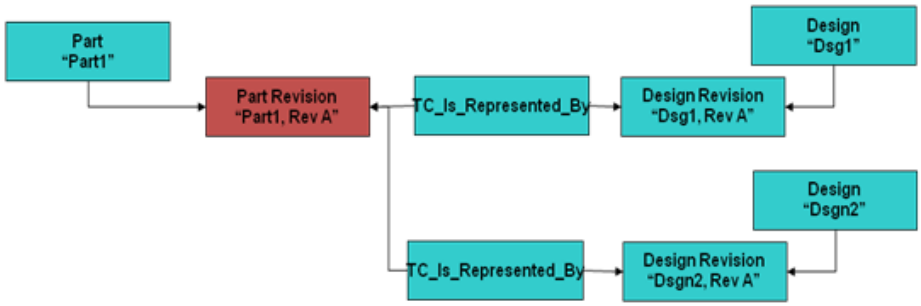
The **checkOtherSideOneToMany** condition checks if the target item revision **Part1, Rev A** is related multiple times to the same type of **otherSide** item revision. If yes, the condition returns **true**, otherwise **false**. In the example, **Part1, Rev A** is related to two instances, **Dsg1, Rev A** and **Dsgn2, Rev A** of the same **otherSide** item revision type. Therefore, the condition evaluates to **true**.

© 2025 Siemens

Note:

The target and **otherSide** is considered to have a one-to-many relation if the target item revision is related to multiple instances of the specified **otherSide** type through the specified relation type.

In the following figure, **Part1, Rev A** is related to multiple instances of **otherSide** type **Design Revision** using the **TC_Is_Represented_By** relation type. The multiple instances to which the **Part1, Rev A** is related are **Dsgn1, Rev A** and **Dsgn2, RevA**.



SECURED

True. This condition is secured and cannot be edited by another template.

© 2025 Siemens

checkOtherSideOneToOne

TEMPLATE

Foundation

SIGNATURE

(DeepCopyRule dr, ItemRevision target, POM_object otherSide)

- **dr**

  The deep copy rule business object.

- **target**

  The item revision on which the **Revise** or **Save As** operation is performed.

- **otherSide**

  Any **POM_object** object related to the target.

DESCRIPTION

This condition is to be used only in deep copy rule definitions.

The condition checks if the target and **otherSide** item revision object has a one-to-one relation.

For example, in the following figure, the target item revision **Part1, Rev A** is related using the **TC_Is_Represented_By** relation to the **otherSide** item revision **Dsg1, Rev A**.



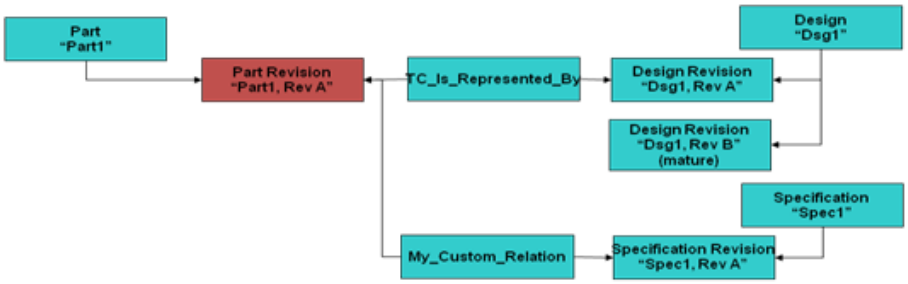
Therefore, the **checkOtherSideOneToOne** condition checks if the **otherSide** item revision **Dsg1, Rev A** has a one-to-one relation with the target **Part1, Rev A**. If the condition evaluates to **true**, then the condition returns true, otherwise **false**. In the example, there is one-to-one relation between **Part1, Rev A** and **Dsg1, Rev A**. Therefore, the condition evaluates to **true**.

© 2025 Siemens

Note:

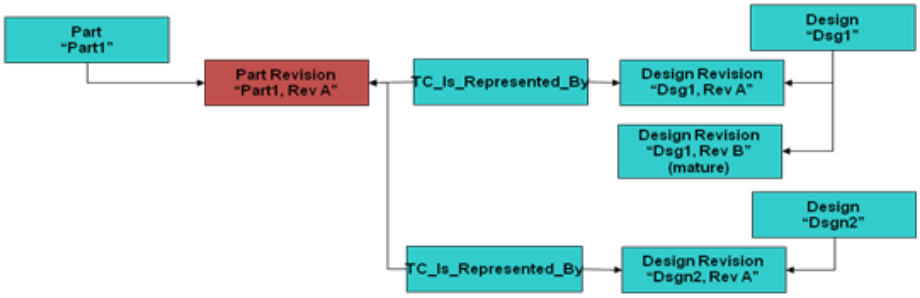
The target and **otherSide** is considered to have a one-to-one relation if the target item revision is related only to revisions having the same name and specified type as the **otherSide** object through the specified relation type.

In the following figure, **Part1, Rev A** and **Dsgn1, Rev A** has a one-to-one relation.



An example of invalid one-to-one relation follows:

In the following figure, **Part1, Rev A** is related to two item revisions, **Dsgn1, Rev A** and **Dsgn2, Rev B** of the same type, using the same **TC_Is_Represented_By** relation. This is not a valid one-to-one relation.



SECURED

True. This condition is secured and cannot be edited by another template.

© 2025 Siemens

isOneToOneAndMature

TEMPLATE

Foundation

SIGNATURE

(DeepCopyRule dr, ItemRevision target, ItemRevision otherSide)

- **dr**

  The deep copy rule business object.

- **target**

  The item revision on which the **Revise** or **Save As** operation is performed.

- **otherSide**

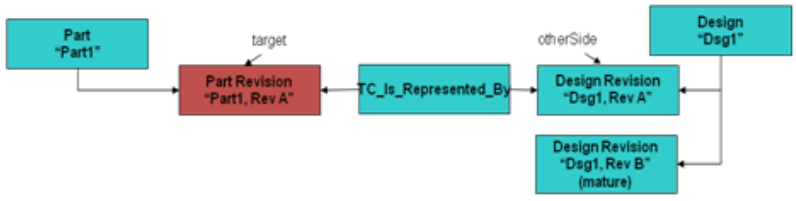
  Any **ItemRevision** object related to the target.

DESCRIPTION

This condition is to be used only in deep copy rule definitions.

The condition checks if the target and **otherSide** item revision object has a one-to-one relation and if the **otherSide** latest item revision is mature.

For example, in the following figure, the target item revision **Part1, Rev A** is related using the **TC_Is_Represented_By** relation to the **otherSide** item revision **Dsg1, Rev A**.



Therefore, the **isOneToOneAndMature** condition checks if the **otherSide** item revision **Dsg1, Rev A** has a one-to-one relation with the target item revision **Part1, Rev A** and if the **otherSide** latest revision is mature. If both the conditions evaluate to **true**, then the condition returns **true**, otherwise false. In the example, there is one-to-one relation between **Part1, Rev A** and **Dsg1, Rev A** and also the latest revision **Dsg1, Rev B** is mature. Therefore, the condition evaluates to **true**.

© 2025 Siemens

SECURED

True. This condition is secured and cannot be edited by another template.

© 2025 Siemens

isOtherSideLatestMature

TEMPLATE

Foundation

SIGNATURE

(DeepCopyRule dr, ItemRevision target, ItemRevision otherSide)

- **dr**

  The deep copy rule business object.

- **target**

  The item revision on which the **Revise** or **Save As** operation is performed.

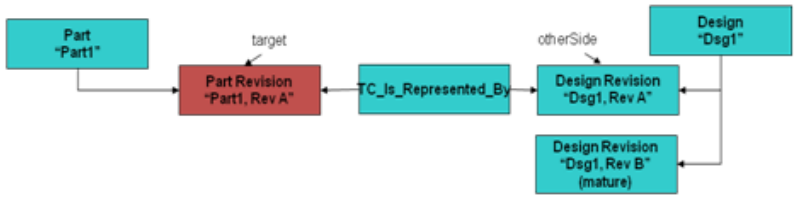
- **otherSide**

  Any item revision object related to the target.

DESCRIPTION

This condition is to be used only in deep copy rule definitions. The condition checks if the **otherSide** item revision object related to the target is mature or not.

For example, in the following figure, the target item revision (**Part1, Rev A**) is related using the **TC_Is_Represented_By** relation to the **otherSide** item revision (**Dsg1, Rev A**).



This condition checks if the other side item revision (**Dsg1, Rev A**) contains a latest revision that is mature. If yes, it returns **true**, otherwise, it returns **false**. In this example, the condition finds the latest revision **Dsg1, Rev B** associated with the **Dsg1** design to be mature. Therefore, the condition evaluates to **true**.

> Note:
>
> An item revision is considered as mature if its status matches with one of the status values in the **MaturityStatuses** attachment for the business object constant on that item revision type.

© 2025 Siemens

For example, if the constant attachment values for **MaturityStatuses** on the **Design Revision** business object is **Approved, Released** and if the status of the **Dsgn1, Rev B** in the figure was set to **Approved**, this item revision instance is considered mature.

SECURED

True. This condition is secured and cannot be edited by another template.

© 2025 Siemens

isOtherSideReplica

TEMPLATE

Foundation

SIGNATURE

(DeepCopyRule dr, POM_object target, POM_object otherSide)

- **dr**

  The deep copy rule business object.

- **target**

  The object on which the **Revise** or **Save As** operation is performed.

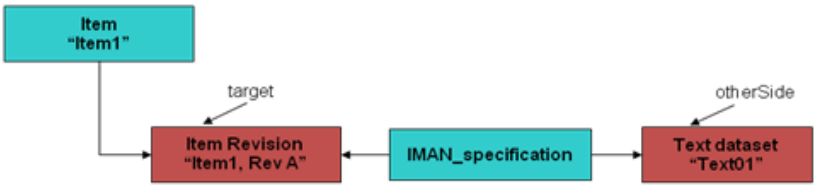
- **otherSide**

  Any **POM_object** related to the target.

DESCRIPTION

This condition is to be used only in deep copy rule definitions. The condition checks the **otherSide** object related to the target is locally created in the current site database or is replicated from another site's database.

For example, in the following figure, the target item revision (**Item1,Rev A**) is related by the **IMAN_specification** relation to the **otherSide** text dataset (**Text01**).



This condition checks if the text dataset (**Text01**) is local to the database or was replicated from another site's database.

SECURED

True. This condition is secured and cannot be edited by another template.

© 2025 Siemens

Propagation rules

Propagating security-related property values

To manage security of related objects, it is useful to automatically propagate security-related property values, such as project and license assignments, from a source business object to related destination business objects. A Teamcenter propagation rule is a definition for propagating values when one of the following actions occurs:

- A security property value is assigned to or removed from an object occurrence.
- A relation is created between two objects, if a propagation rule is configured for the relation and business object types and the source object has propagation data.
- A create, save, saveas, revise, baseline, export/import, or check-in operation is performed, and the object involved has propagation data.
- A reference property is modified, and a propagation rule is configured for that property.

Example:

Suppose that

- On a workspace object, the project property is configured for propagation. Specifically, for the project_list property, the property constant **Fnd0PropagationGroup** is set to **Security Group I**.
- A propagation rule has been defined to automatically propagate the values of properties included in **Security Group I** for a specific relation

  from an item revision
  to any dataset related to the item revision.

Given the conditions above, a user assigns a project value to an item revision. The project value is propagated to datasets referenced by the item revision because of the propagation rule.

Out of the box, several security-related properties on the **WorkspaceObject** business object are enabled for propagation. Out of the box, no other properties are propagation enabled.

Properties	Propagation Group	Propagation Style
project_list	Security Group I	merge

© 2025 Siemens

Properties	Propagation Group	Propagation Style
license_list		
owning_project	Security Group II	fill
ip_classification gov_classification	Security Group III	order

Out of the box, for reference types defined by the **TC_object_type_delegation** preference, access to the named references within a dataset is evaluated based on the above security-related properties of the dataset object. Custom reference types can be added to the **TC_object_type_delegation** preference. Propagation rules are not required to propagate security attributes from Dataset to Named References.

If you want to disable the propagation for any propagation-enabled property, you must change its propagation group to **Propagation Disabled Group**.

By default, the propagation group for properties that are not propagation-enabled is **No Group**. Do not change the group of properties that are not propagation-enabled from **No Group** to **Propagation Disabled Group**.

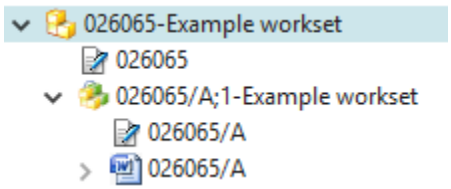
You can create propagation rules and apply them to any persistent property. Propagation rules cannot be applied to runtime properties. For the identified properties, in the business objects specified by the rules, you set the property constant **Fnd0PropagationGroup** to the appropriate propagation group. You can propagate values forward or backward through multiple levels in a product structure by defining a propagation rule at each level in the structure.

To obtain a list of all the properties that have been placed into a property group, and all the propagation rules that use the group, **run the Property Group Usage report**.

In all cases, if an existing property value on a destination object is set explicitly, then propagation rules do not affect the existing value. So, if you do not wish an object to have a propagated value for a property, then you can prevent propagation by explicitly setting the property value on the destination object.

Example of property value propagation

In a simple example of propagation, consider the following island of business object instances:



The item **Example workset** has an item revision, and a Word dataset is attached to the item revision with an IMAN_specification relationship.

© 2025 Siemens

If a user assigns a project value to the **Example workset** item, then the project value automatically propagates to the item revision and the related Word dataset because of two propagation rules:

Direction	Source object	Relation Type / Reference Property	Destination object
Reverse	Item	items_tag	ItemRevision
Forward	WorkspaceObject	IMAN_specification	WorkspaceObject

Propagation groups and styles

To facilitate application of propagation rules to object properties, rules and properties are added to a security group. Out of the box, Teamcenter has a defined security group for each of four propagation styles. In any propagation rules you create, be certain to consistently set matching group and propagation style. While the out of the box groups cover the four available styles, you can define additional groups for your convenience.

Group	Style	Propagation action	Intended for assignment to these properties
Security Group I	**Merge**	Merge with the existing property value. The merge style is applicable to multi-value type properties only.	Properties whose values are merged into a master list, such as **project_list** and **license_list**.
Security Group II	**Fill**	Leave an existing propagated value unchanged.	Properties whose values are filled in, such as **owning_project**. Note: For **owning_project**, if the preferences **TC_allow_remove_owning_project** and **TC_allow_ovewriting_owning_program** are set to True, then the propagation will behave the same as Overwrite style.
Security Group III	**Order**	Leave an existing value unchanged if the value is explicitly set on the business object instance, else compare values	Properties whose values are placed in order of precedence, such as **gov_classification** and **ip_classification**. The order propagation style is defined mainly for handling the properties ip-classification and gov-classification. If you want to use this for a property other than ip-classification and gov-classification, you can do so, but you must create a string classic LOV and attach that LOV to the property so that values can be compared per their definition. If the

© 2025 Siemens

Group	Style	Propagation action	Intended for assignment to these properties
		and fill with the highest value.	values are not there, then Teamcenter keeps the current property value.
Security Group IV	**Overwrite**	Leave an existing value unchanged if the value is explicitly set on the business object instance, else get the value from the nearest parent or nearest related business object instance.	Properties added in a propagation rule (or workspace object) by a customer, where the existing values are overwritten.

General notes for propagation of values

- The style for a security group is defined in the Teamcenter preference **TC_propagation_security_group_style**.
- To apply a propagation rule group to a property, you set its property constant **Fnd0PropagationGroup**.
- For all styles, hierarchy of structure is kept. That is, the propagation of values from the nearest parent is honored when the style is **Overwrite**.
- A propagated value cannot be removed or blanked out from the object to which it is propagated. It can be removed only from the object where it is explicitly set.
- For array-of-values (multi-value) type properties, use **Security Group I** or a custom security group which is bound to the **Merge** propagation style.
- Propagation behavior for single and multi-value properties:

Group (Style)	Single Value Property	Multivalue Property
Security Group I (Merge)	Overwrite (Propagates from the nearest parent)	Merge
Security Group II (Fill)	Fill	Fill
Security Group III (Order)	• Order for *ip_classification*, *gov_classification* and custom classification properties attached with custom LOV for security levels.	Overwrite

© 2025 Siemens

Group (Style)	Single Value Property	Multivalue Property
	• Fill for other properties	
Security Group IV (Overwrite)	Overwrite (Propagates from the nearest parent)	Overwrite (Propagates from the nearest parent)

Examples of propagation for various propagation styles

In the following examples, propagation rules are defined to propagate the respective values from item to item revision and from item revision to related object.

Example:

Group: **Security Group I**
Style: **Merge**
Property: **project_list** (multi-value type)

1. Assign project1 to Item. The project1 value is propagated to the Item revision and Dataset.

Business object	Property Value (project_list)
Item	project1
Item Revision	project1
Dataset	project1

2. Assign project2 to Item Revision. The project2 value is propagated to Dataset.

Business object	Property Value (project_list)
Item	project1
Item Revision	project1, project2
Dataset	project1, project2

3. Assign project3 to Item. The project3 value is propagated to IR and Dataset.

Business object	Property Value (project_list)
Item	project1, project3
Item Revision	project1, project2, project3
Dataset	project1, project2, project3

© 2025 Siemens

Example:

Group: **Security Group II**
Style: **Fill**
Property: **owning_project** (single value type)

1. Assign project1 to Item. The project1 value is propagated to the Item revision and Dataset.

Business object	Property Value (owning_project)
Item	project1
Item Revision	project1
Dataset	project1

2. Assign project2 to Item Revision. The value is changed on the IR because we are explicitly setting the value.

Business object	Property Value (owning_project)
Item	project1
Item Revision	project2
Dataset	project1

3. Assign project3 to Item. The value is not propagated to Item Revision and Dataset because the propagation rule style is **Fill**.

Business object	Property Value (owning_project)
Item	project3
Item Revision	project2
Dataset	project1

Example:

Group: **Security Group III**
Style: **Order**
Property: **IP_classification**

1. Assign "secret" to Item. The value is propagated to Item Revision and Dataset.

© 2025 Siemens

Business object	Property Value (IP_classification)
Item	secret
Item Revision	secret
Dataset	secret

2. Assign "super-secret" to Item Revision. The value propagates to Dataset because the propagation rule style is **Order** and "super-secret" has higher precedence than "secret".

Business object	Property Value (IP_classification)
Item	secret
Item Revision	super-secret
Dataset	super-secret

For more information about order style propagation, see Defining IP clearance and classification levels.

Example:

Group: **Security Group IV**
Style: **Overwrite** Property: **custom** (single value string type)

1. Assign val1 to Item. The value is propagated to the Item revision and Dataset.

Business object	Property Value (custom)
Item	val1
Item Revision	val1
Dataset	val1

2. Assign val2 to Item Revision. The value is changed on Item Revision because we are explicitly setting the value, and it propagates to Dataset because the style is **Overwrite**.

Business object	Property Value (custom)
Item	val1
Item Revision	val2
Dataset	val2

3. Assign val3 to Item. The value is not propagated to Item Revision and Dataset because Val2 is explicitly set on Item Revision, and Dataset gets the value val2 from its nearest parent, Item Revision.

© 2025 Siemens

Business object	Property Value (custom)
Item	val3
Item Revision	val2
Dataset	val2

Resolution of conflicting group/style pairings

Propagation rule definitions should consistently pair propagation rules and styles. In the undesirable case that propagation rules have been defined such that two rules that have the same propagation group name are applicable, but the rules do not have the same propagation style, then Teamcenter resolves the conflict in one of the following ways:

- If the group is listed in the preference **TC_propagation_security_group_style**, then the style listed in the preference applies.
- If the group is not listed in the preference **TC_propagation_security_group_style**, then
  - If the property is multi-value type, then the **Merge** style applies.
  - If the property is a single value type property, then the **Overwrite** style applies.

Example:

Custom Group V has not been listed in the preference **TC_propagation_security_group_style**. The following rules are applied to a multi-value type property.

Rule	Source	Destination	Relation	Group	Style
1	Item Revision	Dataset	IMAN_motion	Custom Group V	Merge
2	Item Revision	Dataset	IMAN_3D_snapshot	Custom Group V	Fill

The **Merge** style applies.

A group style cannot be used for both a multi-value type property and a single-value property. If custom Group V is associated with **Merge** style, then it cannot be used for **Overwrite** style.

Example:

Custom Group VI has not been listed in the preference **TC_propagation_security_group_style**. The following rules are applied to a single value type property.

© 2025 Siemens

Rule	Source	Destination	Relation	Group	Style
1	Item Revision	Dataset	IMAN_motion	Custom Group VI	Overwrite
2	Item Revision	Dataset	IMAN_3D_snapshot	Custom Group VI	Fill

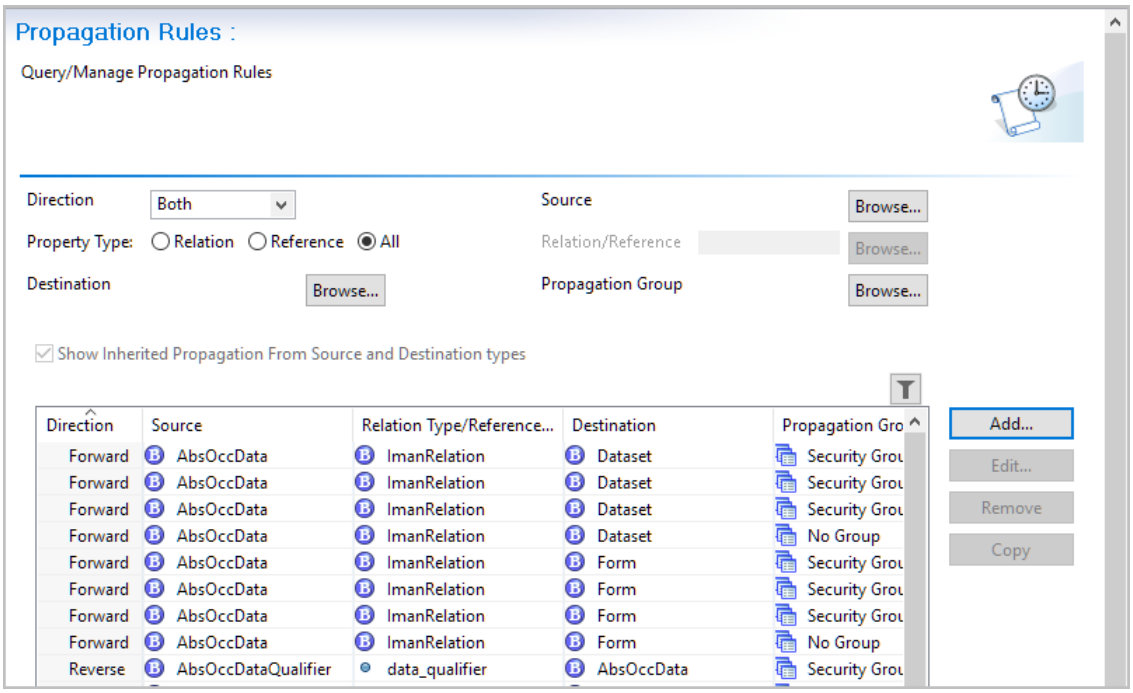
The **Overwrite** style applies.

Create a propagation rule

Define a Teamcenter propagation rule to propagate security-related property values, such as project and license assignments, from a source business object to a destination business object.

1. Choose **BMIDE** > **Editors** > **Propagation Rules Editor**.

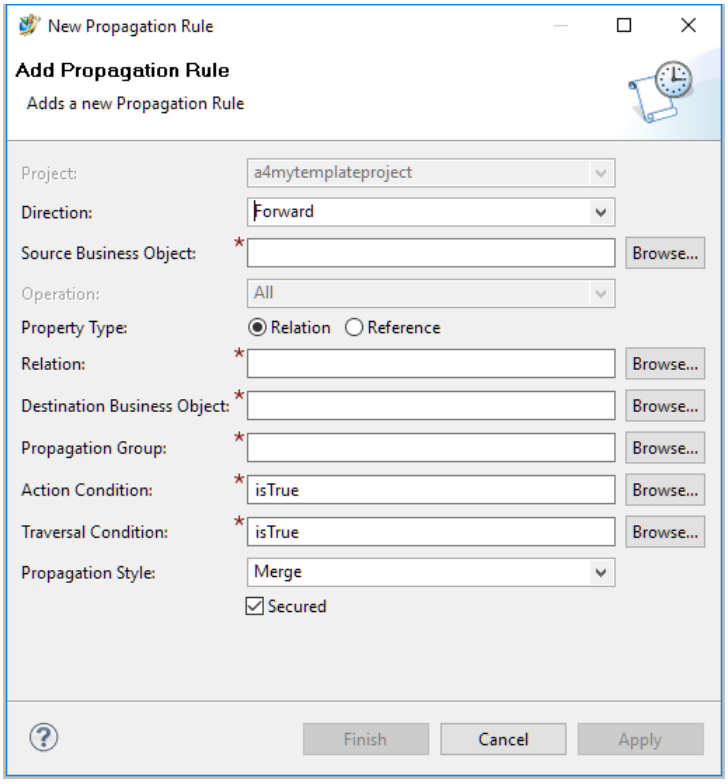
   The **Propagation Rules** editor opens.



2. Click **Add**.

   The **Add Propagation Rule** dialog box opens.

© 2025 Siemens



3. Specify rule parameters.

For this parameter	Do this	
Direction	Select the direction for traversal.	
	Forward	Traverse from the **Source Business Object** to the **Destination Business Object**.
	Reverse	Traverse from the **Destination Business Object** to the **Source Business Object**.
Source Business Object	Specify the source business object type.	
Property Type	Choose the type of connection between the source and destination business objects.	
	Relation	The source object is connected to the destination object by a relationship-type business object.

© 2025 Siemens

<table>
<tr><th>For this parameter</th><th>Do this</th></tr>
<tr><td></td><td>Reference
<table>
<tr><th>If the direction is</th><th>Then</th></tr>
<tr><td>Forward</td><td>The source object has a reference property that makes a connection to the destination object.</td></tr>
<tr><td>Reverse</td><td>The destination object has a reference property that makes a connection to the source object.</td></tr>
</table></td></tr>
<tr><td>Relation / Reference</td><td>The name of this parameter depends on the connection type selected in the Property Type parameter.

Specify the element that connects the source and destination business objects.

Relation — Specify a relationship-type business object.

The object must be a sub-Business Object of the ImanRelation business object.

Reference — Specify the reference property on the source object or destination object, depending on the Direction.

When you select a reference property, the Type Selection list updates to show object types that the selected reference property can potentially connect to. Select one of the available object types.</td></tr>
<tr><td>Destination Business Object</td><td>Specify the destination business object. The method for specifying the destination business object depends on the connection type selected in the Property Type parameter.

Relation — Specify a business object that can have the specified Relation to the specified Source Business Object.

Reference — Select an object from the Type Selection list in the dialog box for selecting a Reference property.</td></tr>
<tr><td>Propagation Group</td><td>Specify the group of properties whose values you want to propagate according to the corresponding propagation style.

The following table describes intended use of out-of-the-box propagation groups.
<table>
<tr><th>Group</th><th>Description</th></tr>
<tr><td>No Group</td><td>No Group is a special-purpose propagation group to prevent potential loss of data. Do not use No Group in rules, except as described in the following case.

If you create a propagation rule, and the rule may be deployed to a database that has legacy data that has not yet been</td></tr>
</table></td></tr>
</table>

© 2025 Siemens

For this parameter	Do this

Group	Description
	migrated to the new relation security approach introduced in Teamcenter 11.2.2, then copy your new rule and in the copy, change the propagation rule to **No Group**.
Propagation Disabled Group	Do not use **Propagation Disabled Group** in a rule. This propagation group is for assignment to business object properties that were previously enabled for propagation, for which you now want to explicitly disable propagation.
Security Group I	Multi-value type properties to be propagated using *merge* propagation style. Example: The **project_list** and **license_list** properties.
Security Group II	Single value properties to be propagated using *fill* propagation style. Example: The **owning_project** property.
Security Group III	Single value properties to be propagated using *order* propagation style. Example: The **gov_classification** and **ip_classification** properties.
Security Group IV	Custom propagation-enabled properties to be propagated using *overwrite* propagation style.

Tip:

For more information about styles and groups, see **Propagation groups and styles**.

The list of property groups that is displayed is defined in the **Fnd0PropertyGroupNames** list of values (LOV). You can create your own property group to add to this list.

© 2025 Siemens

For this parameter	Do this
	To obtain a list of all the properties that have been placed into a property group, and all the propagation rules that use the group, **run the Property Group Usage report**.
Action Condition	Click **Browse** and select the condition that must be met for the propagation to occur. If the condition evaluates to true, then the propagation occurs.
Traversal Condition	Click **Browse** and select the condition that must be met for the propagation to traverse to additional objects. If the condition evaluates to true, then the propagation traverses to the next object.
Propagation Style	Select the propagation style to apply. The style should be consistently paired with the Propagation Group.

Choose this style	To do this
Merge	• For a multi-value type property, add the source property value to the destination property values. • For a single-value type property, overwrite the value with its nearest parent value. Use this style with **Security Group I**.
Fill	If the property is a propagated value on the destination business object instance, then leave the value unchanged. Use this style with **Security Group II**.
Order	Keep the property value with the most restrictive classification (for **IP_** and **ITAR_** properties only). Use this style with **Security Group III**.
Overwrite	Set the value from the nearest parent or relation. However, if the property is explicitly set on the destination business object instance, then leave the value unchanged. Use this style with **Security Group IV**.

Secured	Select ☑ to prevent the propagation rule from being modified or overridden by another template.

4. Click **Finish**.

© 2025 Siemens

The new propagation rule is displayed in the **Propagation Rules** editor.

Test the rule

Before deploying a propagation rule to a production server, deploy the rule to a test server and verify its behavior. Use the propagation_doctor utility to ensure that the rule does not create a potential risk of a looping traversal.

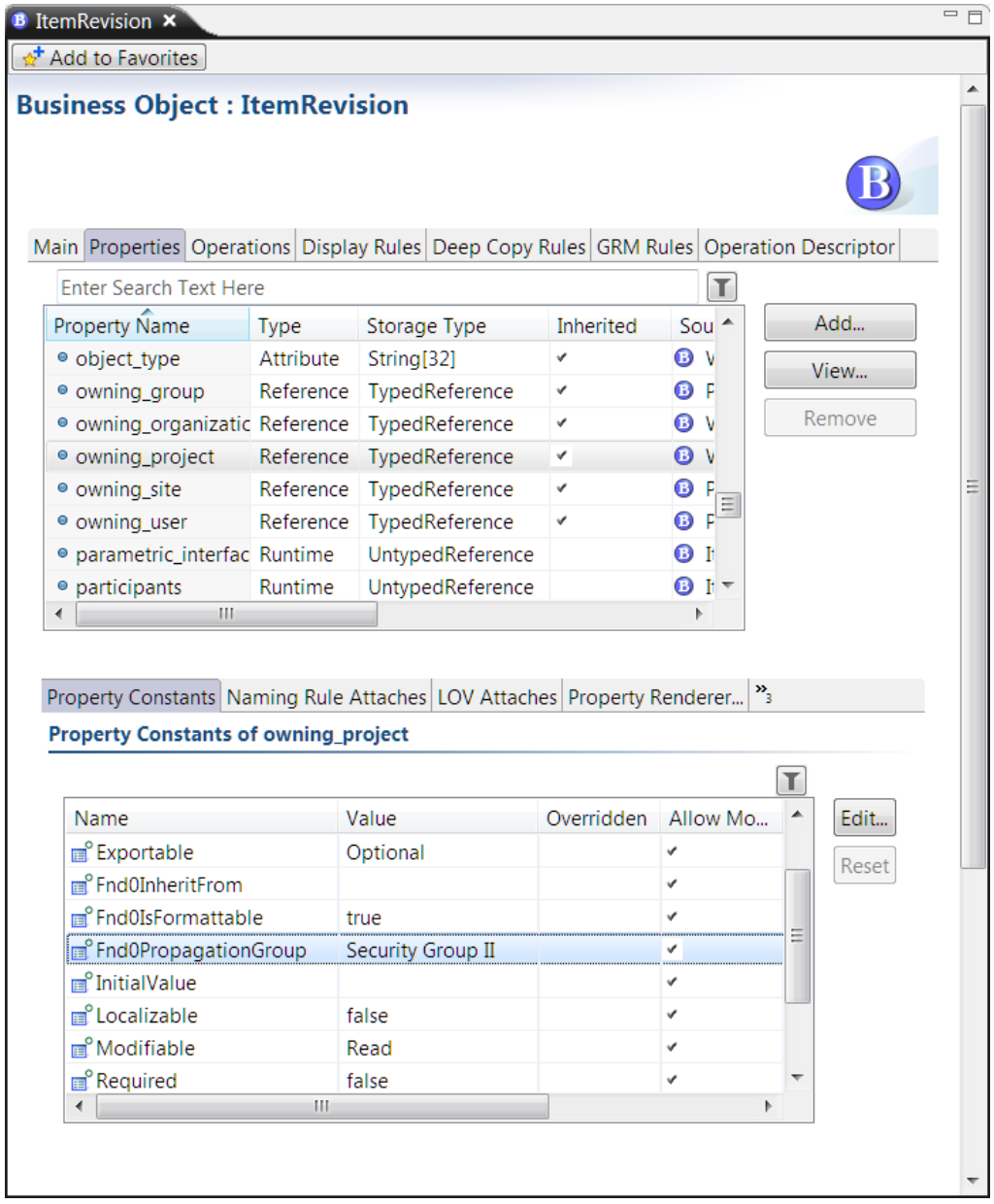
Configure a property for propagation

Use the following procedure to configure a property for propagation.

1. In the Business Modeler IDE, open the business object whose property value you want to propagate to or from.

2. On the **Properties** tab, select the property that you want to add to a propagation group.

© 2025 Siemens



3. On the **Property Constants** tab for the selected property, select the **Fnd0PropagationGroup** property constant and click **Edit**.

4. In the **Property Constant** dialog box, next to **Value**, click **Browse** and select an available propagation group.

   The following table describes the intended use of out-of-the-box propagation groups.

© 2025 Siemens

Group	Intended use
No Group	Note: Do not manually assign the **No Group** propagation group to a property. **No Group** is a special-purpose propagation group to prevent potential loss of data if a new propagation rule is deployed to a database that has legacy data that has not yet been migrated to the new relation security approach introduced in Teamcenter 11.2.2.
Propagation Disabled Group	If the property was previously enabled for propagation, and you now want to explicitly disable propagation of the property, choose the **Propagation Disabled Group** propagation group.
Security Group I	Multi-value properties, to which propagated values are added (*merge* propagation style). Example: The **project_list** and **license_list** properties. When **Security Group I** is assigned to other than a multi-value type property, then the source value overwrites the destination property (*overwrite* propagation style).
Security Group II	Properties with a value that must be filled, which a propagated value may fill (*fill* propagation style). Example: The **owning_project** property. If the destination object already has a propagated property value, then it is not overwritten.
Security Group III	Properties whose value indicates an order of precedence, which propagation sets to the most restrictive classification (*order* propagation style). Example: The **gov_classification** and **ip_classification** properties.
Security Group IV	Properties whose value is overwritten by propagation, provided it is not set explicitly and is a single-valued property (*overwrite* propagation style). This group is used for custom propagation enabled properties.

© 2025 Siemens

Additional information about property groups

The list of property groups is defined in the **Fnd0PropertyGroupNames** list of values (LOV). You can create your own property group to add to this list.

Among all propagation rules, a given propagation group should be associated with only one propagation style. See **Resolution of conflicting group/style pairings**.

> Tip:
>
> To obtain a list of all the properties that have been placed into a property group, and all the propagation rules that use the group, **run the Property Group Usage report**.

Propagate from Item to BOM View and from Item Revision to BOM View Revision

To propagate propagation-enabled properties from Item to BOM View and from Item Revision to BOM View Revision, add the following propagation rules.

Eight rules are required. For all the rules,

Direction = **Forward**.
Operation = **All**
Action Condition and **Traversal Condition** = **isTrue**.
Secured = **yes**
Background = **no**

Source Business Object	Reference	Destination Business Object	Propagation Group	Propagation Style
Item	bom_view_tags	Match All	Security Group I	Merge
Item	bom_view_tags	Match All	Security Group II	Fill
Item	bom_view_tags	Match All	Security Group III	Order
Item	bom_view_tags	Match All	No Group	Fill
ItemRevision	structure_revisions	PSBOMViewRevision	Security Group I	Merge
ItemRevision	structure_revisions	PSBOMViewRevision	Security Group II	Fill
ItemRevision	structure_revisions	PSBOMViewRevision	Security Group III	Order
ItemRevision	structure_revisions	PSBOMViewRevision	No Group	Fill

© 2025 Siemens

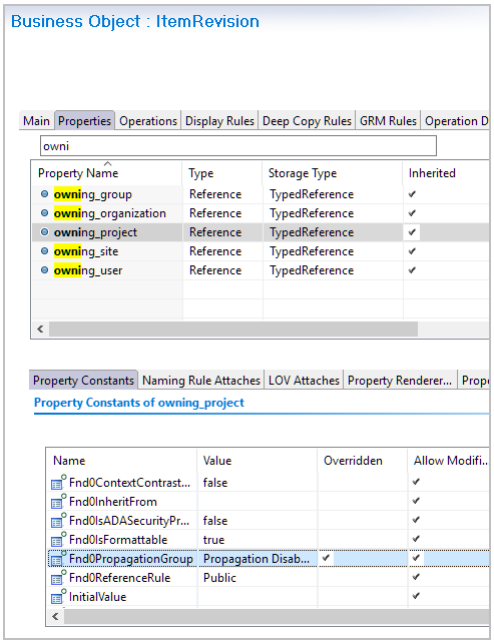
Disable propagation of a property

If you want to disable propagation of an object property that was formerly enabled for propagation, then use the following procedure.

1. Open the business object in Business Modeler IDE and on the **Properties** tab, select the property.

2. On the **Property Constants** tab, for the **Fnd0PropagationGroup** property constant, select the value **Propagation Disabled Group**.

Example:



3. Deploy changes to a test server and verify that you have achieved the desired behavior.

Identify and fix propagation rules with potential traversal risk

Propagation rules can be defined to propagate certain security attributes from a source object to other destination objects. These rules are applied recursively and repeatedly, and could therefore potentially navigate to many different sets of objects. Rule sets that traverse across consistent sets of data (islands of data) in a nested recursive manner, such as Item to Item traversal, can create problematic propagation loops. A loop is said to be formed if traversal from a business object type via propagation rules ends at the same business object type or a parent or child of that business object type.

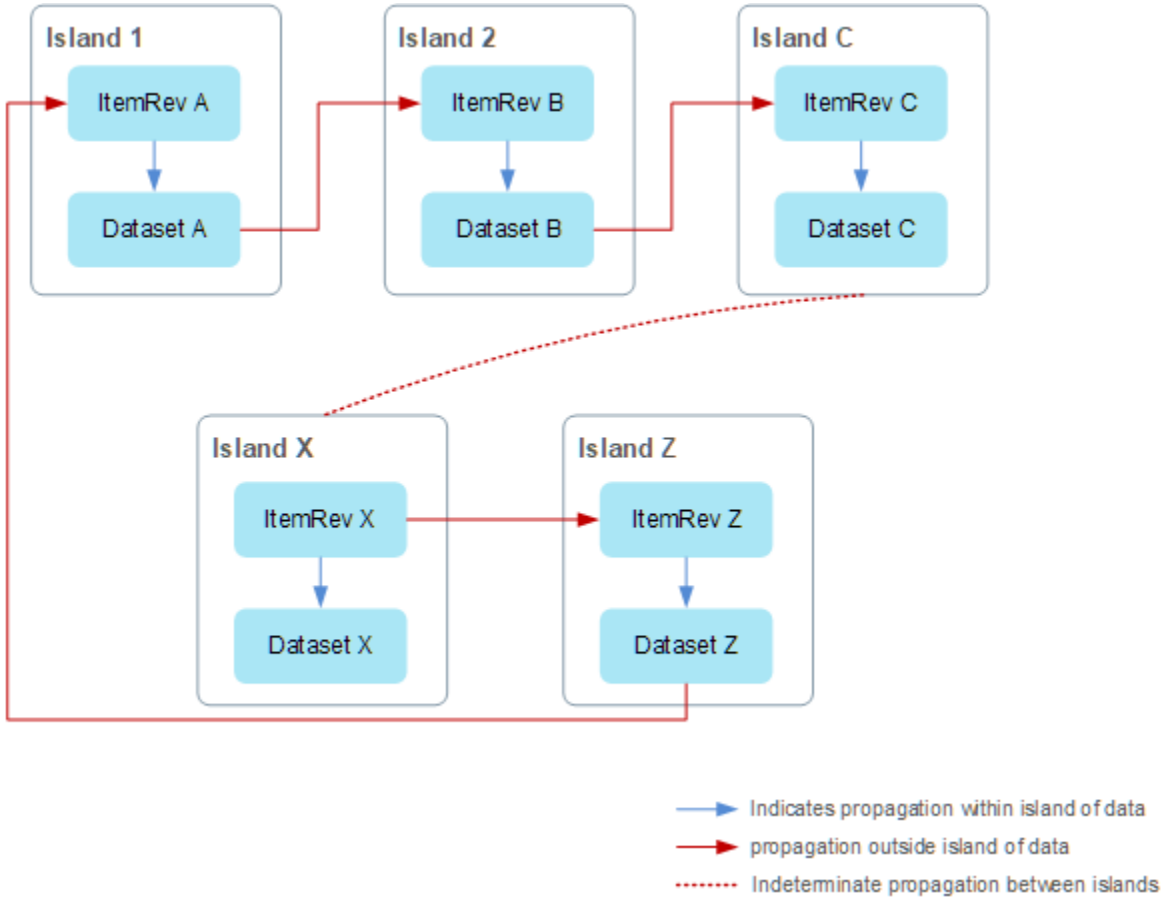
Propagation loops potentially generate a very large set of destination business objects. If this occurs, then performance of user interactions and other Create/Read/Update/Delete (CRUD) activities can be adversely affected.

© 2025 Siemens

Example:

```
Forward ItemRevision IMAN_specification Dataset
Forward Dataset IMAN_external_object_link ItemRevision
```

The above propagation rule set forms a loop because the traversal starts from ItemRevision to Dataset and then traverses from Dataset back to ItemRevision.



Example:

```
Reverse Item items_tag ItemRevision
Forward ItemRevision IMAN_specification WorkspaceObject
```

The above propagation rule set forms a loop because the traversal starts from Item to ItemRevision and then traverses from ItemRevision to WorkspaceObject, which is a parent of Item.

Example:

```
Forward WorkspaceObject IMAN_specification Dataset
Forward Dataset IMAN_external_object_link ItemRevision
```


© 2025 Siemens

The above propagation rule set forms a loop because the traversal starts from WorkspaceObject to Dataset and then traverses from Dataset to ItemRevision, which is a child of WorkspaceObject.

Test the set of propagation rules

Before deploying a new set of propagation rules to a production database, deploy these rules to a test server containing all the propagation rules present on the production database. Run the propagation_doctor utility to ensure that the rule does not create a potential risk of a looping traversal.

If a potential looping traversal risk is identified, then the utility reports the combination of propagation rules that create the loop. Use one of the following approaches to reduce or eliminate the risk.

- **Fix 1: Re-formulate the rule with more specific source/destination classes**.
- **Fix 2: Navigate to the relevant destination objects from some other source objects**.
- **Fix 3: Use custom post actions to populate the data, rather than a propagation rule**.
- **Fix 4: Re-evaluate whether this particular propagation rule is required in the first place**.

Interpreting output of the propagation_doctor utility

The propagation_doctor utility output file writes out problematic propagation loops similar to the following examples. While examining the propagation rule sets (loop), ignore the COTS (commercial off the shelf) propagation rules and consider non-COTS propagation rules.

Example 1:

```
Forward WorkspaceObject IMAN_specification WorkspaceObject
Forward Dataset IMAN_external_object_link ItemRevision
```

The two propagation rules create a loop because the traversal starts from **WorkspaceObject** (source) to another **WorkspaceObject** (destination), and then from this **WorkspaceObject** (destination) to **Dataset** (a sub-class of **WorkspaceObject**) to **ItemRevision** (a sub-class of **WorkspaceObject**).

The rule that is problematic is the second rule, `Forward Dataset IMAN_external_object_link ItemRevision`, which is a custom (non-COTS) rule.

Example 2:

```
Forward AbsOccData ImanRelation Dataset
Forward UGMASTER IMAN_specification Item
Forward WorkspaceObject IMAN_manifestation WorkspaceObject
Reverse PSBOMViewRevision qualifier_bvr AbsOccDataQualifier
Reverse AbsOccDataQualifier data_qualifier AbsOccData
```


© 2025 Siemens

This set of rules creates a loop because the traversal starts from **AbsOccData** (source) through a series all the way back to **AbsOccData** (from where it begins). The loop is created because of the custom propagation rule `Forward UGMASTER IMAN_specification Item`. If this rule is not deployed in the system, then there is no problematic loop.

Fix 1: Re-formulate the rule with more specific source/destination classes

Consider a custom propagation rule that is too generic:

```
<PropagationRule sourceType=" ItemRevision" relRefPropName="TC_Is_Represented_By"
referenceType="Relation" destinationType="ItemRevision" operation="All"
traversalCondition="isTrue" actionCondition="isTrue" propagationGroup="Security Group I"
direction="Forward" propagationStyle="Merge" background="false" secured="false"/>
```

The requirement is to propagate the project from **PartRevision** to **DesignRevision** related via **TC_Is_Represented_By** relation. The propagation rule can be re-formulated to be more specific:

```
<PropagationRule sourceType=" PartRevision" relRefPropName="TC_Is_Represented_By"
referenceType="Relation" destinationType="DesignRevision" operation="All"
traversalCondition="isTrue" actionCondition="isTrue" propagationGroup="Security Group I"
direction="Forward" propagationStyle="Merge" background="false" secured="false"/>
```

Fix 2: Navigate to the relevant destination objects from some other source objects

Instead of propagating from a primary object residing in some other island, propagate from the primary object of the same island by using propagation rules which traverse reference or relation links keeping the context inside the same data scope.

Example:

A user has the following data structure:

Item1
ItemRevision 1
Dataset1 (Dataset1 is related to IR1 with IMAN_specification relation)

Item2
ItemRevision 2
Dataset2 (Dataset2 is related to ItemRevision 2 with IMAN_specification relation)

The user wants to set the same project **P1** to **ItemRevision2** and **Dataset2** when project **P1** is assigned to **Item1**.

Assume that the user wants to use this new custom propagation rule `Forward Dataset IMAN_external_object_link ItemRevision Security Group I` in combination with the COTS propagation rule `WorkspaceObject IMAN_specification WorkspaceObject Security Group I` to propagate from Dataset1 to ItemRevision2 and from ItemRevision2 to Dataset2, but this rule set has a potential traversal risk, so the user should not use this custom rule.

Instead, either

© 2025 Siemens

- Assign project **P1** to **Item2**and it propagates to **ItemRevision2** and **Dataset2**
- Assign project **P1** directly to **ItemRevision 2** and it propagates to **Dataset2**

Fix 3: Use custom post actions to populate the data, rather than a propagation rule

Write a custom post action to assign the security attribute to the business objects of a different island.

Example:

A user has the following data structure:

Item1
ItemRevision 1
Dataset1 (Dataset1 is related to ItemRevision 1 with IMAN_specification relation)

Item2
ItemRevision 2
Dataset2 (Dataset2 is related to ItemRevision 2 with IMAN_specification relation)

The user wants to assign the same project **P1** to **Item2** when project **P1** is assigned to **Item1**.

Assume that the propagation rule `Forward Item IMAN_Rendering Item Security Group I` has a potential traversal risk, so the user should not use this rule.

Instead, write a custom post action to assign the same project to **Item2** that is assigned to **Item1**

Fix 4: Re-evaluate whether this particular propagation rule is required in the first place

If the value propagation is not essential, then simply remove the rule.

Propagation rules preferences

Use the following preferences to aid propagation rules:

TC_copy_security_data_exclusion_list

Specifies the list of security property values that are not copied over to the new object during the **Save As**, **Revise**, and **Baseline** operations. The default values are **project_list** and **owning_project**.

By excluding these property values from simply being copied, the property values can instead be implicitly set by propagation rules.

TC_propagation_exclusion_list

Specifies object types that are excluded from propagation. Data from these objects is not propagated to or from where these objects are referenced.

TC_propagation_processing_in_same_process

© 2025 Siemens

Controls whether propagation is run in the same process or not, despite what the propagation rules state. By default, this preference is **true**. To run rules in background mode, create this preference and set it to **false**.

TC_propagation_security_group_style

Specifies the list of security groups and associated styles. This preference helps resolve conflicts created by the undesirable case that rules with different propagation styles for a given propagation group name are applicable for an operation.

> Note:
>
> Out of the box, for reference types defined by the **TC_object_type_delegation** preference, access to the named references within a dataset is evaluated based on the security-related properties of the dataset object (project_list, license_list, owning_project, ip_classification, and gov_classification). Custom reference types can be added to the **TC_object_type_delegation** preference. Propagation rules are not required to propagate security attributes from Dataset to Named References.

Migrate propagation preference rules

> Note:
>
> The procedure described in this topic is needed when you upgrade from a Teamcenter version prior to version 11.
>
> The procedure described in this topic is not needed when you upgrade from Teamcenter version 11 or later.

The method for performing propagation changed at Teamcenter version 11. If you upgrade from a Teamcenter version prior to version 11, the upgrade process uses the **migrate_propagation_preferences** utility to migrate propagation preferences to propagation rules and write them to a file, and displays a message similar to the following:

```
The customization of Propagation Rules has been migrated from ProjectPreference
objects to the xml file "TC_DATA\model\propagation_preference_rules.xml". In
order to complete the migration process, import this file in BMIDE Project and
deploy it. Failure to do so means that the custom Propagation Rules will no
more be in effect.
```

To complete the Teamcenter upgrade, you must perform the following steps in the Business Modeler IDE to get your custom propagation rules into your upgraded Teamcenter environment.

1. Choose **File→Import**.

   The Import wizard runs.

2. In the **Select** dialog box, choose **Business Modeler IDE→Import template file**.

© 2025 Siemens

3. Click **Next**.

   The Import template file dialog box is displayed.

4. In the **Project** box, select the project to which you want to import the propagation preference rules. Typically, this is a project that contains other custom data model for your Teamcenter installation.

5. Click the **Browse** button to the right of the **Template file** box and select the *TC_DATA\model\propagation_preference_rules.xml* file.

6. Click **Finish**.

   The *propagation_preference_rules.xml* file is imported to the template project.

7. Save the template and use Teamcenter Environment Manager (TEM) to install the template to your Teamcenter database.

Extension rules

Introduction to application extensions

Application extensions allow for the configuration of applications using a decision table. This extension point defines the table and the inputs and outputs that customers can configure against it.

Application extensions can be used to configure business logic on the server, a Teamcenter rich client application (such as My Teamcenter), or any application. You can use application extensions for anything that calls an input and output, from user interface changes (icons, colors, and so on) to actions. Application extensions use the rules based framework (RBF).

Adding an application extension can be a multistep process:

First, you must add the application extension point.
Next, you can add a rule that governs when the extension point is applied.
Finally, you must ensure that you have the API in the code that calls the extension point.

The extension point ID and input arguments are passed in to the Teamcenter rich client application. Therefore, you must work with an application developer to ensure that the ID and arguments have the proper format to pass in to the application.

> Caution:
>
> Application extension points are like a contract. Once they are defined, others can configure application extension rules on them. Do not modify or delete application extension points after they are created, because they may have been deployed, or application extension rules may have been configured against them.

© 2025 Siemens

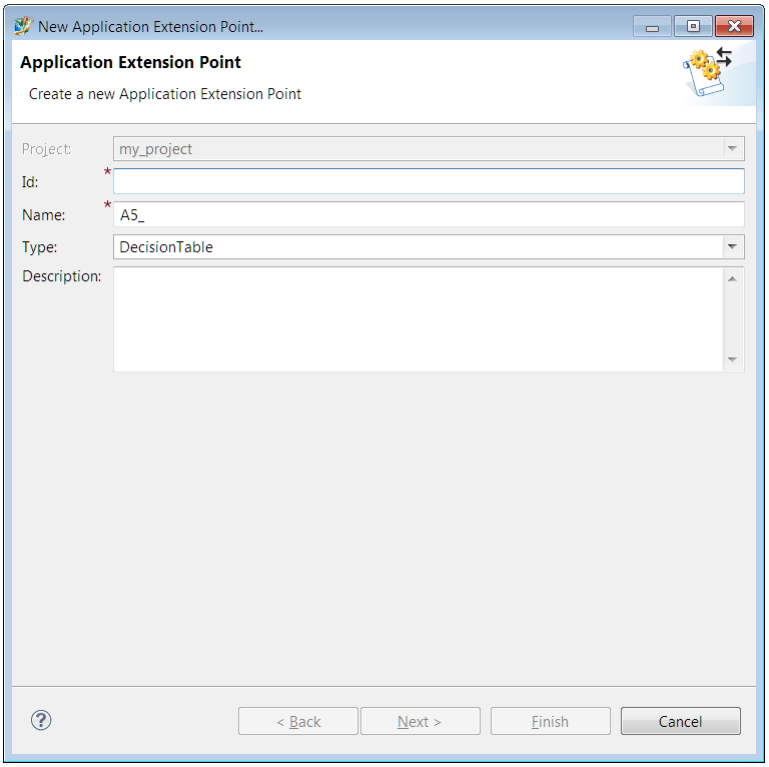
Add an application extension point

An *application extension point* allows for the configuration of applications using a decision table. This extension point defines the table and the inputs and outputs that customers can configure against it. You can use this decision item in a Teamcenter rich client application. After you create an application extension point, you must **create an application extension rule** to govern when it is used.

1. Open the **Extensions\Rules** folders.

2. Right-click the **Application Extension Points** folder and choose **New Application Extension Point**.

   The New Application Extension Point wizard runs.



3. Perform the following steps on the **Application Extension Point** dialog box:

   a. The **Project** box defaults to the previously-selected project.

   b. In the **ID** box, type the ID you want to assign to the extension point. The ID must be all lowercase and have no spaces. This ID is used by the extension rule, and is also passed in to the Teamcenter rich client application.

      For convenience, you may want to include the namespace where the application extension point is used.

© 2025 Siemens

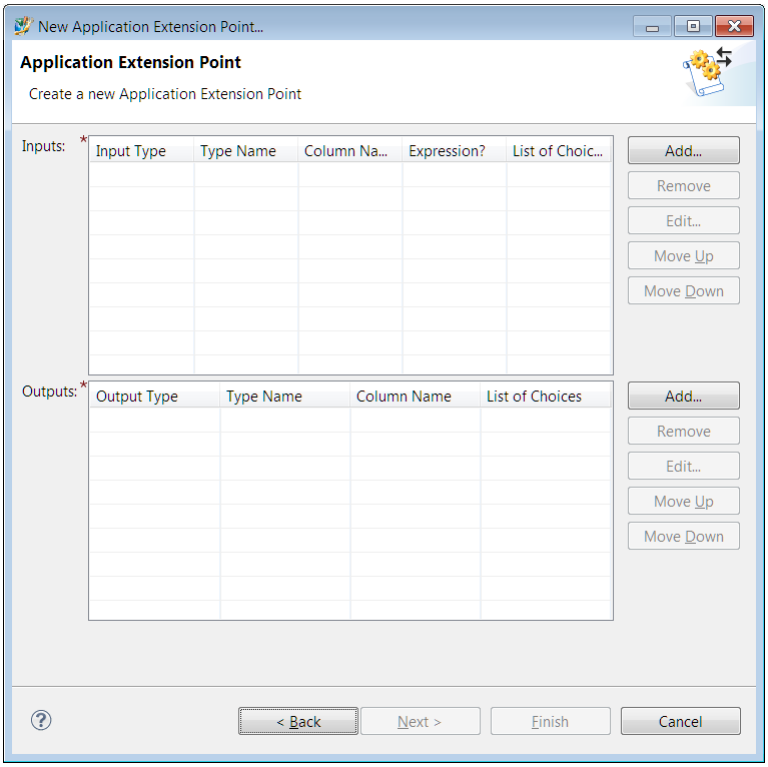
c. In the **Name** box, enter the name you want to assign to the new application extension point.

d. The **Type** box defaults to **DecisionTable**. The decision table lists the inputs and outputs of the extension point.

e. In the **Description** box, enter a description of the application extension point.

f. Click **Next**.

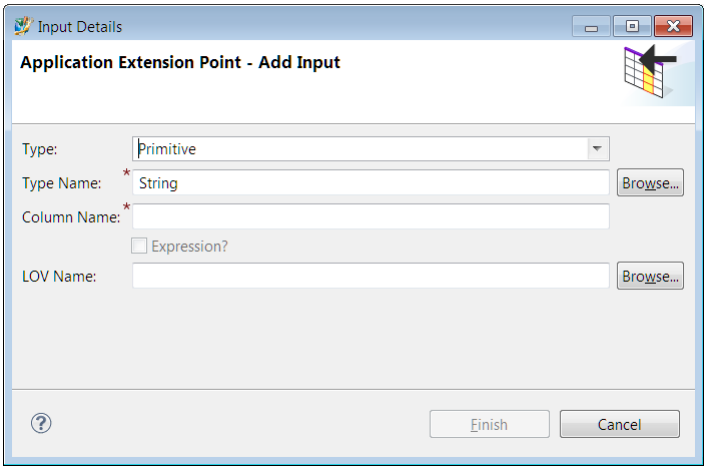
Inputs and **Outputs** tables appear.



4. Click the **Add** button to the right of the **Inputs** table.

The **Add Input** dialog box is displayed.

© 2025 Siemens



Perform the following steps on the **Application Extension Point - Add Input** dialog box:

a. In the **Type** box, select **Primitive** or **Business Object** as the type of extension point input.

   Select **Primitive** if you want to use a data format for input (date, double, float, integer, logical, or string). Select **Business Object** if you want to use properties for input. (You select the properties later, when you create the rule.)

b. Click **Browse** to the right of the **Type Name** box to select a standard data type, either **Boolean**, **Date**, **Double**, **Float**, **Integer**, or **String**.

c. If you previously selected a primitive, in the **Column Name** box, type the name you want for the input column in the extension rule decision table. This column name appears when you **create the extension rule** for this point.

d. If you previously selected a double, float, or integer primitive, select the **Expression?** check box if the input accepts a condition expression (for example, **True** or **False**).

e. If you previously selected a primitive, and you want to use an LOV for the application extension point, click the **Browse** button to the right the **LOV Name** box.

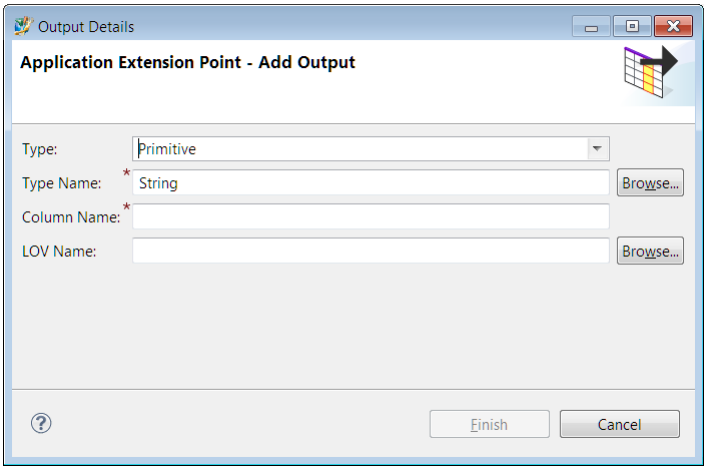
f. Click **Finish**.

   The input appears on the **Inputs** table.

5. Click the **Add** button to the right of the **Outputs** table.

   The **Add Output** dialog box is displayed.

© 2025 Siemens



Perform the following steps in the **Add Output** dialog box:

a. The **Type** box defaults to **Primitive**.

b. Click the **Browse** button to the right of the **Type Name** box to select a type (string, float, date, and so on). Enter an asterisk (*) in the search dialog to see the available types.

c. In the **Column Name** box, type the name you want for the output column in the extension rule decision table. This column name appears when you create the extension rule for this point.

d. If you want to use an LOV for the extension point, click the **Browse** button to the right of the **LOV Name** box

e. Click **Finish**.

The output appears in the **Outputs** table.

6. Continue adding inputs and outputs as desired by clicking the **Add** button to the right of the **Inputs** and **Outputs** tables.

7. After you create inputs and outputs, click **Finish**.

The application extension point is created and appears in the **Application Extension Points** folder.

8. To save the changes to the data model, choose **BMIDE→Save Data Model**, or click the **Save Data Model** button on the main toolbar.

If you want to create a rule that defines when the application extension point is used, you can create an application extension rule.

© 2025 Siemens

Add an application extension rule

An *application extension rule* determines when an application extension point is used, and defines inputs and outputs. When the input is matched, the rule engine returns the output to the application that called the extension point. Before you create an application extension rule, you must have already created an application extension point.

1. Open the **Extensions\Rules\Application Extension Points** folders.

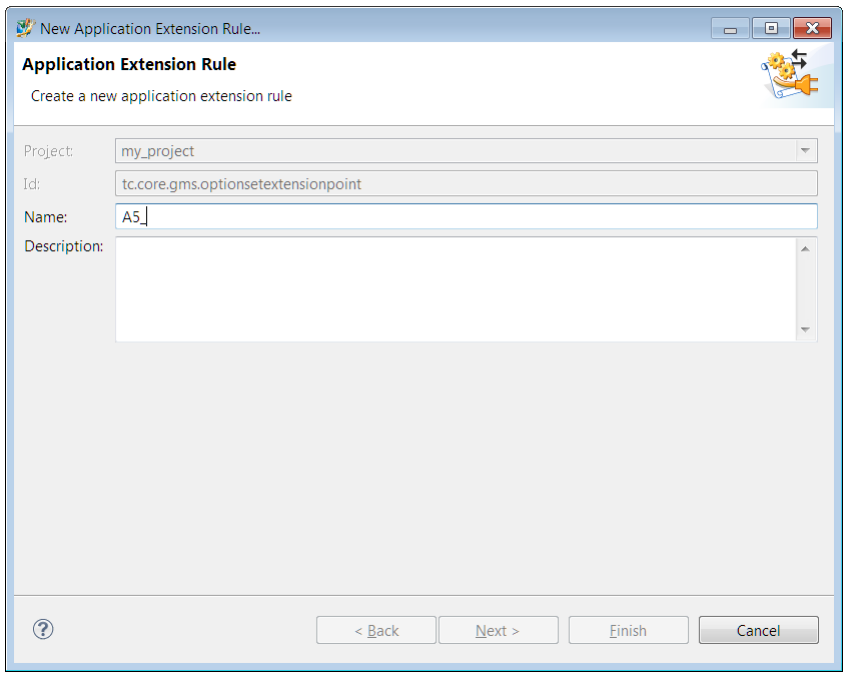
2. Right-click the application extension point against which you want to create the rule and choose **Open**.

   The rule appears in a new view.

   It is helpful to have the extension point open so you can see the input and output details while you create the rule.

3. Right-click the application extension point against which you want to create the rule and choose **Add→Application Extension Rule**.

   The New Application Extension Point Rule wizard runs.



4. Perform the following steps in the **Application Extension Rule** dialog box:

   a. The **Project** box defaults to the project in which you want to create the new rule, and the **ID** box contains the ID for the previously selected application extension point.

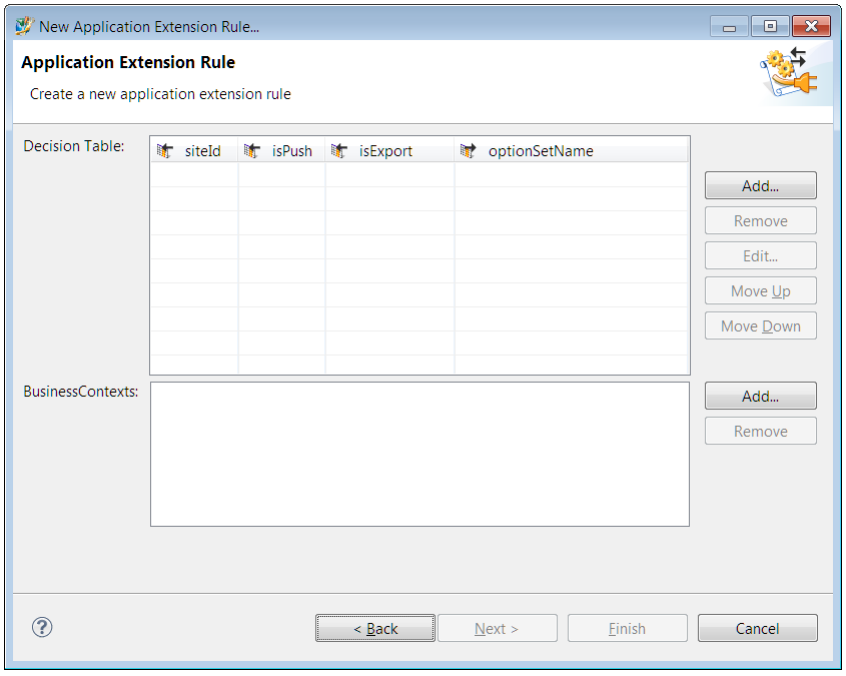
© 2025 Siemens

b. In the **Name** box, enter the name you want to assign to the new rule.

c. In the **Description** box, enter a description of the application extension rule.

d. Click **Next**.

A decision table and a business contexts list appear.

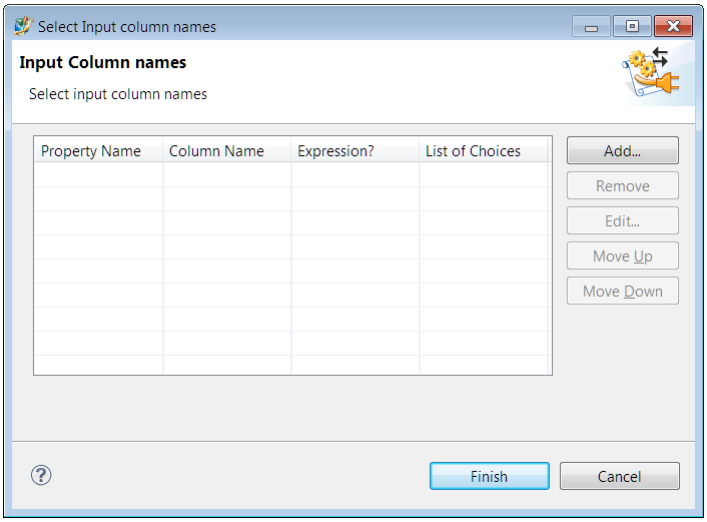


5. The dialog box is different depending on whether you chose a primitive or a business object when you created the extension point. If you chose a primitive for the input on the extension point, proceed to step 6.

If you previously selected a business object for the extension point, a **Config Inputs** button appears to the right of the **Decision Table**. Click the **Config Inputs** button.

The Select Input Column Names wizard runs.

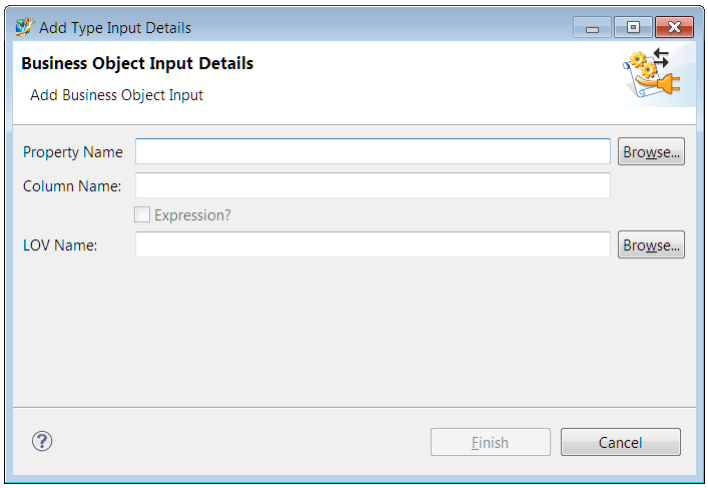
© 2025 Siemens



Perform the following steps on the **Input Column Names** dialog box:

a. Click the **Add** button.

The **Business Object Input Details** dialog box appears.



b. Click the **Browse** button to the right of the **Property Name** box. Select a property to use for input.

c. The **Column Name** box defaults to the property name. Change the name if you want.

d. Click the **Browse** button to the right of the **LOV Name** box if you want to use an LOV for the input. The LOV type must match the property type. For example, if the property is a string type, the LOV must also be a string type.

e. Click **Finish** in the **Business Object Input Details** dialog box.

© 2025 Siemens

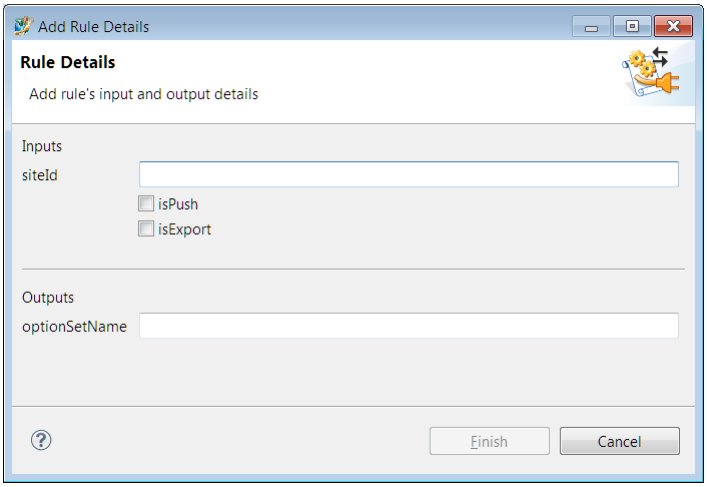
f. Add more properties if you want by clicking the **Add** button.

g. Click **Finish** in the **Input Column Names** dialog box.

6. The decision table displays the input and output column names. For primitives (string, date, and so on), these are created when you made the extension point. For business objects, these are created when you selected the properties on the rule.

   Click the **Add** button to the right of the **Decision Table**.

   The **Rules Detail** dialog box is displayed.



   Perform the following steps on the **Rules Detail** dialog box:

   a. In the **Inputs** box, type the inputs for the point. (The inputs are different for every rule and are provided by the application extension point that the rule is created against.)

   b. In the **Outputs** box, type the outputs for the point. (Like the inputs, the outputs are different for every rule and are provided by the application extension point that the rule is created against.)

   c. Click **Finish**.

7. If you want to define the user group or roles for whom the rule applies, click **Add** button to the right of the **BusinessContexts** pane to **add a business context**.

8. After you create the rule, click **Finish**.

   The application extension rule is created and appears below the application extension point in the **Application Extension Points** folder.

© 2025 Siemens

9. To save the changes to the data model, choose **BMIDE→Save Data Model**, or click the **Save Data Model** button on the main toolbar.

10. Deploy your changes to the server. Choose **BMIDE→Deploy Template** on the menu bar, or select the project and click the **Deploy Template** button on the main toolbar.

11. To verify the application extension rule on the server, run the **execute_rbf_rules** utility.

    This utility validates the application extension rules that are deployed.

Adding an application extension is a three-step process. First you must add the application extension point, and then you must add the rule that governs when the extension point is applied. Finally, you must ensure that you have the API in the code that calls the extension point.

Add a business context

A *business context* defines the user groups for whom a rule applies. For example, if you only want a rule to apply to members of the project administration group, create a business context that identifies the project administration group. Then apply that context to the appropriate rule. Business contexts can be set for application extension rules.

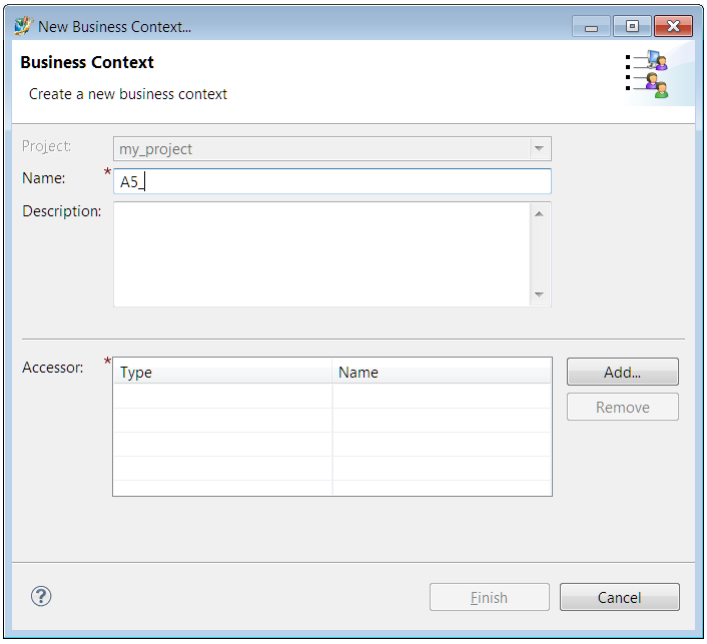
1. Choose one of these methods:

   - On the menu bar, choose **BMIDE→New Model Element**, type **Business Context** in the **Wizards** box, and click **Next**.

   - Open the **Extensions\Rules** folders, right-click the **Business Contexts** folder, and choose **New Business Context**.

   The New Business Context wizard runs.

© 2025 Siemens



2. Perform the following steps in the **Business Context** dialog box:

   a. The **Project** box defaults to the already-selected project.

   b. In the **Name** box, type the name you want to assign to the business context in the database.

      When you name a new data model object, a prefix from the template is automatically affixed to the name to designate the object as belonging to your organization, for example, **A4_**.

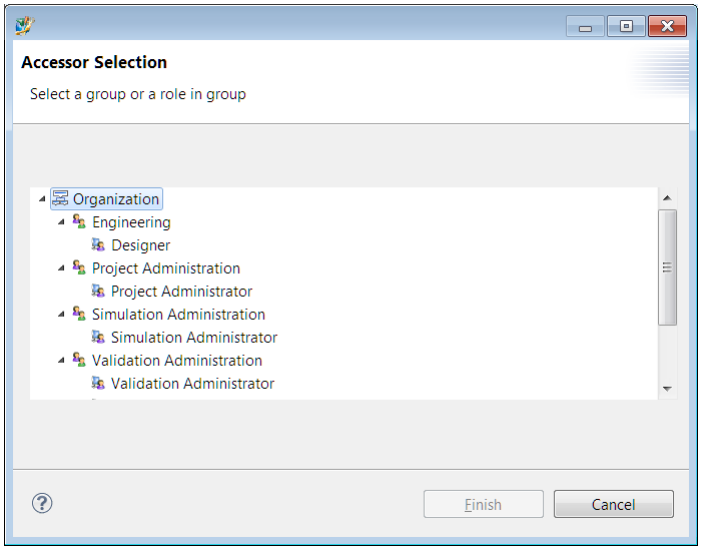
   c. In the **Description** box, type a description of the business context.

   d. Click the **Add** button to the right of the **Accessor** table to choose a group and role for the business context.

      The **Teamcenter Repository Connection wizard** prompts you to log on to a server to look up its available groups and roles.

   e. Select the group and role from the **Accessor Selection** dialog box.

© 2025 Siemens



f. Click **Finish**.

The new business context appears under the **Business Contexts** folder.

3. To save the changes to the data model, choose **BMIDE→Save Data Model**, or click the **Save Data Model** button on the main toolbar.

Now that you have created the business context, it is available for use by other rules. Once the business context is applied to a rule, that rule can only be invoked for the user groups identified in the business context. Business contexts can be set for application extension rules.

Sample application extension APIs

About the sample application extension APIs

When you create an application extension point, you must ensure that you have the application programming interface (API) in the code that calls the extension point. You can call a services-oriented architecture (SOA) API, or an Integration Toolkit (ITK) API.

- SOA API

  The services-oriented architecture (SOA) API evaluates and executes application extension rules contained in the specified application extension point on the given input name-value pairs, and returns the result of the execution in the output name-value pairs. The **executeRbfRules** operation is used for this processing.

- ITK API

  The **RBF_execute** Integration Toolkit (ITK) API is called to evaluate input supplied against the contents of the knowledge base using the rules engine for a specific application extension point (AEP)

© 2025 Siemens

definition. The output results are provided based on that same AEP definition for the calling routine to continue to process. It is the responsibility of the calling routine to decide what processing occurs based on the output received.

© 2025 Siemens

RBF_execute

This set of APIs can be used by an external developer to interact with the rules engine. The declarations can be found in the **rbf.h** include file.

CONSTANT DEFINITIONS

```
#define RBF_VALUE_DATATYPE_STRING                    “STRING”;
#define RBF_VALUE_DATATYPE_BOOLEAN                   “BOOLEAN”;
#define RBF_VALUE_DATATYPE_INTEGER                   “INTEGER”;
#define RBF_VALUE_DATATYPE_DOUBLE                    “DOUBLE”;
#define RBF_VALUE_DATATYPE_FLOAT                     “FLOAT”;
#define RBF_VALUE_DATATYPE_DATE                      “DATE”;
#define RBF_VALUE_DATATYPE_TAG                         “TAG”;
```

Use these constant values as the **dataType** value in the following **RBF_value** structure.

DATA STRUCTURES

```
struct RBF_value
{
    char          *dataType;
    char          *stringValue;
    logical       booleanValue;
    int           integerValue;
    double        doubleValue;
    float         floatValue;
    date_t        dateValue;
    tag_t         tagValue;
    RBF_value()
{
    dataType      = 0;
    stringValue   = 0;
    booleanValue  = false;
    integerValue  = 0;
    doubleValue      = 0;
    floatValue    = 0;
    dateValue     = NULLDATE;
    tagValue      = NULLTAG;
}
};
struct RBF_name_value
{
    char       *name;
    RBF_value  value;
};
```


© 2025 Siemens

These data structures are used to pass input into and receive output from the **RBF_execute** ITK API call. There are helper ITK APIs to help create, populate, and free the array of structures used as both input and output to the **RBF_execute** ITK API:

- **RBF_build_name_value_pairs(**

```
int   RBF_build_name_value_pairs(
const char       *nvp_name,       /* Input */
const char       *nvp_data_type,  /* Input */
const char       *nvp_value,      /* Input */
int              *count,          /* I/O */
RBF_name_value   **values )       /* I/O  (RBF_free_name_value) */
```

  This ITK API builds the array of name/value pair structures for input into the **RBF_execute** ITK API. If the values parameter is **NULL**, then the count and values are initialized with a new parameter. If the values parameter is not **NULL** (already populated), then the count is incremented and values are reallocated and a new parameter added. Call this ITK API (starting with a **NULL** values parameter) for as many times as there are inputs defined for the application extension point ID that is going to be used in the call to the **RBF_execute** ITK API call.

  Note that the input **nvp_value** is always a character string. This ITK API converts that character string to the appropriate data type (based on the input **nvp_data_type** value) for storage into the appropriate value field in the **RBF_value** structure. For example, if the **nvp_data_type** is **RBF_VALUE_DATATYPE_TAG**, then the input **nvp_value** should be a tag string.

- **RBF_free_name_value**

```
int   RBF_free_name_value(
int             *count,      /* Input */
RBF_name_value  **values )   /* I/O */
```

  This ITK API frees the space allocated for an array of name/value pair structures. Call this ITK API for both the input that was created for and the output that was returned from the **RBF_execute** ITK API call.

SIGNATURE

```
int   RBF_execute(
const char          *id,             /* Input */
int               in_count,          /* Input */
RBF_name_value    *in_values,        /* Input */
int               *result_count,     /* Output */
RBF_name_value    **result_values ) /* Output  (RBF_free_name_value) */
```

This API first queries for the application extension point with the given ID. If one is found, the input names are validated to make sure they align with what the application extension point expects. The input (names and values) are then presented to the rules engine in a format understandable by the rules

© 2025 Siemens

engine. The output resulted from any matches found by the rules engine are returned as an array of name-value pairs that are ready to be used by the client.

Input to this service includes an application extension point ID and a count/array of name-value pairs.

Output from this service is a count/array of name-value pairs.

EXAMPLE

```
int  exampleTest( const char *inValue1,
                  const char *inValue2 )
{
    int inCount = 0;
    int ifail = ITK_ok;
    int outCount = 0;
    RBF_name_value *input_values = 0;
    RBF_name_value *output_values = 0;
    /* Build the first name/value pair input for the query
       to the knowledge base */
    ifail = RBF_build_name_value_pairs( "AEP-name-1",
                                        RBF_VALUE_DATATYPE_STRING,
                                        inValue1,
                                        &inCount,
                                        &input_values );
    /* Build the second name/value pair input for the query
       to the knowledge base */
    if ( ifail == ITK_ok )
    {
        ifail = RBF_build_name_value_pairs( "AEP-name-2",
                                            RBF_VALUE_DATATYPE_INTEGER,
                                            inValue2,
                                            &inCount,
                                            &input_values );
    }
    /* Execute the query against the knowledge base */
    if ( ifail == ITK_ok )
    {
        ifail = RBF_execute( "AEP-ID",
                             inCount,
                             input_values,
                             &outCount,
                             &output_values );
    }
    /* Verify expected results for inValue(s) */
    if ( ifail == ITK_ok )
    {
        /***  Do whatever you need to with the results
              found in outCount and output_values  ***/
```


© 2025 Siemens

```
    }
    /* Free allocated storage */
    RBF_free_name_value( inCount,
                         &input_values );
    RBF_free_name_value( outCount,
                         &output_values );
    return( ifail );
}  /* End of exampleTest */
```

In the example, replace *AEP-name-1*, *AEP-name-2*, and *AEP-ID* with values from the application extension point.

© 2025 Siemens

executeRbfRules

Provides the capability to evaluate and execute application extension rules on the server from a Java client. This operation tells the rules engine to apply the set of application extension rules that belong to the specified application extension point on the specified input name-value pairs and returns the result of the execution in the output name-value pairs.

DATA STRUCTURES

```
struct RbfNameValue
{
    std::string name;
    RbfValue value;
};
* name: The "name" is a "string" that identifies the input or output column
* on the Application Extension Rule.
* value: The "value" is specified using an "RbfValue" struct.
struct RbfValue
{
    std::string dataType;
    std::string stringValue;
    bool booleanValue;
    int integerValue;
    double doubleValue;
    float floatValue;
    Date dateValue;
    Tag tagValue;
};
* dataType: indicates the type of data that the struct is holding for the
* specified input or output column on the Application Extension Rule.
* It will will have one of the following values -
* "STRING", "BOOLEAN", "INTEGER", "DOUBLE", "FLOAT", "DATE", "TAG".
* value: the value for the specified input or output column on the
* Application Extension Rule.
struct ExecuteRbfRulesResponse
{
    std::vector < RbfNameValue > outputs;
    Teamcenter::Soa::Server::ServiceData serviceData;
};
* outputs: Outputs are specified as name-value pairs using an "RbfNameValue" struct.
* serviceData: A "ServiceData" struct containing the status of the operation.
```

SIGNATURE

```
ExecuteRbfRulesResponse executeRbfRules( const std::string id,
const std::vector < RbfNameValue > &inputs );
```

DESCRIPTION

An application extension point defines and externalizes a place in the Teamcenter server where a decision can be made.

An application extension rule is an occurrence of an application extension point that is configured based on the constraints provided in the application extension point. Each rule defines a series of inputs and

© 2025 Siemens

outputs. When the input is matched by the rules engine that is running on the server, the rules engine returns the output to the application that called the extension point to be evaluated.

The **executeRbfRules()** operation provides the capability to evaluate and execute application extension rules on the server from a Java client. This operation tells the rules engine to apply the set of application extension rules that belong to the specified application extension point on the specified input name-value pairs, and returns the result of the execution in the output name-value pairs.

INPUT

The **id** value is unique application extension point ID. The **inputs** value is specified as name-value pairs using an **RbfNameValue** structure.

OUTPUT

An **ExecuteRbfRulesResponse** structure containing outputs specified as name-value pairs using an **RbfNameValue** structure and a **ServiceData** structure containing the status of the operation.

The **ServiceData** type is also a way of returning objects that are relevant to the service call (created, deleted, updated, or queried objects) and handling errors. There are methods on **ServiceData** to populate and retrieve the created, deleted, updated, and plain objects, as well as methods to populate and retrieve the error stack.

Attaching extensions to operations

Introduction to extension rules

An *extension* business rule is an independent function or method defined and made available for attachment to a specific business object operation or property as a precondition, preaction, or postaction. The extension adds predefined behavior to the business object operation or property operation. You can also use predefined extensions to perform actions.

In the BMIDE view, predefined extension rules can be viewed under the **Extensions\Rules\Extensions** folder. Attachment is done through the **Extensions Attachments** tab for a business object operation or property.

Using the Business Modeler IDE, you can create your own *custom extensions* for Teamcenter in C or C++ and attach the extension business rules to predefined extension points in Teamcenter, and then subsequently attach the custom rules to an operation of a business object or business object property.

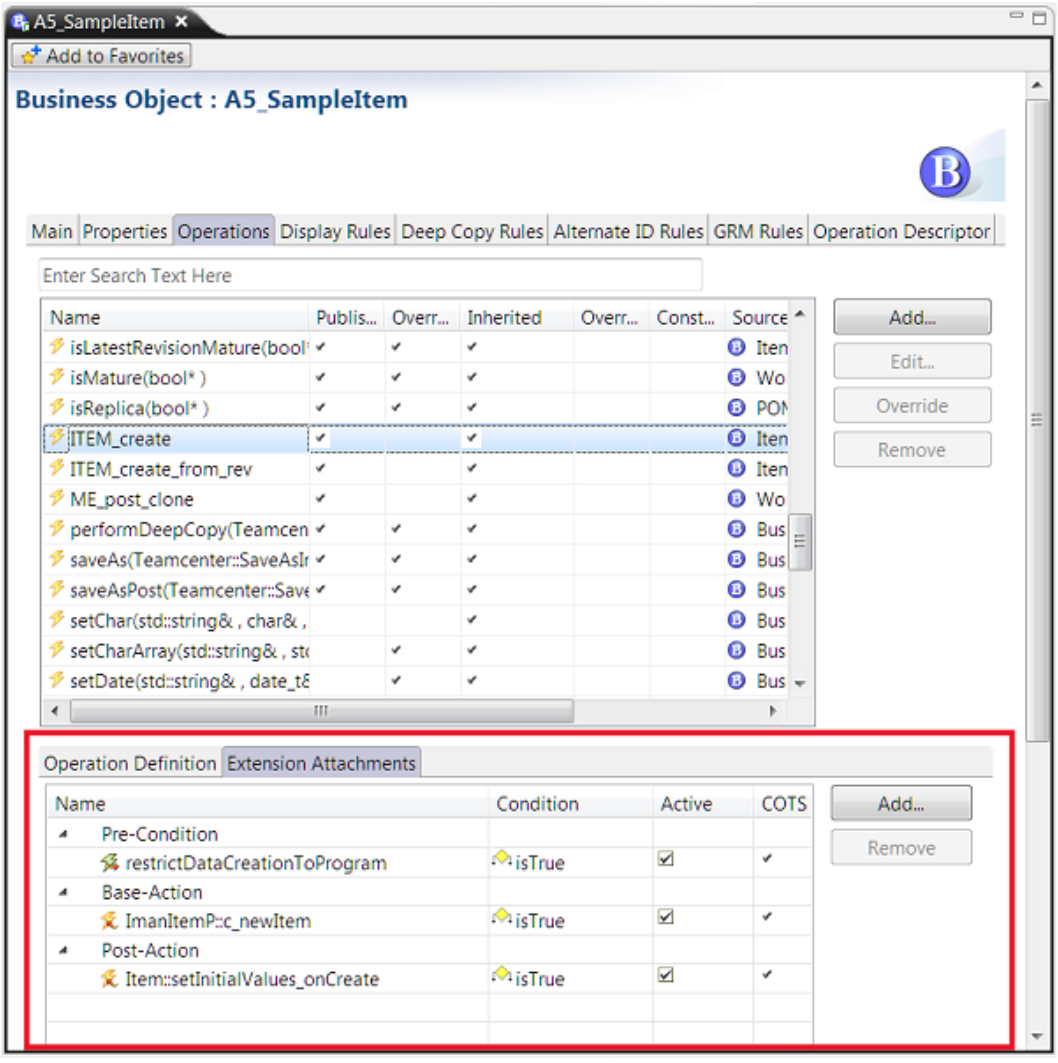
> Note:
>
> Attachment of extension business rules is limited to operations of customer-facing business objects such as Item, Form, and Dataset. System-managed business objects do not typically support extensions.

© 2025 Siemens

Extensions defined for a business object *property* operation can be attached only to the business object for which they were made available.

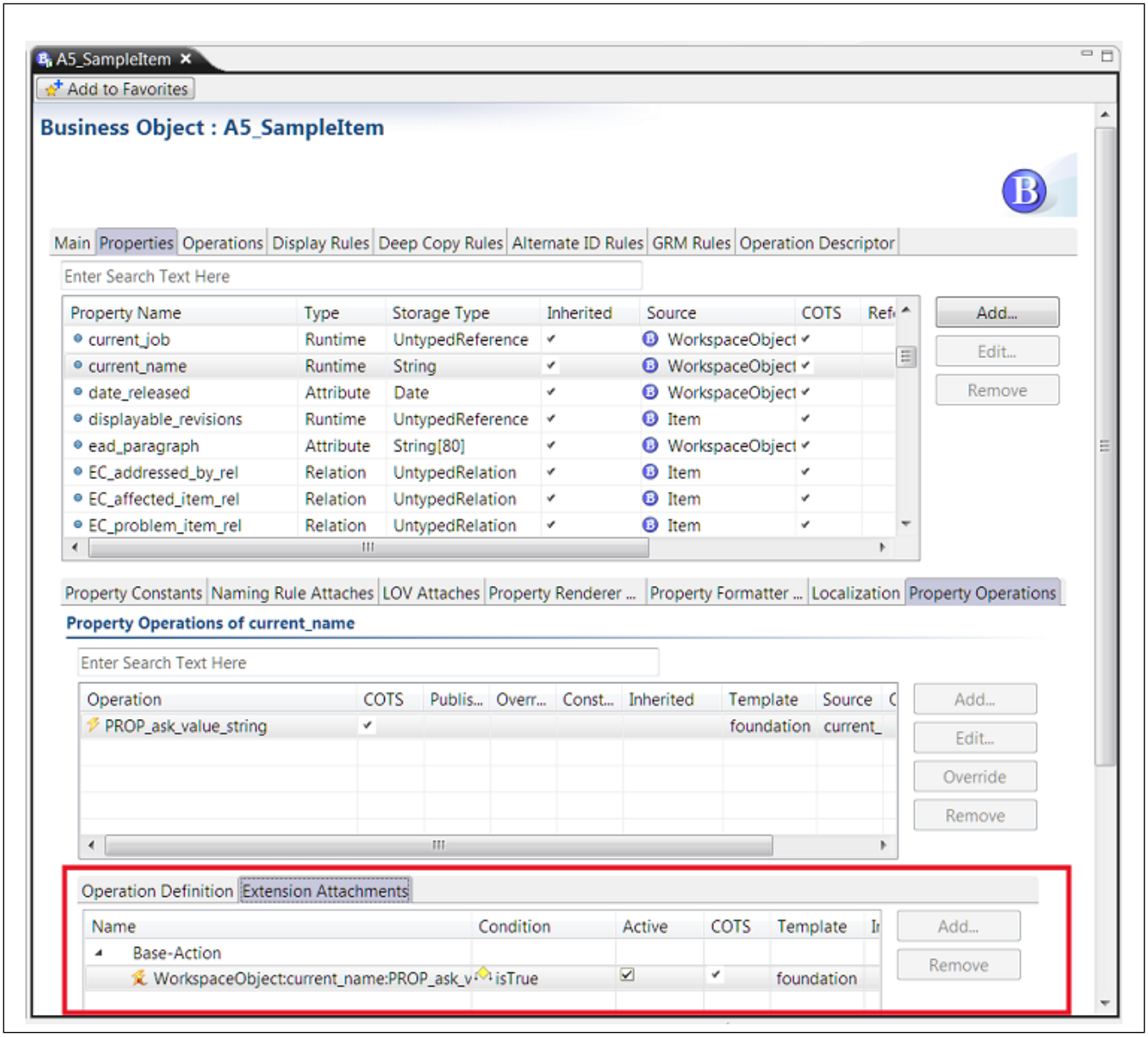
Extensions defined for a *business object* operation can be attached to a child business object of the business object for which they were made available.

Examples of extension attachments in the BMIDE user interface



Extension Attachments tab for business object operations.

© 2025 Siemens



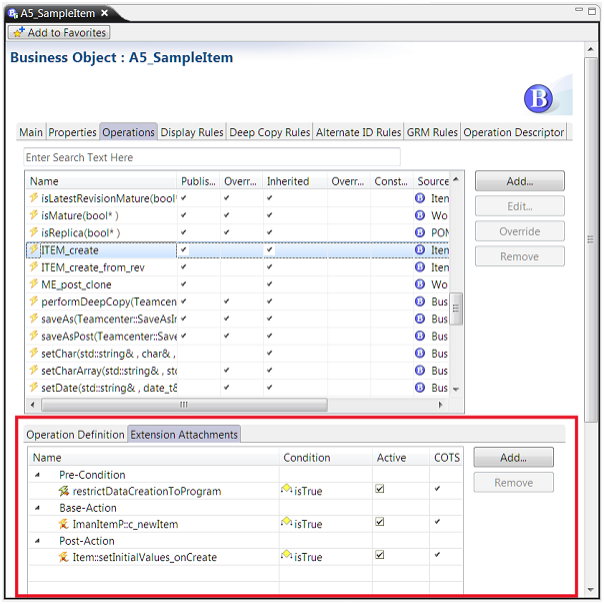
Extension Attachments tab for property operations.

About extension attachments

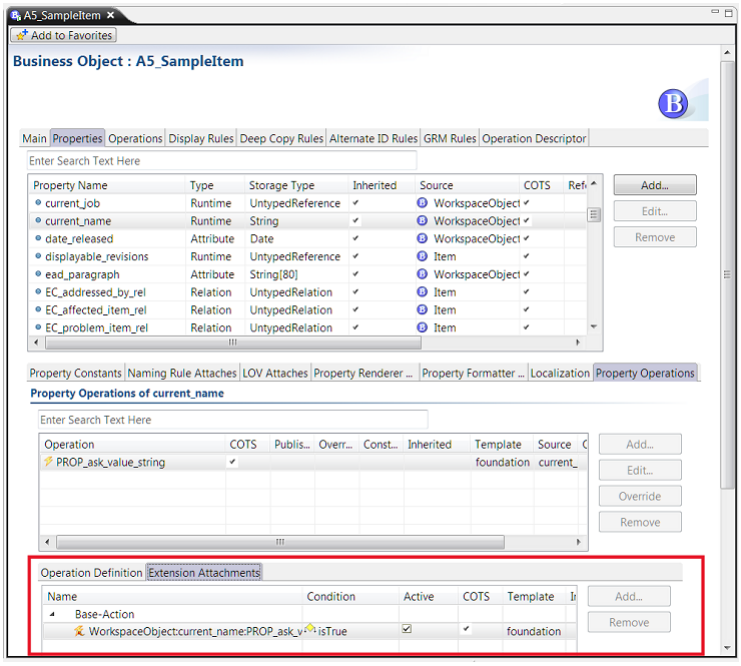
To see operations defined for a business object operation, open the business object and click the **Operations** tab. To see operations defined for a property operation, select the property and click the **Property Operations** tab. To see the extension points defined for an operation, select the operation and click the **Extensions Attachments** tab.

Following is an example of the **Extension Attachments** tab for business object operations.

© 2025 Siemens



Following is an example of the **Extension Attachments** tab for property operations.



Extension points include:

- **Pre-Condition**

  Places limits before an action. For example, limit individual users by their work context to create only a certain item type. Preconditions are executed first in an operation dispatch. If any of the preconditions fails, the operation is aborted. Typical examples of preconditions are naming rules.

© 2025 Siemens

- **Pre-Action**

  Executes code before an action. For example, add user information to the session prior to translation. Preactions are executed after preconditions and before the base action. If any of the preactions fail, the operation is aborted. A typical example is an initial value rule that needs to set an initial value before the save base action is invoked.

- **Base-Action**

  Executes code for an action. The base action is the actual operation implementation, and cannot be replaced. **Base-Action** is used only for user exits operations.

- **Post-Action**

  Executes code after an action. For example, automatically start an item in a workflow. If any of the post-actions fail, the operation is aborted.

Add a predefined extension to a business object

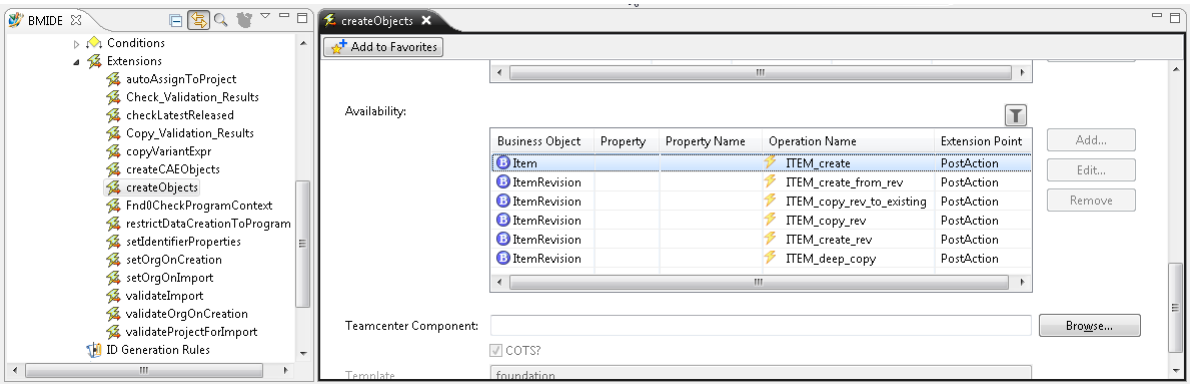
You can use predefined (commercial off the shelf) extensions to perform actions. For example, you can use the predefined **createObjects** extension definition to automatically create a related Microsoft Word dataset whenever an item is created.

1. In the **Extensions** view, expand the project and the **Rules\Extensions** folders.

2. Double-click the predefined extension definition you want to use, for example, **createObjects**.

   The details of the extension appear in a new **Extension Definition** view.

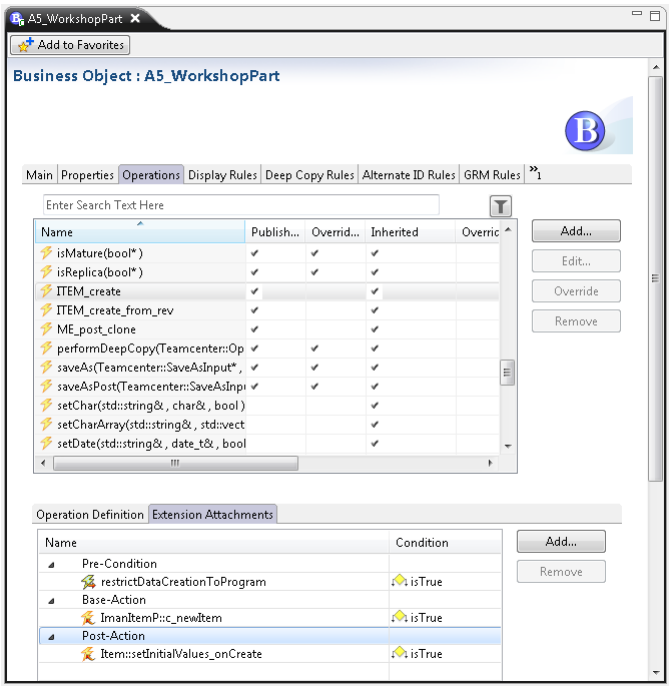
   In the **Availability** table, view the business object, operation, and extension point for which the extension can be used. Decide which business object and operation for which you want to use the extension, and observe the extension point where it can be used.



© 2025 Siemens

3. In the **Business Objects** view, right-click the business object on which you want to use the extension, choose **Open**, and click the **Operations** tab in the resulting editor. The available operations for the business object are found in the table.

4. Select the operation and click the **Extensions Attachments** tab.



5. Click the **Add** button on the **Extension Attachments** tab.

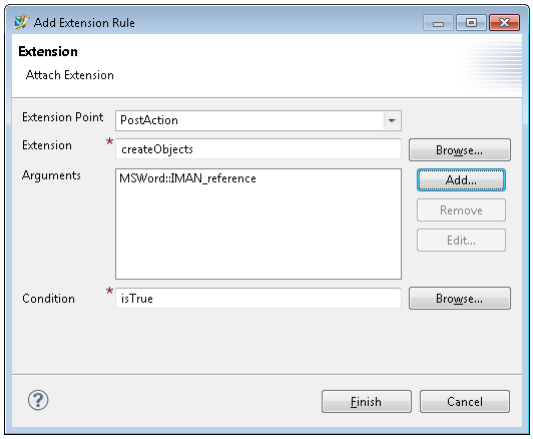
> Note:
>
> This must be an operation and an extension point that appears in the **Availability** table for the extension.

The Add Extension Rule wizard runs.

6. In the **Extension** dialog box, select the extension point (**Pre-Condition**, **Pre-Action**, **Base-Action**, or **Post-Action**) and click the **Browse** button to the right of the **Extension** box to choose the extension.

7. If arguments are required on the operation, the **Add** button to the right of the **Arguments** box is available. Click the **Add** button to add arguments.

© 2025 Siemens



8. Click the **Browse** button to the right of the **Condition** box to select the **condition** that determines when the extension is applied. If you select **isTrue** as the condition, the extension always applies.

   Note:

   Conditions do not apply to base actions; base actions are always performed.

   Only those conditions appear that have valid signatures. For extension rules, the valid condition signature is as follows:

   *condition-name***(UserSession)**

9. Click **Finish**.

   The extension is added to the table under the extension point.

10. Save the data model by choosing **BMIDE→Save Data Model** on the menu bar, and deploy the data model by choosing **BMIDE→Deploy Template**.

    Verify that the extensions work as expected in the Teamcenter rich client.

    For example, if you use the predefined **createObjects** extension definition to automatically create a dataset whenever a certain item type is created, verify that the dataset is created in the My Teamcenter application.

    Note:

    For the **createObjects** extension, the name field is autogenerated and the initial value constant is ignored.

© 2025 Siemens

Attach an extension

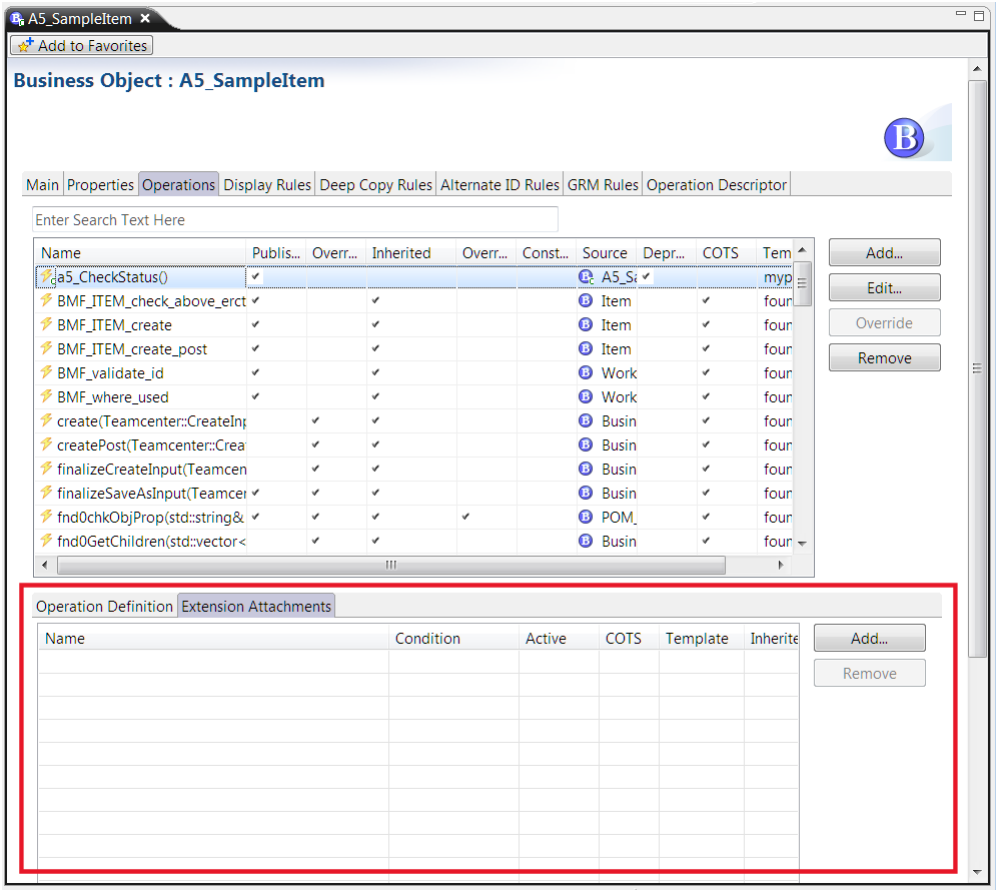
After you define an extension, you can attach it to a business object operation or a property operation so that it is invoked at a particular point (on a precondition, preaction, base action, or postaction on the object).

1. If you made the extension available for use on a business object operation, open the business object and click the **Operations** tab. If you made the extension available for use on a property operation, open the business object the property resides on, click the **Properties** tab, select the property, and click the **Property Operations** tab.

   > Note:
   >
   > You must have already made the rule available to the business object or property in the **Availability** table in the New Extension wizard.

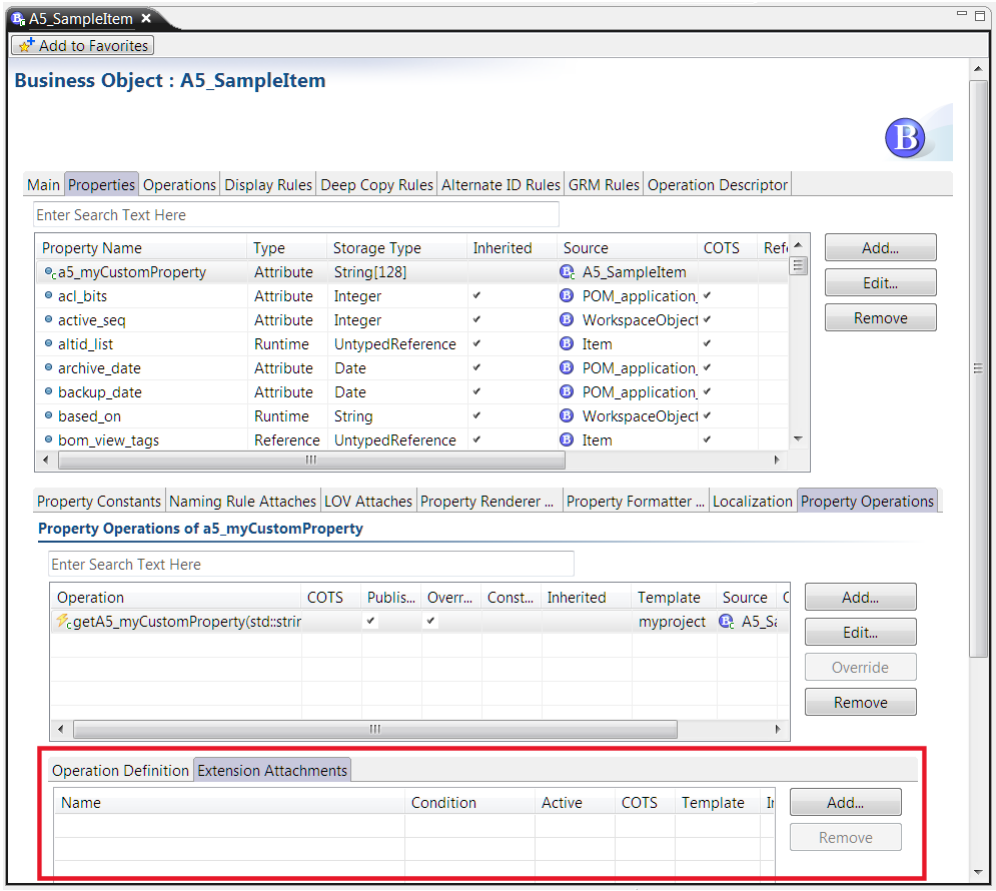
2. To see the extension points defined for an operation, select the operation and click the **Extensions Attachments** tab.

   Following is an example of the **Extension Attachments** tab for a business object operation.



© 2025 Siemens

Following is an example of the **Extension Attachments** tab for a property operation.

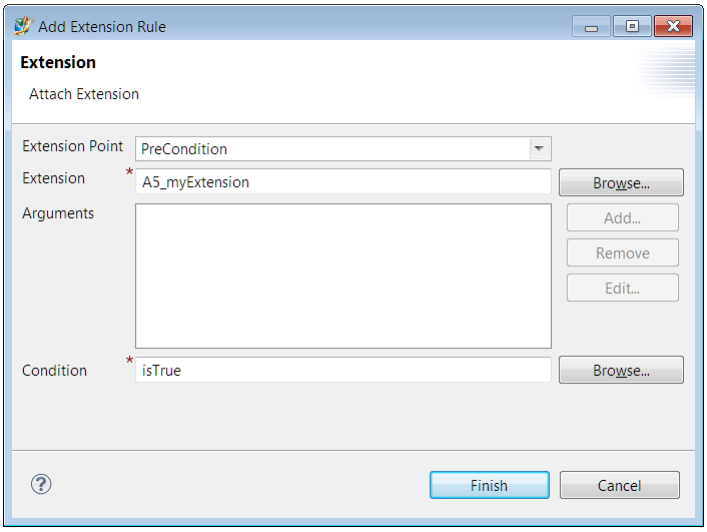


3. Click the **Add** button.

   The **Add** button is available only if there is at least one available extension to attach to this operation.

   The Add Extension Rule wizard is displayed.

© 2025 Siemens



4. Perform the following steps on the **Extension** dialog box:

   a. From the **Extension Point** list, choose the point at which you have made the operation available to run: **PreCondition**, **PreAction**, **BaseAction**, or **PostAction**.

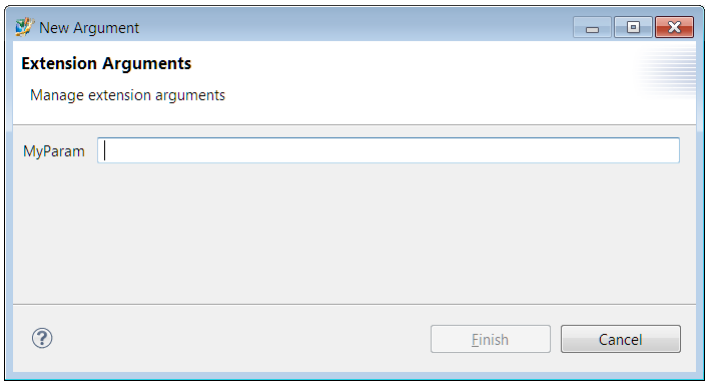
   > Note:
   >
   > The **BaseAction** section is shown only for user exits operations.

   b. Next to **Extension**, click **Browse**. Select the extension you defined earlier and made available.

   Rules appear in the search dialog box only if they have been made available for the business object or property in the **Availability** table in the New Extension wizard.

   c. Click the **Add** button to the right of the **Arguments** box.

   The **Extension Arguments** dialog box is displayed.



© 2025 Siemens

A. In the **Extension Arguments** dialog box, type arguments to append to the rule. Arguments are required if a parameter was set when the extension was created.

B. Click **Finish**.

d. Click the **Browse** button to the right of the **Condition** box to select the **condition** that determines when the extension is applied. If you select **isTrue** as the condition, the extension always applies.

> Note:
>
> Conditions do not apply to base actions; base actions are always performed.
>
> Only those conditions appear that have valid signatures. For extension rules, the valid condition signature is as follows:
>
> *condition-name***(UserSession)**

e. Click **Finish**.

The rule is added to the table and is listed as active (a checkmark appears in the **Active** column).

5. To save the changes to the data model, choose **BMIDE**>**Save Data Model**, or on the main toolbar click **Save Data Model** .

6. Now that you have attached the extension, you must write the code.

Extensions reference

Extensions allow you to use custom functions and predefined methods to extend Teamcenter behavior.

Messages, methods, and method registrations are stored in the database as operations, extensions, and extension points, which can be defined and configured using the Business Modeler IDE. This methodology supports the C and C++ APIs and uses database storage that allows the reuse of extensions.

Before proceeding further, you should become familiar with some terms. *Extensions* allow you to configure system behavior by applying extensions to extension points that are related to business operations in Teamcenter. *Business operations* are actions performed in the system, such as creating and saving an item, fetching or setting a property value, or invoking a user exit. *Extension points* are events in the system, such as a postaction on an operation or a user exit, that allow you to implement custom behavior. *Extensions* contain information about functions associated with Teamcenter business objects and properties.

Business operations can expose one or more extensions points, which can contain zero or more extensions. Extensions within an extension point display the following characteristics:

© 2025 Siemens

- Each extension can have a set of arguments that is unique within the extension point.
- Extensions can be activated and deactivated within an extension point. Inherited extensions can be modified.
- External extensions can be configured in the same manner as the core extensions delivered with Teamcenter.
- An extension can be included more than once in a single extension point with different arguments, and can also be included in multiple extension points depending on availability.
- User exit operations can only be configured from the base extension point (that is, precondition, preaction, and postaction extension points are not available).
- Extensions can execute in any order in a given pre/post set. Consequently, extensions cannot be arranged to execute in a specific sequence within a set.

Operations reference

Operations are actions you can perform in Teamcenter. When you assign an **extension** to a business object or property, you can call an operation on that business object or property.

To see operations defined for a business object, right-click a business object, choose **Open**, and click the **Operations** tab in the resulting view. The following table lists available operations on business objects.

Business object operation	Description
AE_create_dataset	Creates a dataset object in the database.
AE_delete_dataset	Deletes a dataset object from the database.
AE_export_file	Exports a dataset file.
AE_import_file	Imports a dataset file.
AE_save_dataset	Saves a dataset object in the database.
BOM_variant_config	Evaluate a BOM window variant configuration.
GRM_create	Creates a new GRM relation of the given type, linking the specified primary and secondary objects.
IMAN_delete	Deletes an object from the database.

© 2025 Siemens

Business object operation	Description
IMAN_export	Exports an object from the database. (Not supported as a preaction or postaction.)
IMAN_import	Imports an object to the database. (Not supported as a preaction or postaction.)
IMAN_refresh	Reloads (or loads if not previously loaded) the given object from the database. If lock is true, the object is locked for modification. If the lock if false, the object is loaded no-lock.
IMAN_save	Saves the given object to the database.
IMANTYPE_create	Create a new business object.
IMANTYPE_create_props	Creates properties on a business object.
IMANTYPE_init_user_props	Performs user-defined initialization of properties on a business object.
IMANTYPE_viewer_props	Creates viewer properties on a business object.
ITEM_baseline_rev	Produce a new baseline for the given existing revision.
ITEM_copy_rev	Produce a new working revision based on the given existing revision.
ITEM_copy_rev_to_existing	Produce a new revision from an existing item revision based on the **Allow_copy_as_rev** preference. If this preference is set to **true** or **1**, this operation copies the contents of one item revision below an item to a newly created item revision below another existing item. Otherwise, it creates a new item.
ITEM_create	Create a new item with initial working revision.
ITEM_create_from_rev	Create a new item based on an existing item revision.
ITEM_create_rev	Create a new (empty) working revision for an existing item.

© 2025 Siemens

Business object operation	Description
ITEM_deep_copy	Reads the preference rule set for deep copy operation, and performs the following based on the copy rules set for each of the item revision attachments: • **copy_as_object** A new object is created from the existing attachment object. The newly created object has distinct behavior different from its parent. • **copy_as_reference** A symbolic link is created between the attachment of the parent item revision and the newly created item revision. If the attachment of the parent changes, the newly created item revision's corresponding attachment also changes. • **no_copy** The attachment is detached from the newly created item revision.
LOV_ask_disp_values	Converts an LOV to a list of strings.
LOV_ask_num_of_values	Asks how many values are in a LOV.
LOV_ask_value_descriptions	Asks for the list of value descriptions of an LOV.
LOV_ask_values	Asks for the list of values of an LOV.
LOV_ask_values_by_coworker	Asks for values using your own method.
LOV_create	Creates a new LOV.
LOV_insert_values	Inserts values in a LOV.
LOV_is_valid	Determines if the value is valid with a specified LOV.
LOV_set_usage	Sets usage into an LOV.

© 2025 Siemens

Business object operation	Description
LOV_set_value_descriptions	Sets value descriptions in a LOV. The size of the descriptions array must be equal to the size of the values in the LOV.
LOV_set_values	Sets values in a LOV.
LOV_valid_by_coworker	Validates a value by your own method.
ME_create_processoperation	Creates a new process operation object.
ME_clone_template_action	Creates a new message for process cloning.
OBJIO_SM_create	Creates a new storage medium.
WSO_copy	Copies a workspace object.
WSO_create	Creates a workspace object.

To see operations for a property, select the property and click the **Property Operations** tab. The following table lists available operations on properties.

Property operation	Description
PROP_ask_lov_chars	Asks for allowable values as characters that can be set into this property.
PROP_ask_lov_dates	Asks for allowable values as dates that can be set into this property
PROP_ask_lov_doubles	Asks for allowable values as doubles that can be set into this property.
PROP_ask_lov_ints	Asks for allowable values as integers that can be set into this property
PROP_ask_lov_logicals	Asks for allowable values as logicals that can be set into this property.

© 2025 Siemens

Property operation	Description
PROP_ask_lov_strings	Asks for allowable values as strings that can be set into this property.
PROP_ask_lov_tags	Asks for allowable values as tags that can be set into this property.
PROP_ask_value_char	Asks for the value of a single-valued character property. The property cannot be an array or list.
PROP_ask_value_char_at	Asks for the value of a multivalued (that is, list or array) character property at a particular index position.
PROP_ask_value_chars	Asks for one or more values of a character property. The property can be single-valued or multivalued (that is, array or list).
PROP_ask_value_date	Asks for the value of a single-valued date property. The property cannot be an array or list.
PROP_ask_value_date_at	Asks for the value of a multivalued (that is, list or array) date property at a particular index position.
PROP_ask_value_dates	Asks for one or more values of a date property. The property can be single-valued or multivalued (that is, array or list).
PROP_ask_value_double	Asks for the value of a single-valued double property. The property cannot be an array or list.
PROP_ask_value_double_at	Asks for the value of a multivalued (that is, list or array) double property at a particular index position.
PROP_ask_value_doubles	Asks for one or more values of a double property. The property can be single-valued or multivalued (that is, array or list).
PROP_ask_value_int	Asks for the value of a single-valued integer property. The property cannot be an array or list.
PROP_ask_value_int_at	Asks for the value of a multivalued (that is, list or array) integer property at a particular index position.

© 2025 Siemens

Property operation	Description
PROP_ask_value_ints	Asks for one or more values of a integer property. The property can be single-valued or multivalued (that is, array or list).
PROP_ask_value_logical	Asks for the value of a single-valued logical property. The property cannot be an array or list.
PROP_ask_value_logical_at	Asks for the value of a multivalued (that is, list or array) logical property at a particular index position.
PROP_ask_value_logicals	Asks for one or more values of a logical property. The property can be single-valued or multivalued (that is, array or list).
PROP_ask_value_string	Asks for the value of a single-valued string property. The property cannot be an array or list.
PROP_ask_value_string_at	Asks for the value of a multivalued (that is, list or array) string property at a particular index position.
PROP_ask_value_strings	Asks for one or more values of a string property. The property can be single-valued or multivalued (that is, an array or list).
PROP_ask_value_tag	Asks for the value of a single-valued tag property. The property cannot be an array or list.
PROP_ask_value_tag_at	Asks for value of a multivalued (that is, list or array) tag property at a particular index position.
PROP_ask_value_tags	Asks for one or more values of a tag property. The property can be single-valued or multivalued (that is, array or list).
PROP_is_modifiable	Indicates whether a property can be modified by the user. You can register a postaction to the default method that does additional checks.
PROP_set_value_char	Sets a value on a single-valued character property.

© 2025 Siemens

Property operation	Description
PROP_set_value_char_at	Sets the value of a multivalued (that is, list or array) character property at a particular index position.
PROP_set_value_chars	Sets one or more values on a character property. The property can be single-valued or multivalued (that is, array or list).
PROP_set_value_date	Sets a value on a single-valued date property.
PROP_set_value_date_at	Sets the value of a multivalued (that is, list or array) date property at a particular index position.
PROP_set_value_dates	Sets one or more values on a date property. The property can be single-valued or multivalued (that is, array or list).
PROP_set_value_double	Sets a value on a single-valued double property.
PROP_set_value_double_at	Sets the value of a multivalued (that is, list or array) double property at a particular index position.
PROP_set_value_doubles	Sets one or more values on a property. The property can be single-valued or multivalued (that is, array or list).
PROP_set_value_int	Sets a value on a single-valued integer property.
PROP_set_value_ints	Sets one or more values on a integer property. The property can be single-valued or multivalued (that is, array or list).
PROP_set_value_int_at	Sets value of a multivalued (that, list or array) property at a particular index position.
PROP_set_value_logical	Sets a value on a single-valued logical property.
PROP_set_value_logical_at	Sets the value of a multivalued (that is, list or array) logical property at a particular index position.
PROP_set_value_logicals	Sets one or more values on a logical property. The property can be single-valued or multivalued (that is, array or list).

© 2025 Siemens

Property operation	Description
PROP_set_value_string	Sets the value on a single-valued string property.
PROP_set_value_string_at	Sets the value of a multivalued (that is, list or array) string property at a particular index position.
PROP_set_value_strings	Sets one or more values on a string property. The property can be single-valued or multivalued (that is, array or list).
PROP_set_value_tag	Sets a value on a single-valued tag property.
PROP_set_value_tag_at	Sets the value of a multivalued (that is, list or array) tag property at a particular index position.
PROP_set_value_tags	Sets one or more values on a tag property. The property can be single-valued or multivalued (that is, array or list).
PROP_UIF_ask_value	Asks the display value of any type of property.
PROP_UIF_set_value	Sets the value of any type of property using a display value as input.
PROP_validate_lov_char	Validates the value of a property as a character by LOV.
PROP_validate_lov_date	Validates the value of a property as a date by LOV.
PROP_validate_lov_double	Validates the value of a property as a double by LOV.
PROP_validate_lov_int	Validates the value of a property as an integer by LOV.
PROP_validate_lov_logical	Validates the value of a property as a logical by LOV.
PROP_validate_lov_string	Validates the value of a property as a string by LOV.
PROP_validate_lov_tag	Validates the value of a property as a tag by LOV.

Extension inheritance reference

The following guidelines apply to the inheritance behavior of extensions:

- Extensions on business objects are propagated to (inherited by) sub-business objects.

© 2025 Siemens

- Inherited extensions cannot be modified at the sub-business object level. Modifications must be made at the parent level.

- When a new extension is assigned to a sub-business object, the sub-business object inherits the extensions of the parent business object.

Implications of the autoAssignToProject extension on propagation rules

An administrator can attach the **autoAssignToProject** extension to a business object operation **PostAction** extension point. When the action occurs, the extension automatically assigns the user's current project defined by the work context or user settings.

Configuring the **autoAssignToProject** internal extension for a business object has implications on the project propagation rules. Project propagation rules determine which secondary objects are assigned to a project when a primary business object is assigned. When there is a conflict between a propagation rule and the execution of the **autoAssignToProject** extension, the extension takes precedence.

Improving performance for bulk operations

As of Teamcenter release 14.2, enhancements have been made to improve performance in the case of bulk operations. When the **autoAssignToProject** extension is attached to the type-specific object and object-set operations listed in column **Attach to the following operations**, then the project assignment extension is called only once per object type and its subtypes. If the extension is already attached to the operations listed in column **Detach from following operations**, then detach the extension from these operations to gain the improved performance.

Type (and all its subtypes)	Attach to the following operations	Detach from following operations
Item	**IMAN_save** **fnd0Save**	**IMAN_import** **ITEM_create** **ITEM_create_from_rev**
Item revision	**IMAN_save** **fnd0Save**	**ITEM_baseline_rev** **ITEM_copy_rev** **ITEM_copy_rev_to_existing** **ITEM_create_rev**
Dataset	**AE_save_dataset** **fnd0Save**	
Form	**IMAN_save** **fnd0Save**	

© 2025 Siemens

Interactions with propagation rules

The following scenarios illustrate the relationship between extensions and propagation rules when assigning objects to projects.

Note:

- If a current project is not specified for the user, this extension is ignored and the object is not automatically assigned. In addition, when the **autoAssignToProject** extension is configured for an item or ECO, the project name is preselected in the **Assign to Projects** page of the item or ECO create, revise, and save as dialog boxes.
- If you want users to be able to remove objects from an owning project, you must create the **TC_allow_remove_owning_project** preference before using the **autoAssignToProject** extension. If this preference is not set, objects assigned to owning projects cannot be removed using the **Project→Remove** command.

The following points must be considered when implementing the **autoAssignToProject** extension:

- The **autoAssignToProject** extension applies only to newly created objects; whereas, propagation of related objects to projects occurs whenever a relation between two objects is created, modified, or deleted.
- The **autoAssignToProject** extension explicitly assigns objects to projects; therefore, the objects can only be removed from the project by explicitly right-clicking the object in the Teamcenter rich client and choosing **Project→Remove**.
- Propagation rules implicitly assign secondary objects to projects. Therefore, when the primary object is explicitly removed from the project, the secondary object is also removed from the project.
- When importing or exporting project or program data in a Multi-Site Collaboration environment, you must first create the project on both the export and the import site prior to using the **autoAssignToProject** extension.

Scenario	Project assignment behavior
The **autoAssignToProject** extension is configured for types **P** (primary object) and types **S** (secondary object). A user creates an object of type **P** and an object of type **S** related by the **Requirements** relation.	Both objects are automatically assigned to the current project, regardless of whether the **Requirements** relation is specified in the propagation rule list.
The **autoAssignToProject** extension is configured for types **P** (primary object), but not for types **S** (secondary object). A user creates an object of type **P** and an object of type **S** related by the **Requirements** relation.	The object of type **P** is automatically assigned to the current project based on the **autoAssignToProject** extension. If the **Requirements** relation is specified in the propagation rule list, the type **S** object is also assigned to the project. If the **Requirements**

© 2025 Siemens

Scenario	Project assignment behavior
	relation is not specified in the propagation rule list, the secondary object is not assigned to the project.
The **autoAssignToProject** extension is configured for types **P** (primary object) and types **S** (secondary object). In addition, the **Requirements** relation is defined as a propagation rule. The user creates an object of type **P** and an object of type **S**. After creating the objects, the user attaches the secondary object to the primary object using the **Requirements** relationship.	Both the primary and secondary object are automatically assigned to the project based on the configuration of the extension, resulting in an explicit assignment rather than the implicit assignment that occurs when an object is assigned to a project based on propagation rules.

Alternate ID rules

Add an alternate ID rule

Alternate identifiers store information (such as part numbers and attributes) about the same part from different perspectives. They allow different types of users to display an item according to their own rules rather than according to the rules of the user who created the object. Only **Item** business objects or its children use alternate IDs. Alias IDs are similar to alternate IDs. **Alias IDs** store information for similar parts.

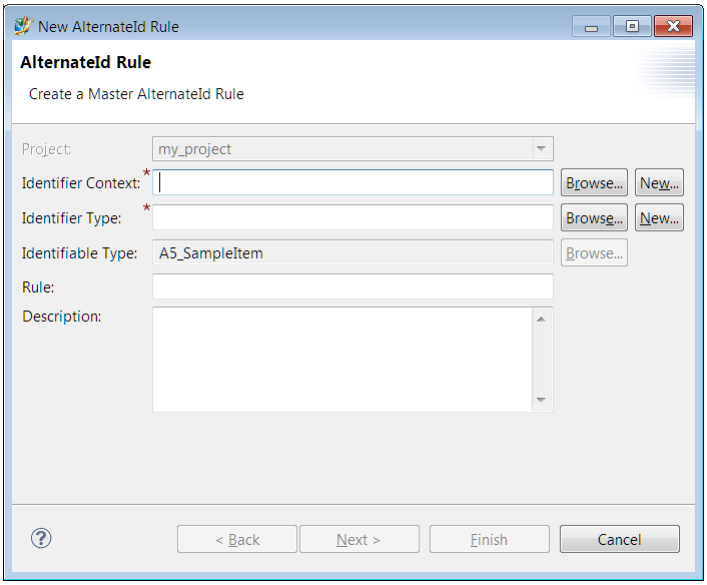
Before creating an alternate ID rule, you must **create Identifier business objects** whose properties hold information such as a supplier's address and cost data. You also need to **create Id Context** objects that specify when the alternate ID is used, such as for a supplier or a department.

1. Right-click the **Item** business object or one of its children, choose **Open**, and click the **Alternate ID Rules** tab.

2. Click the **Add** button to the right of the **Master Alternate ID Rules** table.

   The New Alternate ID Rule wizard runs.

© 2025 Siemens



3. In the **Alternate ID Rule** dialog box, enter the following information:

   a. The **Project** box defaults to the already-selected project.

   b. Click the **Browse** button to the right of the **Identifier Context** box to choose the context when the rule is to be applied. To **create a new Id Context** option, click the **New** button.

   c. Click the **Browse** button to the right of the **Identifier Type** box to choose the **Identifier** or **IdentifierRev** business object as the parent type for this rule. To **create a new Identifier** business object, click the **New** button.

   d. The **Identifiable Type** box defaults to the already-selected business object. This is the business object that this rule applies to.

   e. Use the **Rule** box to govern how many alternate parts can be used for the original one. This is like a cardinality rule that is used for GRM rules.

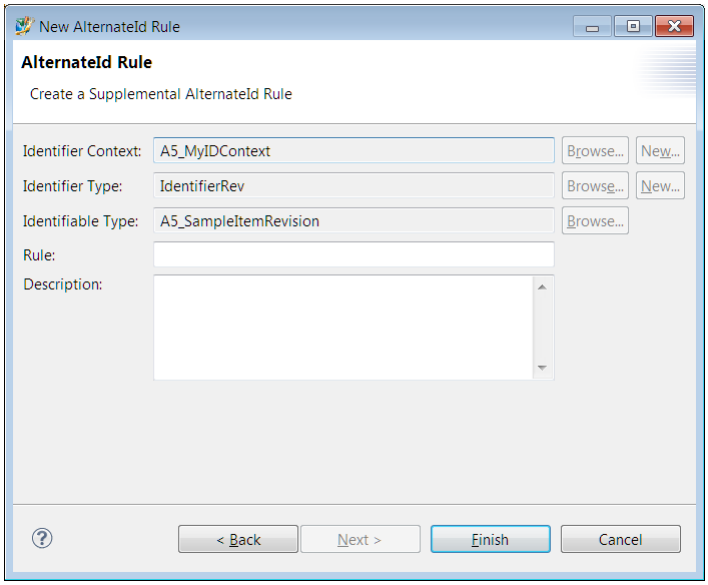
      Leave the **Rule** box empty if you want to allow more than one identifier object associated with the identifiable type. However, if you only want to allow one, then enter a unique string.

   f. In the **Description** box, type a description of the rule.

   g. Click **Next**.

      The dialog box appears for creating a supplement rule.

© 2025 Siemens



4. Create a supplemental alternate ID rule, which is comparable to a revision. In the **AlternateId Rule** dialog box, enter the following:

   a. The value in the **Identifier Context** box defaults to the identifier context of the master alternate ID rule.

   b. The value in the **Identifier Type** box defaults to the identifier type of the master alternate ID rule.

   c. The value in the **Identifiable Type** box defaults to the identifiable type of the master alternate ID rule.

   d. In the **Rule** box, enter how many alternate parts can be used for the original one. Leave the **Rule** box empty if you want to allow more than one identifier object associated with the identifiable type. However, if you only want to allow one, then enter a unique string.

   e. In the **Description** box, type a description of the supplemental rule.

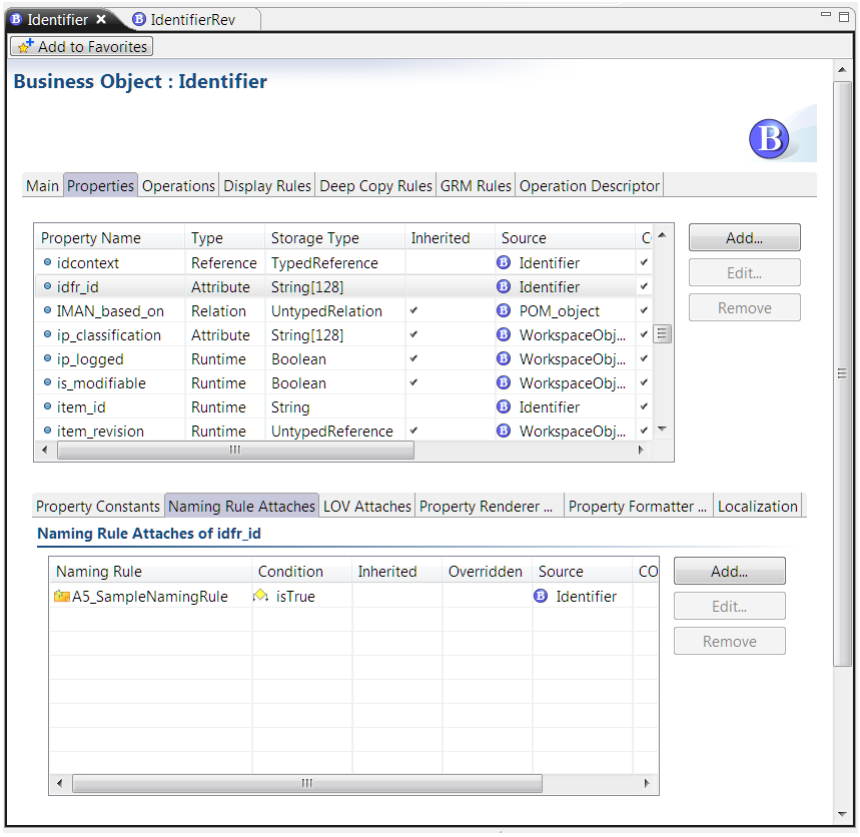
   f. Click **Finish**.

      A master and supplemental rule are created.

5. You must **attach a naming rule** to the ID property of the business object type you used for the identifier. If you do not, you receive an error when you create the alternate ID in the client.

   For example, if you used the **Identifier** business object, you must attach a naming rule to the **idfr_id** property.

© 2025 Siemens



6. To save the changes to the data model, choose **BMIDE→Save Data Model**, or click the **Save Data Model** button on the main toolbar.

7. Deploy your changes to the test server. Choose **BMIDE→Deploy Template** on the menu bar, or select the project and click the **Deploy Template** button on the main toolbar.

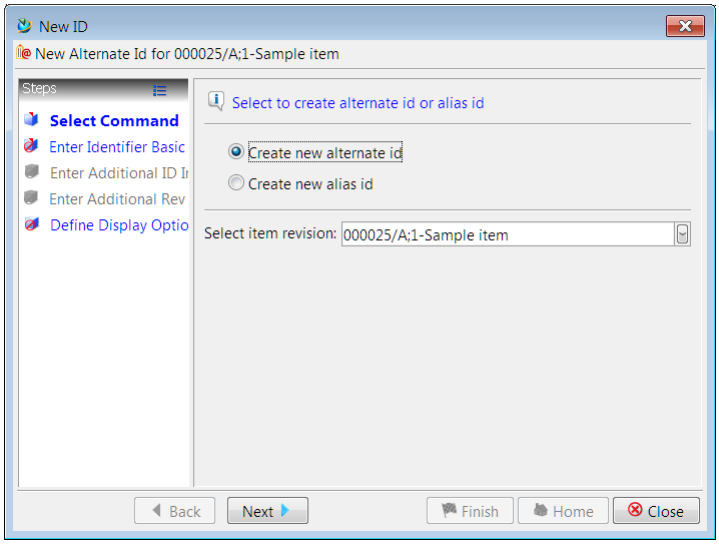
8. After deployment, test your new alternate ID rule in the Teamcenter rich client:

    a. In the My Teamcenter application, select an item you want to apply the rule to. This must be the same kind of business object you applied the rule to in the **Identifiable Type** box.

    b. Choose **File→New→ID**.

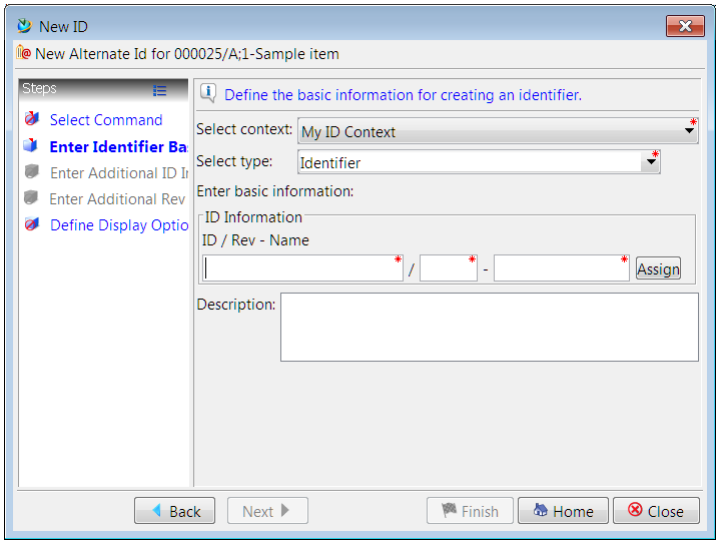
    The **New ID** dialog box is displayed.

© 2025 Siemens



c. In the **New ID** dialog box, choose **Create new alternate ID** and click **Next**.

The dialog box displays for creating an alternate ID.

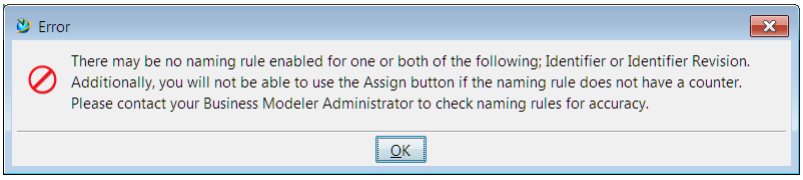


d. Click the arrow in the **Select context** box to choose the ID context you defined on the rule, and click the arrow in the **Select type** box to select the identifier business object you defined on the rule.

e. Click the **Assign** button to assign an ID to the identifier and the identifier revision.

If you did not assign a naming rule to the identifier or identifier revision, you see the following error. Ensure you attached a naming rule to the identifier business object as described earlier.

© 2025 Siemens



Alternate ID rule example

To create an alternate ID rule, right-click the **Item** business object or one of its children, choose **Open**, click the **Alternate ID Rules** tab, and click the **Add** button to the right of the **Master Alternate ID Rules** table.

To illustrate a rule for alternate creation, consider that you have two item business objects, **PartDesign** and **PartMfg**; four identifier business objects, **Identifier**, **IdentifierRev**, **MfgIdentifier**, and **MfgIdentifierRev**; and four contexts, **Production Part**, **Temporary Part**, **Prototype Process**, and **Production Process**. To ensure that the design and manufacturing groups each have their own valid combinations, you could define rules using the following combinations in the New Alternate ID Rule wizard:

- **Identifier Context: Production Part**

  Identifier Type: Identifier

  Identifiable Type: PartDesign

- **Identifier Context: Production Part**

  Identifier Type: IdentifierRev

  Identifiable Type: PartDesign Revision

- **Identifier Context: Temporary Part**

  Identifier Type:Identifier

  Identifiable Type: PartDesign

- **Identifier Context: Temporary Part**

  Identifier Type:IdentifierRev

  Identifiable Type: PartDesign Revision

- **Identifier Context: Prototype Process**

  Identifier Type: Identifier

© 2025 Siemens

Identifiable Type: PartMfg

- **Identifier Context: Prototype Process**

  Identifier Type: IdentifierRev

  Identifiable Type: PartMfg Revision

- **Identifier Context: Production Process**

  Identifier Type: MfgIdentifier

  Identifiable Type: PartMfg

- **Identifier Context: Production Process**

  Identifier Type: MfgIdentifierRev

  Identifiable Type: PartMfg Revision

> Note:
>
> When defining rules for alternates, you must have a rule for the item business object and a rule for the corresponding item revision business object. You cannot create alternate IDs unless both rules are defined. The identifier business object for an item revision must be the name of the identifier business object associated with the item appended with **Rev**.

All of the combinations listed previously are valid; however, they do not define cardinality or how many of each identifier can exist for a given instance of an item. Without a cardinality rule, a **PartDesign** item and item revision can have as many alternate IDs in the **Production Part** or **Temporary Part** contexts as desired.

To allow items to have more than one part number in the **Temporary Part** or **Production Part** contexts, but to limit item revisions to one alternate in those same contexts, you can define the following cardinality rule:

- **Identifier Context: Production Part**

  Identifier Type: Identifier

  Identifiable Type: PartDesign

  Rule: NULL

- **Identifier Context: Production Part**

© 2025 Siemens

Identifier Type: IdentifierRev

Identifiable Type: PartDesign Revision

Rule: OneProdPart

- **Identifier Context: Temporary Part**

  Identifier Type: Identifier

  Identifiable Type: PartDesign

  Rule: NULL

- **Identifier Context: Temporary Part**

  Identifier Type: IdentifierRev

  Identifiable Type: PartDesign Revision

  Rule: OneTempPart

> Note:
>
> The values **OneProdPart** and **OneTempPart** are user-defined keys. You can use any string, such as the values **A** and **B**, to achieve the same results. Once the key value is not null, you can have only one combination of **IDContext/Identifier Type/Identifiable Type** that exists for a given instance of an identifiable.

These rule combinations allow a **PartDesign** item to have many part numbers in production and temporary context, but restricts **PartDesign Revision** to only one part number in production and/or temporary context.

The following combinations allow an item revision to have an alternate ID in either **Production Part** or **Temporary Part** context but not in both contexts:

- **Identifier Context: Production Part**

  Identifier Type: Identifier

  Identifiable Type: PartDesign

  Rule: NULL

- **Identifier Context: Production Part**

© 2025 Siemens

Identifier Type: IdentifierRev

Identifiable Type: PartDesign Revision

Rule: OneEngPart

- **Identifier Context: Temporary Part**

  Identifier Type: Identifier

  Identifiable Type: PartDesign

  Rule: NULL

- **Identifier Context: Temporary Part**

  Identifier Type: IdentifierRev

  Identifiable Type: PartDesign Revision

  Rule: OneEngPart

> Note:
>
> The value **OneEngPart** is a user-defined key. You can use any string, such as the values **A**, to achieve the same results. Once the key value is not null, you can have only one combination of **IDContext/Identifier Type/Identifiable Type** with the same key value that exists for a given instance of an identifiable.

Alternate ID rules characteristics

When **defining rules for alternates**, you must have a rule for the item business object and a rule for the corresponding item revision business object. You cannot create alternate IDs unless both rules are defined.

The identifier business object for an item revision must be the name of the identifier business object associated with the item appended with **Rev**.

The **Rule** box allows the user to restrict the cardinality between the identifiable and the identifier. These are the valid cases for the **Rule** box:

- If the **Rule** box is null for the given combination (ID context rule), the system allows more than one identifier object of that identifier business object to be associated with the same identifiable object. So, this is like cardinality **N**.

© 2025 Siemens

- If the **Rule** box has some string defined in it, and this string is unique across all ID context rules for the given identifier and identifiable business objects, the system allows only one identifier object of that identifier business object to be associated with the identifiable object. So, this is like cardinality **1**.

- If the **Rule** box has some string defined in it, and this string is shared across some ID context rules for the given identifier and identifiable business objects, the system allows only one identifier object of *one* of the identifier business objects to be associated with the identifiable object. So this is like an **OR** condition.

Alias ID rules

Add an alias ID rule

Alias identifiers store part numbers and other attribute information for similar parts. Alias IDs can be associated with many items or item revisions. You can write an *alias ID rule* to define the context when an alias ID can be applied to an item. You can also **create alternate IDs** to store information about the same part from different perspectives.

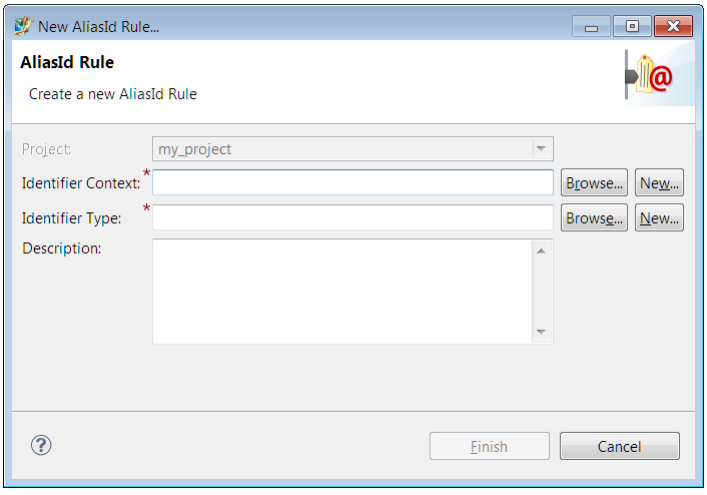
Before creating an alias ID rule, you must **create Identifier business objects** whose properties hold information such as a supplier's address and cost data. You also must **create Id Context options** that specify when the alias ID is used, such as for a supplier or a department.

1. Choose one of these methods:

   - On the menu bar, choose **BMIDE→New Model Element**, type **AliasId Rule** in the **Wizards** box, and click **Next**.

   - Open the **Extensions\Rules** folders, right-click the **AliasId Rules** folder, and choose **New AliasId Rule**.

   The New AliasId Rule wizard runs.



© 2025 Siemens

2. Enter the following information in the **AliasId Rule** dialog box:

   a. The **Project** box defaults to the already-selected project.

   b. Click the **Browse** button to the right of the **Identifier Context** box to select the context when the rule is to be applied.

      Click the **New** button to create a new **Id Context** option.

   c. Click the **Browse** button to the right of the **Identifier Type** box to select the **Identifier** or **IdentifierRev** business object that holds the ID information in its properties.

      Click the **New** button to create a new Identifier business object.

   d. In the **Description** box, type a description of the rule.

   e. Click **Finish**.

      The new rule appears under the **AliasId Rules** folder.

3. To save the changes to the data model, choose **BMIDE→Save Data Model**, or click the **Save Data Model** button on the main toolbar.

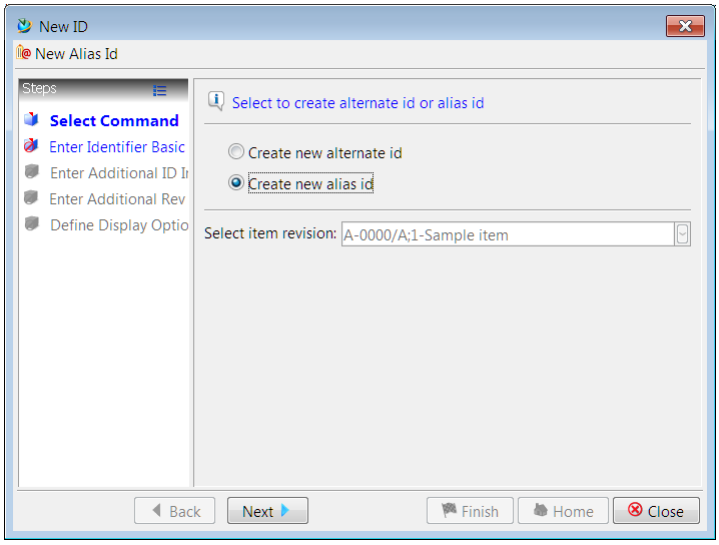
4. Deploy your changes to the server. Choose **BMIDE→Deploy Template** on the menu bar, or select the project and click the **Deploy Template** button on the main toolbar.

5. After deployment, test your new alias ID rule in the Teamcenter rich client:

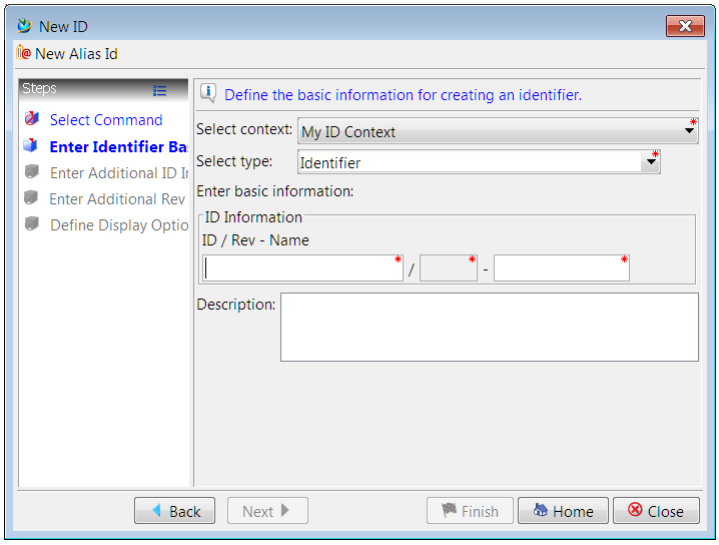
   a. In the My Teamcenter application, select an item you want to apply the rule to and choose **File→New→ID**.

      New Alias ID dialog box is displayed.

© 2025 Siemens



b. In the **New ID** dialog box, choose **Create new alias ID** and click **Next**.



c. Click the arrow in the **Select context** box to choose the ID context you defined on the rule, and click the arrow in the **Select type** box to select the identifier business object you defined on the rule.

d. Type the ID and name and click **Finish**.

Alias ID rules reference

To create an alias ID rule, open the **Extensions\Rules** folders, right-click the **AliasId Rules** folder, and choose **New AliasId Rule**.

© 2025 Siemens

To illustrate a rule for alias creation, consider that you have two defined identifier business objects, **ID1** and **ID2**; two defined contexts, **C1** and **C2**; and four identifiable objects, **Item1**, **Item2**, **ItemRevision1**, and **ItemRevision2**.

You can define two ID context rules that associate identifier business object **ID1** with context **C1** and identifier business object **ID2** with context **C2**. With these rules defined, you cannot create an alias using identifier business object **ID2** with context **C1** or an alias using identifier business object **ID1** with context **C2**.

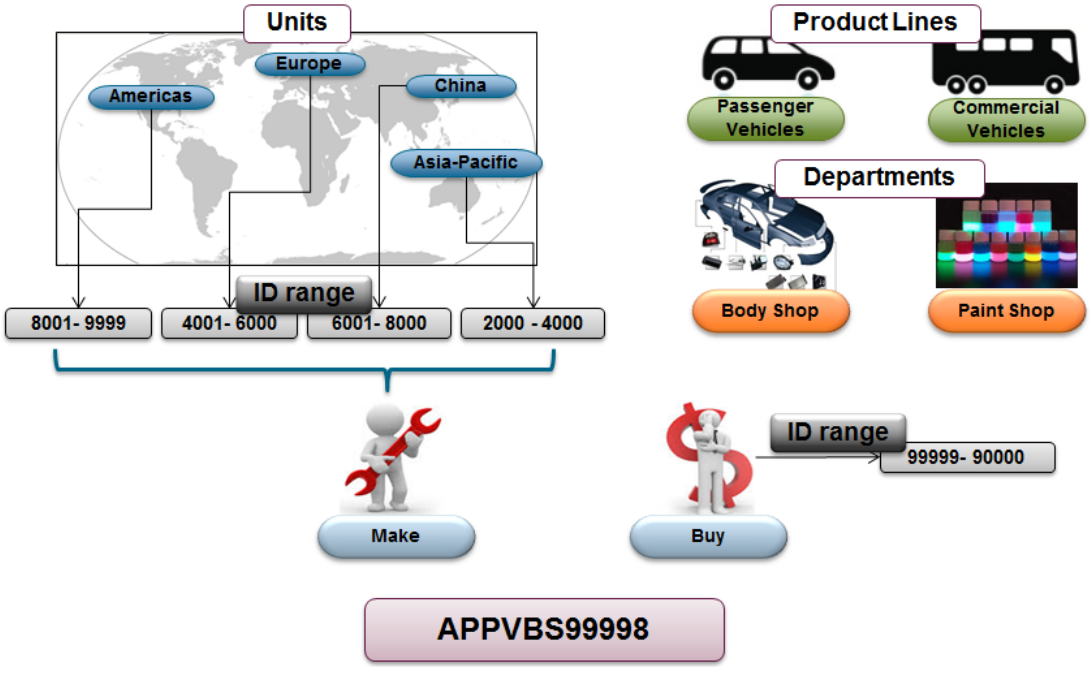
If you do not apply GRM rules, both business objects of alias identifiers, **ID1** and **ID2**, can have an alias relation to all four identifiable objects: **Item1**, **Item2**, **ItemRevision1**, and **ItemRevision2**. If, for example, you want to restrict alias relationships between **Item1** and **ID1**, you can use GRM rules to define the appropriate constraint.

Intelligent part numbering

Intelligent part numbering use case and example

The intelligent part numbering (IPN) system provides a configurable means of generating meaningful item IDs. This use case shows how IPN helps provide a solution to a vehicle manufacturer who wants the item IDs of parts to convey the part's associated unit, product line, and department, and whether the part was manufactured in-house or bought from a supplier.

The manufacturer has two product lines, **Commercial Vehicles** and **Passenger Vehicles**. The company has design centers and departments (such as **Body Shop** and **Paint Shop**) spread across business units on several continents.



© 2025 Siemens

The following table lists the properties and values to be represented in item IDs.

Property	LOV value	LOV display value	Counter range in item ID
Unit	**AP**	**Asia Pacific**	**2000–4000**
	EU	**Europe**	**4001–6000**
	CH	**China**	**6001–8000**
	AM	**Americas**	**8001–9999**
Product Line	**PV**	**Passenger Vehicles**	(Not applicable.)
	CV	**Commercial Vehicles**	(Not applicable.)
Department	**BS**	**Body Shop**	(Not applicable.)
	PS	**Paint Shop**	(Not applicable.)
Make/Buy	**Make**	**Make**	(Represented by **Unit** counter.)
	Buy	**Buy**	**99999–90000** (decrement)

Example:

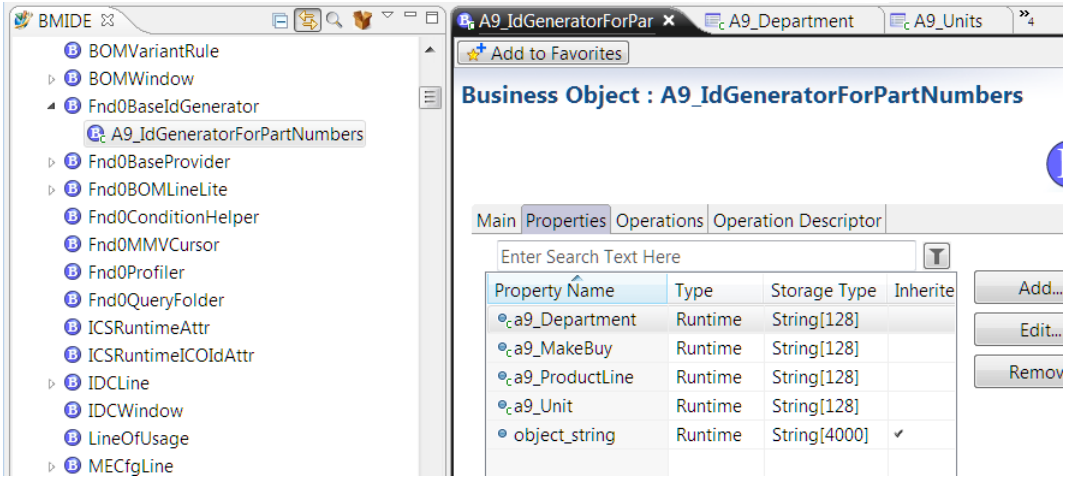
The following example item IDs show how the desired information is represented.

This item ID	Indicates this	
AMPVBS8001	**AM**	(Unit) Americas
	PV	(Product Line) Passenger Vehicles
	BS	(Department) Body Shop
	8001	(Make/Buy) The four-digit number indicates that the item is made rather than bought, and the number value is in the Americas (Unit) counter range
AMPVBS99999	The same information as for **AMPVBS8001**, except that the five-digit number within the buy (Make/Buy) unit range indicates the product is bought.	

The following example procedure shows how to set up intelligent part numbering for this use case. The example parallels the instructions in **Configure intelligent part numbering for a Create operation**.

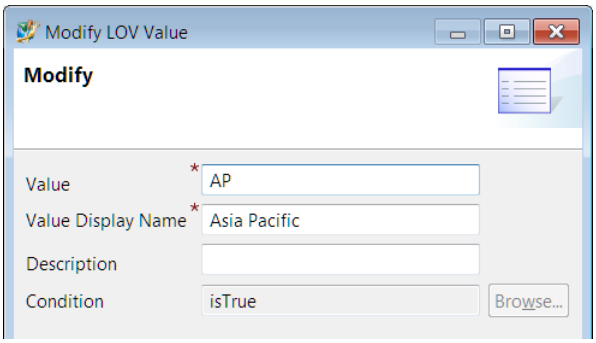
1. Create an ID generator business object and add properties for the unit, product line, department, and make/buy.

© 2025 Siemens

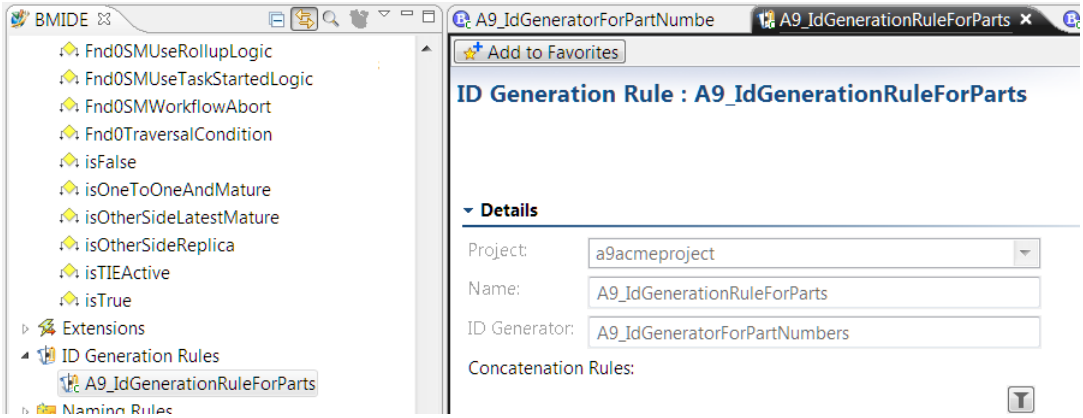


For each property (unit, product line, department, and make/buy), create a list of values (LOV) and attach the LOV to the property.

When you add values to the LOV lists, ensure that in **Value** you type the abbreviation. This is the value that is used in the generated ID.



2. Create an ID generation rule that references the ID generator business object you created.



The following table lists the counters and conditions for each concatenation rule to be added to the ID generation rule. Note that conditions are based only on the properties of the ID

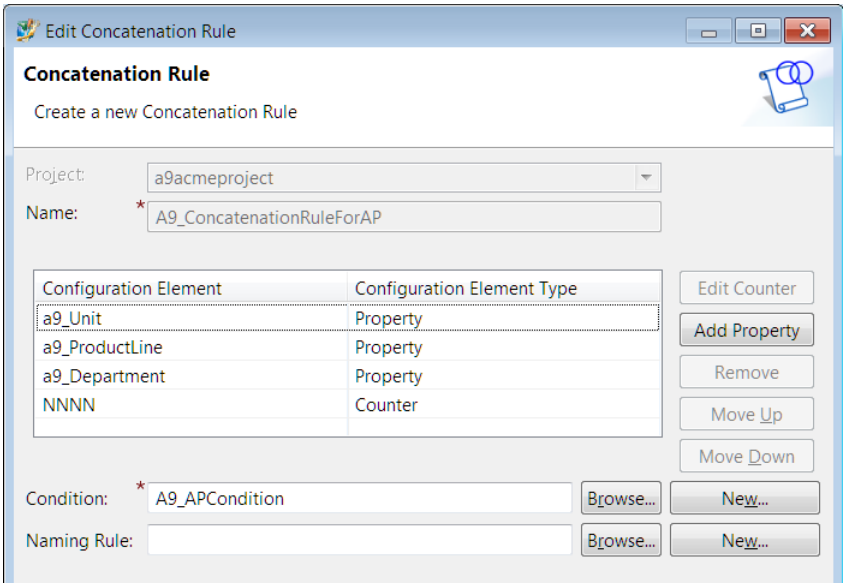
© 2025 Siemens

generator business object that is used in the ID generation rule, and that the conditions among the concatenation rules are mutually exclusive.

Concatenation rule	Counter pattern	Initial value	Maximum value	Condition
AP	**NNNN**	**2000**	**4000**	`o.a9_Unit="AP" and o.a9_MakeBuy="Make"`
EU	**NNNN**	**4001**	**6000**	`o.a9_Unit="EU" and o.a9_MakeBuy="Make"`
CH	**NNNN**	**6001**	**8000**	`o.a9_Unit="CH" and o.a9_MakeBuy="Make"`
AM	**NNNN**	**8001**	**9999**	`o.a9_Unit="AM" and o.a9_MakeBuy="Make"`
Buy	**NNNNN**	**99999**	**90000**	`o.a9_MakeBuy="Buy"`

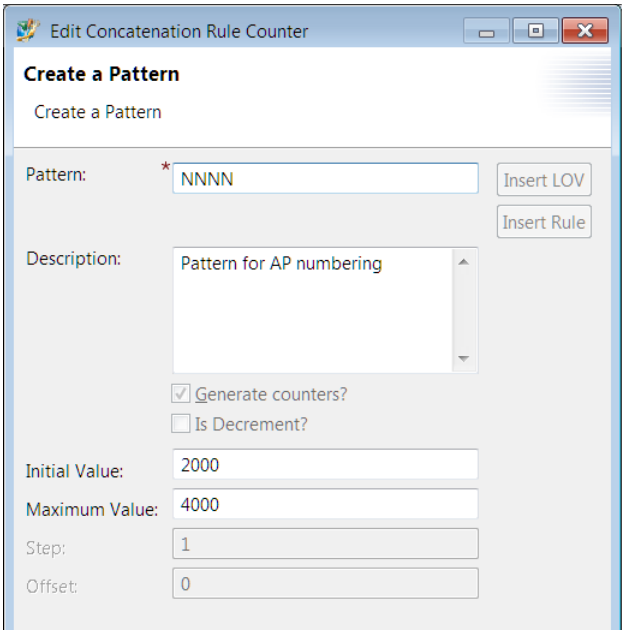
a. Add a concatenation rule for each unit (**AP**, **AM**, **CH**, **EU**) and the **Buy** value of the make/buy property.

In each concatenation rule, add the unit, product line, and department properties.



b. In each concatenation rule, add a counter with the number range for the business unit.

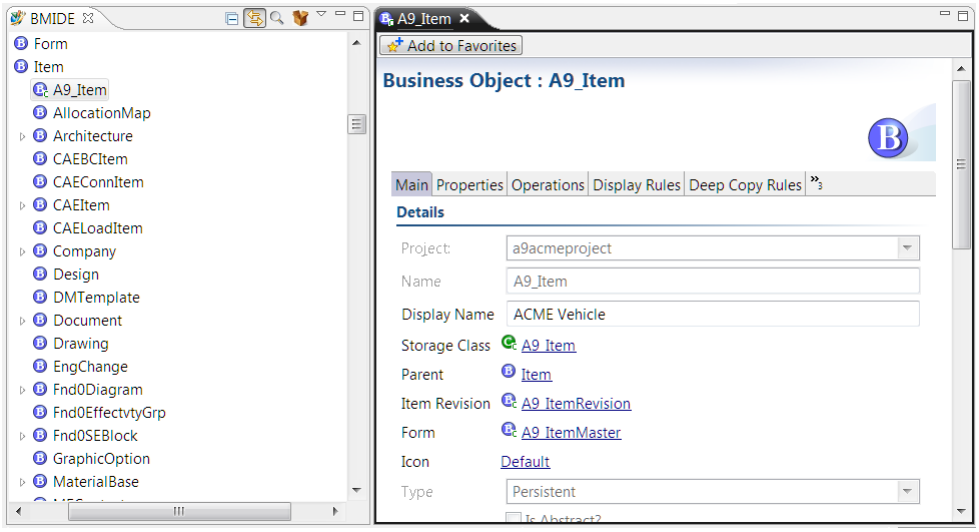
© 2025 Siemens



> Note:
>
> For the **Buy** concatenation rule number pattern, select the **Is Decrement?** check box.

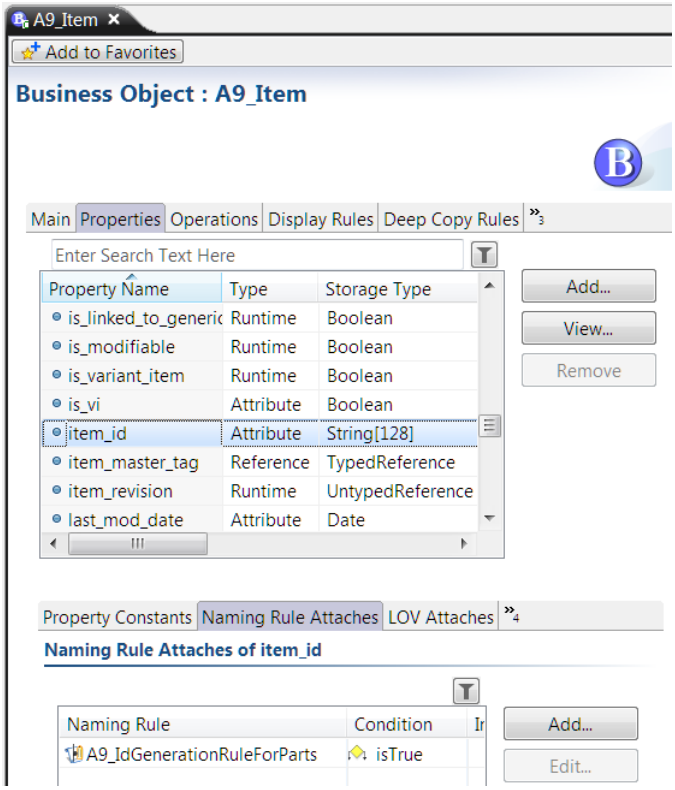
c. For each concatenation rule, next to **Condition**, click **New** and create a condition for using the concatenation rule.

3. Create a custom item business object for which you want to use intelligent part numbering.



To the custom item business object **item_id** property, attach the ID generation rule you created earlier (for this example, **A9_IdGenerationRuleForParts**).

© 2025 Siemens

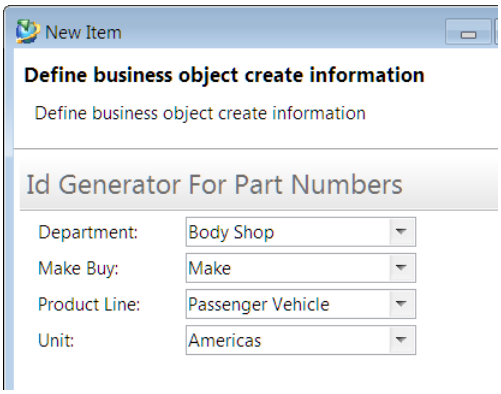


4. Save the project, package the template, and use Teamcenter Environment Manager (TEM) to install the template to Teamcenter.

5. To test intelligent part number for the object, create a new instance.

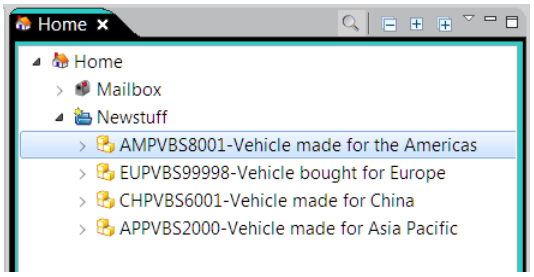
Example:

In the rich client, choose **BMIDE**>**New**>**Item** and choose the custom item business object. When you get to the **Id Generator for Part Numbers** panel, select the properties for the new item.



When you click **Finish**, the ID is generated, and the user-supplied **Name** value is appended to the concatenated item ID.

© 2025 Siemens



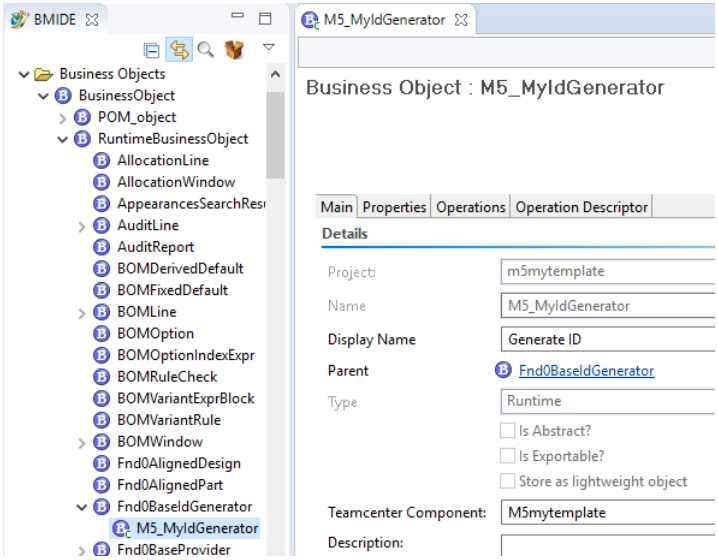
Configure intelligent part numbering for a Create operation

To configure a business object for intelligent part numbering during a create operation, do the following:

1. **Create a custom run-time ID generator business object as a source for concatenation properties**
2. **Create an ID generation rule**
3. **Attach the ID generation rule to the business object where you want to apply IPN**
4. **Create a custom stylesheet for your custom business object**
5. **Test intelligent part numbering for the configured business object**

Create a custom run-time ID generator business object as a source for concatenation properties

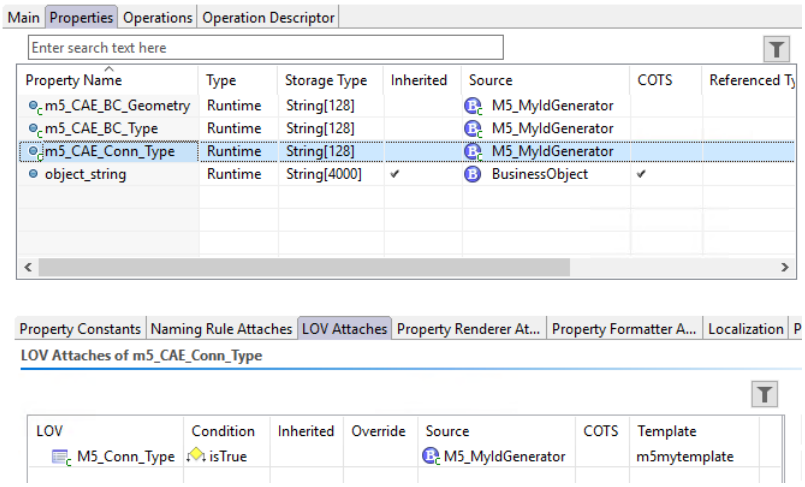
1. In the Business Modeler IDE, create a child of the **Fnd0BaseIdGenerator** runtime business object. Set the **Display Name** to **Generate ID**.



© 2025 Siemens

The display name should reflect the task the end user performs. This name appears as the title of the rich client properties panel for entering the values used in generating the item IDs.

2. Add properties to the custom ID generator business object for the values you want to be concatenated into the generated ID. Typically, you will want to attach lists of values (LOVs) to the properties.



Create an ID generation rule

1. In the BMIDE tree, expand the **Extensions**>**Rules** folder, then right-click the **ID Generation Rule** folder and choose **New ID Generation Rule**.

2. In the **ID Generation Rule** dialog box, specify options for the rule.

For this option	Do this
Name	Type the name you want to assign to the new ID generation rule in the database. The project prefix is automatically supplied.
ID Generator	Click the **Browse** button and select your custom ID generator business object.
Concatenation Rules	Add one or more mutually exclusive concatenation rules. a. In the **ID Generation Rule** dialog box, click **Add** to open the **Concatenation Rule** dialog box. b. In **Name**, enter a name for the concatenation rule. c. Add the counter or properties to be concatenated. • To add a counter pattern to the ID, click **Add Counter**.

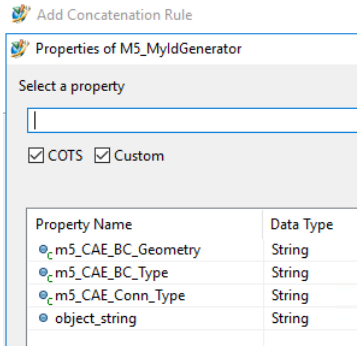
© 2025 Siemens

For this option	Do this

The **Create a Pattern** dialog box is displayed. This is the same dialog box used when you **create a naming rule**.

> Note:
>
> Using **dynamic characters** in a pattern while defining a counter in a concatenation rule is not supported.

- To add a property to be concatenated into the ID, click **Add Property**. The available properties are the ones on the ID generator business object selected for the ID generation rule.



d. Specify a **Condition** for applying the concatenation rule. Make sure that the condition is based only on the properties of the ID generator business object that is used in the ID generation rule.

At run time, the system evaluates the condition associated with each concatenation rule. If a concatenation condition evaluates to true, then the concatenation rule is used to generate the item ID.

> Caution:
>
> The conditions within concatenation rules for an ID generation rule must be mutually exclusive. If two or more conditions evaluate to true, then at run time any of these concatenation rules are randomly picked up for ID generation. In the worst case, instances of the same business object type with different item ID formats could result.
>
> For an example of correct use of multiple concatenation rules with exclusive conditions, see **Intelligent part numbering use case and example**.

e. (Optional) Specify a **naming rule** that validates the **item_id** value for cases where a value for **item_id** is provided.

© 2025 Siemens

For this option	Do this
	If the value of **item_id** is not provided either by manual entry or by importing from a list, then there is no validation against the concatenation rule naming rule.

Tip:

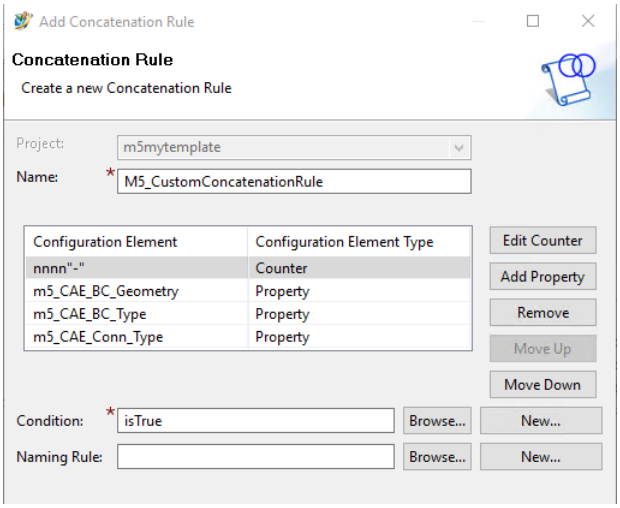
Specify a naming rule with LOVs associated to properties defined in the concatenation rule in the same sequence as the properties in the concatenation rule. This ensures that generation and validation are always in sync.

Providing a naming rule along with a concatenation rule is optional, but it is a best practice to provide a meaningful naming rule that matches the corresponding concatenation rule along with each and every concatenation rule to validate a provided **item_id** value when it is either entered manually by the user or included in the list when importing bulk IDs.

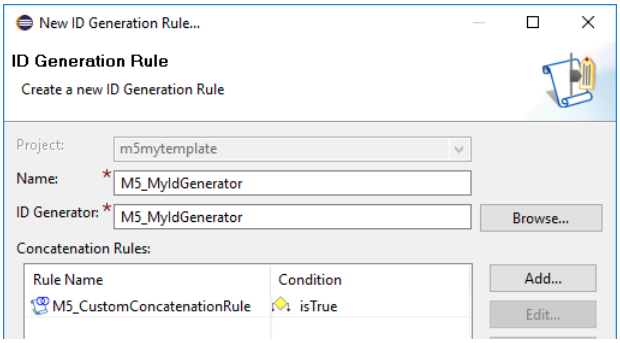
In this incoming item_id value case	This is the effect
The value tests valid against any of the concatenation naming rules within the ID generation rule.	The incoming **item_id** value is accepted.
The value tests invalid against the naming rule configured within the concatenation rule.	If any naming rule is attached to the item type **item_id** property, then the **item_id** value is validated against the naming rule attached to the **item_id** property; otherwise an error message is displayed and the item ID is not generated.
No naming rule is specified in the concatenation rule.	If any naming rule is attached to the item type, then the incoming **item_id** value is validated against the naming rule attached to the **item_id** property; otherwise, the **item_id** value is accepted.

Click **Finish** to complete the concatenation rule.

© 2025 Siemens



3. In the **ID Generation Rule** dialog box, click **Finish** to complete the rule.



Attach the ID generation rule to the business object where you want to apply IPN

1. Attach the ID generation rule to the **item_id** property of an item business object as you would **attach a naming rule**.



> Note:
>
> If both an ID generation rule and a naming rule are associated with an item business object, the ID generation rule always takes precedence.

2. Save changes to the data model and deploy your changes to the server.

© 2025 Siemens

Create a custom stylesheet for your custom business object

By default, the properties on the custom ID generator business object are not displayed in the item creation wizard.

To display the properties during the operation, you must create a custom style sheet for your custom business object based on the item's creation style sheet (such as **ItemCreate.xml**) and edit the property display. For example, you may want to place the properties in a different order or place them on the first item creation page.

Following is tagging that adds the properties in our example to the creation style sheet:

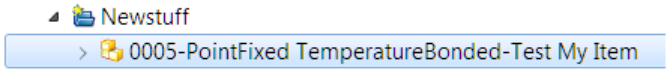
```
<section title="IdGenerator Configuration" titleKey="tc_xrt_IdGenInfo">
   <property name="fnd0IdGenerator:m5_CAE_BC_Geometry"/>
   <property name="fnd0IdGenerator:m5_CAE_BC_Type"/>
   <property name="fnd0IdGenerator:m5_CAE_Conn_Type"/>
</section>
```

Test intelligent part numbering for the configured business object

To test intelligent part numbering for the configured business object (the object to which you attached the ID generation rule), create a new instance of the business object.

> Note:
>
> Do not click **Assign** for the **ID** property. If you click **Assign**, then an error is displayed because ID generation rules are enabled for this business object type.

When you finish specifying the new instance, the ID is generated and the object instance is created. Note that the user-supplied **Name** value is appended to the concatenated item ID.



Configure intelligent part numbering for a Save As operation

Configuring *Intelligent Part Numbering* (IPN) functionality for an object **Save As** operation requires configuring the object as in **Configure intelligent part numbering for a Create operation**, plus some additional configuration.

1. **Set IPN SaveAs preference = True**
2. **Enable IPN on the business object**
3. **Modify the Save As stylesheet for the object**

© 2025 Siemens

Set IPN SaveAs preference = True

To enable IPN **Save As** functionality for the site, do the following one time.

1. Set the **PREF_ITEMSAVEAS_USING_IDGENERATION** preference to **True** and then restart the Teamcenter server and client.

Enable IPN on the business object

For the business object, perform the steps described in **Configure intelligent part numbering for a Create operation**.

Modify the Save As stylesheet for the object

By default, the properties on the custom ID generator business object are not displayed in the item save as wizard.

To display the properties during the operation, you must create a custom style sheet for your custom business object based on the item's save as style sheet (such as **ItemSaveAsdataset** or **Awp0ItemRevSaveAsdataset**) and edit the property display. For example, you may want to place the properties in a different order or place them on the first item creation page.

The required tagging differs depending on whether the stylesheet is for rich client or for Active Workspace.

For this Teamcenter client	Use this pattern

rich client	`<property name = "Idx0<BMIDE property name of your custom ID generation business object>"/>`
	
	Following is example tagging that adds the properties to the rich client save as style sheet:
	
	Example:
	
	In the following example for a rich client style sheet, Department, MakeBuy, ProductLine, and Unit are the custom properties, and a9 is the project ID on the fnd0IdGeneratorsubtype that you created.
	
	`<section title="IdGenerator Configuration" titleKey="tc_xrt_IdGenInfo">`
	`<property name="Idx0a9Department"/>`
	`<property name="Idx0a9MakeBuy"/>`
	`<property name="Idx0a9ProductLine"/>`

© 2025 Siemens

For this Teamcenter client	Use this pattern
	<property name="Idx0a9Unit"/> </section>
Active Workspace	Ensure that the **Stylesheet Type** is set to **Property**. <property name = "items_tag:Idx0<BMIDE property name of your custom ID generation business object>"/> Example: In the following example, Department, MakeBuy, ProductLine, and Unit are the custom properties, and a9 is the project ID on the fnd0IdGeneratorsubtype that you created. <section title="IdGenerator Configuration" titleKey="tc_xrt_IdGenInfo"> <property name="items_tag:Idx0a9Department"/> <property name="items_tag:Idx0a9MakeBuy"/> <property name="items_tag:Idx0a9ProductLine"/> <property name="items_tag:Idx0a9Unit"/> </section>

Multifield keys

Introduction to multifield keys

Multifield keys are identifiers assigned to each object to ensure their uniqueness in the database. Administrators use the **MultiFieldKey** business object constant to assign the key definitions to different business object types. Administrators can add multiple properties to define a key.

The multifield key is composed of a *domain* name and a combination of the object's properties:

*domain***{***properties***}**

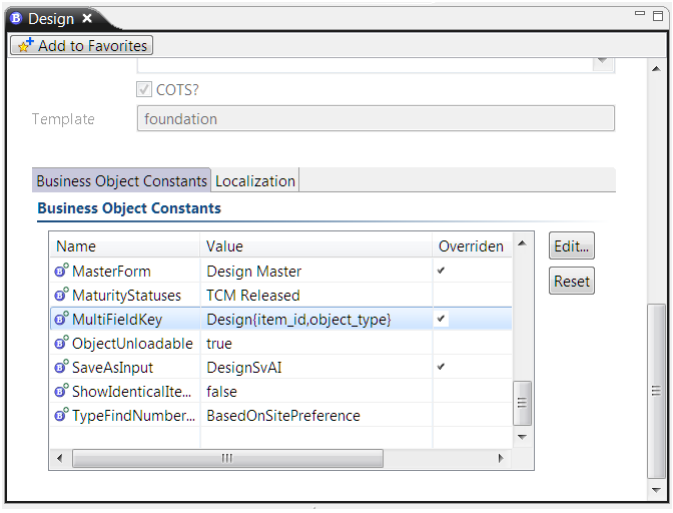
For example, the default multifield key definition for **Item** business objects is **Item{item_id}**. Because children business object types inherit the key definition from their parent, they belong to the same domain as the parent business object.

You can configure multifield keys that allow end users to create multiple related items using the same item ID. For example, suppose end users want to refer to a part and drawing using the same item ID. You can do this by setting up multifield key definitions per domain or object type.

© 2025 Siemens

For example, in the case of part and drawing, the administrator could define the unique key for part business objects and their children as **Part{item_id}** and for drawing business objects and their children as **Drawing{item_id}**. When these definitions are applied, it results in a unique key identifier for each instance of an object type in the database, even though the different object types can share the same item ID. (You could also provide a unique key by adding a property in addition to the **item_id** property, such as **object_type**. However, if you want the **object_type** property to appear in the item name, you must use the **DisplayName** business object constant.)

In the following example, the value for the **MultiFieldKey** business object constant is changed from the default setting of the **item_id** property to also include the **object_type** property:



Note:

The uniqueness of item revisions is not managed with multifield keys. Instead, the revision ID, sequence ID, and item tag (UID of the item) are used.

Creating multifield key definitions

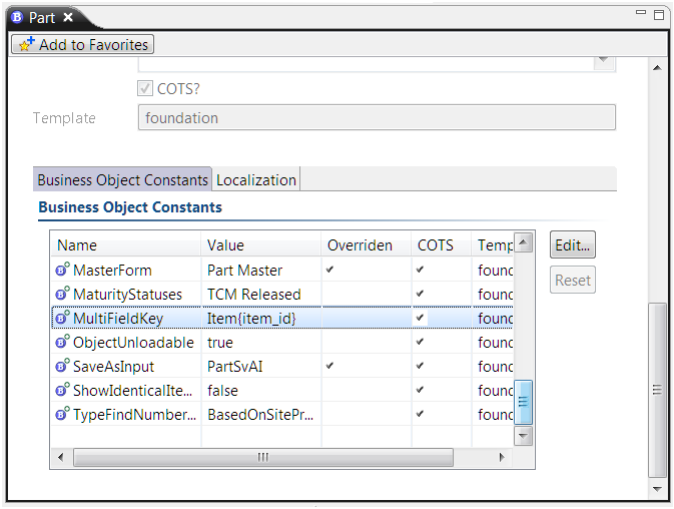
Create a business object's unique key definition using the **MultiFieldKey** business object constant. The key definition is inherited by the child business object types unless a different multifield key definition is created for a child type.

1. Open any business object that is a child of the **POM_Object** business object. For example, open a custom item business object.

2. Select the **MultiFieldKey** constant in the **Business Object Constants** table.

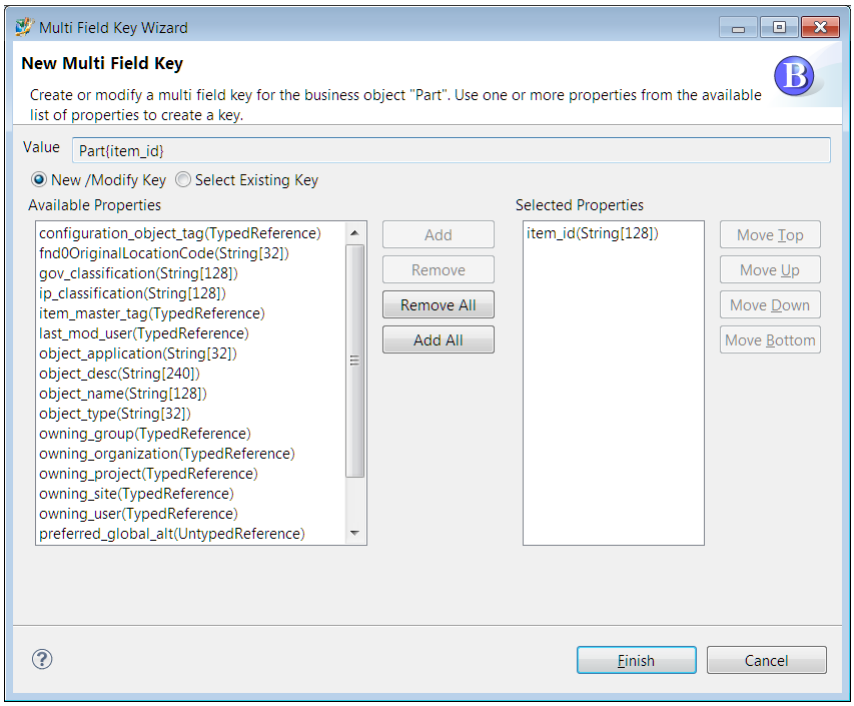
   In the following example, note that the key definition for the **Part** business object is **Item{item_id}**. That's because the **Part** business object inherits the definition from its parent business object, and therefore is in the **Item** business object domain.

© 2025 Siemens



3. Click the **Edit** button to the right of the table.

   The **New Multi Field Key** dialog box is displayed.



4. Create the multifield key:

   - Create a new key

     Select properties in the **Available Properties** box and click the **Add** button.

     The added properties are displayed in the **Selected Properties** box.

© 2025 Siemens

Note:

The **Available Properties** box displays only the properties that are available on the business object and are supported for multifield keys. Multifield keys are supported only for properties with a single value string and reference type properties. Multivalued properties are not supported.

- Use an existing key

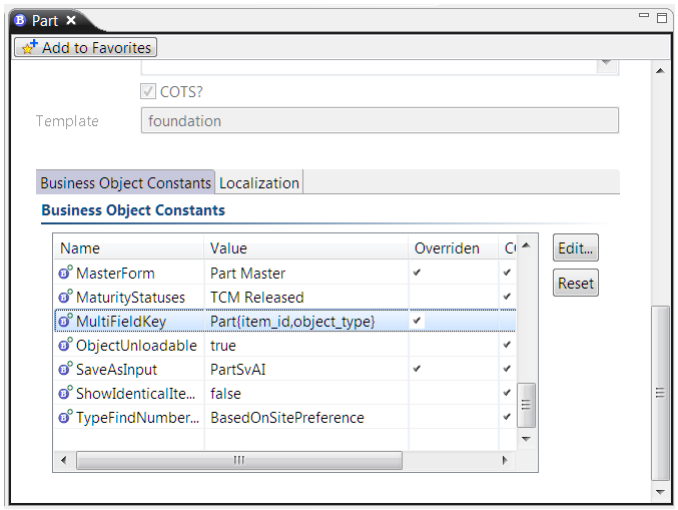
  Select **Select Existing Key** to choose from a list of key definitions in the project that are applicable to the business object receiving the key. This list is empty if no keys are applicable to the business object receiving the key.

  Selecting an existing key means you are adding the business object type to the key *domain*, which is a group of business object types that share the same key definition.

5. Click **Finish**.

The new key definition is displayed.

In the following example, the **MultiFieldKey** value for the **Part** business object has been changed from the default definition of **Item{item_id}** to **Part{item_id,object_type}**. Because all the children of the **Part** business object inherit this definition, it is a key domain for all part types.



Note:

One benefit of using the **item_id** property, but differentiating it by business object type (domain), is that instances of different business object types (items, drawings, parts, documents, and so on) can **share the same item ID** but have unique keys.

© 2025 Siemens

6. If you add properties to a business object multifield key definition, ensure that the same properties are **added to the creation dialog boxes** for the business object type. Because these properties are used to determine object uniqueness, the end user must enter values to the properties if they are to contribute the object's uniqueness. If the value of a property used for a multifield key is empty, that property cannot contribute to the object's uniqueness identification. You can make these properties required on the creation dialog boxes, if desired.

7. Once the keys are installed, keep in mind that they are not visible to the end user in the name of the business object. If you want to use some of the properties in the multifield definition to **define how the name of the business object instances are displayed**, configure the **DisplayName** business object constant.

8. Deploy your custom template to the test server database by using Teamcenter Environment Manager (TEM).

   > Caution:
   >
   > You cannot use **BMIDE**>**Deploy Template** to deploy multifield keys to a test database.

9. To verify that the keys are property defined in the system, use the **get_key_definition** and **get_key_string** utilities.

   Before installing the new multifield key definitions to a production server, you must ensure they do not cause key collisions resulting from identical keys. **Analyze the key definitions** by running the **mfk_update** utility.

10. After verifying that the keys work as designed, you can deploy them a production server by packaging the custom template and installing it using Teamcenter Environment Manager (TEM).

    After deployment, the key is set for this business object type and is inherited by all its children types, unless a separate key is set on a child type.

Multifield key domains

The multifield key is composed of a *domain* name and a combination of the object's properties:

*domain***{***properties***}**

For example, the default multifield key definition for **Item** business objects is **Item{item_id}**.

The key definition is inherited by the child business object types unless a different multifield key definition is created for a child type. Because children business object types inherit the key definition from their parent, they belong to the same domain as the parent business object. All business object types with the same multifield definition constitute a domain. To make a business object a member of an existing key domain, select an existing key when you change the **MultiFieldKey** business object constant value on the business object. Establishing domains allows administrators to identify the object

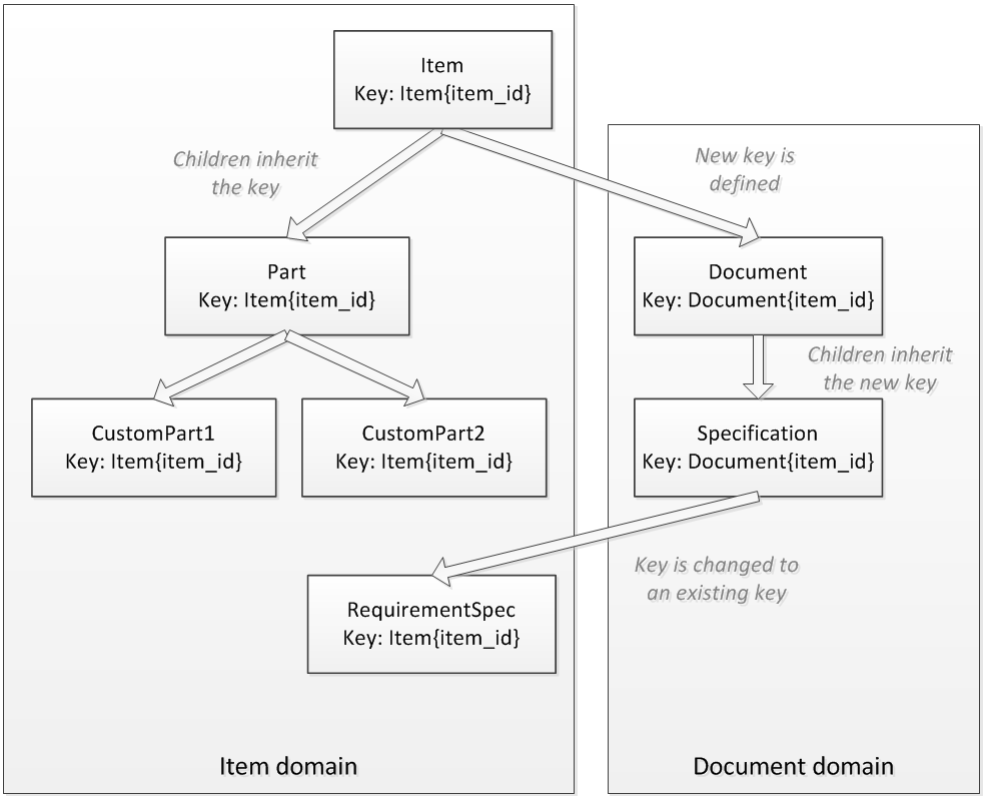
© 2025 Siemens

types that have the same multifield key. This ensures a unique identifier across all the objects in the domain.

> Note:
>
> Multifield key definitions can only be used if all properties in the definition are available on the business objects. You can analyze key definitions using the **mfk_update** utility.

In the following example, a new key is defined on the **Document** business object, creating a new domain. All children of the **Document** business object inherit the new key, and therefore belong to the new domain. Custom business objects also inherit the key from their parents. If you select an existing key for a business object, the business object becomes a member of the existing key's domain.

In the example, the **RequirementSpec** business object is moved to the **Item** domain because the administrator wanted the **item_id** property value for requirements specifications to be unique among all items rather than among all documents.



Creating objects with the same item ID

Through multifield key definitions per domain or object type, instances of different item business object types (such as items, drawings, documents, parts, designs, and specifications) can use the same item ID.

For example, suppose you have a document and a drawing that both describe an item, and you want to give the item, drawing, and document objects the same item ID number so that others can see at

© 2025 Siemens

a glance that they all describe different aspects of the same thing. Using multifield keys, you can give different item types the same **item_id** value by assigning different domains (business object types) to the key definitions.

In this example, you can use the **Item** business object's default multifield key definition (**Item{item_id}**), but change the multifield key definition for **Drawing** business objects to **Drawing{item_id}** and for **Document** business objects to **Document{item_id}**.

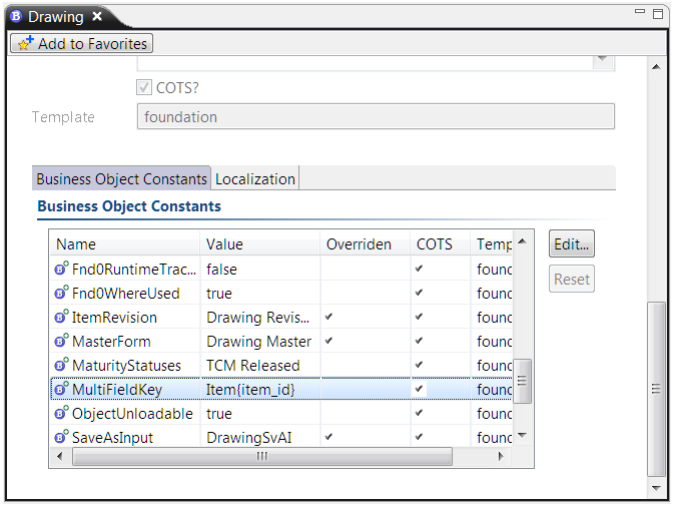
When the keys are installed to the Teamcenter database, end users are allowed to create instances of these different object types with the same item ID because the key definitions each have a different domain (**Item**, **Drawing**, and **Document**).

> Note:
>
> Another way you can use the same item ID is to add the **object_type** property to the key definition used by all the business objects, for example, **Item{item_id,object_type}**. Whereas the domain approach (for example, creating **Drawing{item_id}**) forces uniqueness of the **item_id** value across the **Drawing** domain, if instead the **object_type** property is added at the **Item** domain (for example **Item{item_id,object_type}**), the **item_id** property is unique only for each object type. (Keep in mind that if you want the **object_type** property to display in the item name, you must use the **DisplayName** business object constant.)
>
> Similarly, if you have a situation where you want to allow different parts from different vendors to use the same item ID, you can add the property for the vendor code or the CAGE code to the key definition that already includes the **item_id** property.

1. Open the **Drawing** business object.

2. Select the **MultiFieldKey** constant in the **Business Object Constants** table.

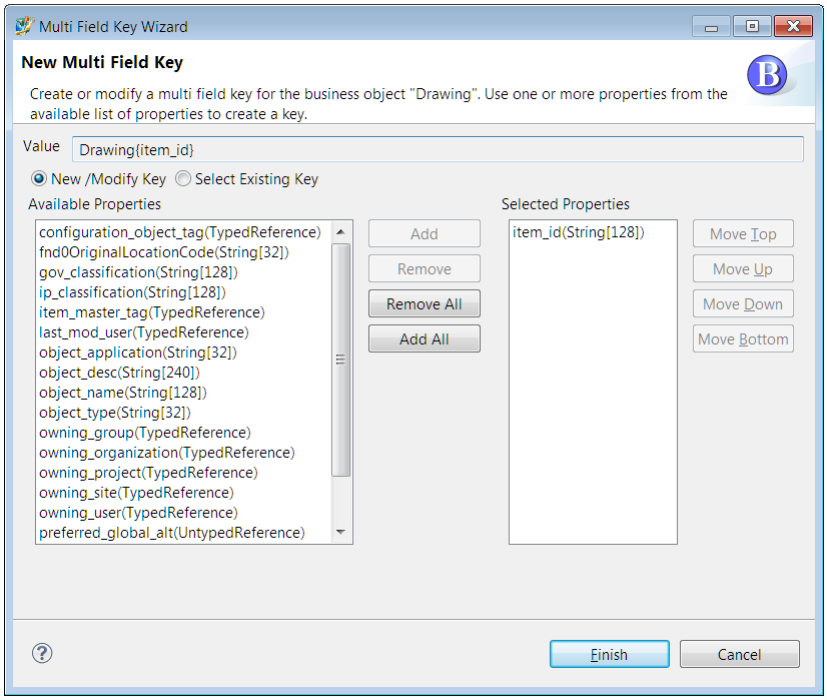
   In the following example, notice how the key definition for the **Drawing** business object is **Item{item_id}**. That's because the **Drawing** business object inherits the definition from its parent business object, and therefore is in the **Item** business object domain.



© 2025 Siemens

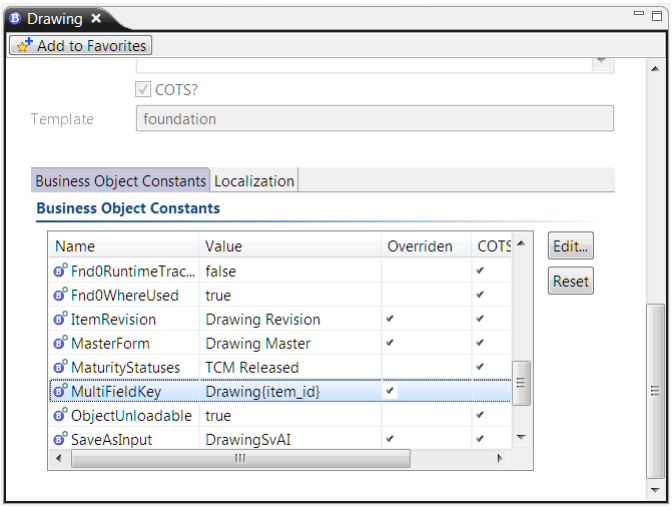
3. Click the **Edit** button to the right of the table.

   The **New Multi Field Key** dialog box is displayed.



4. Click **Finish**.

   This changes the multifield key definition for **Drawing** business objects from **Item{item_id}** to **Drawing{item_id}**.



5. Repeat the same steps for the **Document** business object so that its key definition is changed to **Document{item_id}**.

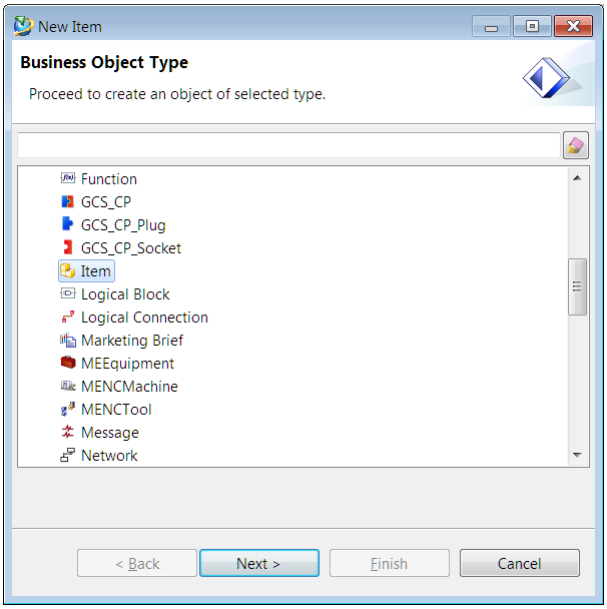
© 2025 Siemens

6. **Package the template** and **install your custom template to the test server database by using Teamcenter Environment Manager (TEM)**.

> Caution:
>
> You cannot use **BMIDE→Deploy Template** to deploy multifield keys to a test database.

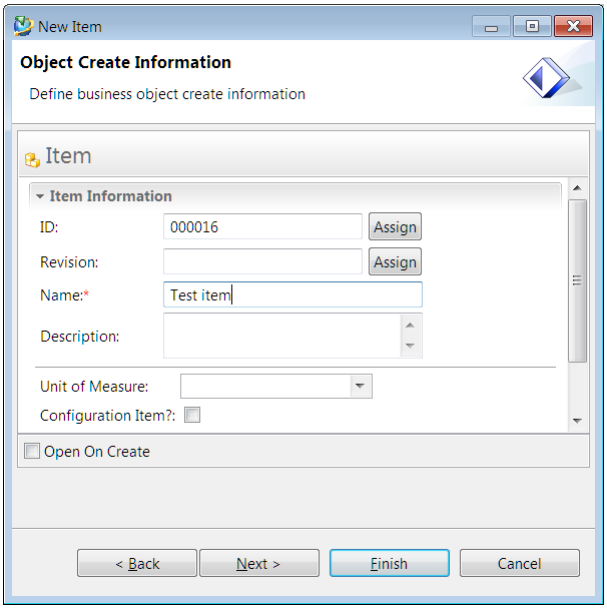
7. To verify the behavior, create an item, drawing, and document that all have the same item ID:

   a. In the rich client, choose **File→New→Item**, select **Item** from the list, and click **Next**.



   b. Type the item ID number and do not click the **Assign** button.

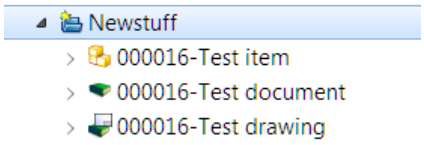
© 2025 Siemens



c. Click **Finish**.

d. Repeat the steps for the drawing and document types, making sure to type the same item ID number for each.

When you are done, you have object instances of different business object types that each have the same item ID.

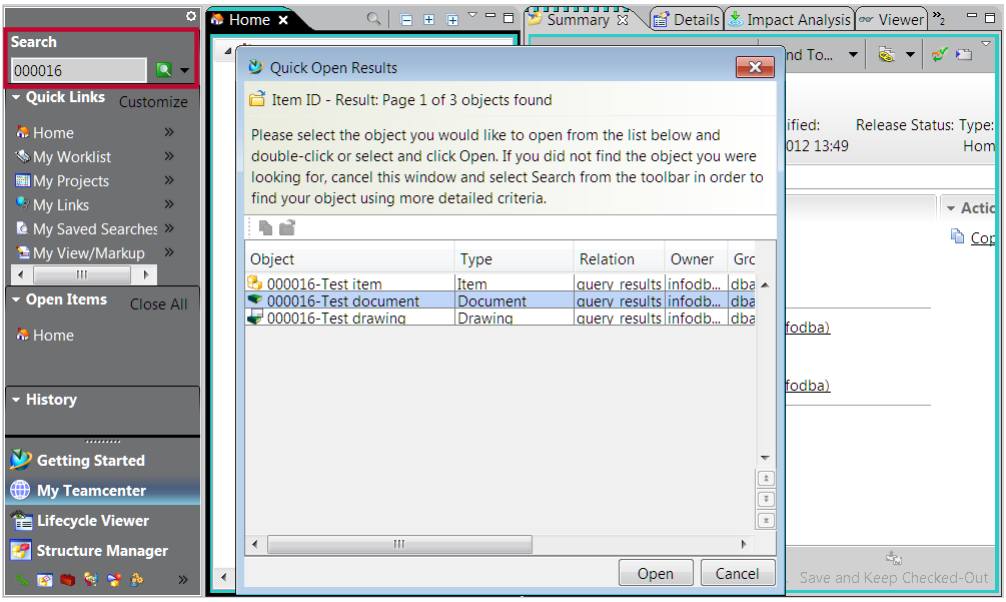


The system allows you to do this because although the key definitions for the item, drawing, and document business object types each use the **item_id** property, they each have a different domain (**Item**, **Drawing**, and **Document** respectively). If you try to create two items with the same item ID (or two documents or two drawings), the system displays an error.

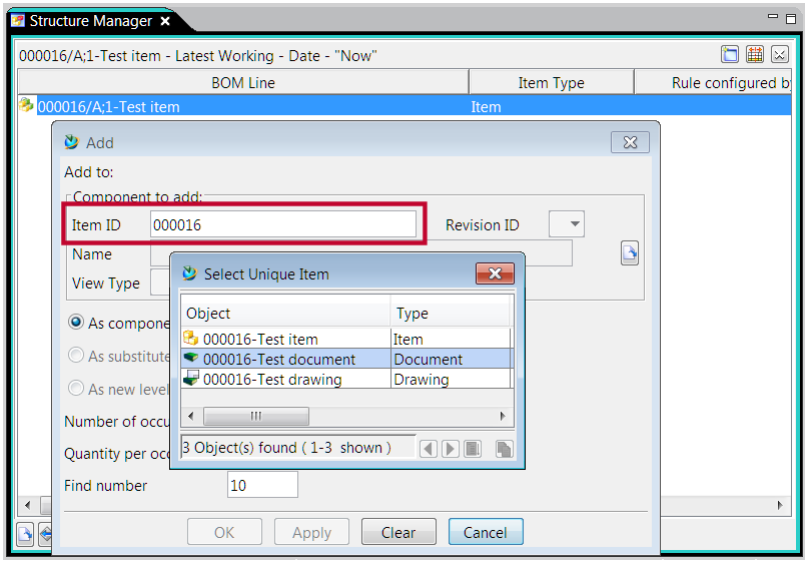
e. To further test the behavior, search for the item ID that is used by multiple objects. A dialog box appears allowing you to choose the desired object.

The following example shows searching in the rich client for an item ID used by multiple objects.

© 2025 Siemens



Similar dialogs are presented in Structure Manager when searching for an item ID that is used by multiple objects. For example, if you choose **Edit→Add** in Structure Manager and type an item ID used by multiple objects, a selection dialog box is displayed.



Managing multifield keys

Use the following settings to manage multifield keys:

- Global constants
  - **Fnd0MultiFieldKeyExclusions**

© 2025 Siemens

Excludes selected business object types from having their multifield key definitions changed. Business object types that already that have a multifield key defined cannot be added to this exclusion list. This constant prevents setting a multifield key by disabling the **Edit** button on the **MultiFieldKey** business object constant.

- **Fnd0SecuredMultiFieldKey**

  Ensures that dependent templates do not change the **MultiFieldKey** business object constant on specified business objects. If the administrator of a dependent template tries to change the **MultiFieldKey** business object constant, the administrator cannot because the **Edit** button is disabled. (The **Edit** button is still enabled on the current template.)

- Business object constants

  - **ShowIdenticalItemIdAndName**

    As defined by the **DisplayName** business object constant, the default display name for business objects comprises the **item_id+object_name** properties. If the values of these two properties are identical, and the **ShowIdenticalItemIDAndName** business object constant is set to its default of **false**, the **object_name** value is dropped from the displayed string. If you want to allow identical item ID and name values to be displayed for an item instance, set the value to **true**.

- Preferences

  - **TC_MFK_DEFAULT_DOMAIN**

    Defines the domain used by the **ITEM_find_item** ITK function when the multifield key unique identifier is not used.

Analyzing multifield keys

After you create multifield key definitions, you must ensure they do not cause key collisions resulting from identical keys. Use the following utilities to analyze the multifield keys in the database:

- **mfk_update**

  Updates multifield key definitions in the key table in the database. Normally, during upgrade the system runs this utility so that business object instances on the server are migrated to the new multifield key definitions. As an administrator, you can also use this utility to:

  - Evaluate proposed multifield key definitions in a template before installing the template to the production server. Specifically, you can determine whether a multifield key is deployable or not (that is, unique across business object instances in the particular domain, all properties found and of the right type, and so on).

© 2025 Siemens

The utility reports potential collisions (duplicate multifield key values) and the quantity of collision instances.

- Analyze the multifield key definitions on the server, and if there are corrupt or inconsistent key definitions, rebuild the key table on the database.
- Rebuild all multifield key values in the system, thereby guaranteeing correctness.

For example, to perform an analysis of proposed keys in a custom template:

```
mfk_update -u=tc_admin_1 -p=pwd -g=dba
-check -file=C:\delta.xml -log=C:\mfk_check_template.log
```

To perform an analysis of the keys in the database:

```
mfk_update -u=tc_admin_1 -p=pwd -g=dba -check
-log=C:\mfk_check_database.log
```

To rebuild the key table for all multifield key definitions in the database:

```
mfk_update -u=tc_admin_1 -p=pwd -g=dba -rebuild -log=C:\mfk_rebuild.log
```

> Note:
>
> Because keys are a string of concatenated property values, they can be quite large. To ensure that you have enough space in the key table for long keys, you can set the **TC_MFK_INDEX_KEY_SIZE** environment variable to specify the byte size limit of the index key size when creating a key in **POM_KEY** table. The default value is **900**.

- **get_key_definition**

  Gets the multifield key definition for a business object type.

  At a command prompt, type:

  ```
  get_key_definition -class=business-object-name
  ```

  The results are displayed as:

  ```
  Key Definition for business-object-name is multifieldkey-definition
  ```

  For example, typing:

  ```
  get_key_definition -class=T5_MyItem
  ```

  results in:

© 2025 Siemens

```
Key Definition for T5_MyItem is T5_MyItem{item_id,object_type}
```

- **get_key_string**

  Gets the multifield key value of an item instance as a string containing property names and values. This utility can be run on the **Item** business object or any of its children. Use the **-item** or **-key** argument:

  - **-item**

    Finds the key values for the item. Type the command using the following format:

    ```
    get_key_string -item=item-ID
    ```

    The results are displayed as:

    ```
    Item Type is business-object-name, Key String is property1=property1-value,property2=property2-value,etc
    ```

    For example, typing:

    ```
    get_key_string -item=000016
    ```

    results in:

    ```
    Item Type is Item, Key String is item_id=000016
    ```

    If other properties are added to the multifield key definition for an item business object, they are displayed separated by commas. For example, if the **item_id** and **object_type** properties comprise the multifield key for a custom item business object named **T5_MyItem**, the output of the command is:

    ```
    Item Type is T5_MyItem, Key String is
    item_id=000016,object_type=T5_MyItem
    ```

  - **-key**

    Finds the business object type of the item. This is not a utility, but an argument used with utilities.

    Follow this format to use the argument with a utility:

    ```
    get_key_string -key=property=property-value
    ```

    The results are displayed as:

    ```
    Item Type is business-object-name, Key String is property=property-value
    ```


© 2025 Siemens

For example, typing:

```
get_key_string -key=item_id=000016
```

results in:

```
Item Type is Item, Key String is item_id=000016
```

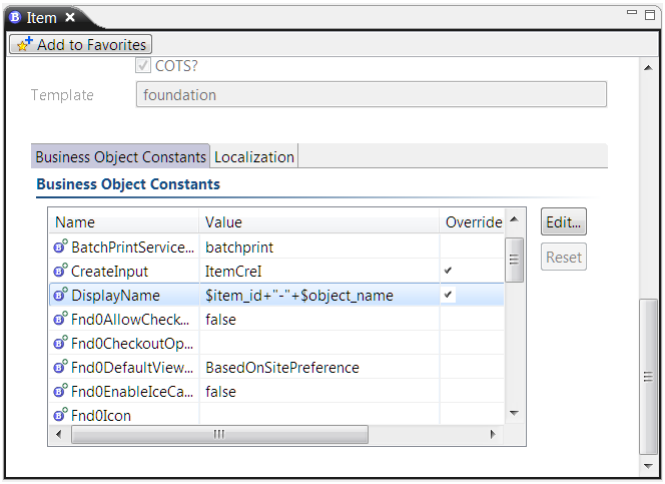
Configure the displayed name of business object instances

Multifield keys are not visible to the end user and only identify object instances in the database. If you want to select the properties to display item instance names in the user interface, use the **DisplayName** business object constant. This constant provides the value used for the **object_string** property, which displays names in the **Object** box on property dialog boxes and in other places in the user interface.

> Note:
>
> The **DisplayName** business object constant should not be confused with **naming rules** or the localized **display name** of business objects.

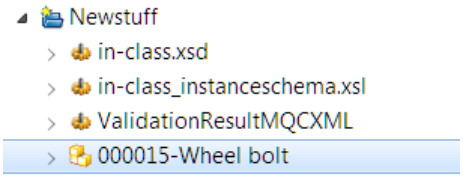
1. Open any business object that is a child of the **WorkspaceObject**, **RevisionAnchor**, or **Fnd0AuditLog** business object and select **DisplayName** on the **Business Object Constants** table.

   For example, open the **Item** business object and select the **DisplayName** business object constant:

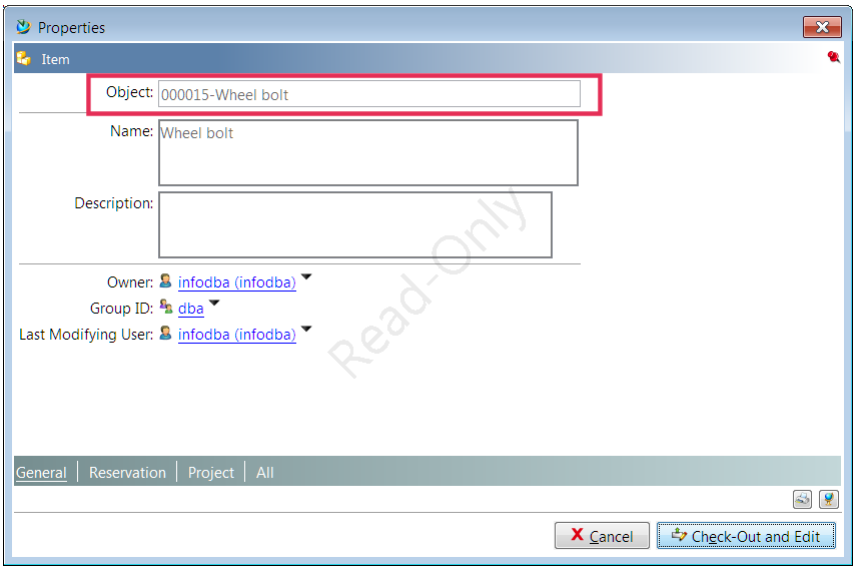


   By default, the constant is set to **$item_id+"-"+$object_name** for **Item** business objects. This displays the item name in the user interface.

© 2025 Siemens

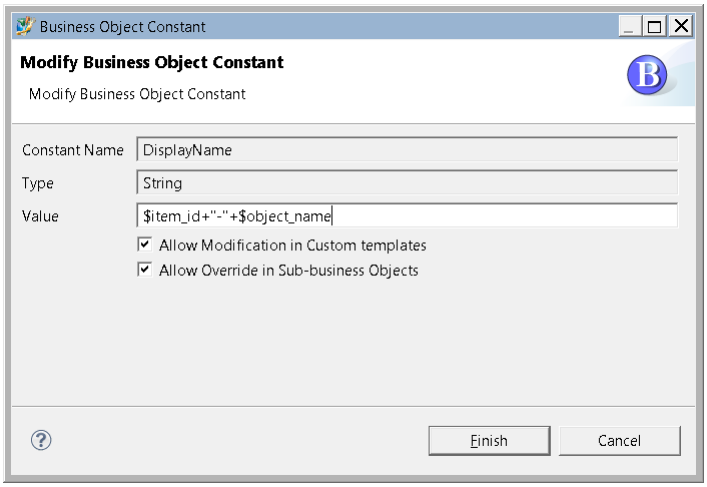


You can also see the displayed name in the **Object** box (provided by the **object_string** property) on dialog boxes and views.



2. To change the constant, select the **DisplayName** constant in the **Business Object Constants** table and click the **Edit** button to the right of the table.

   The **Modify Business Object Constant** dialog box is displayed.

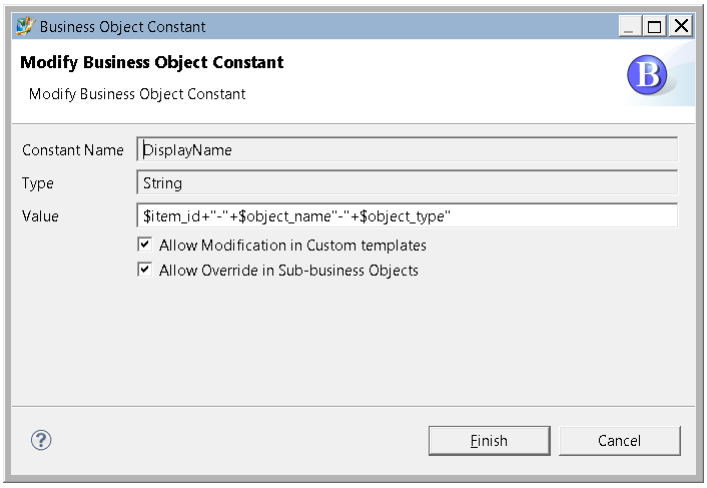


3. In the **Value** box, type the properties you want to display using this format:

© 2025 Siemens

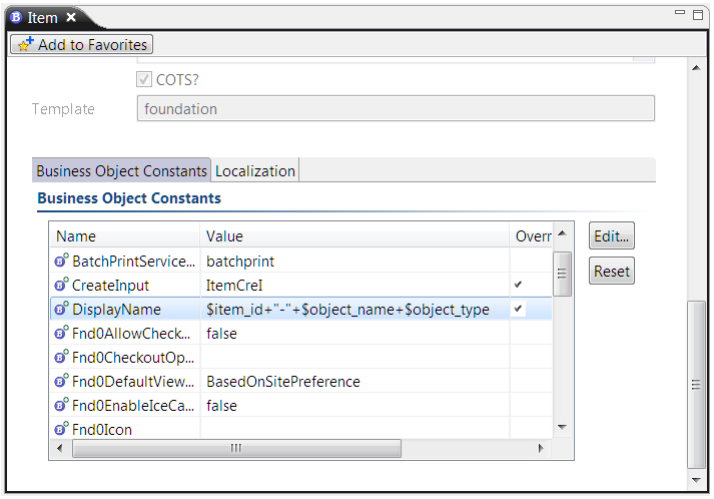
+$*property*

For example, if you want to add the **object_type** property (which shows the name of the business object type), add it as shown.



4. Click **Finish**.

The new constant definition for this business object type is shown.



5. Deploy your custom template to the database.

To deploy to a production server, package the template and install it using Teamcenter Environment Manager (TEM).

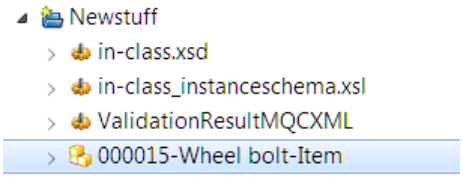
After deployment, the value set for the displayed name is used for this business object type and is inherited by all children types, unless a separate displayed name is set on a child type.

© 2025 Siemens

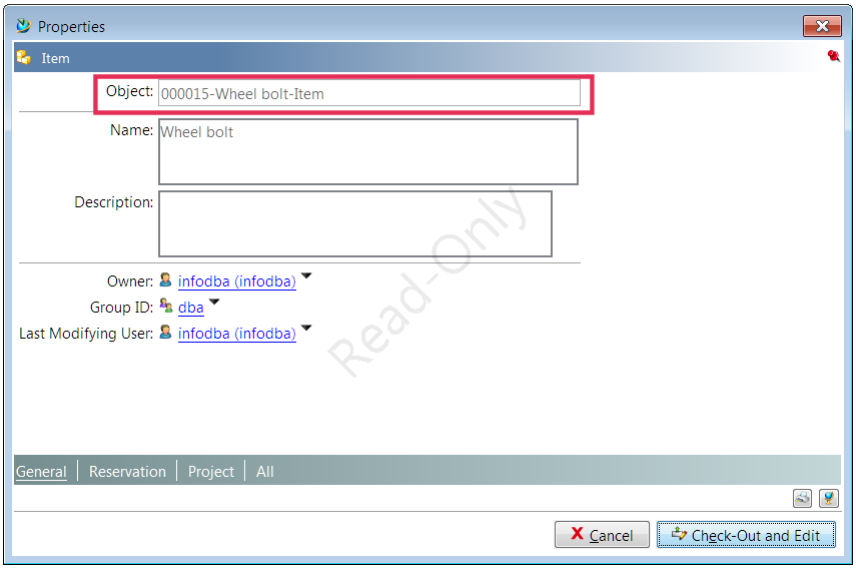
> Note:
>
> The display name of the object is different than the key defined by the **MultiFieldKey** business object constant. However, you can use the same properties for the display name that you use for the key definition.

6. Verify the modified display name format in the user interface.

   For example, view an item instance in the user interface:



   Also view the modified displayed name in the **Object** box (provided by the **object_string** property) in dialog boxes and views.



Considerations for using multifield keys

Keep in mind the following when implementing multifield keys:

- Specifications

  The **Revise** operation for specifications requires the **Revision Id** property (**mdl0revision_id**) to be part of the multifield key for **Specification** objects. If the multifield key does not include the **mdl0revision_id** property, the following error is returned:

© 2025 Siemens

```
The Multifield key for Specification objects does not include the Specification
Revision ID
(mdl0revision_id) property. As a result, creation of revisions is not supported.
If you would like to enable the "revise" operation for Specification objects,
please contact your administrator to add the Specification Revision ID
(mdl0revision_id) property to the Multifield key property for Specification objects.
```

Similarly for the **Save As** operation, the following error is returned if the multifield key does not include the **Specification Id** property (**lbr0SpecificationID**) and the **Specification Name** property (**object_name**):

```
The Multifield key for Specification objects does not include either the Specification
ID
(lbr0SpecificationID) or Specification Name (object_name) properties.
As a result, the "saveAs" operation is not supported.
If you would like to enable the "saveAs" operation for Specification objects,
please contact your administrator to add the Specification ID (lbr0SpecificationID)
or Specification Name (object_name) properties to the Multifield key property
for Specification objects.
```

- Utilities

  Arguments containing **-*key***, such as **-key**, **-itemkey**, and **-keyFileName**, are used in some utilities. These key arguments specify the multifield key ID of the item to act on, and provide an alternative to the **-item** argument as a method to specify items.

  However, to use the **-*key*** arguments, you must know the multifield key ID to enter. To obtain the key ID for an item, use the **get_key_string** utility.

Teamcenter applications support the use of multifield keys. However, there are some considerations that you must keep in mind:

- Classification

  You can use the graphics builder feature in Classification if multifield key support is enabled.

- Structure Manager

  - When using multifield keys with modular variants, avoid giving a new option the name as an existing option in the same namespace. This can cause performance issues.

  - The **PSEAutoViNewItemPopup** preference should be set to **true** when creating automatic variants in a system with multifield keys where the multifield key definition includes an attribute in addition to the item ID. This allows the user to enter the required multifield key information.

- Visualization

  Most visualization features do not require any special configuration to work with multifield key data, with the following exceptions:

© 2025 Siemens

- ClearanceDB

  The managed product name must include the **__PLM_ITEMREV_UID** value for the item revision, and the *clearance.cfgproduct* file must include the multifield key properties for the item.

- MDS stamping

  The **MetaDataStamp_template** preference must specify the values of the multifield key properties associated with the item containing the **MDS_default_styles_template** dataset.

- CAD applications

  Teamcenter integrates with CAD applications that do not support multifield keys. These CAD applications attempt to find items in Teamcenter using the **item_id** property only. To use one or more of these non-multifield key compliant CAD applications with custom multifield key definitions, you must have a domain in Teamcenter with a multifield key that contains only the **item_id** property. The domain must include all the business object classes used by the non-multifield key compliant CAD applications. The domain for the CAD application business objects is specified in the **TC_MFK_DEFAULT_DOMAIN** preference.

  When using a non-multifield key compliant CAD application, keep in mind the following:

  - If no custom multifield key definitions are created, the CAD application integration functions properly because the **item_id** property is unique in the Teamcenter deployment.

  - If custom multifield key definitions are created, a domain must be defined that includes all business objects used by the CAD application integration. The multifield key definition for this domain must be the **item_id** property only. The **TC_MFK_DEFAULT_DOMAIN** preference must be set to this domain.

    > Note:
    >
    > If no CAD integration data yet exists in Teamcenter, you can define a domain first, set the preference to that domain, and then configure the CAD integration to create types that belong only to the defined domain.

Administration data candidate keys

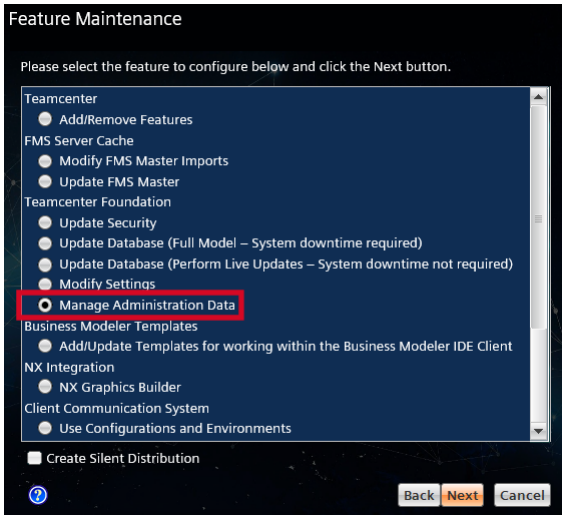
Introduction to administration data candidate keys

Administration data candidate keys are IDs assigned to administration data objects to ensure their uniqueness in the database.

Administration data are data such as ACLs, preferences, organization objects, and the like, which are not maintained in templates that can be imported and exported between sites.

© 2025 Siemens

To import or export administration data, in the **Feature Maintenance** panel of Teamcenter Environment Manager (TEM), select **Manage Administration Data**.



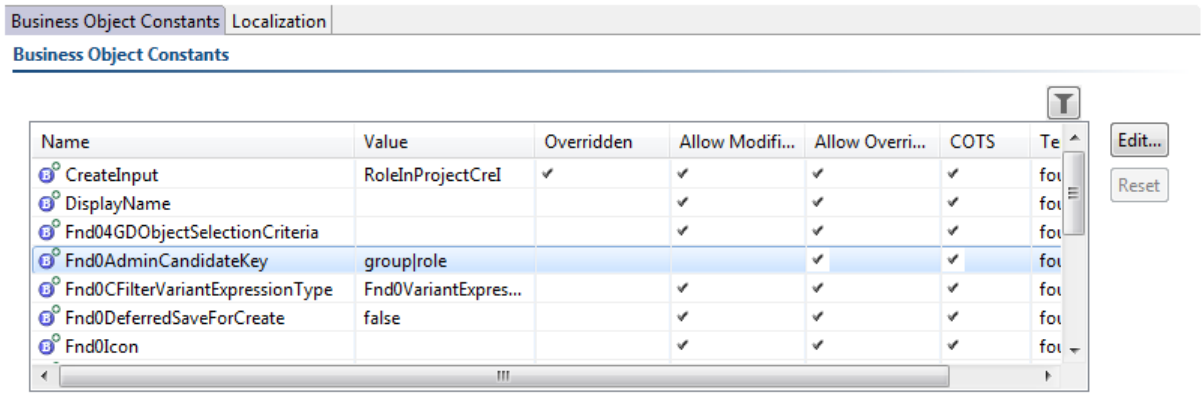
In TC XML export and import, a stable ID is used for business objects to identify the objects for the purpose of re-import and update. Only if the already imported object can be identified uniquely is it possible to re-execute a data transfer and correctly update the object. In Teamcenter, each business object has a UID that serves as its stable identification number. Upon replication to another site, this UID is unique and stable, and the corresponding object can then be updated during repeated re-replication of the original.

For administration data, this approach does not work because administration objects are usually individually created at their respective owning sites and therefore have different UIDs to begin with. So, if you relied on the UID as the stable identity, upon first-time import of such an administration object from another site, you would not find any object with this UID at the local site and therefore assume no such administration object existed. This would be incorrect because the same logical object may have already been locally created with a different UID as these are assigned uniquely by Teamcenter upon object creation.

The solution to this issue is the administration data candidate key, a stable identity definition for administration objects. Such a stable identification number (SID) is a string composed of a sequence of certain attributes of the administration object and possibly other related objects, such that the combination of this string identifies the administration object uniquely. Take the example of a user ID. There can only be one user with the given user ID at a given site. If you import this user to another site, and a user with the same name already exists, they conflict. To resolve the issue, you must logically identify the user at each site. Assigning an administration data candidate key to uniquely identify the user from each site solves the problem.

Administrators use the **Fnd0AdminCandidateKey** business object constant to **create administration data candidate key definitions**. Administrators create the keys for custom business objects that are used to define administrative data.

© 2025 Siemens



The keys can take the following forms:

- Simple candidate key

  Uses one or more primitive type of attributes of the same class as the candidate key.

  For example, the **Role** business object uses the **role_name** property as its candidate key.

- Compound candidate key

  Uses a combination of primitive attributes and/or typed reference attributes of the administration object pointing to other objects.

  For example, the candidate key for the **RoleInProject** business object is **group|role** that is a combination of the **group** and **role** typed referenced properties.

Create administration data candidate keys

Administration data candidate keys are IDs assigned to administration data objects to ensure their uniqueness in the database.

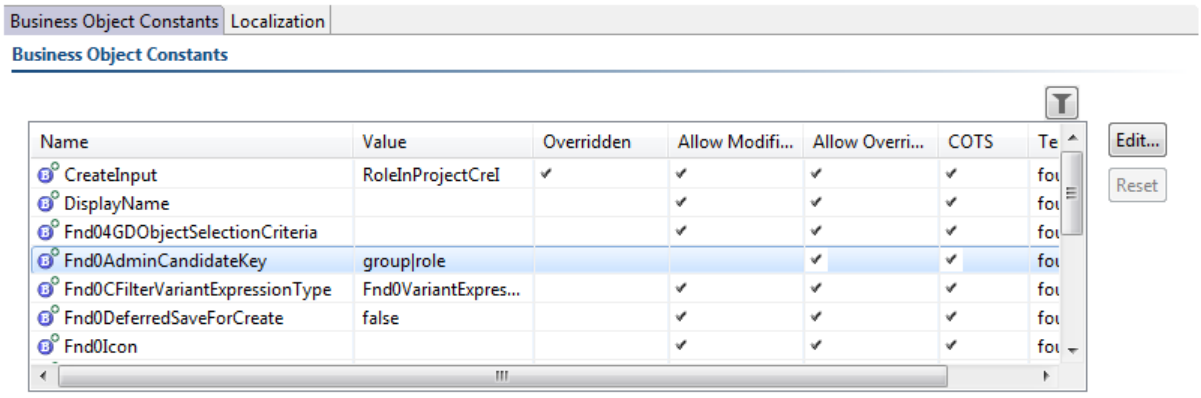
Much like **multifield keys**, administration candidate keys are composed of a combination of the object properties.

The process to create an administration data candidate key is similar to creating a multifield key. You must select the business object to which you want to assign the key and change the value of the **Fnd0AdminCandidateKey** business object constant:

1. In the **Business Objects** folder, right-click the business object and choose **Open**.

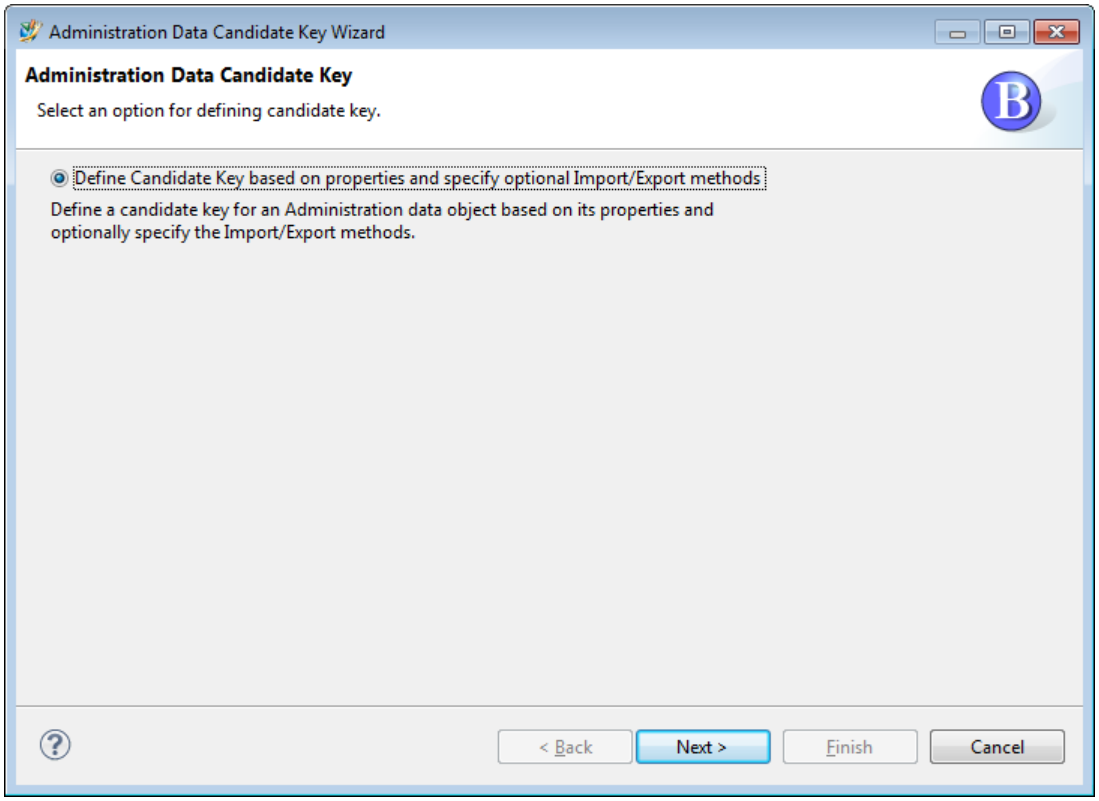
   The constant appears in the **Business Object Constants** table on the **Main** tab.

© 2025 Siemens



2. Click the **Edit** button.

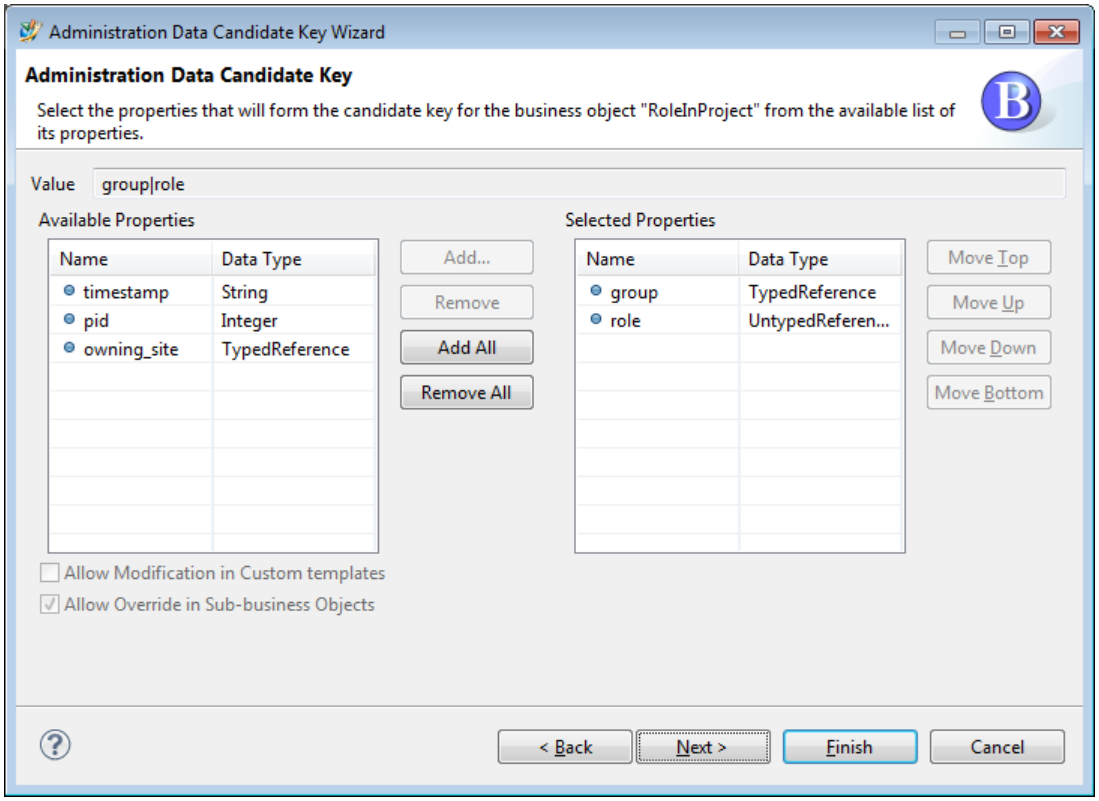
   The Administration Data Candidate Key Wizard is displayed.



3. Click **Next**.

   A dialog box is displayed that allows you to select properties to use for the key.

© 2025 Siemens



4. Select from the available properties and click the **Add** button to add them to the selected properties list. Use the **Move** keys to determine their order.

> Note:
>
> The order of attributes selected to form a candidate key is important and cannot be altered once the key has been deployed to production.

5. When you are satisfied with the properties, click **Finish**.

   The value is added to the business object constant and is displayed in the **Business Object Constants** table.

Conditions

Conditions overview

A *condition* is a definition that identifies data to be evaluated and contains an **expression** that resolves to true if the expression is true for data supplied by the calling function, and resolves to false if the expression is false for the supplied data. Conditions can be used to evaluate objects or user sessions to limit application of a rule or to deliver only certain results.

© 2025 Siemens

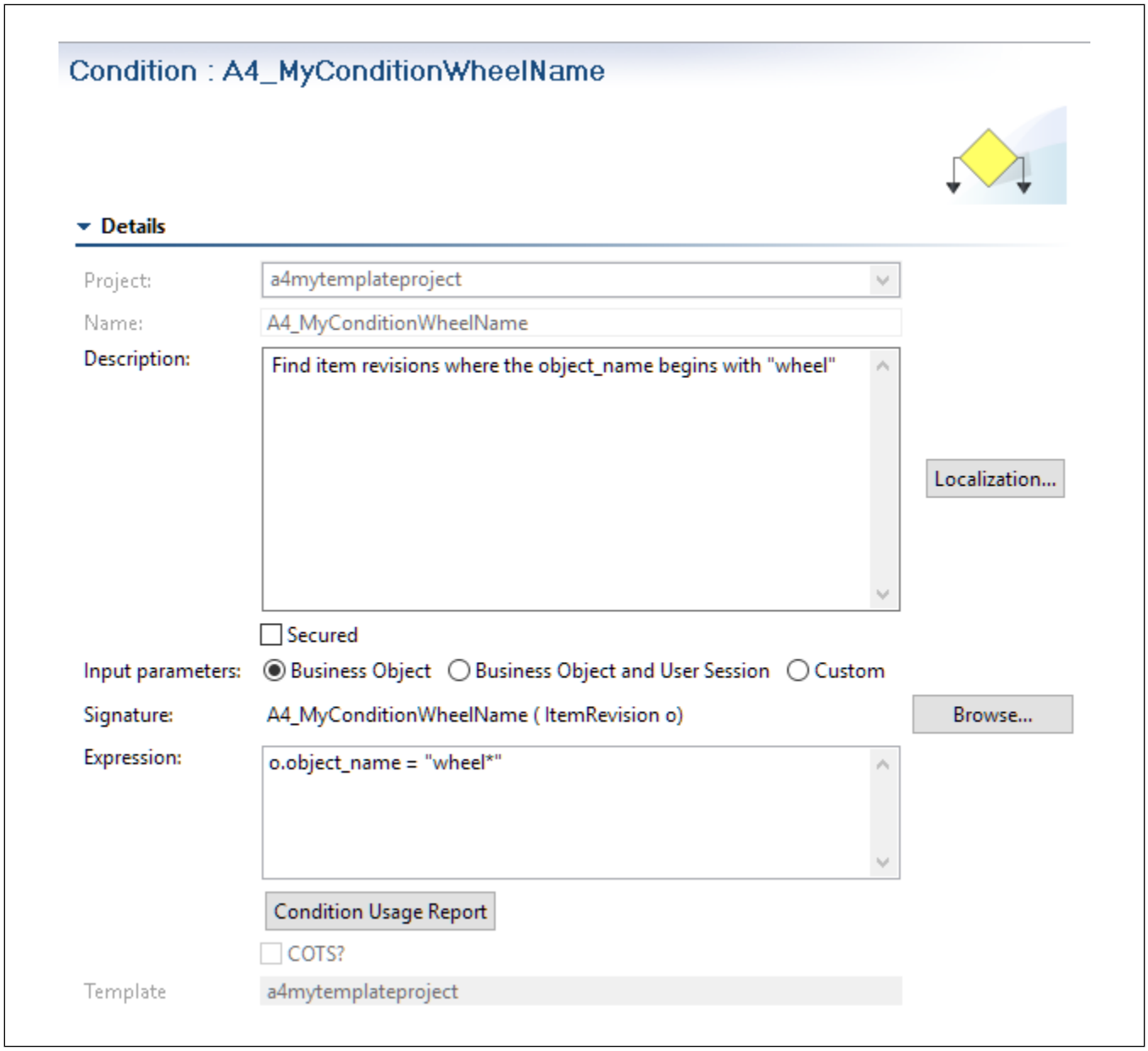
Conditions are never used by themselves, but are only used by other objects. There are many kinds of objects that can use conditions, such as business object display rules, deep copy rules, IRDCs, LOVs, and naming rules, among others.

Example:

1. You define a condition that an object has a specification relationship to an item revision.
2. You attach the condition to a deep copy rule for a document business object.
3. When an item revision instance with a specification relationship to a document business object instance is revised, then the document object is copied forward to the new item revision.

You have a great deal of flexibility when creating the expression for a condition. You can write expressions that contain strings, logical statements, dates, tags, and even other conditions.

© 2025 Siemens



Example details of a condition definition

Application of a rule with a condition

When a rule with a condition is run against an object, application of the rule is divided into two parts: an **IF** clause and a **THEN** clause.

condition (**IF** clause)	evaluates the condition expression for the object instance
rule (**THEN** clause)	describes an action or access permission on the object

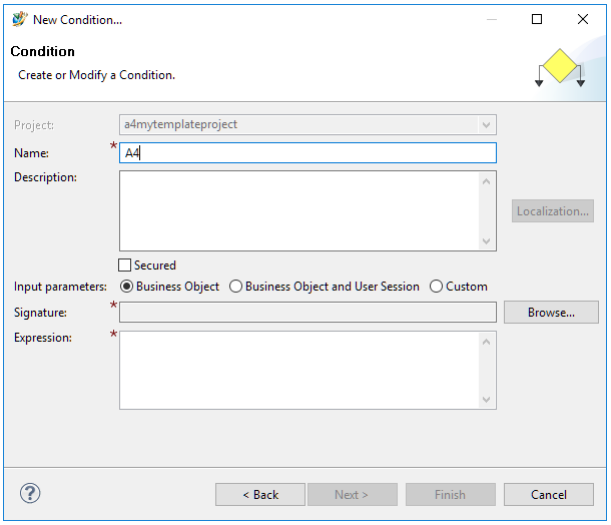
© 2025 Siemens

Add a condition

1. In the Business Modeler IDE, start the new condition wizard in one of these ways:

   - On the menu bar, choose **BMIDE**>**New Model Element**, in the **Wizards** box type **condition**, and click **Next**.

   - Open the **Extensions\Rules** folders, right-click the **Conditions** folder, and choose **New Condition**.



2. In the **Condition** dialog box, define the condition parameters.

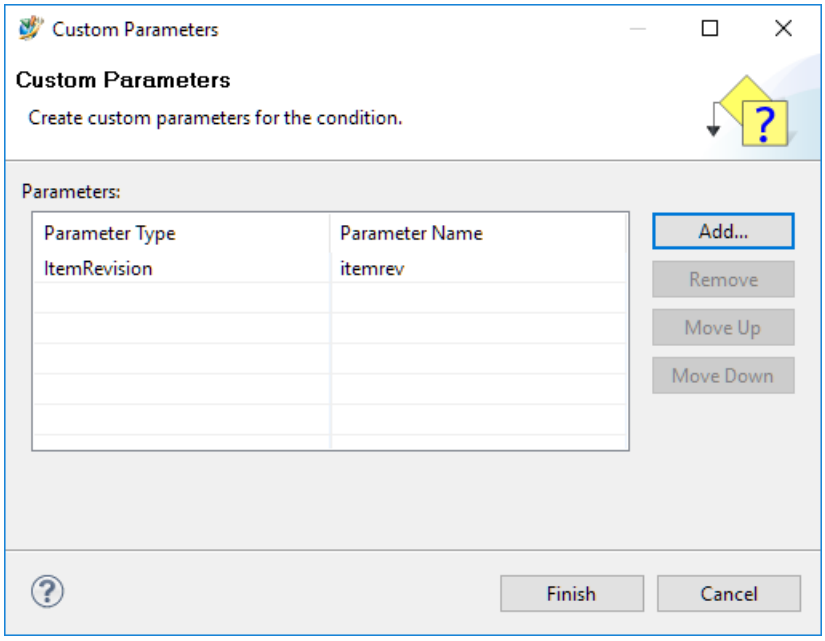
For this parameter	Do this
Name	Type the name you want to assign to the condition in the database. The template prefix is required and is automatically entered.
Description	Type a description of the condition.
Secured	Select **Secured** to prevent the condition expression from being modified or overridden by another template.
Input parameters	Select one of the following: • **Business Object** if you want the condition to be applied to a business object. You can also select this option if you want to use only the **UserSession** business object as a parameter. • **Business Object and User Session** if you want the condition to be applied to a business object in the context of a user's work session (via the **UserSession** object).

© 2025 Siemens

For this parameter	Do this
	• **Custom** if you want to define a condition with two parameters where one of the parameters is not a **UserSession** business object, or to define a condition with three or more parameters.
Signature	Click **Browse** and specify the list of data (the business objects against which you want to run the condition) to be supplied by the calling function. You may end up with more objects in the signature than you use in the expression. Objects defined in the signature do not have to be used in the expression; inclusion in the signature simply makes the object available for use in the expression. The dialog box that opens and procedure for specifying the list varies depending on the selection for **Input parameters**. • **Business Object** input parameters • **Business Object and User Session** input parameters Example: Suppose you want to write a condition to evaluate item revision business objects for a particular group of users. Select the **ItemRevision** business object and click **OK**. Notice that in the **Signature** box an **o** parameter is assigned to the business object, and a **u** parameter is assigned to the user session: `A4_MyCondition ( ItemRevision o , UserSession u)` Later, when you write the condition's evaluation formula in the **Expression** box, you use the **o** parameter to represent the business object and the **u** parameter to represent the user session. • **Custom** input parameters The Custom Parameters wizard runs. Use this wizard to define a condition with two parameters where one of the parameters is not a **UserSession** business object, or to define a condition with three or more parameters.

© 2025 Siemens

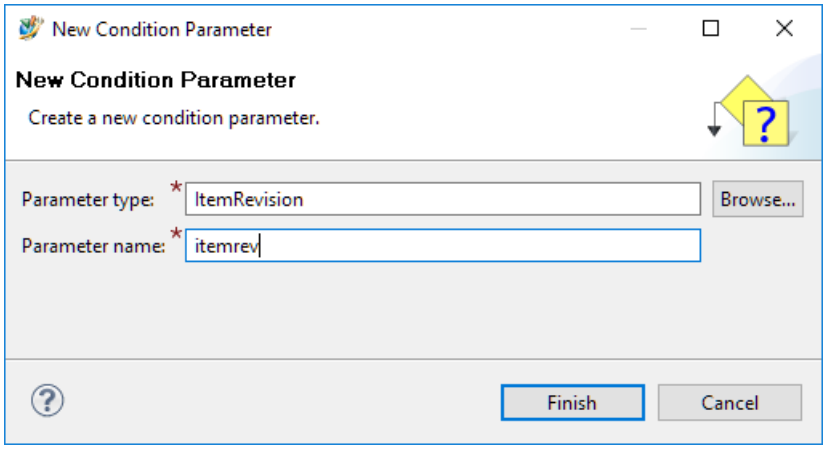
For this parameter	Do this



Perform the following steps in the **Custom Parameters** dialog box:

a. Click **Add**.

The New Condition Parameter wizard runs.



b. Click **Browse** and choose the business object containing the data, then click **OK**.

© 2025 Siemens

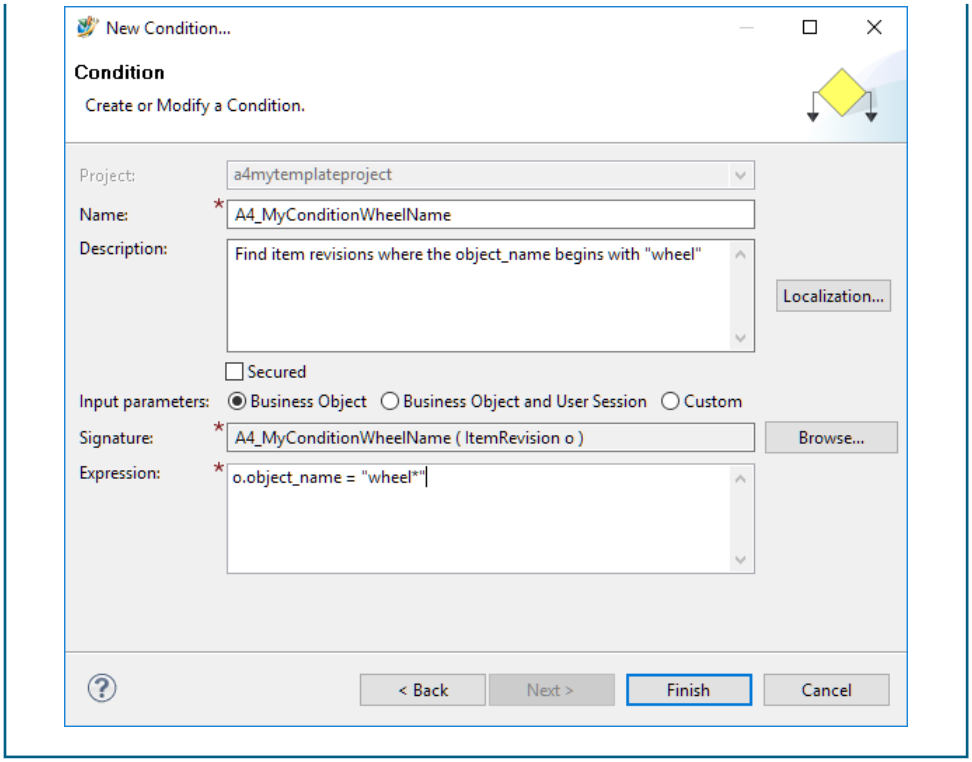
For this parameter	Do this
	c. In **Parameter Name**, type a parameter name to use for the selected business object, then click **Finish**. You can type anything you want for the parameter name, as long as it does not have spaces. Earlier you saw how by default the **o** parameter was chosen by the system to represent a business object, and **u** to represent a user session. Now, type the parameter name. For example, if you selected an **ItemRevision** business object, type **itemrev** for the parameter name. The parameter appears in the **Parameters** table. d. Repeat to add as many parameters as you want, and click **Move Up** or **Move Down** to change the order of the parameters in the condition signature. e. When you are done adding custom parameters, click **Finish**. The custom parameters are added to the signature. Example: Suppose you want to write a condition to evaluate the IDs on item revision business objects. First you select an **ItemRevision** business object and enter **itemrev** for the parameter, and then select an **ItemIdRecord** business object and enter **itemid** for the parameter. The resulting **Signature** box would contain the following: `MyConditionItemRevID ( ItemRevision itemrev , ItemIdRecord itemid)` Later, when you write the condition evaluation formula in the **Expression** box, you use the **itemrev** to represent the item revision business object and **itemid** to represent the item ID record business object.
Expression	Type the condition statement you want to evaluate using the data defined in the signature. You can write **expressions** that contain strings, logical statements, dates, tags, and even other conditions. The condition engine utilizes the CLIPS (C Language Integrated Production System) external rules engine to process condition data. Because of the CLIPS engine rules, condition expressions and the values for the business object properties must be specified using the standard US7ASCII character set.

© 2025 Siemens

For this parameter	Do this
	Tip: Use the following tips for writing an expression: • For a selection of valid entries to choose from, place your cursor in the box and press Ctrl + space bar to activate the content assistant. The content assistant cannot be shown if the business object is an untyped reference or untyped relation. • Type a period after a business object parameter to choose from a list of properties or operations on that business object. For example, type **o.** for a business object, **u.** for a user session, or if you have a custom parameter, type the parameter followed by a period (for example, **itemrev.**). • The expression is validated as you type it in. If the expression is not valid, a message displays at the top of the **Condition** dialog box. Your condition is valid if the **Finish** button is available and no error message displays at the top of the dialog box. • Condition expressions must use the standard US7ASCII character set.

	Example:
	Suppose you want to write a condition to help find all item revisions whose names begin with **wheel**. To define the signature, you select the **Business Object** input parameter and choose the **ItemRevision** business object. The signature displays similar to the following:
	`A4_MyConditionWheelName ( ItemRevision o)`
	In **Expression**, type the following:
	`o.object_name = "wheel*"`
	o is the parameter that represents the item revision, and **object_name** is the name attribute on item revisions. This statement means to evaluate the **object_name** attribute on all item revisions and find those that begin with **wheel**.

© 2025 Siemens

For this parameter	Do this
	

3. Click **Finish**.

   The new condition appears under the **Conditions** folder.

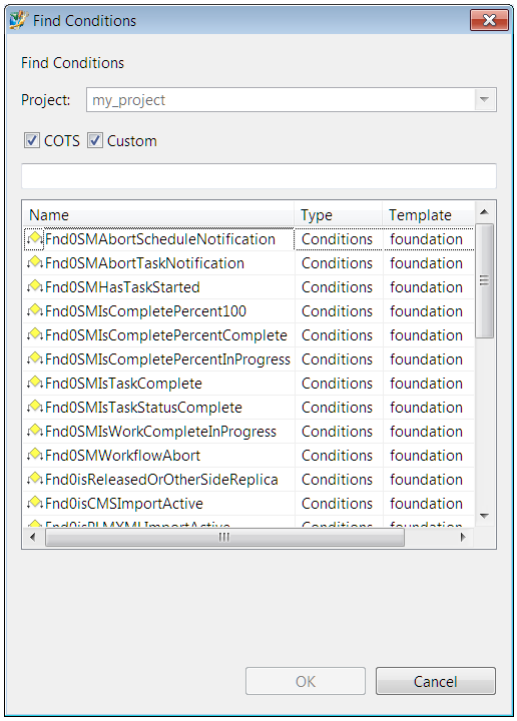
4. To save the changes to the data model, choose **BMIDE**>**Save Data Model**, or on the main toolbar click **Save Data Model** .

Search conditions

1. Right-click the **Conditions** folder and choose **Find Conditions**.

   The Find Conditions wizard runs.

© 2025 Siemens



2. Type the characters to search for. Use an asterisk ***** for wildcard searching.

   The conditions that match the filtering appear in the table.

3. To open a condition, double-click it in the table.

Condition examples

Examples provide a good way to understand how to structure conditions. The examples below are for discussion purposes only.

- Primitives
  - String

```
isStringCondition1( Item o ) := Expression: "EqualString" =
"EqualString"
isStringCondition2( Item o ) := "UnequalString1" != "UnequalString2"
```

> Note:
>
> Wild cards can be specified in strings using ***** and **?** characters:

© 2025 Siemens

- ***** matches 0 or more characters
- **?** matches one character

For example:

```
isCondition1( UserSession o ) := o.role_name = "Design*"
```

This condition evaluates to **TRUE** for all users who have roles like **Designer** or **Design Manager**.

```
isCondition2( UserSession o ) := o.user_id = "?ary"
```

This condition evaluates to **TRUE** for all users who have user IDs like **Mary** or **Gary**.

- Character

```
isCharCondition1( Item o ) := 'a' = 'a'
isCharCondition2( Item o ) := 'a' != 'b'
isCharCondition3( Item o ) := 'a' < 'b'
isCharCondition4( Item o ) := 'a' <= 'a'
isCharCondition5( Item o ) := 'b' > 'a'
isCharCondition6( Item o ) := 'a' >= 'a'
```

- Integer

```
isIntCondition1( Item o ) := 1 = 1
isIntCondition2( Item o ) := 0 != 1
isIntCondition3( Item o ) := 0 < 1
isIntCondition4( Item o ) := 0 <= 1
isIntCondition5( Item o ) := 1 > 0
isIntCondition6( Item o ) := 1 >= 0
```

- Float

```
isFloatCondition1( Item o ) := 1.5 = 1.5
isFloatCondition2( Item o ) := 0.5 != 1.5
isFloatCondition3( Item o ) := 0.5 < 1.5
isFloatCondition4( Item o ) := 0.5 <= 1.5
isFloatCondition5( Item o ) := 1.5 > 0.5
isFloatCondition6( Item o ) := 1.5 >= 0.5
```

- Boolean

```
isBoolCondition1( Item o ) := true
isBoolCondition2( Item o ) := false
```

- Date

© 2025 Siemens

```
isDateCondition1( Item o ) := o.creation_date = "01-Jan-2009 00:00"
isDateCondition2( Item o ) := o.creation_date != "29-Feb-2008 23:59"
isDateCondition3( Item o ) := o.creation_date < "01-Jan-2009 00:00"
isDateCondition4( Item o ) := o.creation_date <= "01-Jan-2009 00:00"
isDateCondition5( Item o ) := o.creation_date > "29-Feb-2008 23:59"
isDateCondition6( Item o ) := o.creation_date >= "29-Feb-2008 23:59"
```

The date literal has to be specified in the following format: **"dd-MMM-yyyy HH:mm"**.

Date element	Description
dd	Two-digit date, for example, **31**
MMM	Three-character month in English only, for example, **Jan**
yyyy	Four-digit year, for example, **2009**
HH	Two-digit hour specified in 24-hour format, for example, **23** for 11:00 p.m.
mm	Two-digit minutes, for example, **59**

- Properties on business objects
  - Properties on any business object

    ```
    isPropCondition1( MyItem o ) := o.color = "red"
    isPropCondition2( MyItem m, YourItem y ) := m.color = y.color
    isPropCondition3( MyItem m, YourItem y ) :=
    m.owning_project.project_id = y.owning_project.project_id
    ```

  - Properties on any business object when there are spaces in the property name

    ```
    isCondition4( BOMLine bl1, BOMLine bl2 ) := bl1.`UG NAME` = bl2.`UG
    NAME`
    ```

  - Properties on **UserSession**

    ```
    isPropCondition5( MyItem o, UserSession u ) := o.owning_user =
    u.user
    isPropCondition6( MyItem o, UserSession u ) := o.owning_group =
    u.group
    isPropCondition7( MyItem o, UserSession u ) := o.owning_project =
    u.project
    ```


© 2025 Siemens

```
isPropCondition8( MyItem o, UserSession u ) := u.user_id = "tc_user"
isPropCondition9( MyItem o, UserSession u ) := u.group_name =
"Engineering"
isPropCondition10( MyItem o, UserSession u ) := u.project_name =
"Concept"
isPropCondition11( MyItem o, UserSession u ) := u.role_name = "DBA"
```

- Operations on business objects

```
isOperationCondition1( DeepCopyRule dr, ItemRevision
target, ItemRevision otherside ) :=
otherSide.items_tag.isLatestRevisionMature() = true
isOperationCondition2( DeepCopyRule dr, ItemRevision target,
POM_object otherside ) := otherSide.isReplica() = true
isOperationCondition3( DeepCopyRule dr, ItemRevision target,
ItemRevision otherside ) :=
target.checkUniqueItems( otherSide,dr.relation,dr.is_target_primary,1,
-1 ) = true
```

- Nested conditions

```
isNestCondition1( DeepCopyRule dr, ItemRevision target, ItemRevision
otherside ) :=
( Condition::checkOtherSideOneToOne( dr, target,otherside ) = true )
AND
( Condition::isOtherSideLatestMature( dr, target, otherside ) = true )
```

> Note:
>
> All nested conditions need to be prefixed with **Condition::**.

- **INLIST**

> Note:
>
> All **INLIST** conditions need to be prefixed with **Function::**.

Search for a primitive in an array of typed/untyped references

```
isNestCondition2( WorkspaceObject o, UserSession u ) :=
Function::INLIST( u.project_name, o.project_list, "project_name" )
```

- String functions
  - **ToUpper**

© 2025 Siemens

```
isToUpperCondition1( Item o, UserSession u ) :=
Function::ToUpper( u.role_name ) = "DBA"
```

> Note:
>
> All **ToUpper** conditions need to be prefixed with **Function::**.

- **ToLower**

```
isToLowerCondition1( Item o, UserSession u ) :=
Function::ToLower( u.user_id ) = "mgr"
```

> Note:
>
> All **ToLower** conditions need to be prefixed with **Function::**.

- Operators

  Operators include **!**, **!=**, **<**, **<=**, **=**, **>**, **>=**, **AND**, **NOT**, and **OR**.

  For an example that uses operators, see the **Fnd0SMIsCompletePercentInProgress** condition:

```
Fnd0SMIsCompletePercentInProgress (ScheduleTask task, Schedule sched,
UserSession session) =:
task.complete_percent<100 and ( (task.complete_percent> 0 and
task.fnd0state="not_started") or (task.complete_percent>=
0 and task.fnd0state="complete") ) and (task.task_type=0 or task.task_type=1
or task.task_type=4) and sched.is_template=false
```

- Lists of values (LOVs)

  The ability to **place conditions directly on LOVs** or sub-LOVs is deprecated. Instead, you can apply the conditions when attaching the LOVs to properties.

- **Naming rules**

- **Deep copy rules**

- IRDCs

  For an example, see the **Fnd0DMTemplateCondition** condition:

```
Fnd0DMTemplateCondition (DMTemplateRevision 0) :=
o.ApplicationName="RM"
```


© 2025 Siemens

Condition system

Condition syntax

The code syntax for a condition is as follows:

condition-name (*argument-list*) **:=** *expression*

where

condition-name	is the name of the condition.
argument-list	is the list of parameters (object types and names) that can be included in the expression. The arguments identify and apply to data that is supplied for an operation.
:=	separates the condition name and argument list from the expression. The separator symbol does not appear in the **Condition** dialog box when you create a condition.
expression	is the expression statement.

Example:

```
MyCondition ( ItemRevision o ) := o.color = red
```

This condition resolves to true if the **color** attribute on an **ItemRevision** business object has the value **red**.

Condition expressions

Condition *expressions* are statements that evaluate data supplied by a calling function. When used for a condition, expressions are resolved to true if an evaluation of the statement is true, and are resolved to false if an evaluation of the statement is false. Expressions are of two kinds: unary and binary.

unary Unary expressions evaluate a single data point.

Syntax of a simple unary expression:

expression-value

binary Binary expressions compare two data points, and comprise a left expression (left operand) and a right expression (right operand), with an operator such as **=** between them.

© 2025 Siemens

Syntax of a simple binary expression:

expression-value comparison-operator expression-value

Both unary and binary expressions can be combined with other unary or binary expressions using the **AND/OR** operators to form complex expressions.

Syntax of a complex expression:

(unary-expression-value | binary-expression-value) [(AND | OR)
(unary-expression-value | binary-expression-value)]

Expressions can contain the following:

- Primitive data types (string, character, integer, float, Boolean, date)
- Business objects
- Properties on business objects
- Operations on business objects
- User session object
- Nested conditions
- Functions such as **INLIST**, **ToUpper**, and **ToLower**
- Equality operators (=, !=)
- Relational operators (<, >, <=, >=)
- Logical operators (**AND**, **OR**)
- Unary operators (!)

Condition expressions and the values for the business object properties must be specified using the standard US7ASCII character set.

Condition calls

You can call conditions from within conditions. You must supply the name of the condition to call and the object arguments to pass. Like other calls, condition calls have the following syntax:

condition-name(*argument-list*) := *expression*

Replace *condition-name* with the name of the condition you are creating, replace *argument-list* with the list of objects to be supplied by the calling program, and replace *expression* with the condition statement that calls the condition.

In the following example, the **isCondition1** condition calls the **checkOtherSideOneToOne** and **isOtherSideLatestMature** conditions:

```
isCondition1( DeepCopyRule dr, ItemRevision target, ItemRevision
otherside ) :=
  ( Condition::checkOtherSideOneToOne( dr, target,otherside ) = true )
```


© 2025 Siemens

```
AND
  ( Condition::isOtherSideLatestMature( dr, target, otherside ) = true )
```

Comparison operators

Binary expressions compare an expression on the left side (left operand) with an expression on the right side (right operand). Binary expressions follow this syntax:

expression-value comparison-operator expression-value

The following Boolean operators are used to compare the left operand to the right operand.

Operator	Description
!	Not true (when used with a stand-alone operand)
!=	Not equal to
<	Less than
<=	Less than or equal to
=	Equal to
>	Greater than
>=	Greater than or equal to
AND	Both operands are evaluated to true
NOT	The following operand is not true.
OR	Either operand is evaluated to true

Mathematical operators like **+**, **-**, **/**, *****, and **%** are not supported, nor are trigonometric functions like **sin**, **cos**, **tan**, and so on.

Valid binary expression operands

Binary expression syntax is validated at creation time when you **add a condition**. Valid operand and operator combinations are shown in the following table.

Left operand	Valid operators	Valid right operands
String property String operation String literal	**=**, **!=**	String property String operation

© 2025 Siemens

Left operand	Valid operators	Valid right operands
		String literal **ToUpper** / **ToLower**
String literal	**=**, **!=**, **>**, **<**, **>=**, **<=**	Date attribute Date operation
Numerical attribute Numerical operation Numerical literal Character literal Character property Character operation	**=**, **!=**, **>**, **<**, **>=**, **<=**	Numerical attribute Numerical operation Numerical literal Character literal Character property Character operation
Logical attribute Logical operation Condition **INLIST** Logical literal (**TRUE**, **true**, **FALSE**, or **false**)	**AND**, **OR**, **=**, **!=**	Logical attribute Logical operation Logical literal Condition **INLIST**
Date attribute Date operation	**=**, **!=**, **>**, **<**, **>=**, **<=**	Date attribute Date operation String literal
Tag property Tag operation	**=**, **!=**	Tag property Tag operation **NULL** value
NULL value	**=**, **!=**	Tag property Tag operation
ToUpper / **ToLower**	**=**, **!=**	String property String operation String literal
Stand-alone	**!**	Logical attribute Logical operation Condition **INLIST**

To check whether a given expression is valid, look up the left operand to see which operators and right operands are valid for it.

Example:

In the expression

© 2025 Siemens

```
obj1.name != obj2.name
```

both the left operand and the right operand are string properties. When you check the table, you see a left operand string property can use **=** or **!=** operators when the right operand is a string property. So the expression is valid.

Condition operation rules

Not all operations defined on a business object can be used in a condition expression. Only an operation that satisfies the following criteria shows up in the content assistant (press Ctrl + space bar) and can be used in condition expressions:

- Returns an **int**
- Has exactly one output that is the last parameter in the operation
- Has input only parameters (except the last parameter)
- Does not have input/output parameters
- Uses one of the following parameter data types in a condition expression:

  bool
  char
  date_t
  double
  float
  int
  long
  std::string
  tag_t

- Uses **tag_t** instead of **Teamcenter::***business-object* as parameters in condition expressions
- Does not take vectors, maps, or any template data type as parameters
- Does not take SOA service objects as parameters

Because there is no provision to define variables in a condition expression, the actual return value on the operation is ignored. Instead, the last output only parameter on the operation is treated as a return value.

© 2025 Siemens

Valid condition signatures

You can create your own conditions to use for LOVs, IRDCs, GRM rules, deep copy rules, and so on. In the wizard for adding a condition to an object, only the conditions with signatures that are valid for the object are shown. Depending on the object type, valid condition signatures vary.

Object type	Valid condition signatures
Extension rule LOV value Naming rule Revision naming rule Sub-LOV Business object display rule	*condition-name***(UserSession)**
LOV attachment	*condition-name***(UserSession)** *condition-name***(POM_object o)** *condition-name***(POM_object o, UserSession u)** When conditions for LOV attachments are evaluated, the attribute values shown in the client user interface are used. That is, the LOVs shown are based on what the user sees in the user interface, not the attributes on the object as stored in the database.
Deep copy rule	*condition-name***(UserSession)** *condition-name***(DeepCopyRule,ItemRevision,POM_object)** *condition-name***(DeepCopyRule,ItemRevision,POM_object,UserSession)** For the second and third condition signatures: • The second argument is the target business object used in the deep copy rule (that is, **ItemRevision**). • The third argument is the object type used in the deep copy rule (that is, **POM_object**).
GRM rule	*condition-name***(UserSession)** *condition-name***(ImanGRM, POM_object, POM_object)** *condition-name***(ImanGRM, POM_object, POM_object, UserSession)** For the second and third condition signatures:

© 2025 Siemens

Object type	Valid condition signatures
	• The first argument is the current GRM rule. • The second argument is the primary business object used in the GRM rule. • The third argument is the secondary business object used in the GRM rule.
IRDC	*condition-name*(**ItemRevision**) *condition-name*(**ItemRevision, UserSession**)

Troubleshooting conditions evaluations

Perform the following checks to determine why a condition is not evaluating as expected:

1. Verify that the client system is refreshed.

   Log off and log on to refresh the conditions in the system.

   You can also use the **KnowledgeBaseRefreshInterval** global constant to automatically check for new conditions by setting its value to **0**.

2. Verify that the condition signature of the condition is valid for the specific functionality. For a list of valid condition signatures by functionality, see **Valid condition signatures**.

   > Note:
   >
   > When conditions for LOV attachments are evaluated, the attribute values shown in the client user interface are used. That is, the LOVs shown are based on what the user sees in the user interface, not the attributes on the object as stored in the database.

3. Verify that the condition is deployed correctly.

   a. Run the **business_model_extractor** utility to verify if the condition has been deployed to the database, for example:

   ```
   business_model_extractor
     -u=username -p=password -g=group
     -mode=all
     -outfile=dbextract_model.xml
     -log=dbextract_model.log
   ```

   b. Open the resulting output file (for example, **dbextract_model.xml**) and search for the condition.

   c. If you find the condition exists in the extracted model XML file and condition parameters and condition expression are correct, proceed to the next troubleshooting step. Otherwise,

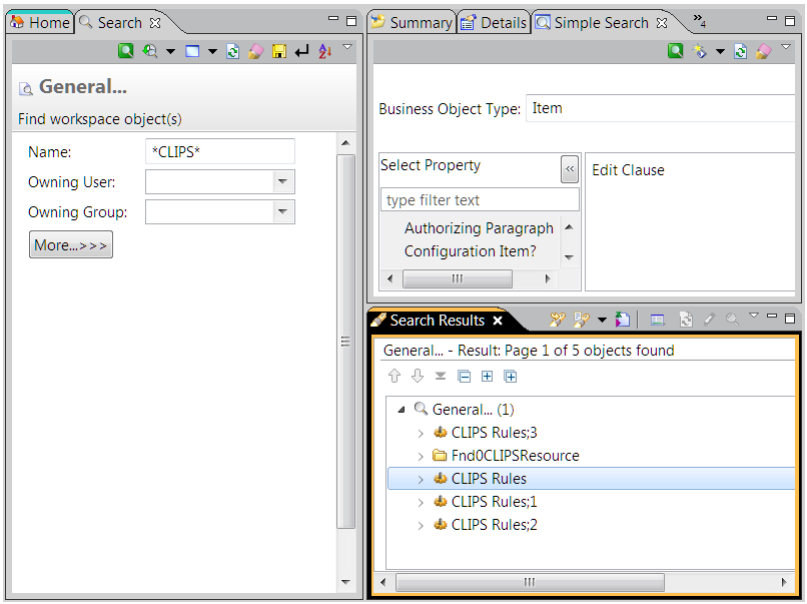
© 2025 Siemens

investigate why the condition did not get deployed; perhaps the deployment failed and the failure was ignored and so the condition did not get deployed to the database.

4. Verify that the CLIPS rules file was generated correctly.

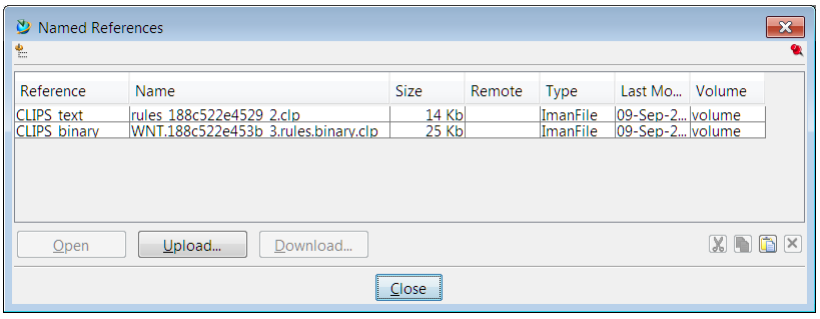
   a. Launch the rich client.

   b. Search for the **CLIPS Rules** dataset instance.



   If you do not see a **CLIPS Rules** dataset instance, it means that something failed during deployment and prevented the **CLIPS Rules** dataset from being instantiated. Try manually regenerating it by executing the **bmide_setupknowledgebase** utility.

   c. Right-click the **CLIPS Rules** dataset instance and choose **Named References**.



   d. Select the **CLIPS_text** named reference and click the **Download** button to export it.

   If you are unable to export the **CLIPS_text** named reference, check if FMS and volumes are set up correctly. These have to be set up correctly for the CLIPS rules file to get generated and

© 2025 Siemens

uploaded as named references. Once you fix FMS and volume related issues, try exporting the **CLIPS_text** named reference again.

e. Open the exported CLIPS rules file (for example, **rules_*id*.clp**) in a text editor and search for the condition.

If you find the condition exists in the CLIPS rules file and condition parameters and condition expression seem correct, then the problem could be with the evaluation code. Otherwise, try regenerating the CLIPS rules file manually by running the **bmide_setupknowledgebase** utility to see if that resolves the issue.

```
bmide_setupknowledgebase
  -u=user-name
  -p=password  -g=group
  -regen=true
  -log=bmide_setupknowledgebase.log
```

ITK APIs for conditions

The following ITK APIs are the only ones required for condition engine processing:

- **CE_evaluate_condition**

  Evaluates the condition using the specified condition tag and condition parameters.

```
extern CE_API int CE_evaluate_condition(
                    const tag_t  condition_tag,  /**< (I)  */
                    const int    parm_count,     /**< (I)  */
                    const tag_t  *parm_tags,     /**< (I)  */
                    logical      *result         /**< (O)  */
                    );
```

  For example:

```
// .... some code .... //
 ifail = CE_evaluate_condition( condition_tag,
                                parmCount,
                                parmArray,
                                &local_result );
 // .... some code .... //
```

  Always pass in the **NULLTAG** value for the **UserSession** object parameter. The condition engine code automatically retrieves the **UserSession**object.

  The value of the tag of all objects represented in the condition signature (with the exception of the **UserSession** object for which you pass in the **NULLTAG** parameter) must be provided in the order they are defined in the condition signature. For example, given an **ExampleCondition(Item o,**

© 2025 Siemens

ItemRevision ir, UserSession u) condition signature, provide the actual tag of an **Item** object in the first parameter, the actual tag of an **ItemRevision** object in the second parameter, and **NULLTAG** for the third parameter. The **parm_count** number is three, and the parameters are loaded into the array of tags in that order.

The **CE_evaluate_condition** API does not invoke the rules engine if the condition name is **isTrue** or **isFalse**, so there is no need for callers to retrieve the condition name using the condition tag to check if the condition name is **isTrue** or **isFalse**.

- **CE_find_condition**

  Returns the condition tag for the specified condition name attribute value.

```
extern CE_API int CE_find_condition(
                const char   *condition_name,  /**< (I)  */
                tag_t        *condition_tag    /**< (O)  */
                );
```

- **CE_ask_condition**

  Returns the condition name attribute value for the specified condition tag.

```
extern CE_API int CE_ask_condition(
                const tag_t  condition_tag,    /**< (I)  */
                char         **condition_name  /**< (OF)  */
                );
```

  The return value is **OF**, so the output must be freed by **SM_free**.

View condition engine service information

Details regarding the condition engine service can be viewed in the **Foundation** template in the Business Modeler IDE.

1. Open the **Extensions\Code Generation\Services\TcSoaBusinessModeler** folders.

2. Double-click the **ConditionEngine** object to view the details.

Following is the **ConditionEngine** service API:

```
Connection connection = SoaSession.getConnection();
ConditionEngineService conditionEngineService =
ConditionEngineService.getService( connection );
   EvaluateConditionsResponse response = null;
   try
   {
       ConditionInput[] inputs = new ConditionInput[2];
```


© 2025 Siemens

```
    // true condition
    ConditionInput conditionInput1 = new ConditionInput();
    conditionInput1.conditionName = "isTrue";
    inputs[0] = conditionInput1;

    // false condition
    ConditionInput conditionInput2 = new ConditionInput();
    conditionInput2.conditionName = "isFalse";
    inputs[1] = conditionInput2;

    // evaluate conditions
    response = conditionEngineService.evaluateConditions( inputs );

    // process results
    assertEquals( response.outputs[0].exitCode, 0 );
    assertEquals( response.outputs[0].result, true );

    assertEquals( response.outputs[1].exitCode, 0 );
    assertEquals( response.outputs[1].result, false );
}
catch ( Exception e )
{
    assertNull( e );         }
```

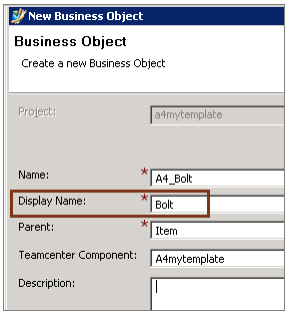

© 2025 Siemens

13. Setting display names

What is a display name?

A display name is the name that displays for data model objects in Teamcenter user interface clients. You can use a Business Modeler IDE to set the display name for your custom objects as well as override display names on COTS (standard) Teamcenter objects.

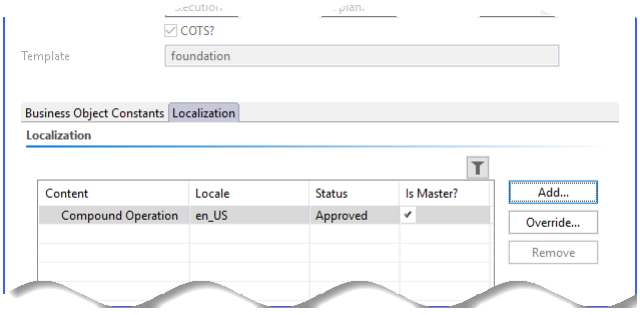
For example, if you create a new part business object type named **A5_Bolt**, you can use the **Display Name** box in the creation dialog box to set the display name as **Bolt**.



Add a display name for the object in other languages

1. Open the new business object type.

2. To the right of the **Localization** table, click **Add**.



3. In the **Localization** dialog box, select the locale and enter the display name for the locale.

> Caution:
>
> If you change the display name of an object in the master locale, you must also change the display names for the object in all locales. If you do not, the status of display names in other locales is marked as invalid.

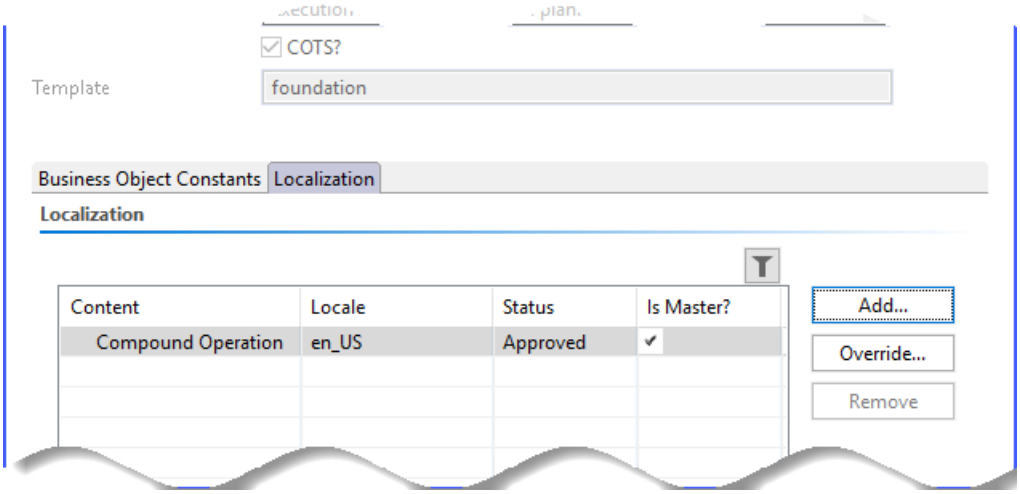
© 2025 Siemens

Change display names for business objects and option types

You can change the display name of any business object, as well as the following object types: ID contexts, note types, occurrence types, statuses, and view types.

1. In the Business Modeler IDE, open a custom object or a COTS (standard) object.

2. Open the **Localization** tab.

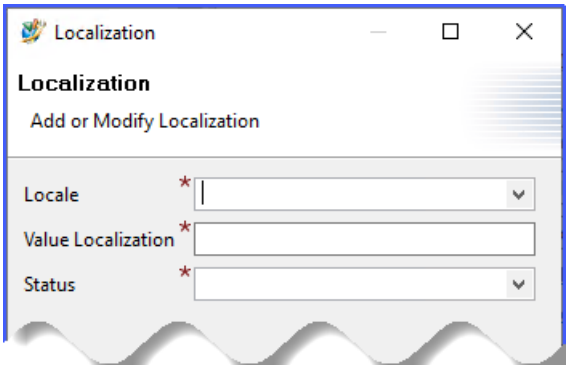


3. In the **Localization** table, select a row and click a command at the right side of the table:

Use this command	To do this
Add	Add a new display name.
Override (For a COTS object)	Change the selected display name.
Edit (For a custom object or an already overridden COTS object)	Change the selected display name.
Remove	Remove the selected display name.

The **Localization** dialog box appears.

© 2025 Siemens



4. In the **Localization** dialog box, specify the options for the localization:

For this option	Do this
Locale	Select the language locale where the text is to be used. Available locales depend on the languages set in the global constant **Fnd0SelectedLocales**. **cs_CZ** Czech as spoken in the Czech Republic **de_DE** German as spoken in Germany **en_US** English as spoken in the United States **es_ES** Spanish as spoken in Spain **fr_FR** French as spoken in France **it_IT** Italian as spoken in Italy **ja_JP** Japanese as spoken in Japan **ko_KR** Korean as spoken in Korea **pl_PL** Polish as spoken in Poland **pt_BR** Portuguese as spoken in Brazil **ru_RU** Russian as spoken in Russia **zh_CN** Chinese as spoken in China **zh_TW** Chinese as spoken in Taiwan
Value Localization	Type the value for the display name.
Status	Select the status of the text change in the approval life cycle: **Approved** The text change is approved for use. **Invalid** The text change is not valid. **Pending** The text change is pending approval. **Review** The text change is in review.

© 2025 Siemens

For this option	Do this
	Tip: You can export text for translation based on its status using the **l10n_import_export** utility.

5. Click **Finish**.

Set display names for properties

Properties display information about objects in Teamcenter, such as name, creation date, owner, and so on. Using the Business Modeler IDE, you can define the name that displays for properties in the Teamcenter user interface.

1. In the Business Modeler IDE, open a business object and click the **Properties** tab.

2. Select a string property in the table for which you want to change the display name, for example, **object_name**.

3. Click the **Override**, **Edit**, or **Add** buttons to the right of the **Localization** table to change the text.

Note:

To be proficient with properties, you need to know both the internal name of the property and its display name. You can change the settings in the rich client to display the internal name of a property in the user interface. Log on to the rich client as an administrator, choose **Edit**>**Options**, and in the left pane of the **Options** dialog box, choose **Options**>**General**>**UI**. In the right pane, click the **Sys Admin** tab and select **Real Property Name**. To verify the change, select an item in the rich client and choose **View**>**Properties**.

Tip:

You can change how values for properties are displayed in the Active Workspace user interface by using **property formatters**.

Set display names for lists of values (LOVs)

Lists of values (LOVs) are the pick lists displayed in Teamcenter when end users click an arrow in a data entry box. Using the Business Modeler IDE, you can define the text that displays for string LOVs in the Teamcenter user interface.

1. In the Business Modeler IDE, open the **Extensions\LOV** folders.

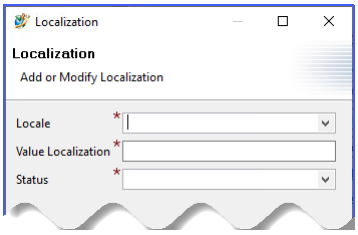
© 2025 Siemens

2. Open an LOV and select a value in the LOV table.

3. Click the **Localization** button to the right of the table.

   The **LOV Value Localization** dialog box is displayed.

   The **Localization** dialog box appears.



4. In the **LOV Value Localization** dialog box, click the **Override**, **Edit**, or **Add** button.

   The **Localization** dialog box is displayed.

5. Perform the following in the **Localization** dialog box:

   a. In the **Value Localization** box, type the value as you want it to display in the user interface.

   b. Click the arrow in the **Locale** box to select the language locale where the text is used.

   c. If a description was previously entered for the LOV value, in the **Description Localization** box, type a description for the display text.

   d. In the **Status** box, select the status of the text change in the approval life cycle.

   e. Click **Finish**.

6. Click **Finish** in the **LOV Value Localization** dialog box.

Validating localizations

The display names (localizations) entered in the Business Modeler IDE are validated to ensure that the characters used in the display names are valid for a given locale. The same validations are done by the Business Modeler IDE parser whenever you open, reload, or import a Business Modeler IDE project to the Business Modeler IDE client.

To achieve this validation, the Business Modeler IDE now installs the **textsrv_text.xml** file into the *TC_ROOT*/**lang/textserver/no_translation** directory.

© 2025 Siemens

The **textsrv_text.xml** file contains encodings per locale that are used by the Business Modeler IDE client to validate the display names. This file contains many XML entries, but the following sample code shows how encodings are stored per locale:

```
<!-- SECTION DEFINING THE SMALLEST OR CUSTOM ENCODING FOR EACH
LOCALE: THIS IS USED FOR BMIDE LOCALIZATION VALIDATION -->
<key id="locale_validation_encoding_en_US">us-ascii</key>
<key id="locale_validation_encoding_cs_CZ">iso-8859-2</key>
<key id="locale_validation_encoding_pl_PL">iso-8859-2</key>
<key id="locale_validation_encoding_de_DE">iso-8859-1</key>
<key id="locale_validation_encoding_es_ES">iso-8859-1</key>
<key id="locale_validation_encoding_fr_FR">iso-8859-1</key>
<key id="locale_validation_encoding_it_IT">iso-8859-1</key>
<key id="locale_validation_encoding_pt_BR">iso-8859-1</key>
<key id="locale_validation_encoding_ja_JP">shift_jis</key>
<key id="locale_validation_encoding_ko_KR">euc_kr</key>
<key id="locale_validation_encoding_ru_RU">iso-8859-5</key>
<key id="locale_validation_encoding_zh_CN">gb2312</key>
<key id="locale_validation_encoding_zh_TW">big5</key>
```

In the XML sample code, not only are the encodings stored per locale, they are available only for the locales included with standard Teamcenter. Currently, each locale is set to its minimum encoding. For example, the encoding for **en_US** is **US-ASCII**. This means if you enter display names (localizations) in the Business Modeler IDE for **en_US**, the Business Modeler IDE validates that the characters entered in the **Display Name** box are valid in the **US-ASCII** character set. The same logic applies to localizations entered in other locales.

The encodings supplied in the **textsrv_text.xml** file are used *only* by the Business Modeler IDE client for validations on localizations during the loading of a Business Modeler IDE project or when users enter display names in the user interface. These encodings are not used during template deployment. When you install your template to the database, the settings in the **textsrv_text.xml** file are not used. Instead, the Business Modeler IDE deploy utilities use the encoding set for the database and the server host.

Encodings in the **textsrv_text.xml** file are the minimum encodings. You may use a larger encoding because your database supports it. For example, you may want to use the Euro symbol in **en_US** files because your database encoding is set to **ISO-8859-15**. In such cases, you can modify the entries in the **textsrv_txt.xml** file and provide your larger encoding for each of the locales. For example, you change the encoding for **en_US** as **ISO-8859-15**:

```
<key id="locale_validation_encoding_en_US">iso-8859-15</key>
```

You can change the encoding for each of the locales (shown in the previous code example) to use an encoding that is supported in your database. The Business Modeler IDE then uses your custom encoding to validate the localizations in the Business Modeler IDE client.

Teamcenter includes localizations only for a certain list of locales. For example, Teamcenter does not include localization for the Romanian (**ro_RO**) locale. You may be supporting the Romanian locale and if you want the Business Modeler IDE client to validate all localizations entered in the Romanian locale,

© 2025 Siemens

you must modify the **textsrv_text.xml** file and add an entry for this locale. For example, you can add the following:

```
<key id="locale_validation_encoding_ro_RO">iso-8859-16</key>
```


© 2025 Siemens

© 2025 Siemens

14. Configure Teamcenter applications

Access Manager

Configure Access Manager using the Business Modeler IDE

Access Manager defines rules that control who can access objects. Use the Business Modeler IDE to create custom objects used by the Access Manager application. COTS Access Manager objects are provided by the Foundation template. No additional templates are needed.

Create a custom privilege

If you want to create a custom privilege using the Business Modeler IDE, you must define the new privilege in the **am_text_locale.xml** file and then add the new privilege to the database using the Business Modeler IDE. There are two attributes of a custom privilege that you must manually define in the **am_text_locale.xml** file:

- Privilege name

  Specifies the unique identifier that Teamcenter uses to store privileges in the database. (The display name of the privilege name property is maintained in the Business Modeler IDE.)

- Privilege token

  Specifies the single-letter mnemonic for that privilege (for example, **R** is the token for the default read privilege).

1. Manually add the following entries to the *TC_ROOT***lang\textserver***language**\am_text_locale.xml** file:

   - A name entry as follows:

     ```
     <key id="k_am_priv_NAME">NAME</key>
     ```

   - A token entry as follows:

     ```
     <key id="k_am_token_NAME">T</key>
     ```

     > Tip:
     >
     > *NAME* is the name and *T* is the single-letter token you define for this new custom privilege.

2. Add the new privilege to the database using the Business Modeler IDE by adding children to the **AM_privileges** business object.

© 2025 Siemens

ADA License

Configure ADA License using the Business Modeler IDE

ADA License provides support to enforce the International Traffic in Arms Regulations (ITAR) and intellectual property (IP) policies using authorized data access (ADA) licenses. Use the Business Modeler IDE to create custom objects used by the ADA License application. COTS ADA License objects are provided by the Foundation template. No additional templates are needed.

Add ADA License categories

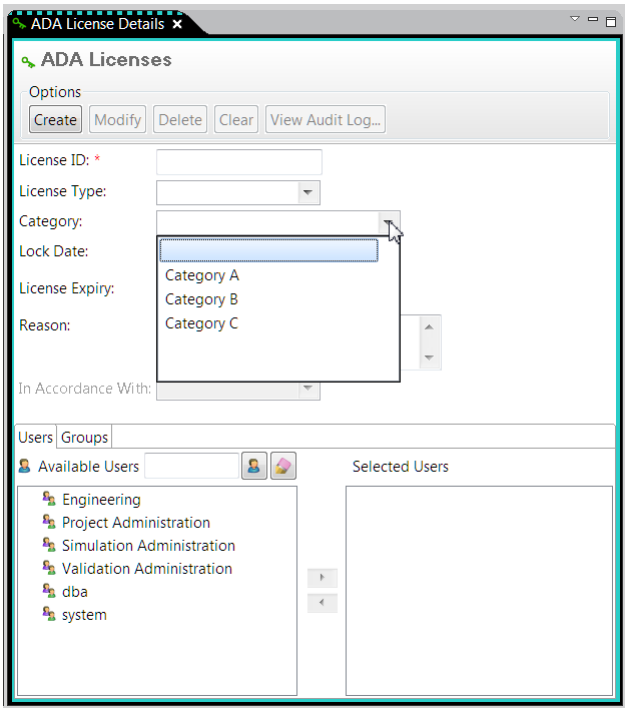
Users of the ADA License application can select license categories from the menu in the **Category** box in the ADA License application.

These categories are defined in the **Fnd0ADALicenseCategories** list of values (LOV) found in the Business Modeler IDE. If you want to change the listed categories, you can change the values in the LOV.

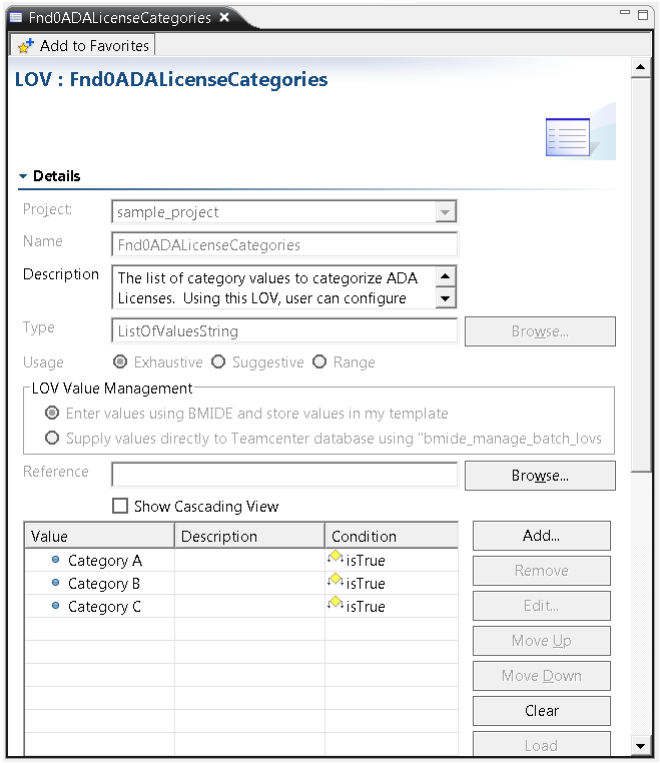
> Note:
>
> You can use a process similar to the following to add a list of citizenships to the **User Citizenships** box in the ADA License application by using the Business Modeler IDE to attach a list of values to the **fnd0user_citizenships** property on the **ADA_License** business object.

1. To see the available default categories in the ADA License application, click the arrow in the **Category** box.

© 2025 Siemens

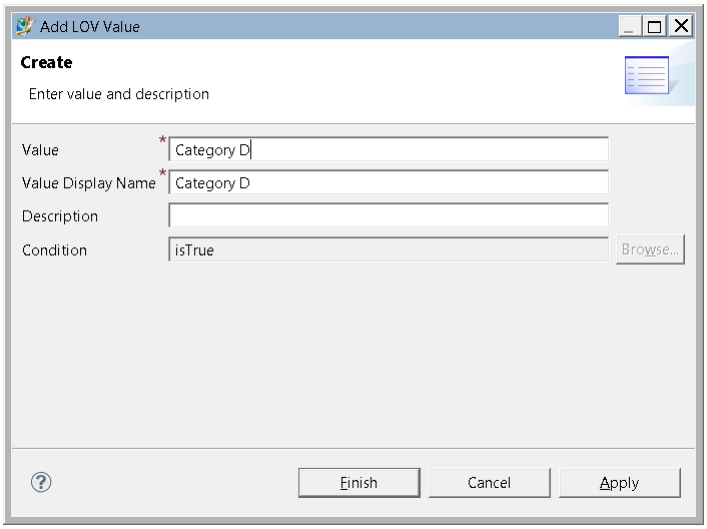


2. In the Business Modeler IDE, open the **Extensions\LOV** folders and double-click the **Fnd0ADALicenseCategories** list of values.



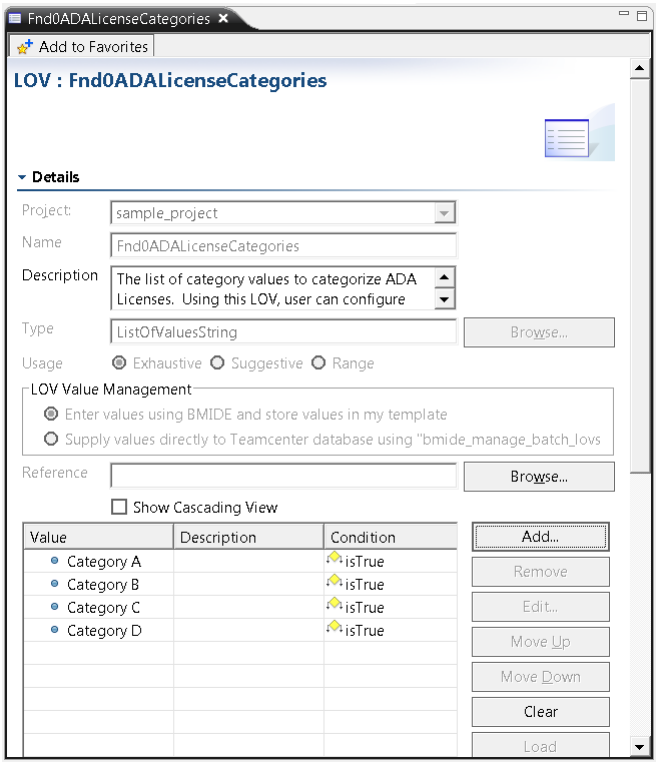
© 2025 Siemens

3. To add a new category, click the **Add** button to the right of the table and type the new value. (You can also remove the default values and add completely new values, if desired.)



4. Click **Finish**.

   The new value is displayed on the table of values.

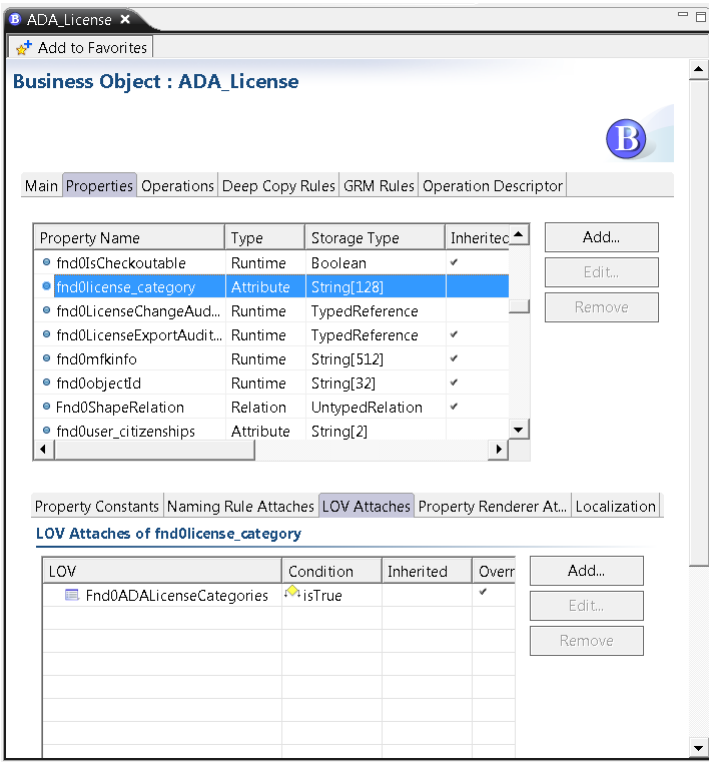


5. The **Category** box in the ADA License user interface is defined by the **fnd0license_category** property. To see how the categories list of values is attached to the **Category** box, open the

© 2025 Siemens

ADA_License business object, click the **Properties** tab, and scroll to the **fnd0license_category** property in the table. The **Fnd0ADALicenseCategories** list of values is shown as attached to this property in the **LOV Attaches** table.

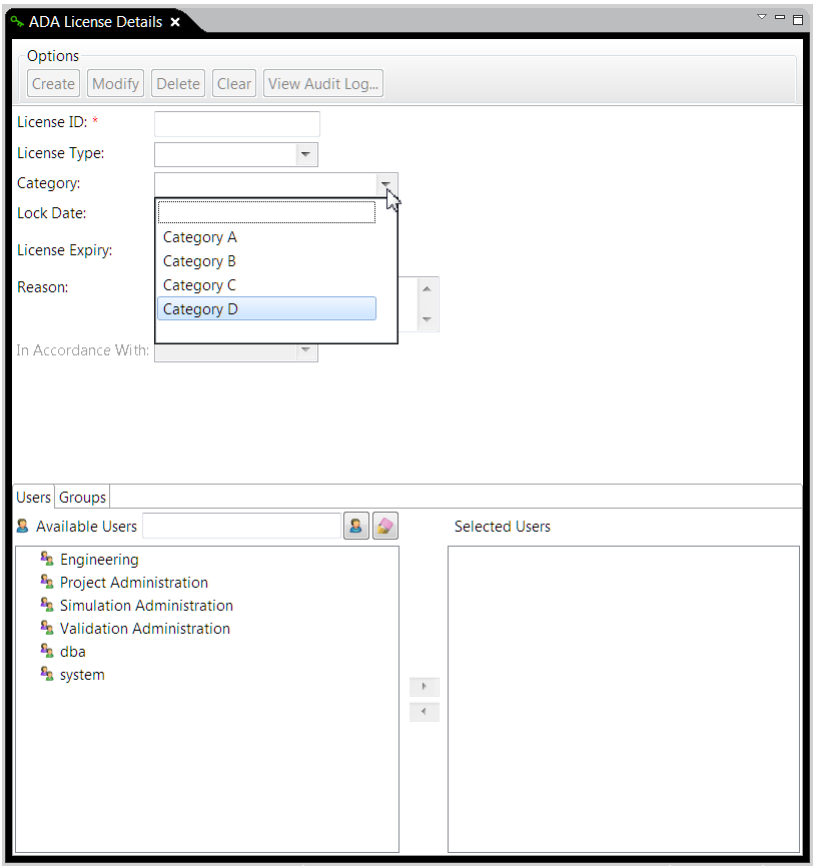
> Note:
>
> If you want to create your own set of license categories in another list of values, you can attach your custom LOV to the **fnd0license_category** property and use it instead of the **Fnd0ADALicenseCategories** LOV.



6. To save your changes to the template, on the menu bar choose **BMIDE→Save Data Model**. Then **package the template** and use Teamcenter Environment Manager to install the packaged template to your server.

7. To see the new category in the ADA License application, click the arrow in the **Category** box.

© 2025 Siemens



8. Now that you have added categories, try the following additional configurations:

   - Configure a different set of categories for each license type.

     a. Create a category LOV for each license type (ITAR, Exclude, and IP), each containing the list of categories unique to that license type.

     b. Attach the appropriate LOV to the **fnd0license_category** property on each of the license type business objects (**ITAR_License**, **Exclude_license**, and **IP_license**).

     c. When the end user opens the ADA License application and selects a license type in the **License Type** box, the corresponding categories appear in the **Category** box.

   - Configure a different set of categories for each user group.

     a. Create a category LOV for each user group (for example, dba, designer, and so on), each containing the list of categories unique to that user group.

     b. Create a condition for each user group, for example:

        Signature: *condition-name* **(UserSession o)**
        Expression: **o.group_name="dba"**

© 2025 Siemens

c. Open the **ADA_License** business object, select the **fnd0license_category** property, and in the **LOV Attaches** table, add each user group category LOV using the condition for that user group.

d. When the end user logs on to the ADA License application as a member of a user group, the categories belonging to that user group appear in the **Category** box.

Aerospace and Defense

Configure Aerospace and Defense using the Business Modeler IDE

Use the Business Modeler IDE to create custom objects used by the Aerospace and Defense solution.

> Note:
>
> Before working with Aerospace and Defense objects, you must **install the following templates** to your project:
>
> - Aerospace and Defense Foundation (**adsfoundation_template.zip** file)
> - Aerospace and Defense Change Management (**adschangemanagement_template.zip** file)
> - Aerospace and Defense Training (**adstrainingprogram_template.zip** file)

Aerospace and Defense business objects

The Aerospace and Defense templates provide the following business objects. You can create your own custom business objects as children of these business objects.

- **ADSDesign**

  Represents the geometric data of a component or assembly. This is a child of the **Design** business object.

- **ADSDrawing**

  Represents a drawing for a technical document. This is a child of the **Drawing** business object.

- **ADSPart**

  Represents a component of a product. This is a child of the **Part** business object.

- **ADSTechDocument**

© 2025 Siemens

Represents the technical document used in document-centric programs. This is a child of the **Document** business object.

- **CommercialPart**

  Represents the common-use parts that have been identified as standard design by a company, an industry, or the military. This is a child of the **Part** business object.

- Aerospace and Defense relation business objects:

  - **ADS_Lists_Parts**

    Represents the association between a technical document revision and an ADS part or ADS design item.

  - **ADS_Lists_PartRevisions**

    Represents the association between a technical document revision and ADS part revisions or ADS design revisions.

  - **ADS_Lists_DrawingRevisions**

    Represents the association between a technical document revision and an ADS drawing revision.

  - **TC_Program_Preferred_Items**

    Represents the association between a standard item and a program.

The following properties control use of location codes, which are used in Aerospace and Defense:

- **fnd0CurrentLocationCode**

  Stores the current location code for the item revision. This property is found on the **ItemRevision** business object.

- **fnd0LocationCodePref**

  Automatically assigns the original location code to an item when it is initially created. This property is found on the **TC_UserContext** business object.

- **fnd0LocationType**

  Captures location type for a company location. This property is found on the **CompanyLocation** business object.

- **fnd0OriginalLocationCode**

© 2025 Siemens

Stores the original location for the item. This property is found on the **Item** business object.

This property is optional by default. You can make it required using the **Required** property constant on Aerospace and Defense business objects (for example, on the **ADSPart**, **ADSDesign**, **ADSTechDocument**, **ADSDrawing**, **Ads0StdNote**, or **Ads0CustomNote** business objects).

Audit Manager

Configure Audit Manager using the Business Modeler IDE

System administrators use Audit Manager to create audit logs. Audit logs track what information has changed and who has changed the information. Use the Business Modeler IDE to create custom objects used in the Audit Manager. COTS Audit Manager objects are provided by the Foundation template. No additional templates are needed.

Use the Business Modeler IDE to create custom audit events, event mapping, and audit definition objects used in the Audit Manager. To create these objects, in the **Extensions** folder, open the **Audit Manager** folder and right-click the **Audit Definitions**, **Event Type Mappings**, or **Event Types** folders and choose the **New** command.

The typical flow for creating Audit Manager objects is as follows:

1. Set preferences.

   - **HiddenPerspectives**

     Remove **AuditManager** from the list of hidden perspectives.

   - **TC_audit_manager**

     Ensure that the preference is set to **ON**.

   - **TC_audit_manager_version**

     Ensure that the preference is set to **3**.

     > Note:
     >
     > Use the **1** value for legacy auditing (only workflow, checkout, and checkin auditing) and the **2** value to use the Audit Manager application in the rich client solely to manage audit definitions.

2. **Create a new event type** to specify the event for which audit logs are to be written.

3. **Create an event type mapping** to connect the event to a business object type.

© 2025 Siemens

4. **Create an audit definition** to define the information that needs to be captured when an event occurs to a specific business object instance.

Create an event type

An *event* is an action that occurs to an object in Teamcenter, for example, when an item is checked out. Teamcenter records audit logs when certain events occur on certain types of objects.

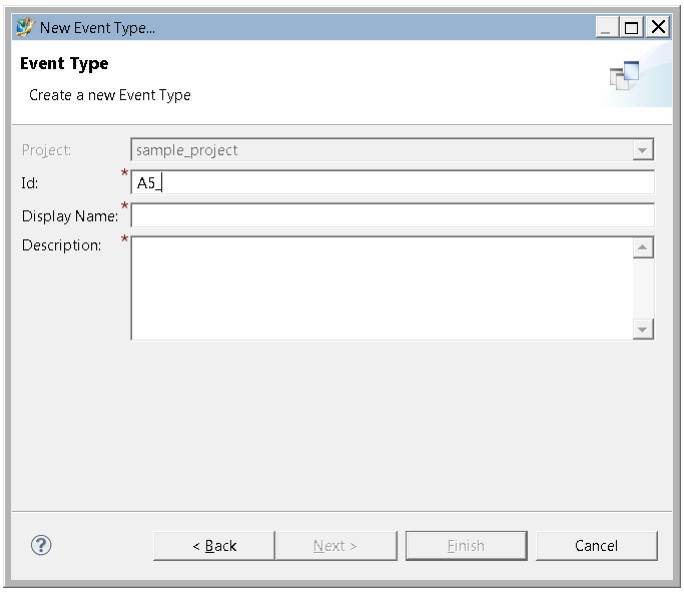
You only need to create a new event type if there is not an existing event type that covers your needs. When you create a type, its name is only a text reminder of the type of information you are looking from in the audit. The actual event information is captured by the audit type selected when you create the event type mapping.

In the past, the **install_event_types** utility was used to create new events. Now you create new event types using the Business Modeler IDE.

1. Choose one of these methods:

   - On the menu bar, choose **BMIDE→New Model Element**, type **Event Type** in the **Wizards** box, and click **Next**.
   - Open the **Extensions\Audit Manager** folders, right-click the **Event Types** folder, and choose **New Event Type**.

   The New Event Type wizard runs.



2. In the **Id** box, type the name of the new event.

3. In the **Display Name** box, type the name that you want the event to have in the user interface.

© 2025 Siemens

4. In the **Description** box, type a description of the new event so that others know what it is used for.

5. Click **Finish**.

6. **Create an event type mapping definition** to connect the event to a business object type.

Create an event type mapping

While an *event* is an action that occurs to an object in Teamcenter, *event mapping* is connecting an event to a business object type. In other words, the event mapping declares that you want to receive an audit log for a certain event on a certain kind of object.

An event mapping must be created for a business object type and event before you use that business object and event type in an audit definition. Event mapping is inherited by child business object types. For example, instances of the **Part** business object type inherit the mapping from the **Item** business object type.

In the past, the event mapping was created using the **install_event_types** utility. Now event mapping is created using the Business Modeler IDE.

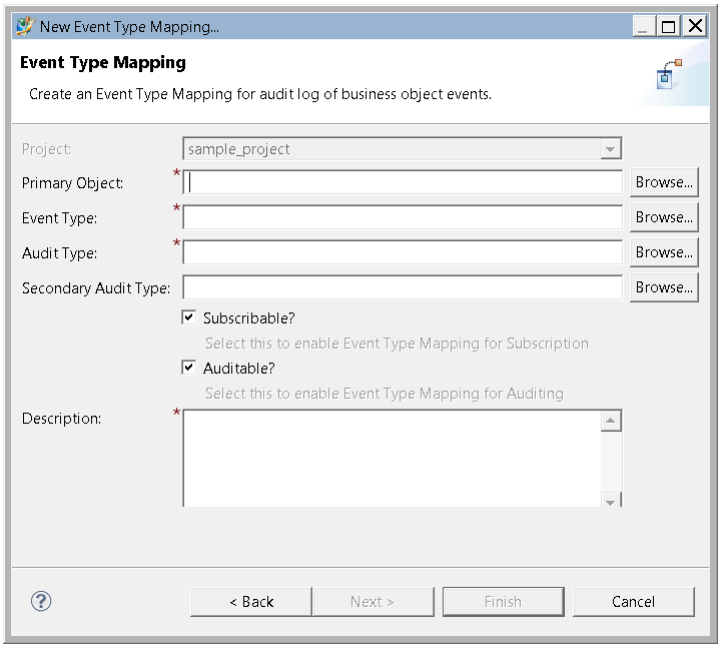
1. Choose one of these methods:

   - On the menu bar, choose **BMIDE→New Model Element**, type **Event Type Mapping** in the **Wizards** box, and click **Next**.

   - Open the **Extensions\Audit Manager** folders, right-click the **Event Types Mappings** folder, and choose **New Event Type Mapping**.

   The New Event Type Mapping wizard runs.

© 2025 Siemens



2. Click the **Browse** box to the right of the **Primary Object** box to select the type of business object you want to audit.

3. Click the **Browse** box to the right of the **Event Type** box to select the event you want to audit for the selected business object.

4. Click the **Browse** box to the right of the **Audit Type** box to select the type of audit to use for this mapping. The audit types are represented by business objects that are children of the **Fnd0AuditLog** business object.

5. Click the **Browse** box to the right of the **Secondary Audit Type** box to select the **Fnd0SecondaryAudit** business object. This **Secondary Audit** object stores information and properties about the secondary objects that are related to the main object being audited.

6. Select the **Subscribable?** check box to specify that the event type mapping can be subscribed to.

7. Select the **Auditable?** check box to specify that the event type mapping can be audited.

8. In the **Description** box, type a description for this mapping so that others know what it is used for.

9. Click **Finish**.

Enable auditing workflow release status objects

Teamcenter supports auditing the **Release Status** property of workflow target objects. Although this is not enabled out of the box, it can be configured using the Business Modeler IDE.

© 2025 Siemens

The following steps describe configuration changes to enable auditing the **Release Status** property of workflow target objects.

1. Start the Business Modeler IDE.

2. (Optional) Create a Business Modeler IDE template project.

3. Expand the project and the **Extensions\Audit Manager** folders.

4. Expand the **Event Type Mappings** folder and double-click the **EPMTask:__Complete** event type mapping.

5. Click the **Browse** box to the right of the **Secondary Audit Type** box to select the **Fnd0SecondaryAudit** business object.

6. From the **Event Type Mappings** folder, double-click the **EPMJob:__End** event type mapping.

7. Click the **Browse** box to the right of the **Secondary Audit Type** box to select the **Fnd0SecondaryAudit** business object.

8. To save the changes to the data model, choose **BMIDE→Save Data Model**, or click the **Save Data Model** button on the main toolbar.

9. Deploy your changes to the server to enable auditing of **Release Status** property of workflow target objects. Choose **BMIDE→Deploy Template** on the menu bar, or select the project and click the **Deploy Template** button on the main toolbar.

10. After deployment, test your new event type mapping in the Teamcenter rich client. Initiate a workflow against a product revision. As needed, customize the process history display and review the progress of the workflow in the **Process History** view.

Create an audit definition

The following procedure shows how to create an *audit definition* that defines information to be captured when an event occurs to a particular kind of object.

1. Ensure the following:

   - An event mapping has been created between the business object type and the event.

   - If you want to set a specific condition for activating logging of an event, then ensure that the condition is defined.

© 2025 Siemens

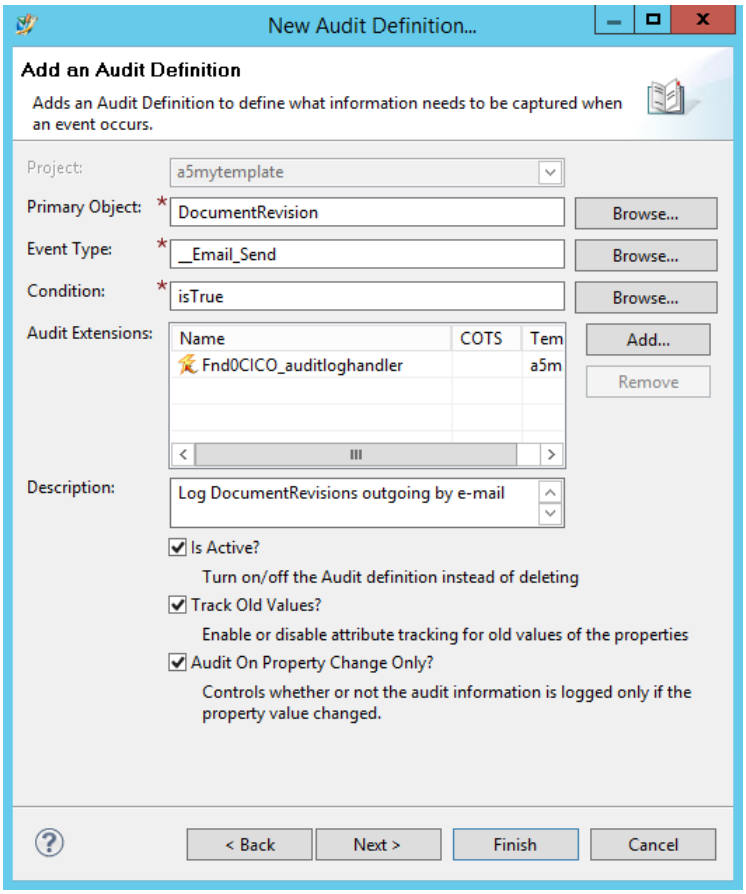
Example:

You may want an audit entry to be written when a user accesses datasets that are export controlled, but not datasets that are not export controlled. Before creating the audit definition, create the condition definition.

2. Start the New Audit Definition wizard in one of these ways:

   - On the menu bar, choose **BMIDE→New Model Element**, type **Audit Definition** in the **Wizards** box, and click **Next**.

   - Open the **Extensions\Audit Manager** folders, right-click the **Audit Definitions** folder, and choose **New Audit Definition**.



3. Specify basic parameters for the audit definition.

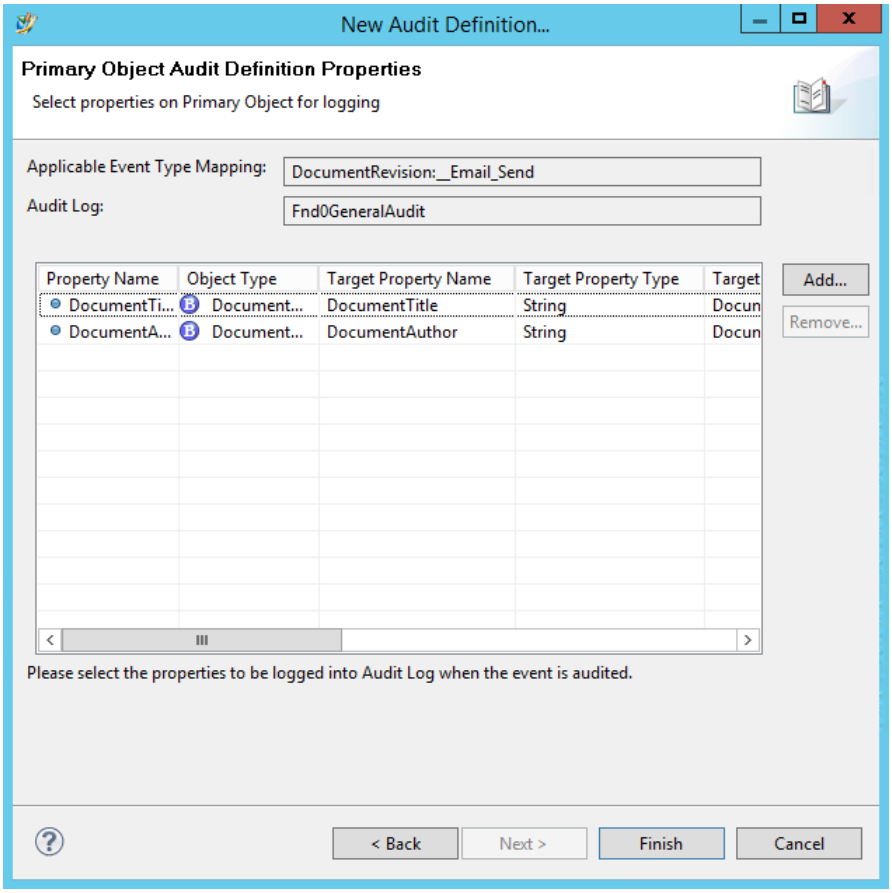
© 2025 Siemens

For this parameter	Do this
Primary Object	Click **Browse** and select the primary business object type that you want to audit.
Event Type	Click **Browse** and select the event that you want to audit for the selected business object.
Condition	Click **Browse** and select the condition under which the audit definition applies. Note: If the specified condition criteria are not matched, the audit log will always create. If you specify the **Delete** event type, then in the condition specify only the **UserSession** parameter.
Audit Extensions	Click **Add** and select the log extensions to use in the definition.
Description	Type a description of the purpose for this audit definition.
Is Active?	Select the check box to turn on the audit definition.
Track Old Values?	Select the check box to enable tracking of the old values of primary object properties. Note: Attribute tracking is available only for primary objects and is not supported for secondary objects.
Audit on Property Change Only?	Select the check box to log the information specified in this audit definition only if the property values change. This functionality is enabled only if the **Track Old Values?** check box is selected.

4. Click **Next**.

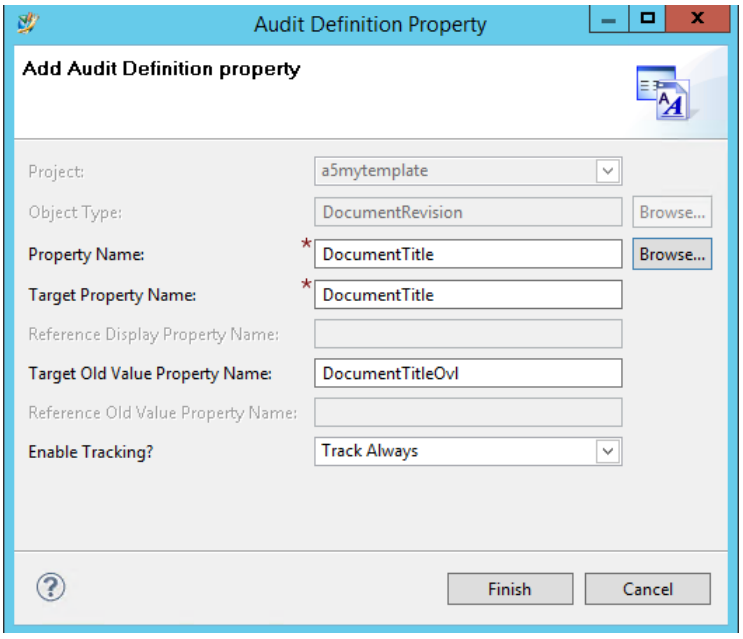
The **Primary Object Audit Definition Properties** dialog box is displayed.

© 2025 Siemens



a. In the **Primary Audit Definition Properties** dialog box, click **Add**.

The **Add Audit Definition property** dialog box is displayed.



© 2025 Siemens

b. In the **Add Audit Definition property** dialog box, to the right of **Property Name**, click **Browse** and select a property to track.

c. To change the display name of the property in audit logs, in **Target Property Name** type a new display name.

d. The **Target Old Value Property Name** box and the **Enable Tracking?** box are enabled if you selected the **Track Old Values?** check box in the **Add an Audit Definition** dialog box. As applicable, enter additional parameters for the audit definition property.

Parameter	Description
Target Old Value Property Name	To change the display name of the old property value in audit logs, type a new display name.
Enable Tracking?	Select the kind of tracking: **Track Always** Always tracks old and new values of properties, even if there are no changes to the property value. **No** Does not track changes to properties. **Track Different** Tracks old and new values of properties only when the property value changes.

e. Click **Finish**.

5. Add more primary object audit definition properties as needed.

6. (If the primary object can have secondary objects) To capture information from secondary objects that are related to or referenced by the main object being audited, click **Next** in the **New Audit Definition** dialog box and add secondary object audit definition properties.

7. When you are done adding properties, click **Finish** in the **New Audit Definition** dialog box.

Test the audit definition

Verify that the audit definition object can create audit logs.

© 2025 Siemens

Post-upgrade steps required for importing custom event types into a template project

If you upgrade a Business Modeler IDE template project from a version earlier than Teamcenter 10, then after you upgrade the Business Modeler IDE template project, you must add its legacy custom event types. This is because Audit Manager objects are now managed using the Business Modeler IDE.

To help you import these custom event types, the system identifies the custom event types definitions during the upgrade process and writes them to a **custom_audit_configurations.xml** file generated under the *TC_DATA***model** directory. At the end of the upgrade process, Teamcenter Environment Manager (TEM) issues a warning if there are any custom event types.

Postupgrade, import these custom event type definitions into your custom template project before deploying any changes to the upgraded database. If not, the next TEM update process or Business Modeler IDE deployment tries to delete these event types, which may or may not pass based on whether there are references to it in the database.

Perform the following steps in the Business Modeler IDE immediately after the successful upgrade to Teamcenter and before deploying any data model changes:

1. Import the **custom_audit_configurations.xml** file from the *TC_DATA***model** directory into your custom template project by choosing **File→Import→Business Modeler IDE→Import template file**.

2. In the **BMIDE** view, right-click the project and choose **Reload Data Model**. Make sure there are no model errors reported in **Console** view.

3. Populate the appropriate display names to the custom event types wherever necessary.

4. Package and deploy the template to the Teamcenter database.

You can configure Audit Manager using the Business Modeler IDE.

Audit Manager data model objects

The Foundation template provides the COTS data model objects used by Audit Manager, including:

- Business objects
  - **Fnd0AuditLog**

    Defines the available types of audit logs to be used in event type mapping.

    - **Fnd0FileAccessAudit**

      Holds file access audit records. Shown as **File Access Audit** in the rich client user interface.

© 2025 Siemens

- **Fnd0GeneralAudit**

  Holds audit records for which the object type and event type combination are not defined in any other audit log business objects. Shown as **General Audit** in the user interface.

- **Fnd0LicenseChangeAudit**

  Holds license change audit records. Shown as **License Change Audit** in the user interface.

- **Fnd0LicenseExportAudit**

  Holds license export audit records. Shown as **License Export Audit** in the user interface.

- **Fnd0OrganizationAudit**

  Holds organization audit records. Shown as **Organization Audit** in the user interface.

- **Fnd0ScheduleAudit**

  Holds schedule audit records. Shown as **Schedule Audit** in the user interface.

- **Fnd0StructureAudit**

  Holds structure audit records. Shown as **Structure Audit** in the user interface.

- **Fnd0WorkflowAudit**

  Holds process and signoff history audit records. Shown as **Workflow Audit** in the user interface.

- Extensions

  The following audit extensions define the log handlers to use in an audit definition:

  - **Fnd0CICO_auditloghandler**

    Logs checkin and checkout information, change ID, and the reason to audit. Applies to checkin and checkout events.

  - **Fnd0OCC_track_position_orientation_audithandler**

    Logs occurrence position and orientation changes of components in structures.

  - **Fnd0PROJInfo_audithandler**

    Logs project names that are assigned to the project. The project names are comma separated.

© 2025 Siemens

- **Fnd0USER_get_additional_log_info**

  Logs workflow information to audit logs. For example, for the **__Assign** event, this handler logs information such as the process name, task type, user comments, and the user ID and user name the workflow is assigned to.

- **Fnd0WriteSignoffDetails**

  Logs the workflow signoff history.

Automotive Edition

Configure Automotive Edition using the Business Modeler IDE

Use the Business Modeler IDE to configure business rules used by the Automotive Edition application.

> Note:
>
> Before working with Automotive Edition objects, you must **install the following templates to your project**:
>
> - Teamcenter Automotive Edition (**tcae_template.zip** file)
> - GM Overlay (**gmo_template.zip** file)

CAE Manager

Configure CAE Manager using the Business Modeler IDE

Use the Business Modeler IDE to create custom objects used by the CAE Manager application.

CAE Manager objects are provided by the Foundation template. No additional templates are needed.

Change Management

Configure Change Management using the Business Modeler IDE

Use the Business Modeler IDE to create custom objects used by the Change Manager application. The custom objects you create are templates of the different change processes to be used at your site. End users create instances of these change objects, such as change requests, and use them in their workflow processes.

© 2025 Siemens

Note:

Before working with these objects, you must **install the Change Management template (cm_template.xml file) to your project**.

Create change business objects if you want to create your own change notices, requests, or problem reports, or if you want to create new change relationships.

1. If you have not already done so, **create a custom template project** to hold your data model changes.

2. Ensure that you have installed the Change Management template (**cm_template.xml** file) to your project.

3. Create children **Change Management business objects** as needed.

   - Item business objects

     To create your own change notices, requests, or problem reports, add children under the **ChangeNotice**, **ChangeRequest**, or **ProblemReport** business objects.

   - Revision business objects

     To create your own change revision objects, add children under the **ChangeNoticeRevision**, **ChangeRequestRevision**, and **ProblemReportRevision** business objects.

   - Relation business objects

     To create your own change relations, add children under the **CMRelation** business object.

4. Make the new business objects creatable by making new conditions.

   Each Change Management business object has a corresponding condition that makes it creatable in the user interface. The conditions follow this naming convention:

   *naming-prefix***Cm0is***business-object-name***Creatable**

   Search on conditions containing the word **Creatable** for examples, and look at these conditions to see how to create your own conditions.

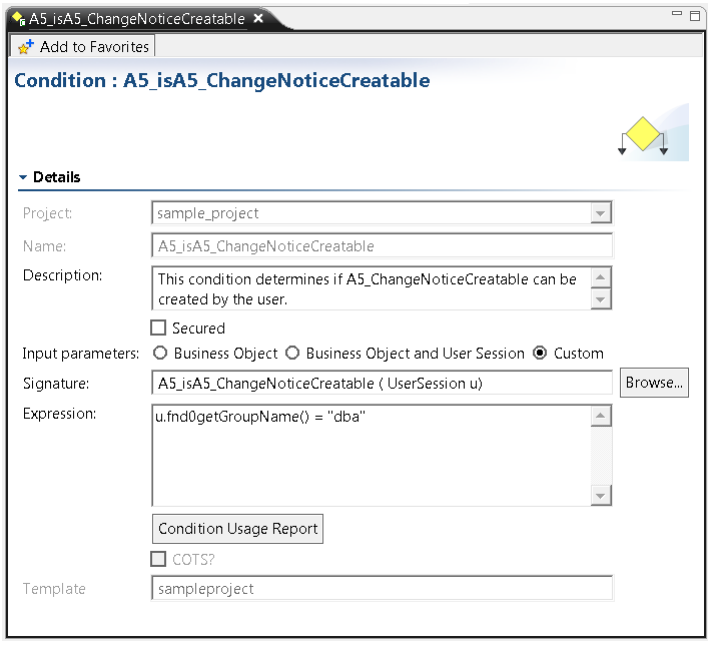
   For example, if you created a child of the **ChangeNotice** business object, look at the **Cm0isChangeNoticeCreatable** condition for an example of how to create yours. Or if you created a child of the **CMHasImpactedItem** relation business object, look at the **Cm0isCMHasImpactedItemCreatable** condition. (The conditions whose names end in **ForPrimary** and **ForSecondary** are for the primary and secondary relationships, and those whose names end in **ForTask** are for schedule tasks.)

© 2025 Siemens

The following example shows how to create a new condition:

a. If the naming prefix for your project is **A5_**, create a child of the **ChangeNotice** business object named **A5_ChangeNotice**. (You can **name it** what you want, but it must have the naming prefix for your project.)

b. **Create a condition** named **A5_Cm0isA5_ChangeNoticeCreatable** based on the **Cm0isChangeNoticeCreatable**.



5. **Create dynamic participants** for the workflow.

   Children of the **Participant** business object represent participants in the workflow process. For Change Management, these participants include **Analyst**, **ChangeSpecialist1**, and **Requestor**, among others. You can create new participant business object types to represent participants in the workflow process. You then assign keywords to these participants to ensure that they are tied to the correct workflows.

6. Create new forms as needed.

7. Configure pseudofolders as needed using preferences.

Change Management business objects

The Change Management template provides the following business objects. You can **create your own custom change business objects** as children of these business objects.

- **ChangeItem**

© 2025 Siemens

Provides general behavior to support planning capabilities, traceability, and Change Management.

> Caution:
>
> When possible, you should subclass from non-abstract rather than abstract classes. You are allowed to subclass from the generic (**Gn**) classes if you are certain that you do not need any of the properties or behaviors (for example, extensions, condition rules, and so on) on the non-abstract classes in your implementation.
>
> Therefore, use caution when creating a custom business object as a child of the **ChangeItem** business object or its immediate children **Cm0GnWorkOrder**, **GnChangeNotice**, **GnChangeRequest**, or **GnProblemReport**.

- **ChangeNotice**

  Provides the means to formulate a detailed work plan for a set of requests and to execute the plan to closure. After change requests are implemented, a notice should be sent to all the stakeholders notifying them of the change.

- **ChangeRequest**

  Provides the means to formulate a business case for the resolution of a set of problems and enable users to provide technical recommendations. When a problem report is reviewed and verified, a change request must be filed to handle the problem.

- **ProblemReport**

  Describes a problem in such a way that another party can repeat the same steps and duplicate the problem precisely.

- **ChangeItemRevision**

  Provides the revision of the **ChangeItem** business object.

  > Caution:
  >
  > When possible, you should subclass from non-abstract rather than abstract classes. You are allowed to subclass from the generic (**Gn**) classes if you are certain that you do not need any of the properties or behaviors (for example, extensions, condition rules, and so on) on the non-abstract classes in your implementation.
  >
  > Therefore, use caution when creating a custom business object as a child of the **ChangeItemRevision** business object or its immediate children **Cm0GnWorkOrderRevision**, **GnChangeNoticeRevision**, **GnChangeRequestRevision**, or **GnProblemReportRevision**.

  - **ChangeNoticeRevision**

    Provides the revision of the **ChangeNotice** business object.

© 2025 Siemens

- **ChangeRequestRevision**

  Provides the revision of the **ChangeRequest** business object.

- **ProblemReportRevision**

  Provides the revision of the **ProblemReport** business object.

- **CMRelation**

  Defines the top-level Change Management relation; all Change Management relations are children of this business object.

  - **CMHasImpactedItem**

    Provides for change planning in terms of associating business item revisions impacted by the change in context.

  - **CMHasProblemItem**

    Relates the **ChangeItem** object (such as **ProblemReport**, **ChangeRequest** and **ChangeNotice** objects) to problem item revisions

  - **CMHasSolutionItem**

    Provides for the traceability of the deliverable that result from the execution of the product plan. It relates **ChangeNotice** objects to a solution item.

  - **CMHasWorkBreakdown**

    Provides for work breakdown relations.

  - **CMImplements**

    Allows **ProblemReport** objects to be implemented by **ChangeRequest** objects, and **ChangeRequest** objects to be implemented by **ChangeNotice** objects. It is used for navigating change resolution.

  - **CMReferences**

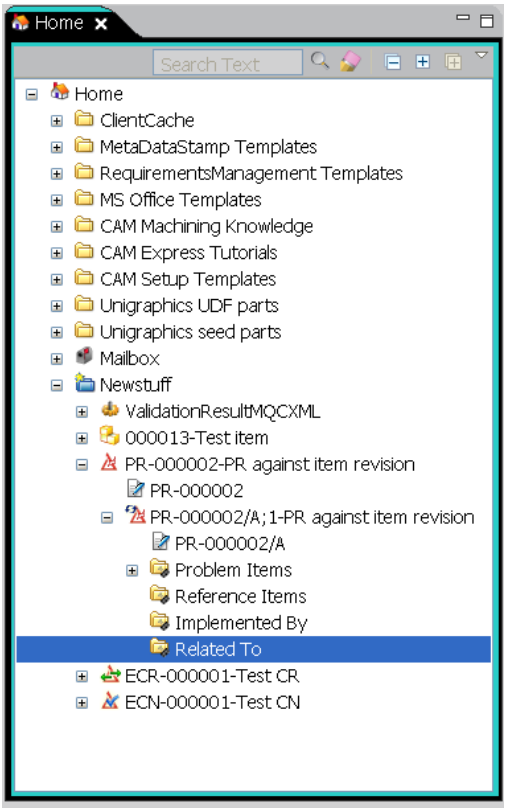
    Allows problem reports, change requests, and change notices to reference items and datasets.

© 2025 Siemens

Create a new relation for use with a Change Management pseudofolder

Folders under Change Management objects (such as change notices, problem reports, and change requests) hold items related to those objects. You can create your own Change Management folder to hold items. You can create the folder for COTS or custom Change Management business objects.

The following example creates a **Related To** folder for problem report objects.



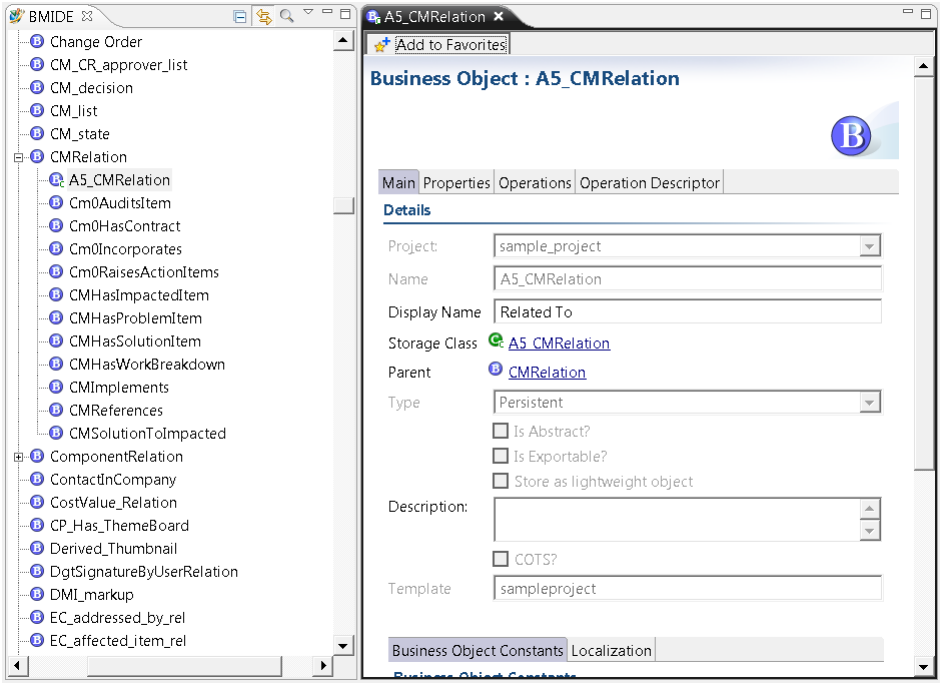
1. If you have not already done so, **create a custom template project** to hold your data model changes.

2. Ensure that you have **installed the Change Management template (cm_template.xml file)** to your project.

3. Create a new relation.

   Create a new business object under the **CMRelation** business object, for example, *naming-prefix***CMRelation**. The display name of the business object is used as the name of the new folder.

© 2025 Siemens



4. Revise the business object to use the new relation.

   a. Open the business object that the relation is to be used for. In this example, because the new folder is to be placed under problem report objects, open the **ProblemReportRevision** business object.

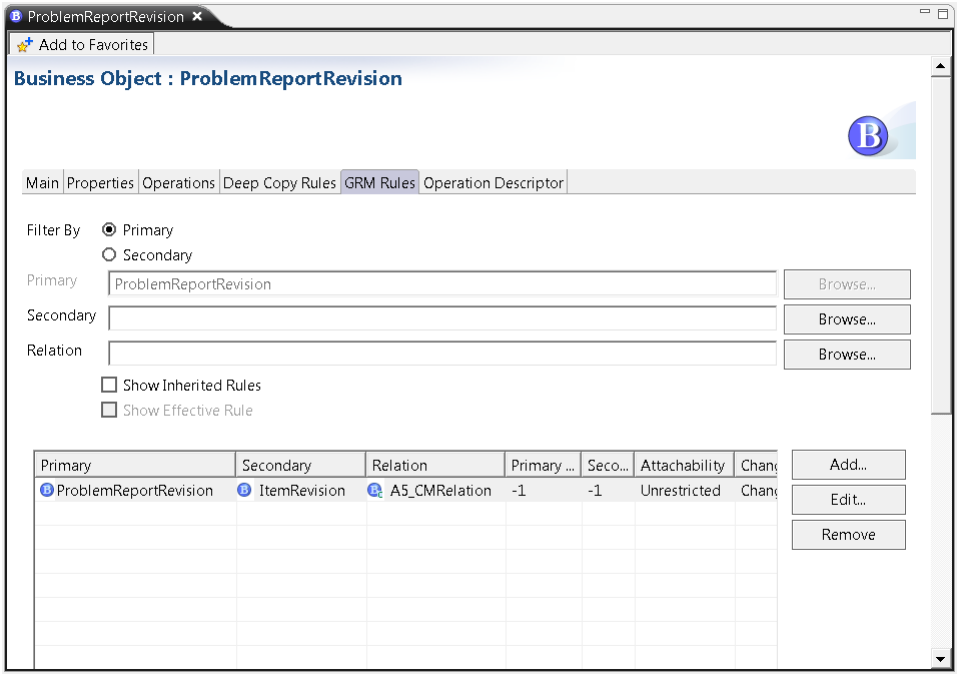
   > Note:
   >
   > You can also open the **ChangeNoticeRevision** business object, **ChangeRequestRevision** business object, or a custom business object that is a child of one of these.

   b. Create a GRM rule.

   Click the **GRM Rules** tab and add any GRM rule based on your business use case.

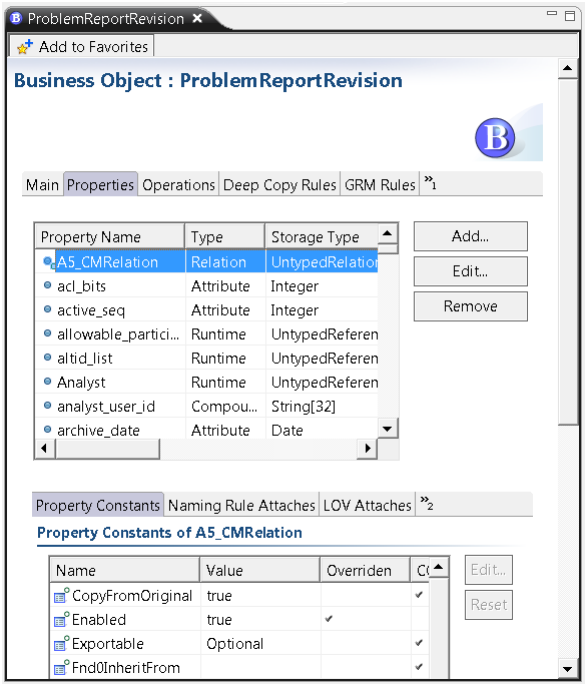
   In this example, select the **ProblemReportRevision** business object as the primary object, select the **ItemRevision** business object as the secondary object, and select the new relation business object as the relation (for example, **CM5_CMRelation**).

© 2025 Siemens



c. Create the relation property.

Click the **Properties** tab and add a relation property with the same name as the new relation business object. (Use the internal name, not the display name.) For examples of relation properties, look at any of the existing relation properties, for example, **CMHasProblemItem**.



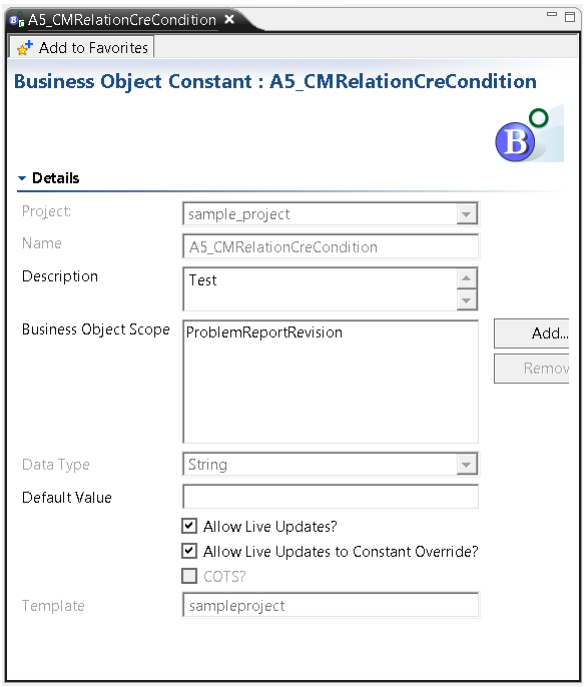
5. Create a business object constant for the new relation.

© 2025 Siemens

a. Create the business object constant.

Add a new business object constant named *new-custom-relation-name***CreCondition** (for example, **CM5_CMRelationCreCondition**).

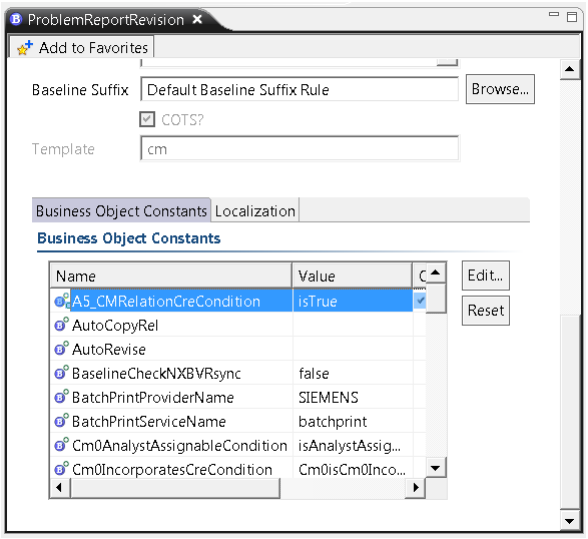
The scope of the constant refers to the scope in which the new relation is applied. Because this example adds the new folder only to problem reports, select the **ProblemReportRevision** business object as the scope. If the relation is to be used across all change objects, select the **ChangeItemRevision** business object.



b. Set the value of the business object constant on the business object.

Open the business object that the constant is scoped to (for this example, the **ProblemReportRevision** business object), and set the condition when the constant applies. For example, if **isTrue** is specified, the creation of the relation is always allowed. The value determines which condition to use to evaluate whether to allow the creation of the relation. You can use any of the existing or custom conditions.

© 2025 Siemens

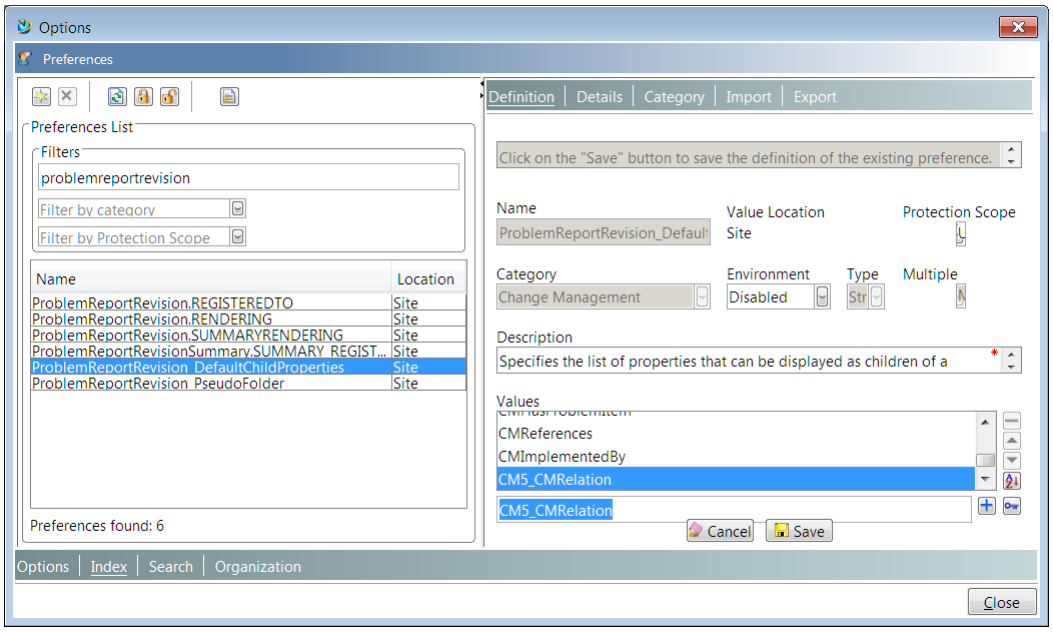


6. Set preferences to use the new relation.

   Add the new relation business object to the value list in the following preferences:

   *business-object-name*_**DefaultChildProperties**
   *business-object-name*_**PseudoFolder**

   For this example, add the new relation to the **ProblemReportRevision** business object.



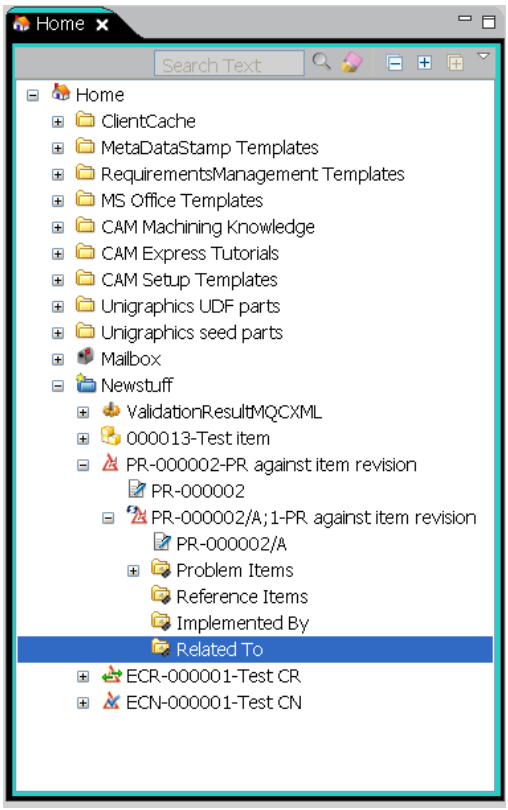
© 2025 Siemens

Note:

These preferences already exist for the COTS Change Management objects, such as the **ProblemReportRevision** business object. For custom business objects, you must create the preferences. The corresponding preferences of the parent object type provide a good starting point for what values should be included.

7. Verify the new folder.

   Deploy the custom template from the Business Modeler IDE to a test server, or package the template and install it using Teamcenter Environment Manager (TEM). When you create a Change Management object for which the new relation was made, the new folder appears.



Add a custom naming rule to standard Change Management objects

If you want to use a custom naming rule to override the COTS naming rule on a standard Change Management object, the **isTrue()** condition cannot be used. You must first create a custom condition that resolves to the **isTrue** condition, and when you override the COTS naming rule, attach the custom naming rule using the custom condition.

For example, you want to override the COTS naming rule on the **item_id** property on the **ProblemReport** business object. Create a custom condition whose expression returns the **isTrue** condition as follows:

© 2025 Siemens

Condition::isTrue()

Once you **override the COTS naming rule** using the custom condition, the custom naming rule works as designed.

Change Management conditions

Change Management conditions validate the objects involved in the change process.

For example, the **isAnalyst** condition determines if the logged-on user is assigned as the analyst. And the **Cm0isCMHasProblemItemCreatable** condition allows the **Problem Items** folder to be updated. To see a listing of all the Change Management conditions, **run the Data Model Documentation report** and look at all the conditions in the **cm** template.

> Note:
>
> You can also obtain a report of the COTS data model in the *Teamcenter Data Model Report* provided in the Teamcenter documentation distribution image. Expand the *DataModelReport.zip* file to a local directory and open *help\en_US\custom\DataModelReport\index.html*.

Some of the conditions have corresponding business object constants. For example, the **Cm0AnalystAssignableCondition** business object constant specifies the condition to determine if the analyst user is assignable and uses the **isAnalystAssignable** condition as its default value. The Change Management conditions begin with **CM**.

Defining deep copy rules for creating changes from another change

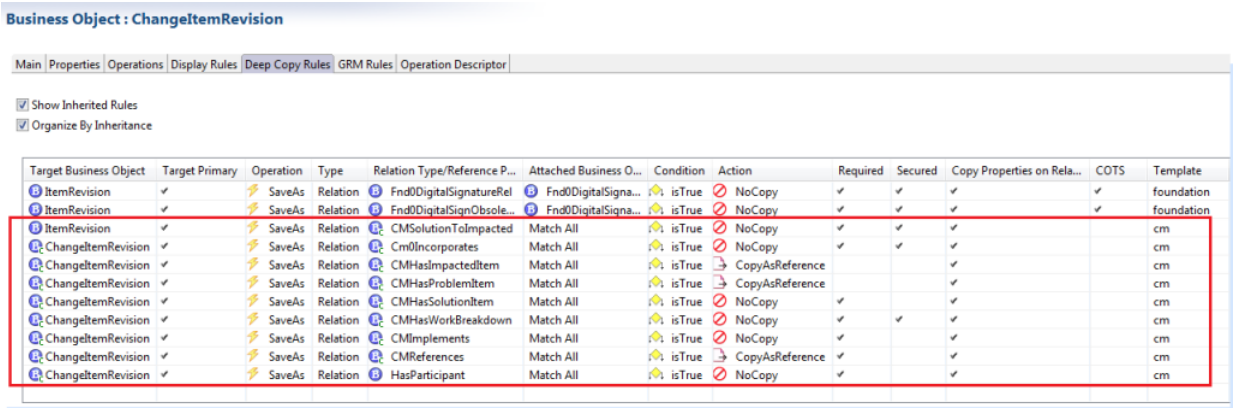
Use the Teamcenter Business Modeler IDE deep copy rules to set what objects and attributes are copied when a user creates a copy of a change from another. Deep copy rules define whether objects belonging to a business object instance can be copied when a user performs a save as or revise operation on that instance. Deep copy rules can be applied to any business object type and are inherited by children business object types.

Using deep copy rules, you can configure whether the following are copied for a change:

- Name, subject, description
- Problem and Impacted Items
- Referenced or related documents

The figure shows the deep copy rules defined by default for a **ChangeItemRevision** in the **Deep Copy Rules** editor. The rules define that when copying a change, copy the problem, impacted, and reference objects, but do not copy the incorporates and solution items.

© 2025 Siemens



Note:

Validate that the setting **copyFromOrignal** is set to true on persistent properties to copy them during a save as or revise operation. If **copyFromOrignal** is set to false on persistent properties, the properties are not copied during save as or revise.

Define deep copy rules for copying options from an ECR to an ECN

Use Teamcenter Business Modeler IDE deep copy rules to configure the copying options when a user derives an ECN from an ECR. Deep copy rules define whether objects belonging to a business object instance can be copied when a user performs a **Derive** operation on that instance. Deep copy rules configure the copy option, providing the ability to select individual objects from the **Affected Items** and **Reference Items** in an ECR.

Using deep copy rules, you can configure whether the following are copied into the ECN:

- **Impacted Items**
- **Problem Items**
- **Reference Items**

You can create rules for the default relation types **CMHasImpactedItem**, **CMHasProblemItem**, **CMReferences**.

Refer to the *BMIDE for Data Model Design* in the Teamcenter collection for more information.

Note:

Validate that the setting **copyFromOrignal** is set to true on persistent properties to copy them during the derive process. If **copyFromOrignal** is set to false on persistent properties, the properties are not copied during derive.

© 2025 Siemens

Create Deep Copy Rules in BMIDE

1. Create a custom template based on the **Change Manager** template.
2. Open the **Change Item Revision Business Object**.

   You can create a **Derive** deep copy rule for **GnChangeRequestRevision** and **GnProblemReportRevision**.

   - Select the **Show Inherited Rules** check box to display all rules inherited from parent business objects.
   - Select the **Organize by Inheritance** check box to sort the rules by parent business object names.
   - Use the **Add**, **Edit**, or **Remove** buttons to work with the deep copy rules.
3. Select the **Deep Copy Rules** tab and click the **Add** button to add a row for each rule.
4. Choose the business object that the deep copy rule is applied to.

   Specify the parameters for each rule.

Parameter	Description	
Target Primary?	Mark or clear the check box as appropriate.	
	☑	When the checkbox is marked, **Target Business Object** is the primary object of the relationship specified in the **Relation Type** box. When the business object instance is revised or saved, the secondary objects are carried forward and related using the relation in the **Relation Type** box.
	☐	When the checkbox is cleared, **Target Business Object** is the secondary object of the relationship specified in the **Relation Type** box. When the business object instance is revised or saved, the primary objects are carried forward and related using the relation in the **Relation Type** box.
Operation Type	Select **Derive**.	
Type	**Relation** creates the deep copy relationship	
Relation / Reference Property	Select **CMHasImpactedItem**, **CMHasProblemItem**, or **CMReferences**. You will supply a rule for each relation.	

© 2025 Siemens

Parameter	Description
Attached Business Object	For **CMHasImpactedItem** and **CMHasProblemItem** types select **ItemRevision**. For **CMReferences** select **WorkspaceObject**.
Condition	Select the condition **isTrue**.
Action	Choose the kind of copying to be allowed for the business object. The available options differ depending on the type of target business object. Select **CopyAsReference**. This creates a new relation between the new revision and the related object. Therefore, modifications performed on the copied object are propagated to the source object. Note that if the deep copy rule **NoCopy** is set as **isTrue** and **Required**, then this relation does not appear in any panel in Active Workspace.
Required	Leave blank.
Secured	Select if you want to prevent the deep copy rule from being modified or overridden by another template.
Copy Properties on Relation	Select if you want persistent properties on relation objects carried forward when the primary objects participating in relations are revised or saved as new objects. If not selected, only mandatory properties are carried forward.

5. Click **Finish**.

   The rule is created and appears in the table in the **Deep Copy Rules** editor.

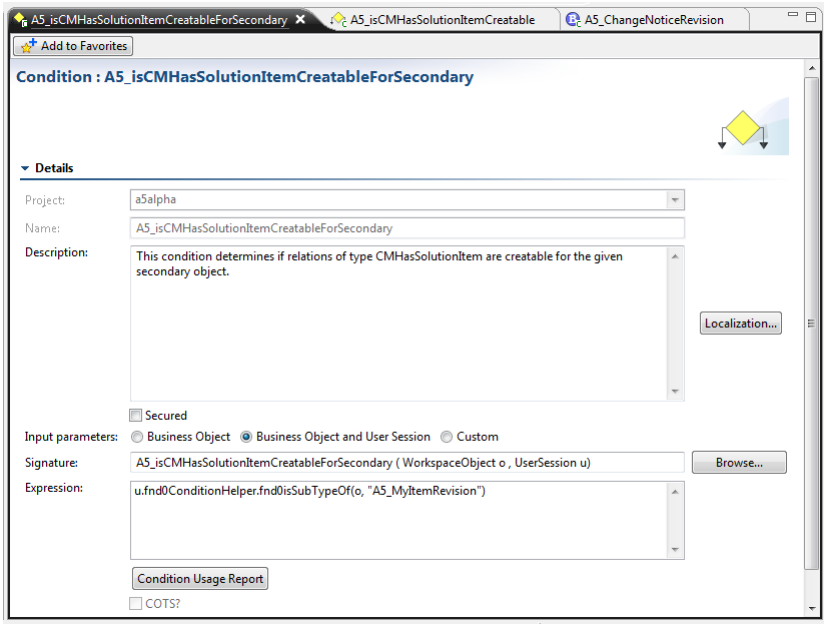
Restrict actions based on the type of applied change object

You can specify that only specific business object types can be used as solution items for a specific change object type. Use the **CMHasSolutionItemCreCondition** business object constant to point to the condition that defines the business objects.

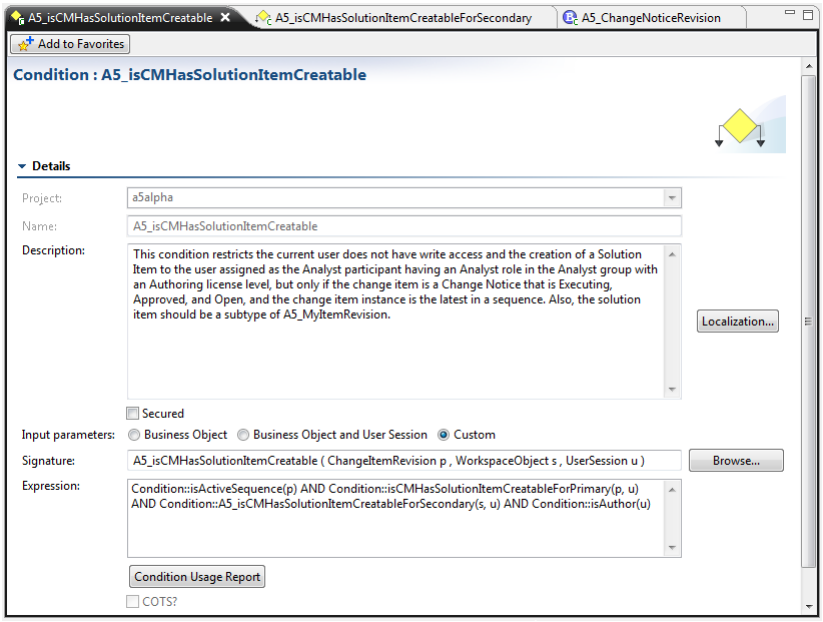
This configuration also works for ECNs. For example, a user can add a solution item to an ECN subtype only when the solution item business object is configured to be added as solution for the ECN.

1. Define a new custom condition similar to the COTS **Cm0isCMHasSolutionItemCreatableForSecondary** condition that defines the business objects that can be used as solution items. Use the **fnd0isSubTypeOf** operation on the **fnd0ConditionHelper** business object to define the business objects allowed. The following example points to the custom **A5_MyItemRevision** business object type. You can add multiple business object types.

© 2025 Siemens

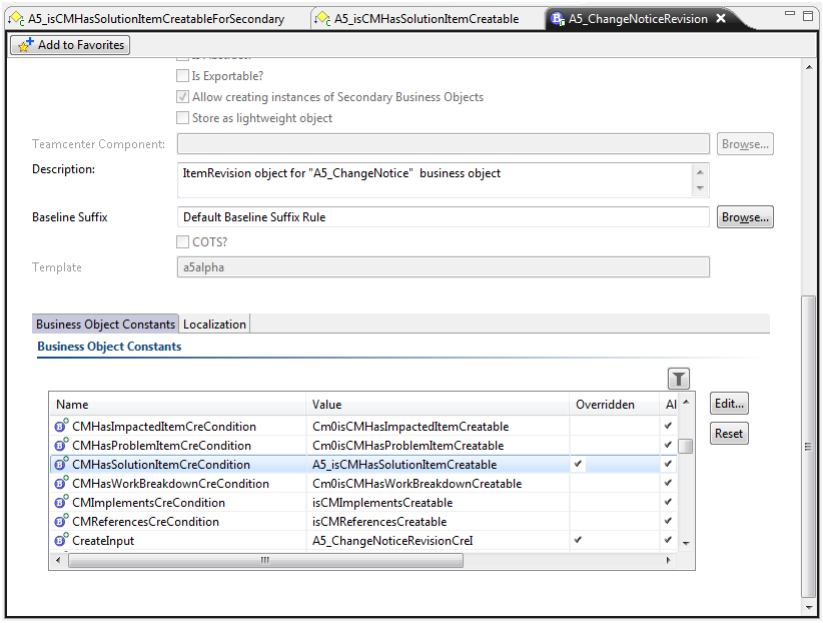


2. Define a new custom condition similar to the COTS **Cm0isCMHasSolutionItemCreatable** condition, except that it uses the new custom condition for the secondary condition instead of the **Cm0isCMHasSolutionItemCreatableForSecondary** condition. The following example points to the custom **A5_isCMHasSolutionItemCreatableForSecondary** condition created in the previous step.



3. Open the change notice revision for which you want to implement this behavior and override the **CMHasSolutionItemCreCondition** business object constant to point to the new custom condition. In the following example, it points to the custom **A5_isCMHasSolutionItemCreatable** condition.

© 2025 Siemens



Classification

Configure Classification using the Business Modeler IDE

Teamcenter can automatically compute attribute values based on other attribute values within the class or view or based on attribute values of the object being classified. It uses custom logic that you assign to a predefined operation in the Business Modeler IDE application. Classification objects are provided by the Foundation template. No additional templates are needed.

Classification extensions

Extensions allow you to write a custom function or method for Teamcenter in C or C++ and attach the rules to predefined hook points in Teamcenter.

Following are the extensions provided for Classification:

- **icsAskSubclassName**

  Determines the **ics_subclass_name** property value for a **workspaceObject** business object. This extension is used as the base action on the **ics_subclass_name** property on the **WorkspaceObject** business object.

- **icsItemRevBaselinePostAction**

  Ensures that during baseline operation of an item revision, classification objects (ICOs) from the item revision being baselined are copied to a new item revision. This extension is used as the postaction on the baseline operation on the **ItemRevision** business object.

© 2025 Siemens

- **icsItemRevSaveAsPostAction**

  Ensures that during the **Revise** operation of an **ItemRevision** business object, classification objects (ICOs) from the item revision being revised are copied to a new classification. This extension is used as postaction on the **Revise** operation on the **ItemRevision** business object. To avoid data corruption, do not use this extension for customization.

- **icsItemRevSaveAsPreCheck**

  Performs a preaction check during the **Revise** operation of an **ItemRevision** business object. Before creating new revision, this extension checks whether the classification objects on the object being revised conform to the minimum-maximum and key-LOV deprecation constraints based on preference settings. This is used as a precondition on the **Revise** operation on the **ItemRevision** business object.

- **icsItemSaveAsPostAction**

  Ensures that during the **SaveAs** operation of an **Item** or **ItemRevision** business object, classification objects (ICOs) from the item being saved are copied to the new item or item revision. This extension is used as the postaction on the **SaveAs** operation on the **Item** or **ItemRevision** business object. To avoid data corruption, do not use this extension for customization.

- **icsItemSaveAsPreCheck**

  Performs a preaction check during the **SaveAs** operation of an **Item** or **ItemRevision** business object. Before creating the new object, this extension checks whether the classification objects on the object being saved conform to minimum-maximum and key-LOV deprecation constraints based on preference settings. This extension is used as a precondition on the **SaveAs** action on **Item** or **ItemRevision** business objects.

- **icsSavePostCheck**

  Controls if classification object IDs are up-to-date for a given **Item** or **ItemRevision** business object instance being saved as and modifies them if needed. This extension is used as a postaction during the **SaveAs** operation of an **Item** business object.

- **isICSObjectClassified**

  Determines the **ics_classified** property value for a **workspaceObject** business object. It is used as the base action on the **ics_classified** property on the **WorkspaceObject** business object.

© 2025 Siemens

Service Lifecycle Management (SLM)

Configure Service Lifecycle Management (SLM) using the Business Modeler IDE

Use the Business Modeler IDE to create custom objects used by Service Lifecycle Management (SLM). SLM is comprised of a suite of applications that you can use to manage assets over their lifetime.

Note:

Before working with SLM objects, you must **install the following templates to your project**:

- As-Built Management (**asbuilt_template.zip** file)
- As-Built and As-Maintained Alignment (**asbasmalignment_template.zip** file)
- MRO Core (**mrocore_template.zip** file)
- Service Planning (**serviceplanning_template.zip** file)
- Service Processing (**serviceprocessing_template.zip** file)
- Service Request Processing (**servicerequest_template.zip** file)
- Service Scheduler (**servicescheduling_template.zip** file)

Consideration when creating a custom physical part

The **PhysicalPart** business object is provided by the MRO Core (**mrocore**) template and is used to designate physical instances of parts. If you create a custom business object as a child of the **PhysicalPart** business object, you must edit the **ServiceEventCreate.xml** XML rendering style sheet so that the custom physical part appears in the list of available physical parts when creating an event in the **Create Service Event** dialog box.

For example, if the custom physical part is named **GT4_PhysicalPart** and the custom physical part revision is named **GT4_PhysicalPartRevision**, in the **ServiceEventCreate.xml** XML rendering style sheet, change:

```
<property name="inprogress_physical_element" searchHint="MRO Physical Part"
objectHint="PhysicalPart,PhysicalPartRevision" />
```

To:

```
<property name="inprogress_physical_element" searchHint="MRO Physical Part"
objectHint="PhysicalPart,PhysicalPartRevision,GT4_PhysicalPart,GT4_PhysicalPartRevision"
/>
```


© 2025 Siemens

Manufacturing Process Planner

Configure Manufacturing Process Planner using the Business Modeler IDE

Use the Business Modeler IDE to create custom objects used by *Manufacturing Process Planner*. Manufacturing Process Planner allows you to design a plan that details how to manufacture a product. The manufacturing process plan includes a top-level structure of the process required to manufacture the product, as well as a detailed design of the individual processes and activities to be included in the plan.

Before working with Manufacturing Process Planner objects, you must **install the following templates to your project**:

- Work Instructions (**cmtmes_template.zip** file)
- MES Integration (**cmtmesinteg_template.zip** file)

> Note:
>
> To provide full functionality to your custom objects based on the **OP** and **Process** objects, including **Automatic Allocation**, in the rich client you must modify the following closure rules:
>
> AccountabilityProcessAllNoAssign
>
> ACPlantBOPDerFirstLvlProcAndOps
>
> In those rules, add your new operation and process objects to the **Conditional Clause** values that contain **OPRevision** and **ProcessRevision**.

3D PDF business objects

Teamcenter enables you to create interactive 3D reports that are derived directly from process data stored in the Teamcenter database. The reports are in PDF format and contain process information and 3D data. The Manufacturing Process Planner templates provide the following business objects used for 3D PDF configuration. You can create your own custom business objects as children of these business objects.

- 3D PDF dataset business objects
  - **Mes0PDFInputs**

    Contains different input files to generate a PDF report (for example, PRC/U3D, XFDF, and so on).
  - **Mes0PDFReport**

© 2025 Siemens

Contains a PDF report generated for a process or operation.

- **Mes0PDFReportTemplate**

  Contains a PDF template file used to generate a PDF report for a process or operation.

- 3D PDF relation business objects

  - **Mes0PDFContentRel**

    Defines a relation that attaches a dataset containing report content (**Mfg0ME3DPDFInputs**) to a report generated from it (**Mes0PDFReport**).

  - **Mes0PDFInputsRel**

    Defines a relation that attaches a dataset containing report content (**Mes0PDFInputs**) to a process or an operation that the content refers to.

  - **Mes0PDFReportInputRel**

    Defines a relation that attaches a report input to a report template (**Mes0PDFReportTemplate**).

  - **Mes0PDFReportRel**

    Defines a relation that attaches a PDF report (**Mes0PDFReport**) to its owning process or operation.

  - **Mes0PDFTemplateRel**

    Defines a relation that attaches a PDF report template (**Mes0PDFReportTemplate**) to a report generated from it (**Mes0PDFReport**).

Enable multi-unit configuration

Multi-unit configuration refers to a combination of multiple end items and range of units for each end item used to configure product structure occurrences. To enable multi-unit configuration (also known as multi-unit effectivities), the administrator must use the Business Modeler IDE to set the **Fnd0EnableMultiUnitConfiguration** global constant to **true** at each site.

In addition, the administrator must use the rich client or the Active Workspace **Preferences** page to set the **MEAccountabilityCheckEnableNetEffectivity** preference to **true**.

© 2025 Siemens

Multi-Structure Manager

Configure Multi-Structure Manager using the Business Modeler IDE

Use the Business Modeler IDE to create custom objects used by the Multi-Structure Manager application.

Multi-Structure Manager objects are provided by the Foundation template. No additional templates are needed.

Configure the automateAndLink extension

If your organization manages designs and parts separately, you must align the CAD designs and the BOM at appropriate times. This is known as *CAD-BOM alignment*.

You can automate the alignment so that when a user creates a CAD design and it is checked in to Teamcenter, the corresponding part is created automatically. To configure this, use the **automateAndLink** extension. Add the **automateAndLink** extension as a postaction to the create operation on the desired source design business object and select the relationship you want to the target part that is created.

You can configure automatic alignment for any children of the **Item** business object. For the following example, the source design is a custom **A5_MyDesign** business object that is a child of the **Design** business object, the target part is a custom **A5_MyPart** business object that is a child of the **Part** business object, and the relationship between them is the **TC_Is_Represented_By** relation.

You are not limited to using the **automateAndLink** extension for CAD-part alignment. You can use it to generate automatically one kind of item business object when another kind is created, and to link the two. Its behavior is similar to the **createObject extension**.

> Caution:
>
> If you are using a custom part type and you add a **BomViewRevision** to a custom design type, you must create custom code at the **BMF_ITEM_search_target_item** user exit, otherwise performing a **Save As...** on the custom design *creates* a copy of the custom part, but does not *attach* it. Contact your Siemens Digital Industries Software representative for example code.

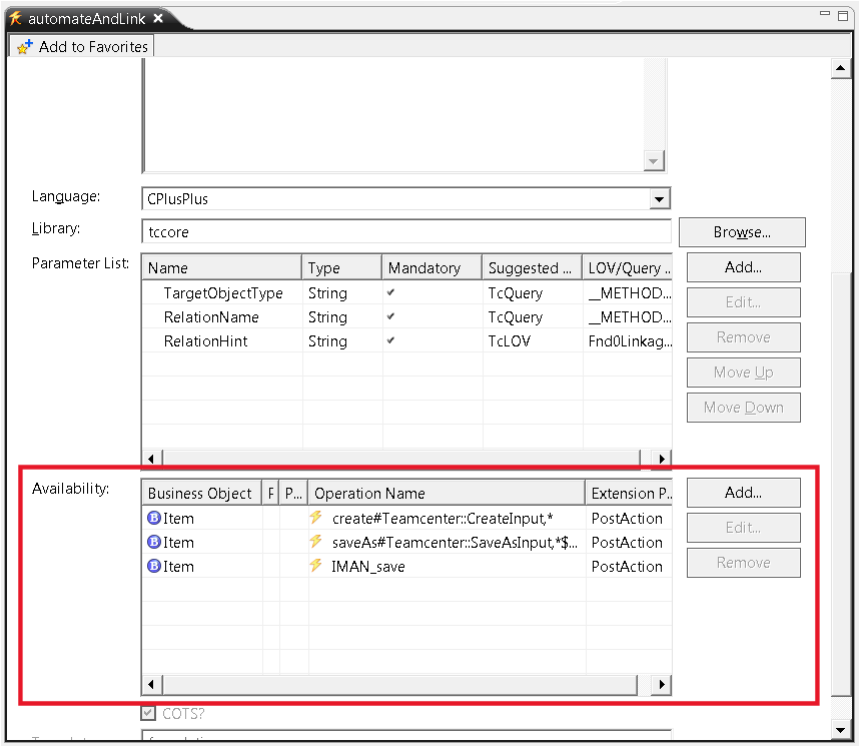
1. If you have not already done so, **create a custom template project** to hold your data model changes.

2. Add the **automateAndLink** extension as a postaction to the create operation for the desired design business object type.

    a. Make sure that the **automateAndLink** extension is available for use on your desired business object type. Open the **automateAndLink** extension and look at the business objects listed in the **Availability** table. This extension is available on the **Item** business object type and all its

© 2025 Siemens

children. Because the **A5_MyDesign** business object is a child of the **Design** business object, which in turn is a child of the **Item** business object, this extension is available for use.

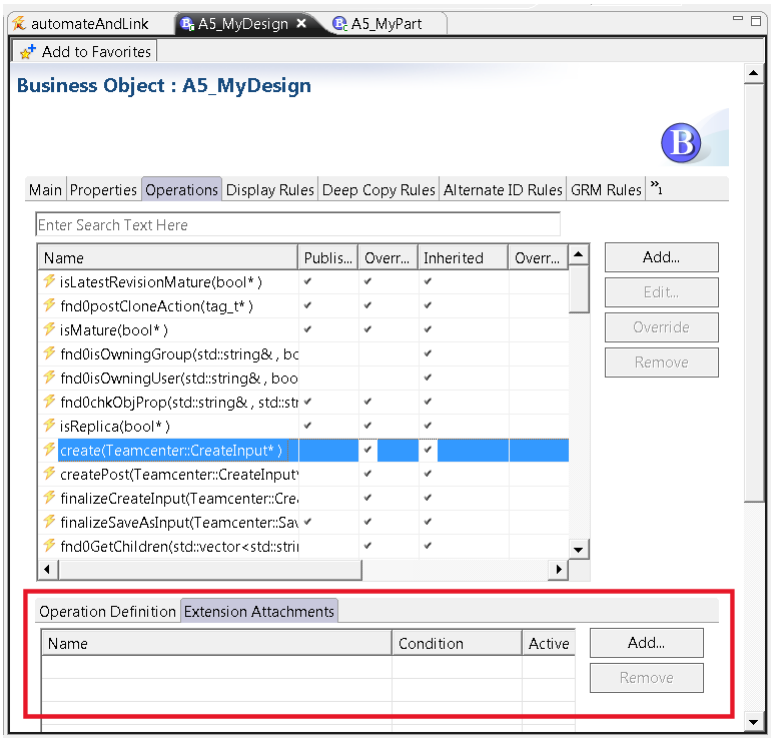


b. Add the **automateAndLink** extension as a postaction to the create operation for the design business object.

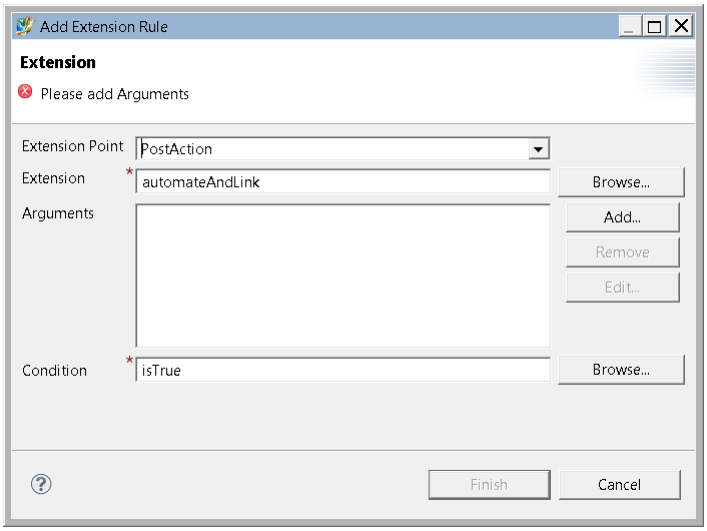
   Open the **A5_MyDesign** business object, click the **Operations** tab, and select the **create(Teamcenter::CreateInput*)** operation.

c. In the **Extension Attachments** tab, click the **Add** button to the right of the table.

© 2025 Siemens



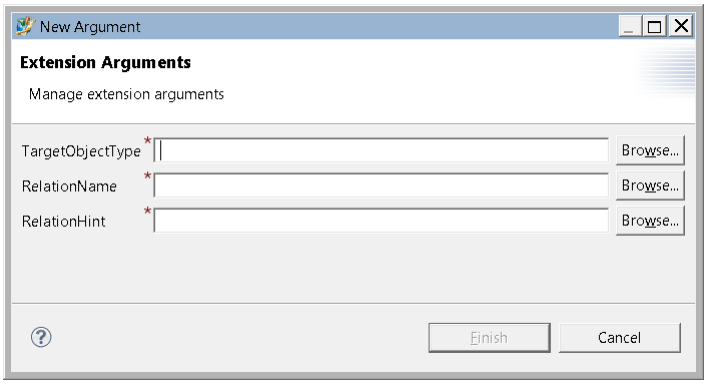
d. In the **Extension** dialog box, the **automateAndLink** extension is selected in the **Extension** box because it is made available as a postaction on the business object type.



e. Click the **Add** button to the right of the **Arguments** box.

The **Extension Arguments** dialog box is displayed.

© 2025 Siemens

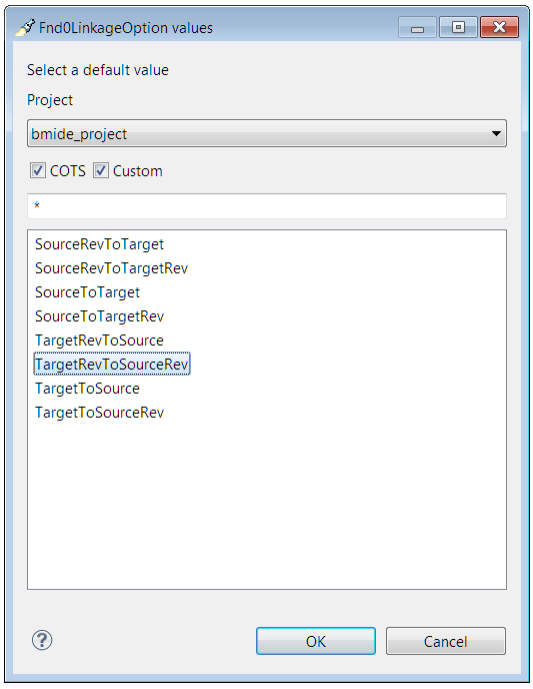


f. Perform the following steps in the **Extension Arguments** dialog box:

A. Click the **Browse** button to the right of the **TargetObjectType** box to select the target part type to be automatically created (for example, **A5_MyPart**).

B. Click the **Browse** button to the right of the **RelationName** box and select **TC_Is_Represented_By** as the relationship between the design and the part.

C. Click the **Browse** button to the right of the **RelationHint** box to select which source or target object is used as the primary object in the relation. For example, select **TargetRevToSourceRev** to link the target revision (the part revision) to the source revision (design revision) where the target revision is the primary object.



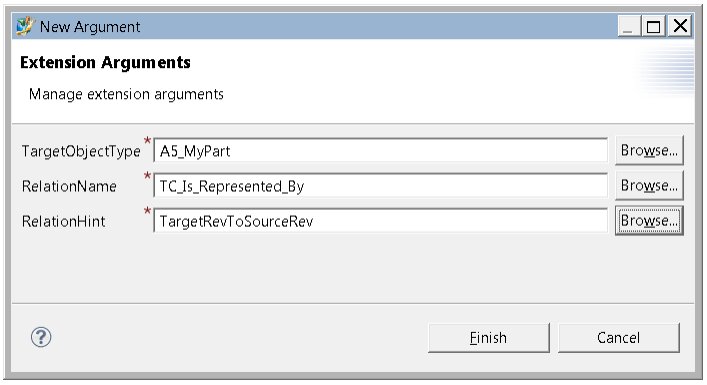
© 2025 Siemens

Note:

This relation hint list is provided by the **Fnd0LinkageOption** list of values. To see these with their descriptions, later you can open the **LOV** folder and open the **Fnd0LinkageOption** list of values.

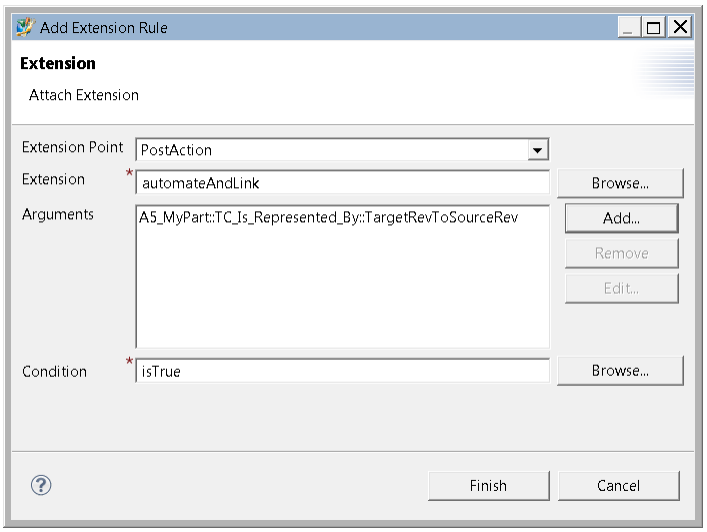
D. Click **OK**.

The argument is displayed.



E. Click **Finish** on the **Extension Arguments** dialog box.

The argument is added to the **automateAndLink** extension.

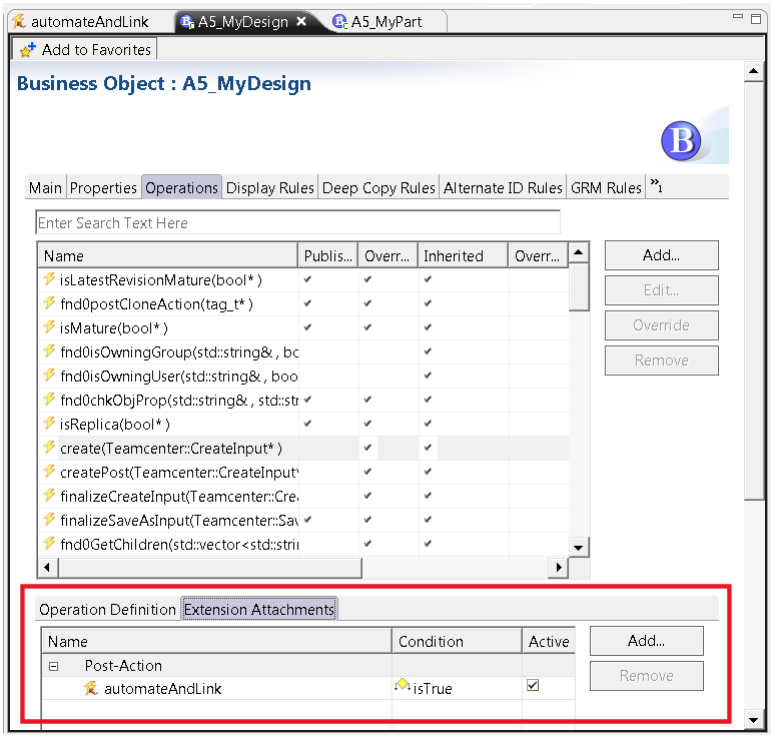


© 2025 Siemens

> Note:
>
> This procedure automatically creates a part when a design is created. If you want to give the end user a choice about whether to create the part automatically, you can **apply a condition to the automateAndLink extension by selecting it in the Condition box**.
>
> You can also create other kinds of conditions to perform other kinds of evaluations.

g. Click **Finish** in the **Extension** dialog box.

The **automateAndLink** extension is added as a postaction of the create operation on the **A5_MyDesign** business object. To see the argument, scroll right in the table to the **Arguments** column.

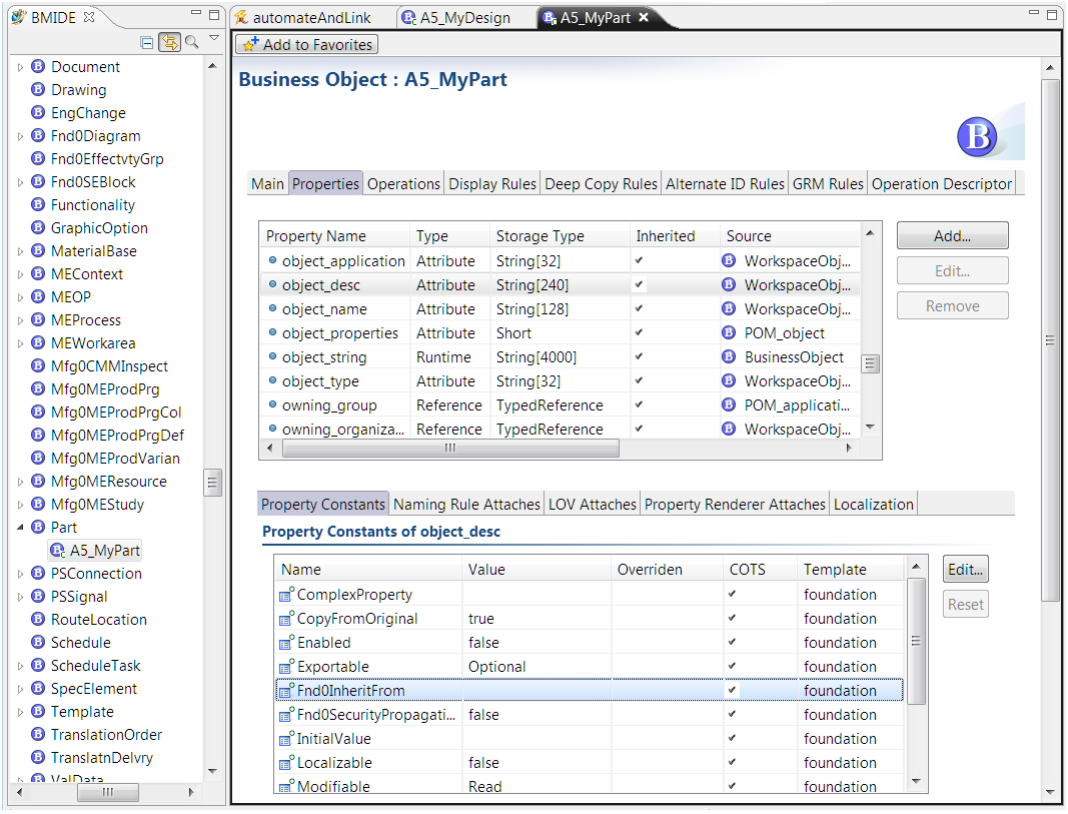


3. You can use the **Fnd0InheritFrom** property constant to inherit property values from the source CAD design business object when the automatic alignment of the design and part occurs.

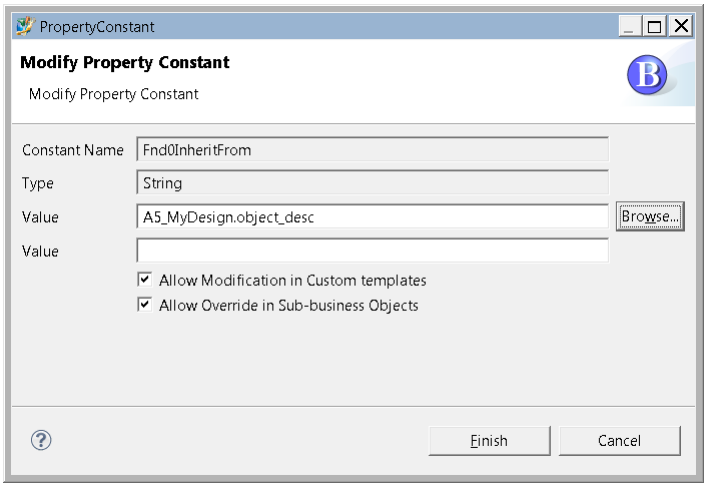
a. Open the **A5_MyPart** business object, click the **Properties** tab, and select a property that you want to inherit from the source design, for example, **object_desc**.

b. In the **Property Constants** table, select the **Fnd0InheritFrom** property constant.

© 2025 Siemens



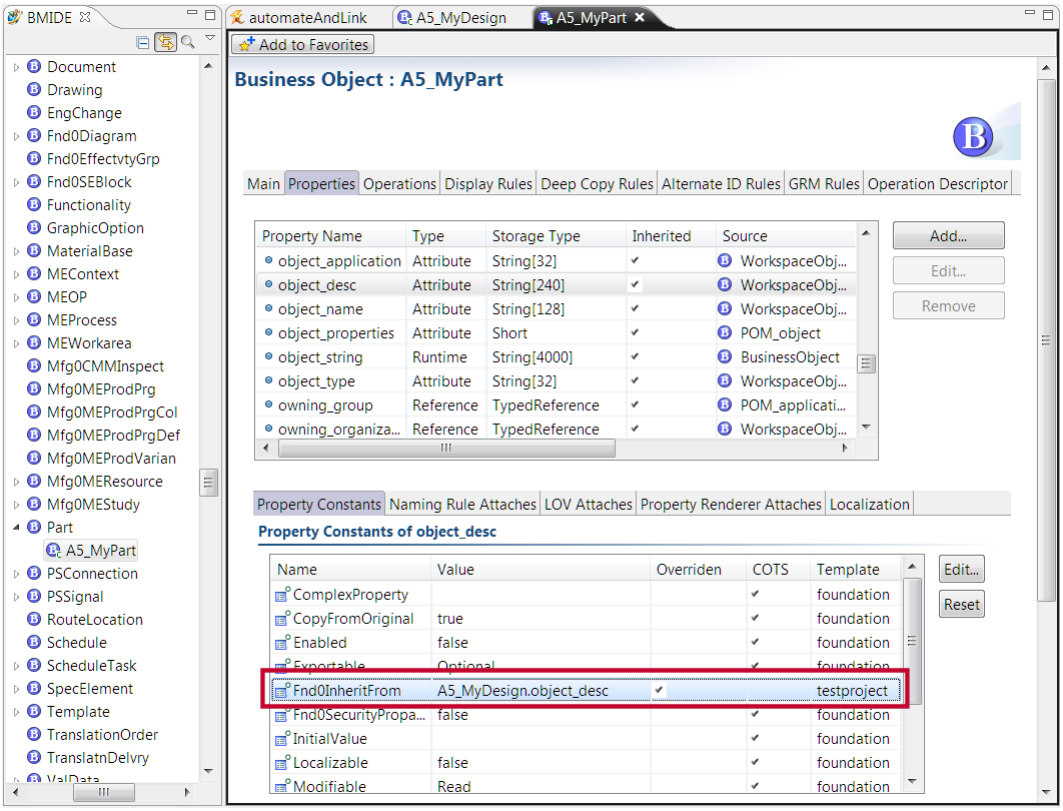
c. Click **Edit** to the right of the **Property Constants** table and browse for the business object and its property that you want to inherit from, for example, **A5_MyDesign.object_desc**. Copy the value to the second **Value** box.



d. Click **Finish**.

The new value for the **Fnd0InheritFrom** property constant is displayed.

© 2025 Siemens



e. Repeat this step for any other properties you want the part to inherit from the design, for example, **object_name** or **item_id**.

> Note:
>
> If you choose to inherit the **item_id** property, you must ensure that you have first configured multifield key definitions to allow two different business object types (in this example, **A5_MyDesign** and **A5_MyPart**) to **use the same item ID**.

4. To save your changes to the template, on the menu bar choose **BMIDE→Save Data Model**. Then **package the template** and use Teamcenter Environment Manager to install the packaged template to your server.

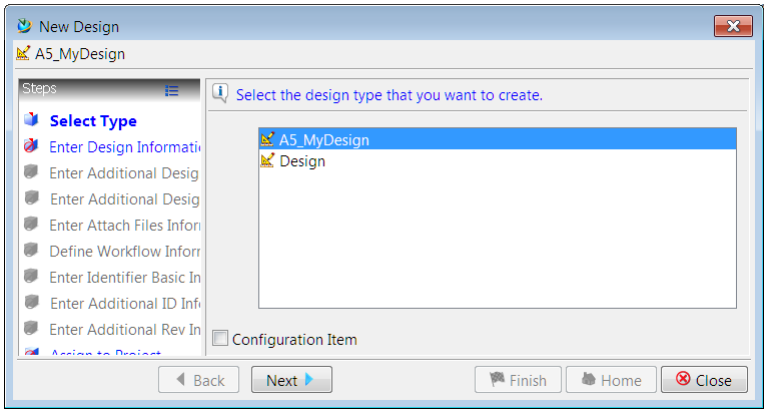
5. Verify that the alignment is performed when you create the **A5_MyDesign** business object.

   a. Log on to the rich client and open Multi-Structure Manager.

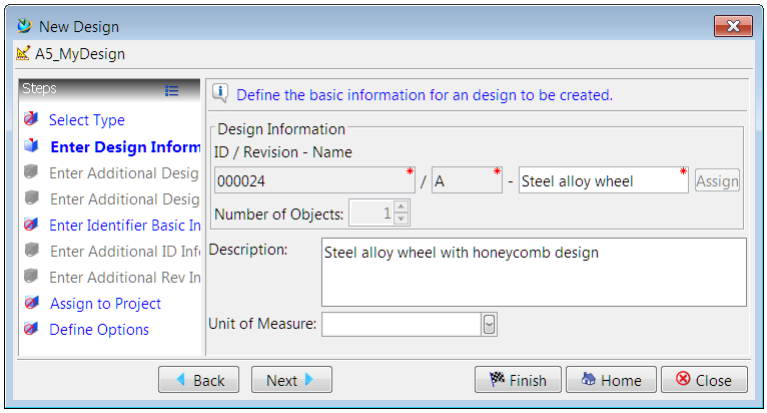
> Note:
>
> Creating the design this way in the rich client is for illustration purposes. Typically, you create the design in a CAD application and save it to Teamcenter directly from that CAD application through an integration to Teamcenter. At the time the design is saved in the CAD application to Teamcenter, the automatic alignment occurs.

© 2025 Siemens

b. Choose **File→New→Design** and select **A5_MyDesign**. Click **Next**.

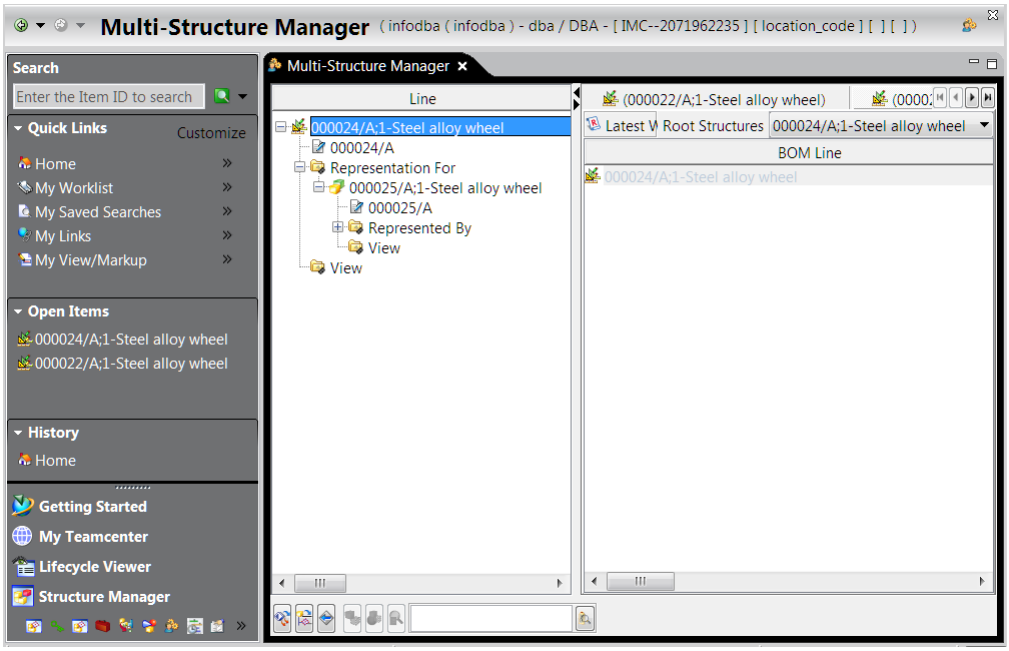


c. In the **New Design** dialog box, give the object an ID, a name, and a description. Click **Finish**.



The design and part are automatically aligned. The part is created and related to the design. Also notice how the object description on the part is inherited from the design.

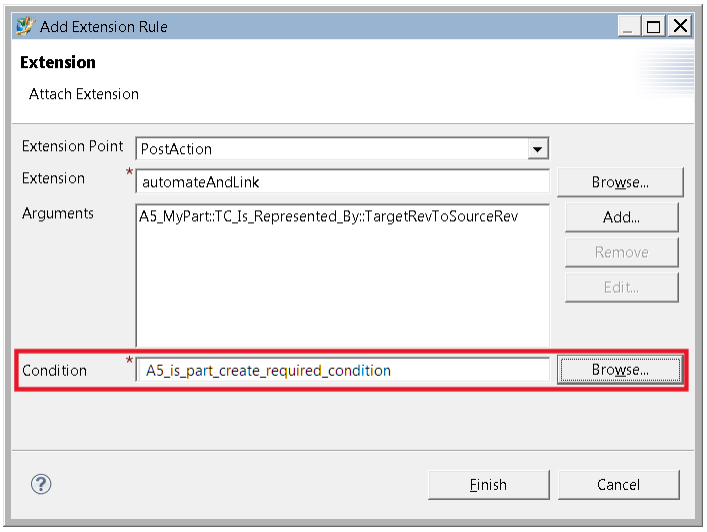
© 2025 Siemens



Using conditions with the automateAndLink extension

You can use the **automateAndLink** extension to automate CAD-part alignment so that when a user creates a CAD design and it is checked in to Teamcenter, the corresponding part is created automatically.

You can apply conditions to the extension to modify how it runs. If you create your own conditions to apply to the automateAndLink extension, input parameters for the condition can be business objects, user session, or both.



Following are some conditions you can use with the **automateAndLink** extension:

- **Fnd0isCMSImportActive**

© 2025 Siemens

Prevents the target object from being created if a classic Multi-Site import session is active.

- **Fnd0isPLMXMLImportActive**

  Prevents the target object from being created if a PLM XML import/export (PIE) import session is active.

Create a condition asking whether to create a part

You can use the **automateAndLink** extension to automate CAD design-part alignment so that when a user creates a CAD design and it is checked in to Teamcenter, the corresponding part is created automatically. But users may not always want to have a part automatically created when they create a design. Depending on the business practice, the criteria to create a part can differ. Therefore, you can create a condition and apply it to the **automateAndLink** extension so that a dialog box is provided asking end users if also they want to have a part automatically created when they create a design.

> Note:
>
> You can configure automatic alignment for any children of the **Item** business object. This example uses design and part business object types.

1. Create a Boolean property on the source design business object. The Boolean property provides a true or false selection to the end user asking them if they want to create the part.

   a. Open the source design business object type and click the **Property** tab.

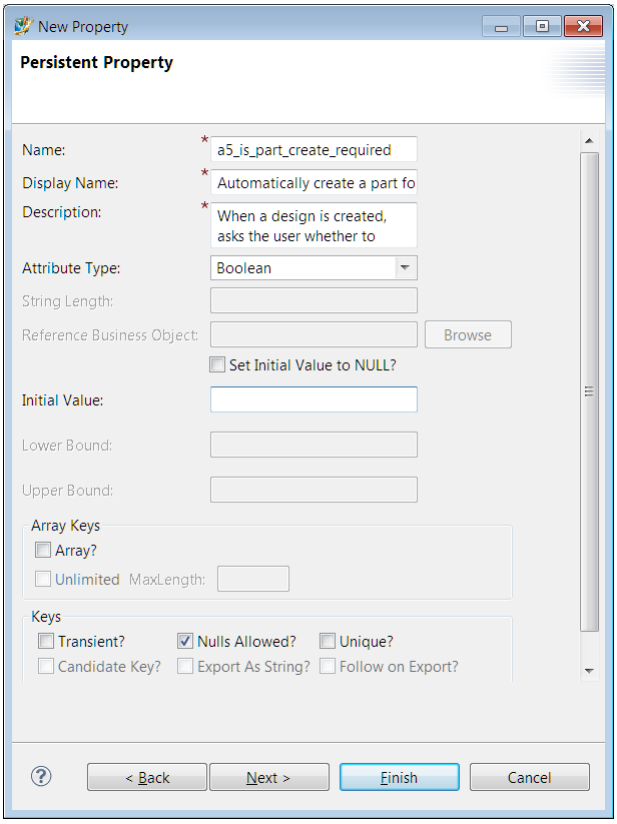
   b. Click the **Add** button to the right of the **Property** table, select **Persistent** and click **Next**.

   c. Type a name for the property in the **Name** box, and in the **Display Name** box type the message you want to display to the user, for example:

      Automatically create a part for the design?

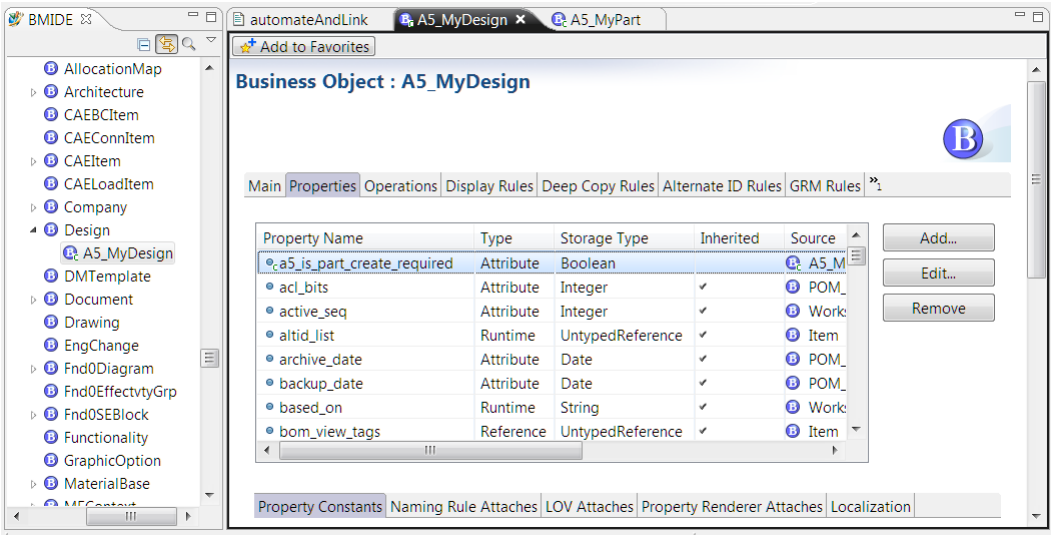
   d. For the **Attribute Type**, select **Boolean**.

© 2025 Siemens



e. Click **Finish**.

The new Boolean property is displayed on the design business object.



2. Add the Boolean property to the **CreateInput** operation on the **Operation Descriptor** tab. This adds the Boolean property to the design creation wizard.

© 2025 Siemens

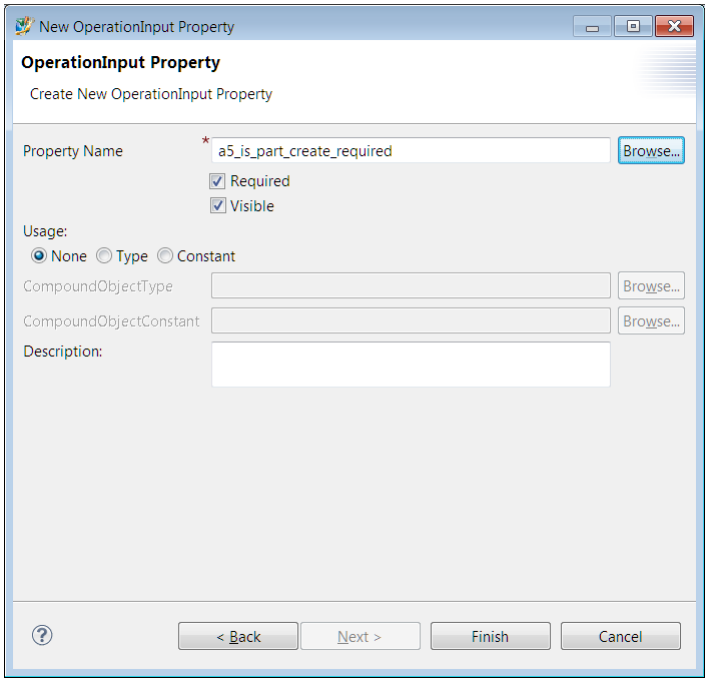
a. Click the **Operation Descriptor** tab on the design business object.

b. In the **Operation** box, ensure that **CreateInput** is selected and click the **Add** button to the right of the table.

c. Select **Add a Property from Business Object** and click **Next**.

d. In the **OperationInput Property** dialog box, click the **Browse** button to the right of the **Property Name** box and select the new Boolean property you previously created.

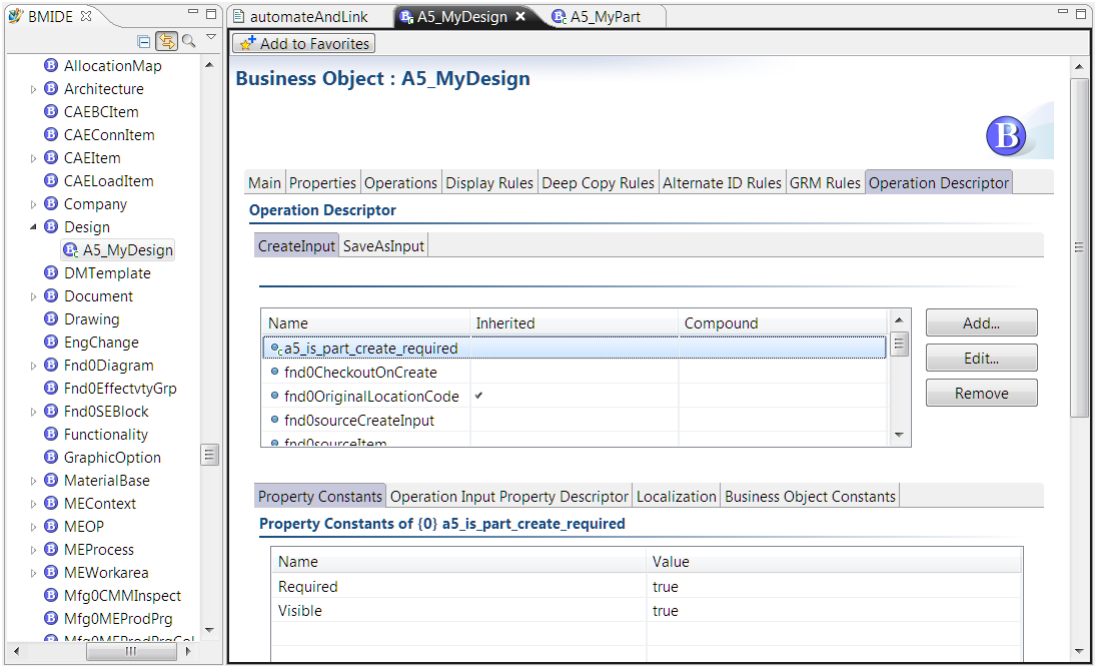
e. Select the **Required** and **Visible** check boxes.



f. Click **Finish**.

The Boolean property appears on the **Operation Descriptor** tab.

© 2025 Siemens



3. Create a condition that uses the Boolean property. This condition is applied to the **automateAndLink** extension to determine whether the part is automatically created when a design is created.

   a. Open the **Extensions\Rules** folders.

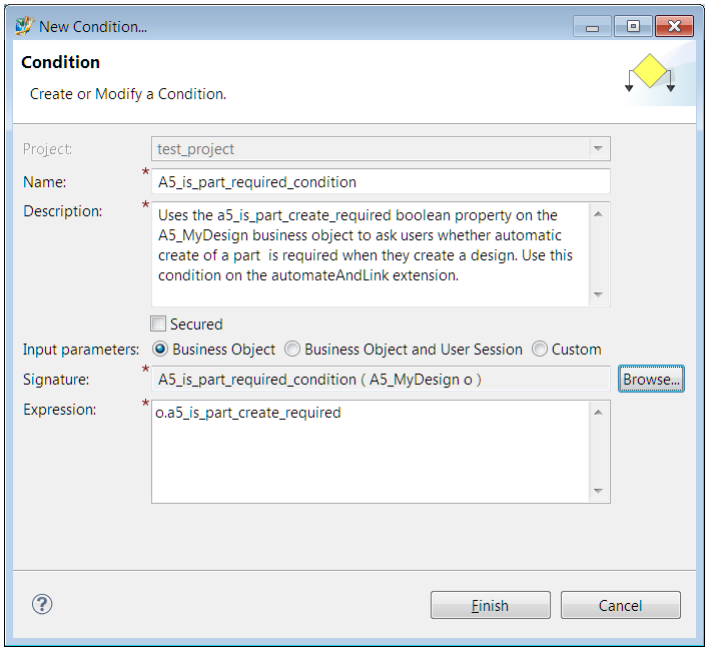
   b. Right-click the **Conditions** folder and choose **New Condition**.

   c. Give the condition a name and a description that clearly state what the condition is used for.

   d. Click the **Browse** button and select the design business object that has the Boolean property.

   e. In the expression box, type **o.** and the name of the Boolean property, for example, **o.a5_is_part_create_required**. This means that the property is used for condition evaluation.

© 2025 Siemens

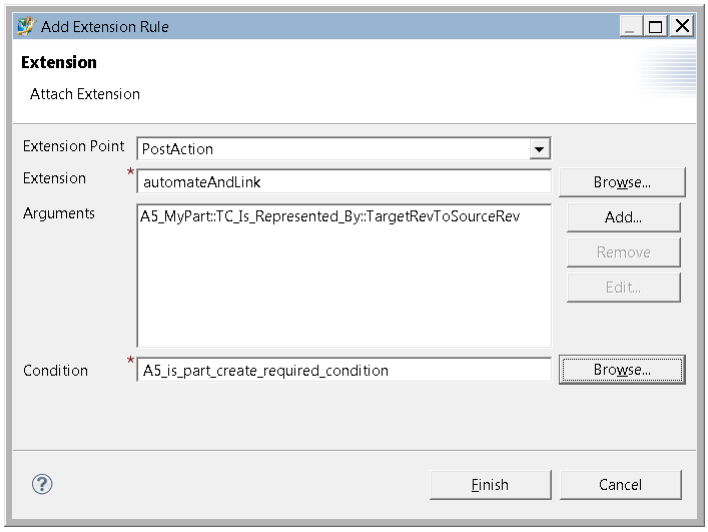


f. Click **Finish**.

4. Apply the condition to the **automateAndLink** extension.

a. **Configure the automateAndLink extension** for the source design type and target part type.

b. Click the **Browse** button to the right of the **Condition** box and select the condition you created that includes the Boolean property.



c. Click **Finish**.

© 2025 Siemens

5. To save your changes to the template, on the menu bar, choose **BMIDE→Save Data Model**. Then **package the template** and use Teamcenter Environment Manager to install the packaged template to your server.

6. Verify that the Boolean property is displayed when you create the design business object.

   a. Log on to the rich client and open Multi-Structure Manager.

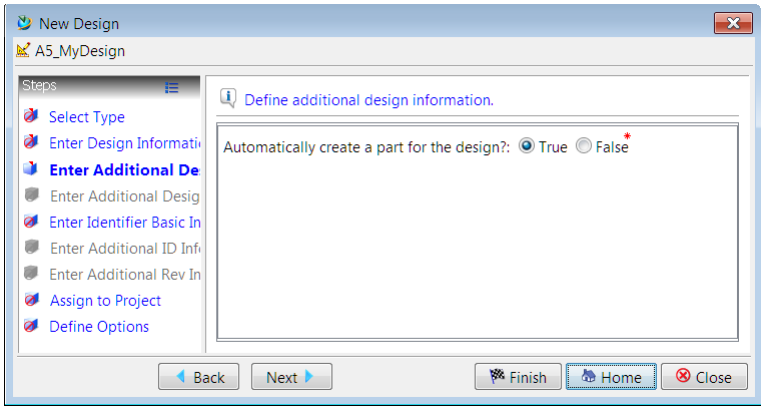
   > Note:
   >
   > Creating the design this way in the rich client is for illustration purposes. Typically, you create the design in a CAD application and save it to Teamcenter directly from that CAD application through an integration to Teamcenter. At the time the design is saved in the CAD application to Teamcenter, the automatic alignment occurs.

   b. Choose **File→New→Design** and select the design type you want to create. Click **Next**.

   c. In the **New Design** dialog box, give the object an ID, a name, and a description. Click **Next**.

      The Boolean property is displayed.

   d. Select **True** to automatically create the part and relate it to the design, or click **False** to create only the design.



   e. Click **Finish**.

      The new design is displayed in Multi-Structure Manager. If you selected **True**, the related part is also created. If you selected **False**, the related part is not created.

© 2025 Siemens

NX

Creating item types with required attributes for NX

You cannot view custom items with required properties in NX unless the required property has an initial value defined or the initial value is listed as **Null**. Use the **Required** property constant to make a property required and the **InitialValue** property constant to set the initial value.

Configure NX CAM Integration using the Business Modeler IDE

Use the Business Modeler IDE to create custom objects used by the NX CAM Integration application.

NX CAM Integration objects are provided by the Foundation template. No additional templates are needed.

Organization

Configure Organization using the Business Modeler IDE

Use the Business Modeler IDE to configure aspects of the Organization application. Organization objects are provided by the Foundation template. No additional templates are needed.

For example, you can add a list of citizenships to the **Citizenships** box in the Organization application by using the Business Modeler IDE to attach a list of values to the **fnd0_citizenships** property on the **User** business object.

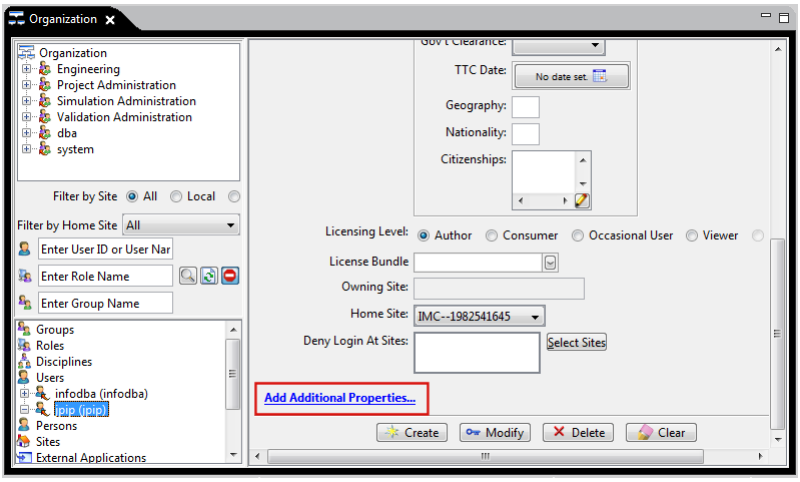
Add additional properties to users

You cannot add custom properties directly to the **User** business object. However, you can add custom properties to the **Fnd0CustomUserProfile** business object. (The **fnd0custom_user_profile** property on the **User** business object points to the **Fnd0CustomUserProfile** business object as the source for the custom properties.)

When you add custom properties to the **Fnd0CustomUserProfile** business object and install the resulting template to a Teamcenter server, an **Add Additional Properties** link appears in the panel in the Organization application used to create or modify users. Click the link to add the custom properties to the user's profile.

After assigning custom properties to a user, you can search on those properties, add them to saved queries, or assign access rules to them for further control of the user.

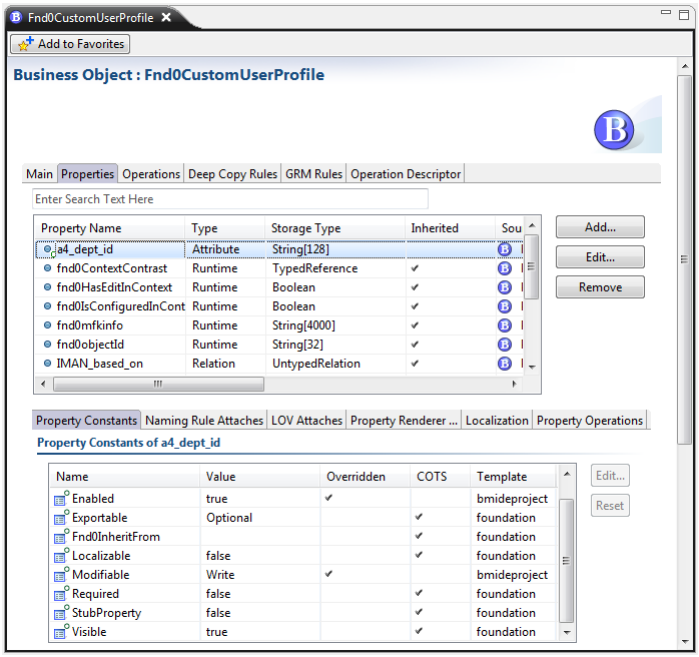
© 2025 Siemens



1. Open the **Fnd0CustomUserProfile** business object, click the **Properties** tab, and click the **Add** button to add a custom persistent property.

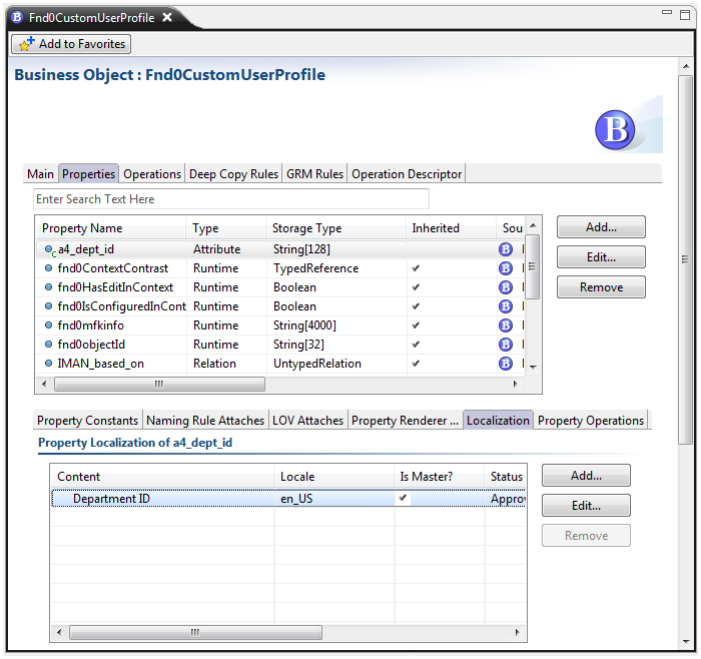
> Note:
>
> You cannot add a relation or compound property to the **Fnd0CustomUserProfile** business object. Nor can you subclass the **Fnd0CustomUserProfile** business object.

In the following example, a custom property has been added to hold the user's department ID. Note that the **Visible** property constant is set to **true** by default. This ensures that the property appears in the end user interface.



© 2025 Siemens

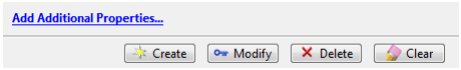
2. Ensure that you click the **Localization** tab to change the property display name to a value you want to appear in the end user interface.



3. **Save your data model**, **package the template**, and **use Teamcenter Environment Manager to install the packaged template to your server**.

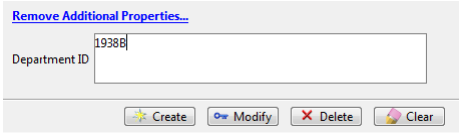
4. Launch the Teamcenter rich client, open the Organization application, and click the **Users** group.

   The **Add Additional Properties** link appears at the bottom of the panel.



5. Click the **Add Additional Properties** link to add values to the custom properties on the user's profile. Click the **Create** or **Modify** button to save the values.

   When you add values to the custom properties, the link name changes to **Remove Additional Properties**.



   Clicking the link toggles it to **Add Additional Properties**.

© 2025 Siemens

Note:

Clicking the **Remove Additional Properties** discards the values entered. That is, if you click the **Remove Additional Properties** link and click the **Modify** button, the values are not stored in the database. To save the values, click the **Add Additional Properties** link, type the values, and then click the **Create** or **Modify** button.

Product and manufacturing information (PMI)

Configure product and manufacturing information (PMI) using the Business Modeler IDE

Use the Business Modeler IDE to create custom objects used by the product and manufacturing information (PMI) functionality.

Siemens Digital Industries Software recommends that you create a **PMIContext** custom business object as a child of the **Item** business object. This custom context business object can then be assigned to the **MEPostAssignIDICMaker** preference, which is used to configure PMI effectivity.

Note:

Before working with PMI objects, you must **install the Configurable PMI Visualization (PMI) template** to your project.

To install PMI to a server, perform the following steps:

1. Open *feature_pmi.xml* in a text editor.

In this case	Edit the *feature_pmi.xml* file found here
You are performing a new install of Teamcenter.	*<KIT>/install/modules*
You want to add the feature to an existing Teamcenter installation.	*<TCROOT>/install/install/modules*

2. If you have `<visible value=”false”/>`, change it to `<visible value=”true”/>`.

3. Launch TEM, in the **Features** panel, under **Product Manufacturing Information**, locate **Configurable PMI Visualization**.

4. Select the **Configurable PMI Visualization** check box and click **Next**.

© 2025 Siemens

Schedule Manager

Configure Schedule Manager using the Business Modeler IDE

Use the Business Modeler IDE to create custom objects used by the Schedule Manager application. Schedule Manager objects are provided by the Foundation template. No additional templates are needed.

Create a custom status for Schedule Manager

You can create custom statuses to add to the available Schedule Manager statuses. Statuses show the state of schedules and tasks, such as **In Progress** or **Complete**.

> Note:
>
> Do not use this method to **add custom status types** for use by Workflow Designer and other areas of Teamcenter. Instead, open the **Extensions\Options** folders, right-click the **Status**, and choose **New Status**.

1. If you have not already done so, **create a custom template project** to hold your data model changes.

2. Add your new status to an existing status LOV.

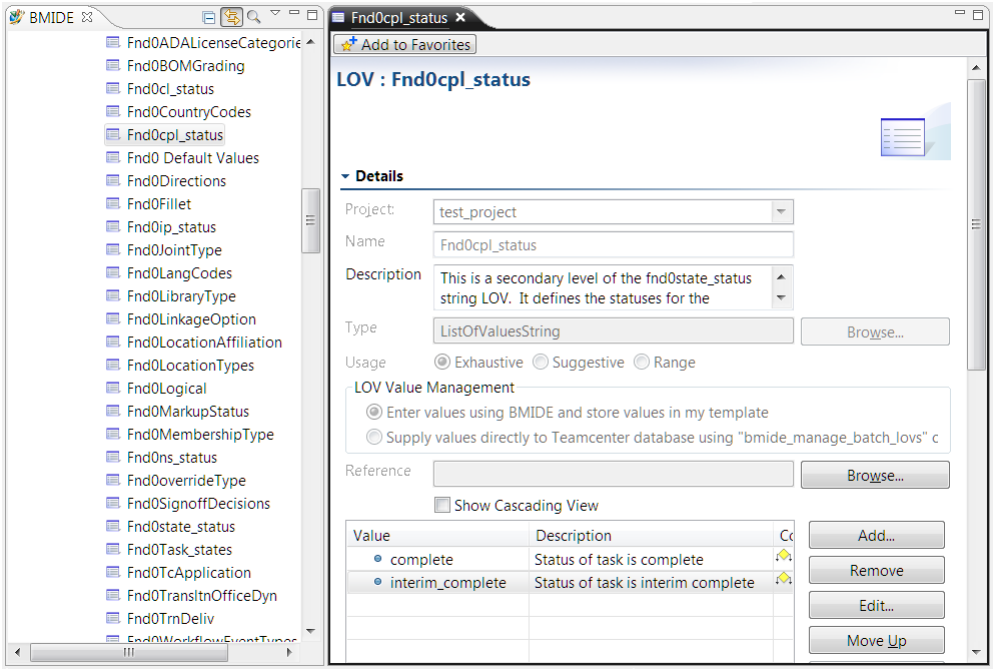
   Add the new status value to an existing **Fnd0*_status** LOV, for example, **Fnd0cl_status** (closed statuses), **Fnd0cpl_status** (complete statuses), **Fnd0ip_status** (in-progress statuses), or **Fnd0ns_status** (not started statuses).

   > Caution:
   >
   > Do not change the **Fnd0state_status** LOV. This is a cascading LOV whose values are added from the status sub-LOVs.

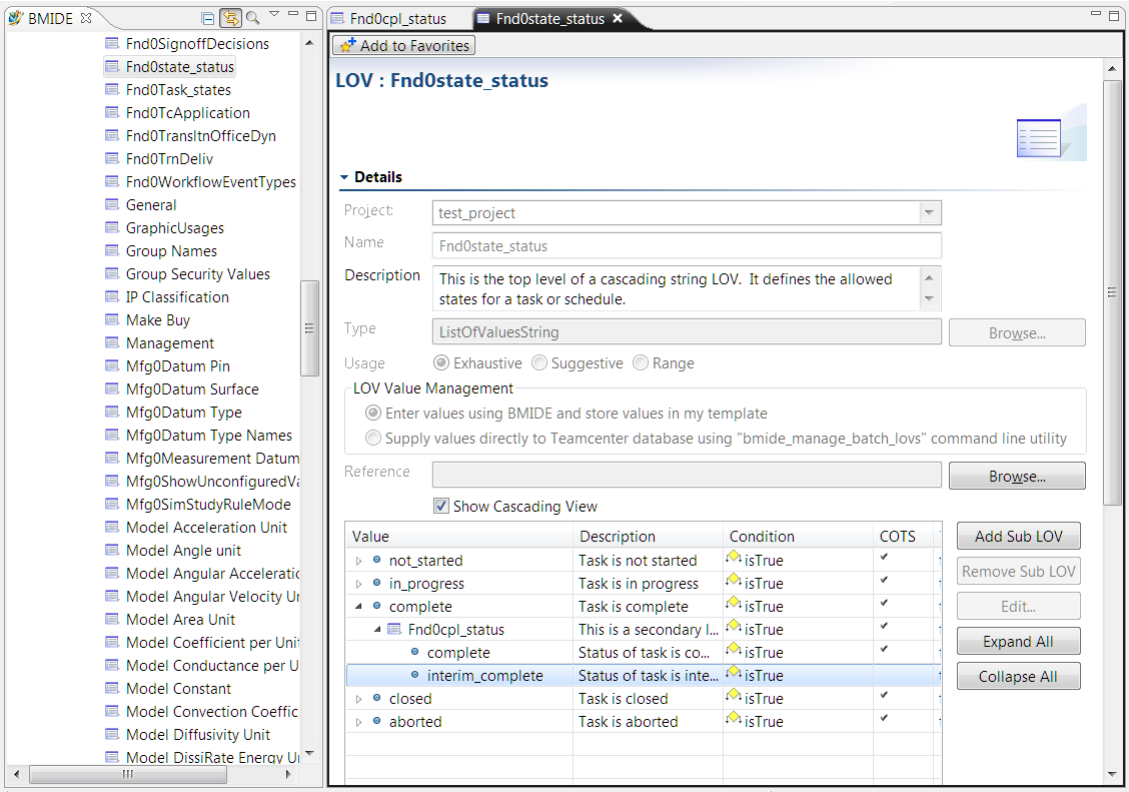
   For example, if you want to create a status that displays as **Interim Complete** in the user interface, open the **Fnd0cpl_status** and add the **interim_complete** value. Make sure to type the value display name and description, because they appear in the Schedule Manager user interface.

   For examples of Schedule Manager status LOVs, see the LOVs named **Fnd0*_status**.

© 2025 Siemens



After you add the new status value to the LOV, the value appears under the sub-LOV within the **Fnd0state_status** LOV.



© 2025 Siemens

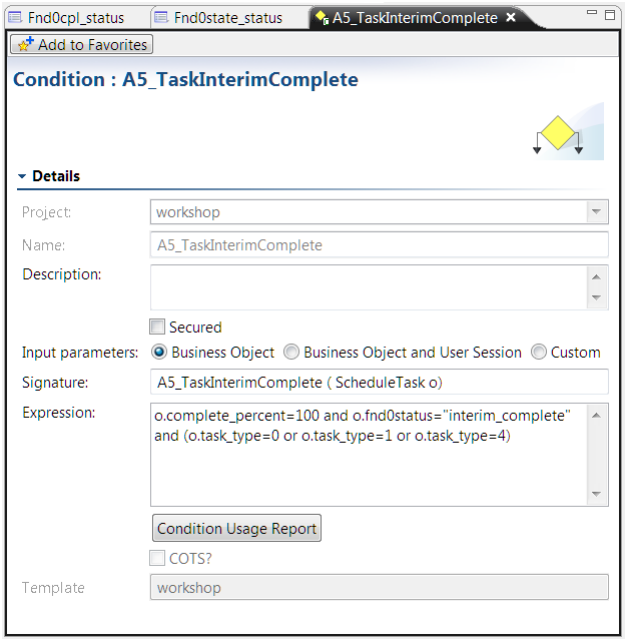
3. Create a custom condition rule to determine when the new status is initiated and pass it in to the **Fnd0SMSetTaskStatus** extension. Base the custom condition on an existing condition. For examples of Schedule Manager conditions, see the conditions named **Fnd0SM***.

   For the interim complete status example, you can create a condition based on the **Fnd0SMIsTaskComplete** condition:

   ```
   S5_TaskInterimComplete ( ScheduleTask o)

   o.complete_percent=100 and o.fnd0status="interim_complete"
   and (o.task_type=0 or o.task_type=1 or o.task_type=4)
   ```

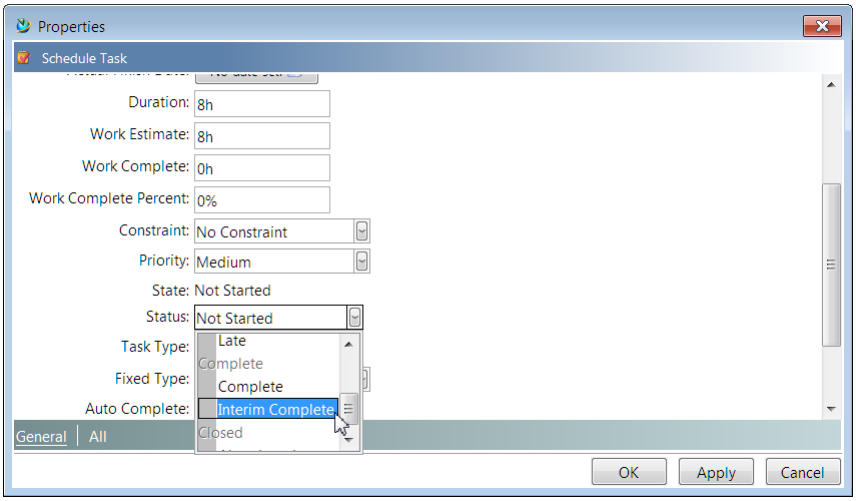
   This condition returns true if the task complete percent is **100** and the **fnd0status** property has the **interim_complete** value.



4. Save your data model, package the template, and use Teamcenter Environment Manager to install the packaged template to your server.

5. To see the new status in Schedule Manager, select a schedule or task, choose **View→Properties** on the menu tool bar, and select the arrow at the end of the **State** or **Status** boxes.

© 2025 Siemens



Schedule Manager operations, extensions, and conditions used for statuses

Operations, extensions, and conditions work together to provide Schedule Manager task status functionality. You can set preaction and postaction operations on object properties so that when the property is set, the preaction or postaction executes a customized action. You can also attach extension points to the preaction or postaction operation, as well as define conditions that can be evaluated in several ways.

In the use case of task status, whenever a specified property's value is set (for example, the **Fnd0Status** property), the postaction on the property operation calls the extension point code that makes an ITK call to evaluate the specified condition. If the condition returns true, the business logic is executed.

Default statuses have their own conditions and extension points. You can modify these conditions to meet your needs for setting the status. To define a new condition, pass in the new condition name to the provided extension to allow the extension to evaluate the new condition. To comply with the provided extensions, the system supports running conditions defined to take **Schedule**, **ScheduleTask**, or **UserSession** business objects. From these objects, you have access to the needed run-time parameters to write a condition.

Following are the property operations used to process statuses. The property operations are placed on properties of the **TaskExecutionFormInfo** business object:

- **setWork_Complete** property operation

  Placed on the on the **work_complete** property. Runs the **Fnd0SMSetTaskStatus** extension as a postaction, which in turn evaluates the **Fnd0SMIsWorkCompleteInProgress** condition against the **in_progress** status.

- **setComplete_percent** property operation

  Placed on the **complete_percent** property. Runs the following extensions as postactions:

© 2025 Siemens

- **Fnd0SMSetTaskStatus** extension

  Evaluates the **Fnd0SMIsCompletePercentInProgress** condition against the **in_progress** status.

- **Fnd0SMSetTaskStatus** extension

  Evaluates the **Fnd0SMIsCompletePercentComplete** condition against the **complete** status.

- **setFnd0Status** property operation

  Placed on the **fnd0state** property. Runs the following extensions as postactions:

  - **Fnd0SMInitializeActualStart** extension

    Evaluates the **Fnd0SMHasTaskStarted** condition

  - **Fnd0SMInitializeActualFinish** extension

    Evaluates the **Fnd0SMIsTaskStatusComplete** condition

  - **Fnd0SMSetPercentComplete** extension

    Evaluates the **Fnd0SMIsTaskStatusComplete** condition when the task is at **100.00** percent complete.

  - **Fnd0SMRollupStatus** extension

    Processes rollup rules logic. The rollup rules logic in this extension is not customizable. But you can create and attach a custom extension to change the rollup logic.

Schedule Manager property operations

Schedule Manager property operations are primarily used to process task status. The postaction on the property operation calls the extension point code which makes an ITK call to evaluate the specified condition. If the condition returns true, the business logic is executed.

Following are the Schedule Manager property operations:

- Following are property operations on the **TaskExecutionFormInfo** business object:

  - **setActual_finish_date**

    This operation is not used for COTS rules. It is attached to the **Fnd0SMSetTaskStatus** extension to allow you to create a postaction.

  - **setActual_start_date**

© 2025 Siemens

This operation is not used for COTS rules. It is attached to the **Fnd0SMSetTaskStatus** extension to allow you to create a postaction.

- **setComplete_percent**

  This operation is used for COTS rules.

- **setFnd0status**

  This operation is used for COTS rules.

- **setApproved_work**

  This operation is not used for COTS rules. It is attached to the **Fnd0SMSetTaskStatus** extension to allow you to create a postaction.

- **setWork_complete**

  This operation is used for COTS rules.

- Following are property operations on the **TaskSchedulingFormInfo** business object:

  These operation are not used for COTS rules, but are attached to the **Fnd0SMSetTaskStatus** extension to allow you to create a postaction.

  - **setDuration**
  - **setFinish_date**
  - **setStart_date**
  - **setWork_estimate**
  - **setWork_remaining**

- Following are property operations on the **ScheduleTaskRevision** business object:

  - **setPriority**

    This operation is not used for COTS rules. It is attached to the **Fnd0SMSetTaskStatus** extension to allow you to create a postaction.

© 2025 Siemens

Schedule Manager extensions

Schedule Manager extensions are primarily used to process task status, and are usually attached to property operations as postactions. The extension point code makes an ITK call to evaluate the specified condition. If the condition returns true, the business logic is executed.

> Note:
>
> The Business Modeler IDE does not support associating the same extension name multiple times to a given operation. Therefore, there are multiple extensions with a similar name that call the main extension that contains the logic to execute.

Following are the Schedule Manager extensions:

- **Fnd0SMInitializeActualStart**

  Initializes the actual start date if the condition is true. If the date is not already set, it is initialized to the scheduled start date or system time depending on the settings in the **DefaultActualToSystemDate** and **SM_EnforceActualDatesBeforeNow** preferences.

- **Fnd0SMInitializeActualFinish**

  Initializes the actual finish date if the condition is true. If the date is not already set, it is initialized to the scheduled finish date or system time depending on the settings in the **DefaultActualToSystemDate** and **SM_EnforceActualDatesBeforeNow** preferences.

- **Fnd0SMSetPercentComplete**

  Sets the task percent complete to the provided percentage if the condition is true. The extension name can be used to pass a different argument.

- **Fnd0SMSetTaskStatus**

  Sets the task status to the provided status if the condition is true. The extension name can be used to pass a different argument.

- **Fnd0SMSetScheduleStatus**

  Sets the schedule status to the provided status if the condition is true. This extension is not used for COTS rules but is provided for your customization use.

- **Fnd0SMRollupStatus**

  Rolls up statuses.

© 2025 Siemens

Subscriptions

Configure subscription conditions using the Business Modeler IDE

You can use the Business Modeler IDE to create conditions that can be used when creating subscriptions.

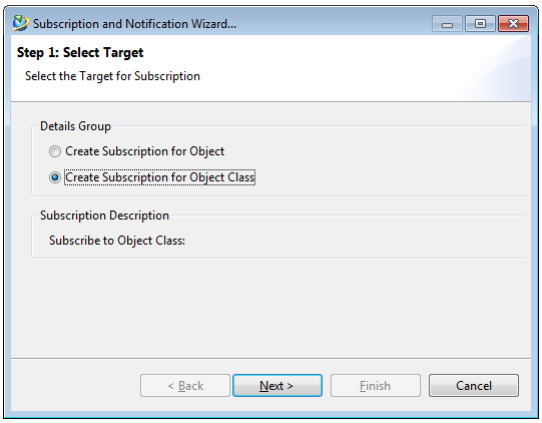
1. Create a condition in the Business Modeler IDE to be used for subscriptions. The condition signature must contain a workspace object and **UserSession** object. The condition appears in the subscription creation dialog box in the client user interface if it is valid for the selected object for which the subscription is being created.

2. Choose **BMIDE→Save Data Model** to save your changes to the template. Then choose **BMIDE→Generate Software Package** to package the template. Finally, use Teamcenter Environment Manager to install the packaged template to your server.

3. View where the new subscription condition appears in the rich client:

   a. In My Teamcenter, select an object for which you want to create a subscription and choose **Tools→Subscribe** on the menu bar, or right-click the object and choose **Subscribe** on the context menu.

   b. Select **Create Subscription for Object Class** and click **Next**.

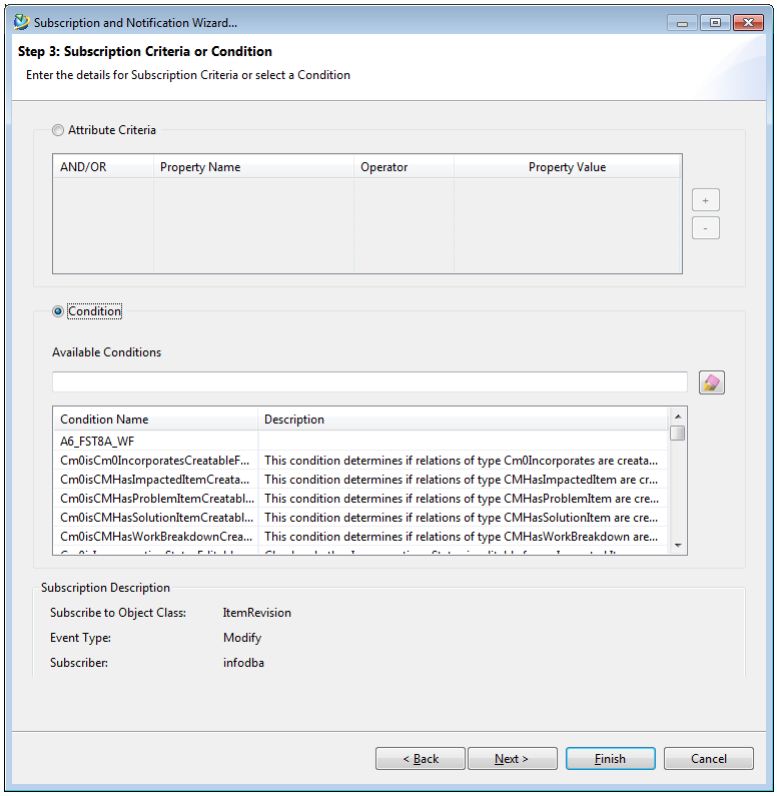


   c. Select the event type and click **Next**.

   d. Click the **Condition** button.

      All the valid conditions are displayed. Conditions are filtered for the type of selected target object. When you create a condition in the Business Modeler IDE to be used in subscriptions, it appears in this dialog box if the condition contains **UserSession** object and a valid workspace object in the signature.

© 2025 Siemens



Supplier Relationship Management

Configure Supplier Relationship Management using the Business Modeler IDE

Use the Business Modeler IDE to create custom objects used by Supplier Relationship Management (SRM). Supplier Relationship Management is also known as Teamcenter Sourcing.

Note:

Before working with Supplier Relationship Management objects, you must **install the following templates to your project**:

- SRM Integration (**srmintegration_template.zip** file)
- Vendor Management (**vendormanagement_template.zip** file)
- SRM Connect (**srmconnect_template.zip**, available for installation from the Supplier Relationship Management installation files)

© 2025 Siemens

Configure lists of values (LOVs) for Supplier Relationship Management

You can use Teamcenter data exchange and Bid Package Exchange to send design and bid package data to Supplier Relationship Management. To prepare for this exchange, you must create properties and accompanying lists of values (LOVs) that contain the partition and template names in the bid packages.

1. If you have not already done so, **create a custom template project** to hold your data model changes.

2. Ensure that you have **installed the following templates** to your project:

   - SRM Integration (**srmintegration_template.zip** file)
   - Vendor Management (**vendormanagement_template.zip** file)
   - SRM Connect (**srmconnect_template.zip**, available for installation from the Supplier Relationship Management installation files)

3. Create partition, template, and event name properties on the **BidPackageRevision** business object.

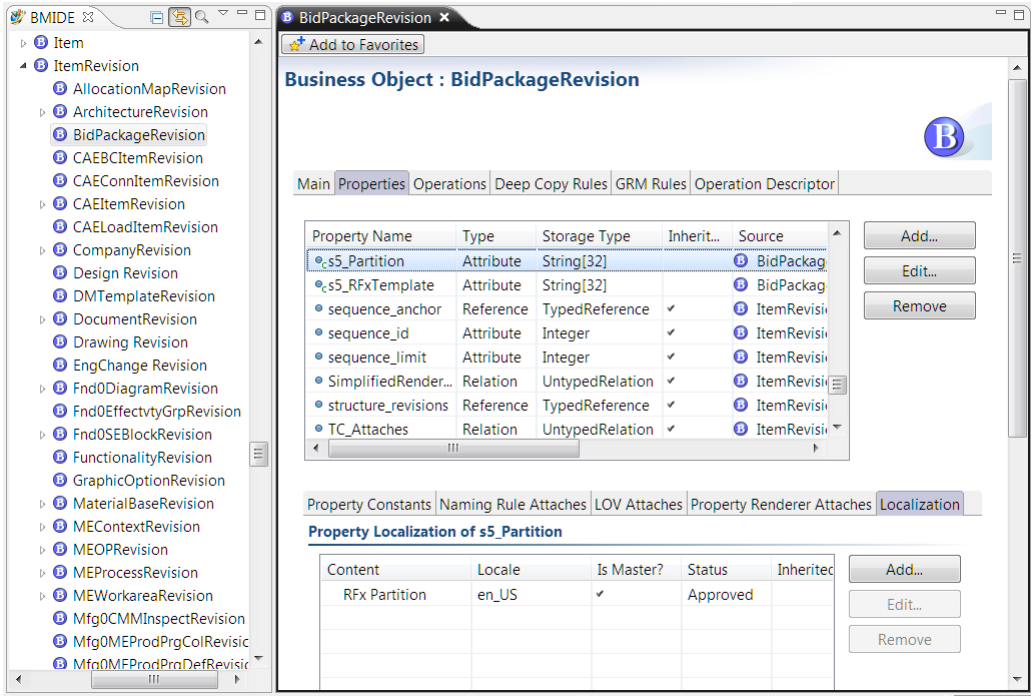
   a. Open the **BidPackageRevision** business object and click the **Properties** tab.

   b. Create a partition property. For this example, create a *project-prefix***Partition** property with the **RFx Partition** display name.

   c. Create a template property. For this example, create a *project-prefix***RFxTemplate** property with the **RFx Template** display name.

   d. Create an event name property. For this example, create a *project-prefix***RFxEventName** property with the **RFx Event Name** display name.

© 2025 Siemens



4. Create the partition and template LOVs and use them to create a cascading LOV.

   a. Create an LOV that holds the names of the partitions.

      For example, create an LOV named *project-prefix***Partition** and add a value for each partition.

      For this example, the values are **Acme** and **General Pinnacle**. The name of the LOV values (not the display name) must match the name of the RFx partitions.

   b. Create an LOV for each partition that holds the names of the available templates for that partition.

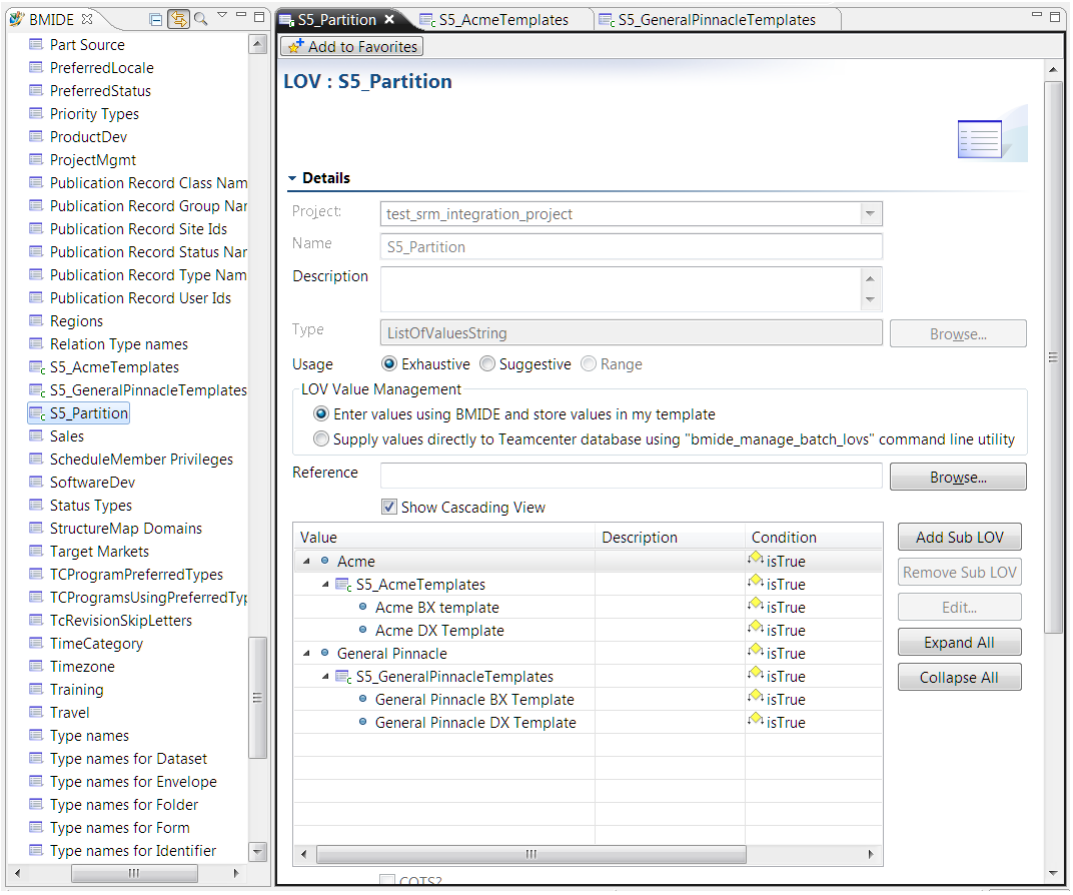
      For example, create a *project-prefix***AcmeTemplates** and *project-prefix***GeneralPinnacleTemplates** and add template values to each.

   c. Open the *project-prefix***Partition** LOV and select the **Show Cascading View** check box.

   d. For each partition LOV value, add the corresponding template LOV.

      For example, add the *project-prefix***AcmeTemplates** LOV as a sub-LOV to the **Acme** value, and add the *project-prefix***GeneralPinnacleTemplates** LOV as a sub-LOV to the **General Pinnacle** value.

© 2025 Siemens



5. Attach the partition LOV to the partition property.

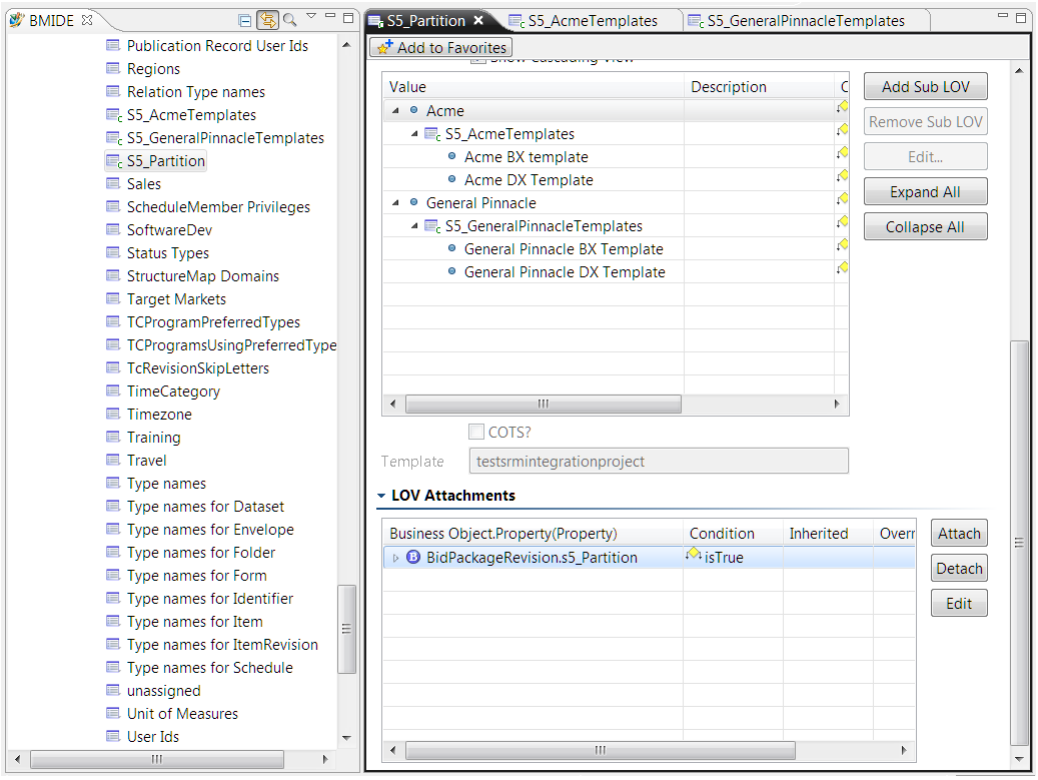
   When the end user selects the **RFx Partition** property in the user interface, a list of partitions appears.

   a. Open the *project-prefix***Partition** LOV.

   b. Scroll down to the **Using the LOV Attachments** table and click the **Attach** button.

   c. Attach the *project-prefix***Partition** LOV to the *project-prefix***Partition** property on the **BidPackageRevision** business object.

© 2025 Siemens



6. Create an interdependent LOV.

   When the end user selects a partition on the **RFx Partition** property box, the **RFx Template** box only shows the templates for the selected partition.

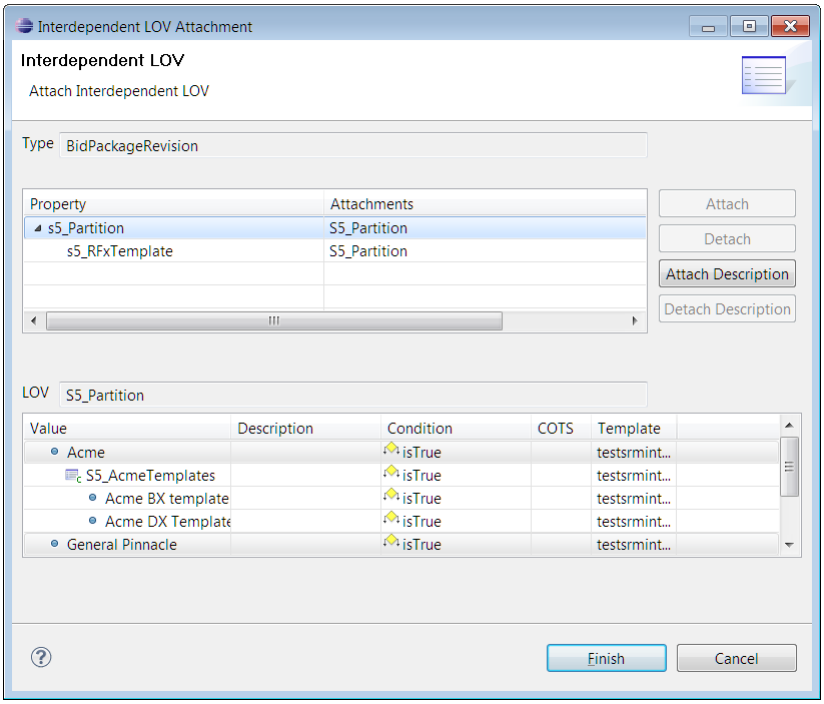
   a. With the *project-prefix***Partition** LOV still open and displaying the **Using the LOV Attachments** table, click the **Edit** button.

   b. In the **Interdependent LOV** dialog box, select the *project-prefix***Partition** property and click the **Attach** button.

   c. Attach the *project-prefix***RFxTemplate** property to the *project-prefix***Partition** property

   d. Click **Finish**.

© 2025 Siemens



7. Verify the new LOVs work correctly in the end-user interface.

   a. **Package the project** and install the resulting template to the server.

   > Note:
   >
   > The server must have the Vendor Management and SRM Integration applications installed.

   b. Log on to the rich client.

   c. Choose **File→New→Vendor Management→Bid Package** and create the bid package.

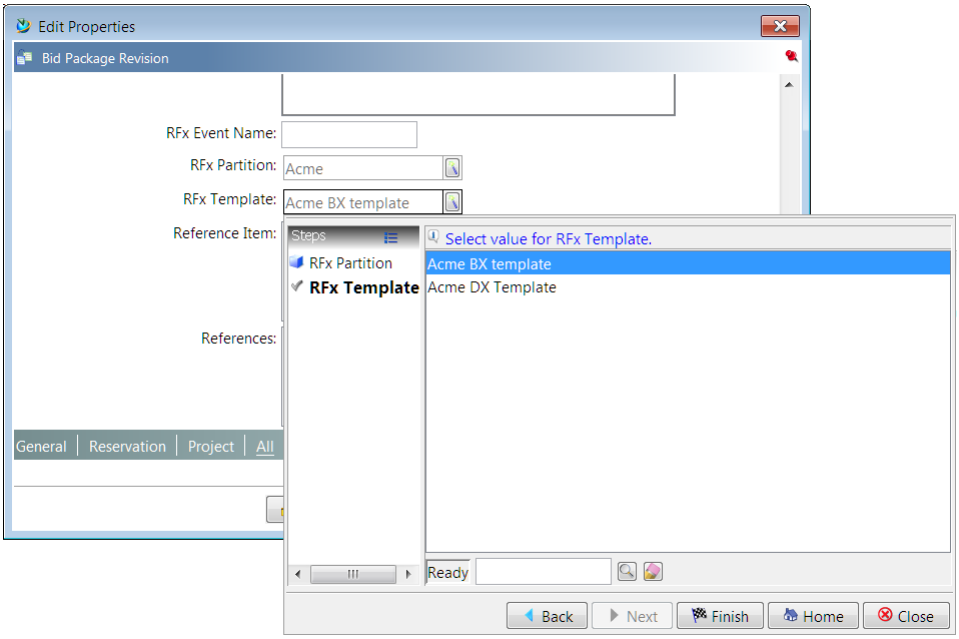
   d. Right-click the bid package revision, choose **Edit Properties**, and check out the bid package revision.

   e. To see all the properties on the bid package revision object, click the **All** link at the bottom of the dialog box, scroll down to the bottom of the dialog box, and select **Show empty properties**.

   f. Scroll to the **RFx Partition** and **RFx Template** properties.

   g. Click the box in the **RFx Partition** box and choose the partition and the template. Note that after you choose the partition, you are prompted to select the template. This is as a result of the interdependent LOV configuration.

© 2025 Siemens



h. Now that the partition and template information is available using the new properties and LOVs, you are ready to transfer data to Supplier Relationship Management.

Systems Engineering

Configure Systems Engineering using the Business Modeler IDE

1. Install the Systems Engineering template (**systemsengineering_template.xml** file) to your Business Modeler IDE reference templates.

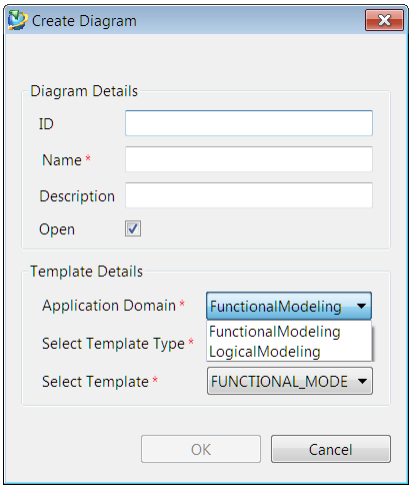
2. Add the **Systems Engineering** template to your project as a dependent template.

3. Configure and create custom objects used by the Systems Engineering application.

Add an application domain for diagrams

In Systems Engineering, a diagram is a graphical representation of a system. You can create diagrams in Systems Engineering by right-clicking an object and choosing **Create Diagram**. When you create a diagram, select the Teamcenter application you want the diagram to be opened in by clicking in the **Application Domain** box.

© 2025 Siemens



Using the Business Modeler IDE, you can add an application domain to the list of available domains using the **Fnd0TcApplication** list of values. Then you can specify the default perspective to use for that application domain using the **NE_diagram_domains_for_perspectives** preference.

The following example adds an application domain for the Systems Engineering application.

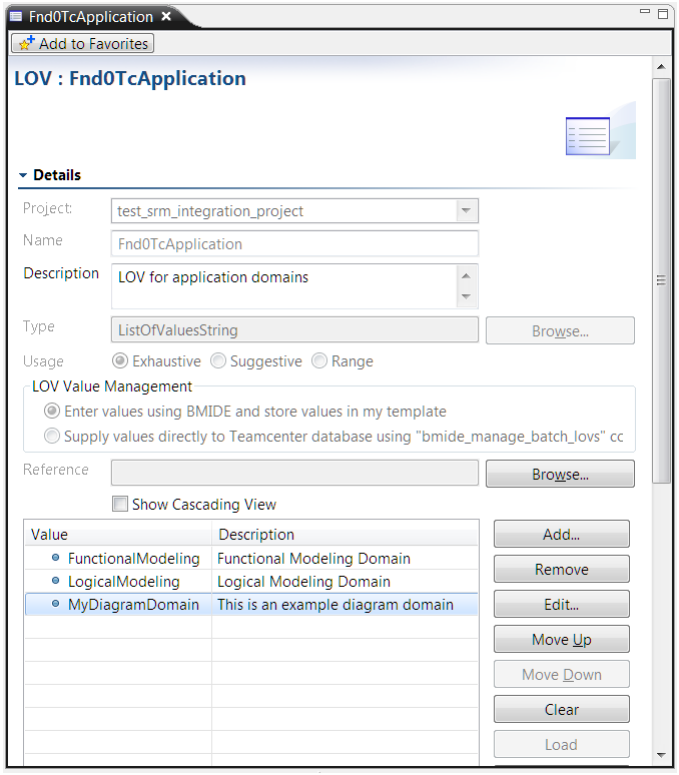
> Note:
>
> This functionality is confined to defining and using a new domain in the Systems Engineering application.

1. If you have not already done so, **create a custom template project** to hold your data model changes.

2. Open the **Extensions\LOV** folders and open the **Fnd0TcApplication** list of values.

3. Click the **Add** button to the right of the values list and add a diagram domain, for example, **MyDiagramDomain**.

© 2025 Siemens



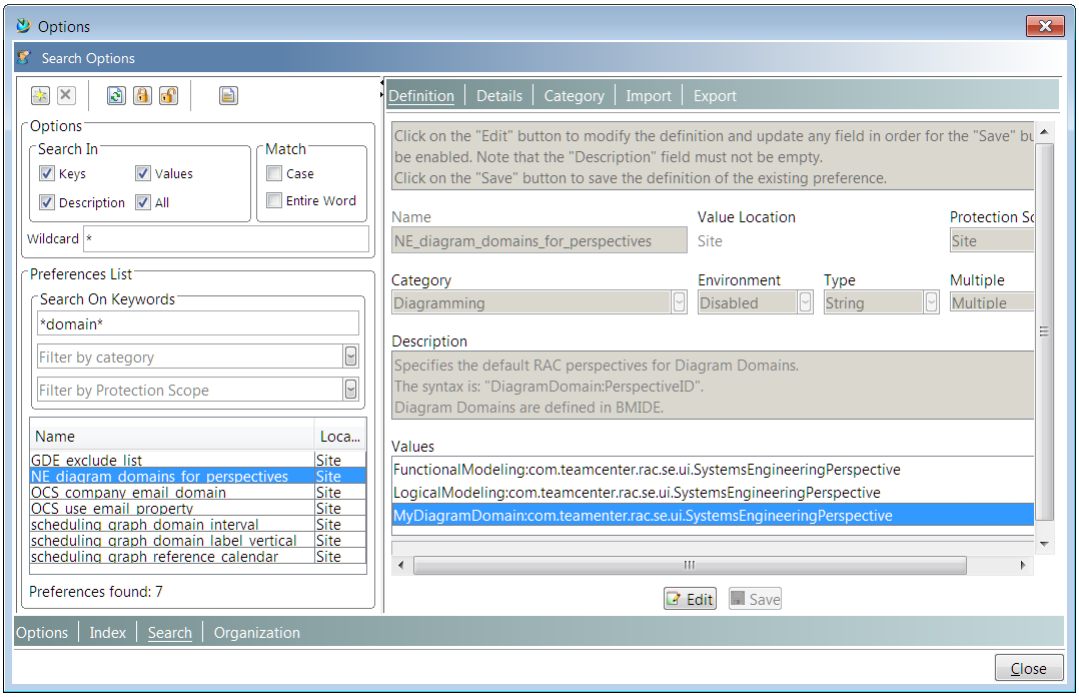
4. Choose **BMIDE→Save Data Model** to save your changes to the template. Then choose **BMIDE→Generate Software Package** to package the template. Finally, use Teamcenter Environment Manager to install the packaged template to your server.

5. Open the **NE_diagram_domains_for_perspectives** preference and add the following value:

   ```
   MyDiagramDomain:com.teamcenter.rac.se.ui.SystemsEngineeringPerspectiv
   e
   ```

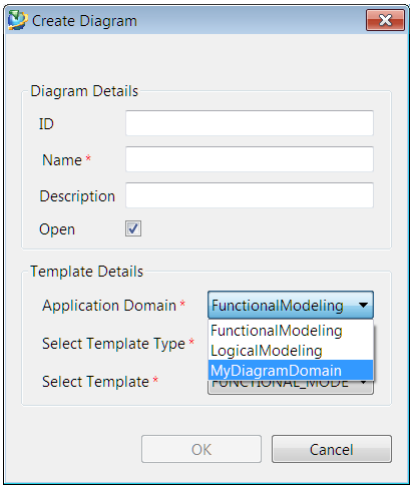
   This maps the **MyDiagramDomain** value entered on the LOV to the Teamcenter application perspective you want the diagram to be opened in, which is Systems Engineering.

© 2025 Siemens



6. In Systems Engineering, right-click an object, choose **Create Diagram**, and click in the **Application Domain** box.

   The **MyDiagramDomain** value you entered to the LOV appears in the menu.



7. After creating a diagram with the new application domain selected, when you open the diagram later, it opens in the perspective defined in the **NE_diagram_domains_for_perspectives** preference.

© 2025 Siemens

Creating an icon for use in the Systems Engineering and Requirements Management BOM view

You can use the **Fnd0Icon** business object constant to **assign an icon to a business object type** and to change the icon based on the state of the property.

Perform the following sample steps in the Business Modeler IDE to set the icon property rendering for use in the Systems Engineering and Requirements Management BOM view. In this example, add a **b4_RiskPriority** property (display name **Risk Priority**) on the revision business object. Attach a **RiskPriorityLOV** list of values to the property that has the **High**, **Medium**, and **Low** values. The icon changes based on the property selection.

1. Import the icons to use (for example, **B4_Normal.png**, **B4_Low.png**, **B4_Medium.png**, **B4_High.png**).

2. Create the **B4_RiskReq** business object as a child of the **Requirement** business object.

3. Add the **b4_RiskPriority** property to the **B4_RiskReqRevision** business object type.

   > Note:
   >
   > The property should be string only.

4. Create the **B4_PriorityRiskLOV** LOV with the **Low**, **Medium**, and **High** values and attach it to the **b4_RiskPriority** property.

5. Attach the **Fnd0Icon** business object constant to the **B4_Normal.png** icon on the **B4_RiskReqRevision** business object.

6. Create a property renderer called **B4_RiskPR**.

7. Add the following render definition:

```
<?xml version="1.0" encoding="UTF-8"?>
     <icons Version="1.0">
       <propertyMap name="PriorityRiskMap">
          <item key="High" value="B4_High.png"/>
          <item key="Medium" value="B4_Medium.png"/>
          <item key="Low" value="B4_Low.png" />
       </propertyMap>
       <primaryIcon source="B4_Normal.png"/>
       <overlayIcon source="b4_RiskPriority"
mapName="PriorityRiskMap" />
     </icons>
```


© 2025 Siemens

8. In the **Property Renderer Attachments** box, attach the *business-object*.**object_string**, in this case, **B4_RiskReqRevision.object_string**.

9. Create a compound property pointing to the **B4_RiskReqRevision.b4_RiskPriority** property called **RiskPriorityBOM** on the **Fnd0RequirementBOMLine** business object.

10. Save and deploy.

11. Perform the following in Teamcenter:

    a. Add the **B4_RiskReq** business object to the **TcAllowedChildTypes_RequirementSpec** preference.

    b. Create a requirement specification and add three **B4_RiskReq** requirements.

    c. Add the **RiskPriorityBOM** property to the requirement BOM view.

Validation Manager

Configure Validation Manager using the Business Modeler IDE

Use the Business Modeler IDE to create custom objects used by the Validation Manager application. Validation Manager objects are provided by the Foundation template. No additional templates are needed.

You can add new business objects that provide extended properties and specialized behaviors in support of the associated validation agent. The **NXCheckMate** and **NXCMValData** business objects support Validation Manager interaction with the Check-Mate application in NX Integration. The **NXRDDV** and **NXRDDVValData** business objects support the Validation Manager interaction with the Requirement-Driven Design Validation application in NX Integration.

Validation Manager business objects

You can create children of the following Validation Manager business objects:

- **NXCheckMate**

  Represents the NX Check-Mate agent object and is a child of the **Item** business object. There is only one instance of the **NXCheckMate** business object allowed in the database.

- **NXCheckMateRevision**

  Represents the NX Check-Mate agent revision object and is a child of the **ItemRevision** business object. There is only one instance of the **NXCheckMateRevision** business object allowed in the database

© 2025 Siemens

- **NXCMValData**

  Represents the NX Check-Mate checker objects and is a child of the **Item** business object.

- **NXCMValDataRevision**

  Represents a unique checker object in the system and is a subclass of the **ItemRevision** business object.

- **NXRDDV**

  Represents the NX RDDV agent object and is a subclass of the **Item** business object. There is only one instance of the **NXRDDV** business object allowed in the database.

- **NXRDDVRevision**

  Represents the NX RDDV agent revision object and is a child of the **ItemRevision** business object. There is only one **NXRDDVRevision** business object allowed in the database.

- **NXRDDVValData**

  Represents the NX RDDV checker objects and is a child of the **Item** business object.

- **NXRDDVValDataRevision**

  Represents a unique checker object in the system and is a child of the **ItemRevision** business object.

- **TC_validation_data**

  Relates an **NXCheckMateRevision** object to an **NXCMValData** object. All **NXCMValData** objects that are linked from an **NXCheckMateRevision** object with this relation represent the available NX Check-Mate checker objects in the system.

- **TC_validation_tool**

  Relates an **NXCheckMateRevision** object to a **Tool** object. The **Tool** object contains necessary data for the system to invoke the defined validation utility for a validation agent revision.

- **ValidationResult**

  Represents a validation result.

Validation Manager properties

The following properties are available on Validation Manager business objects:

© 2025 Siemens

- **allow_override_results**

  Indicates if the execution of the associated **NXCMValDataRevision** or **NXRDDVRevision** objects are skipped for overridden results of an agent revision.

- **is_client_utility**

  Indicates if the validation utility defined by a **Tool** business object is invoked from the client machine. This is a logical property of the **NXCheckMate** and **NXRDDV** business objects.

- **is_mandatory**

  Indicates if an **NXCMValDataRevision** or **NXRDDVValDataRevision** object is a mandatory checker.

- **override_reason_mandatory**

  Indicates if an override reason is required when a result override request is initiated. This is a logical property of the **NXCheckMateRevision** and **NXRDDVRevision** business objects.

- **valdata_name**

  Defines a unique name of an **NXCMValData** object within the **NXCheckMate** agent. This string property is on the **NXCheckMateValData** and **NXRDDVValData** business objects.

- **validation_arguments**

  Stores the command line arguments when the validation utility defined by a **Tool** business object is invoked.

- **validation_category**

  Describes the category of a checker. This is a string property on the **NXCMValDataRevision** business object.

- **validation_closure_rule**

  Points to a **ClosureRule** object in the system. The closure rule object is used to find validation target objects.

- **validation_parameters**

  Points to a **ValidationParams** object. This is a tag reference property on the **NXCMValDataRevision** business object.

© 2025 Siemens

Wiring Harness Design Tools Integration

Configure Wiring Harness Design Tools Integration using the Business Modeler IDE

Use the Business Modeler IDE to create custom objects used by the Wiring Harness Design Tools Integration application.

> Note:
>
> Before working with Wiring Harness Design Tools Integration objects, you must **install the Wire Harness Configuration templates (hrn_template.zip file)** to your project:

You can create the following business objects:

Connection
Functionality
Interface
Network
ProcessVariable
PSSignal
Signal
TC_LINK

Workflow Designer

Configure Workflow Designer using the Business Modeler IDE

Use the Business Modeler IDE to create custom objects used by the Workflow Designer application. Workflow Designer objects are provided by the Foundation template. No additional templates are needed.

You can use the also Business Modeler IDE to create dynamic participants and to register custom workflow handlers.

> Note:
>
> Workflow handlers such as **EPM-set-property** cannot recognize run-time or compound properties. These handlers only set properties that have a persistent attribute on some object, and they cannot influence the setting of run-time or compound properties.

© 2025 Siemens

Create dynamic participants

Dynamic participants are users who are dynamically assigned tasks in workflows based on their roles. These dynamic participants are children of the **Participant** business object, and for change management can include **Analyst**, **ChangeSpecialist1**, **Requestor**, and **Change Contributor**, among others.

You can create new participant business object types to represent participants in the Workflow process.

After you create a new participant type, assign keywords to the participant to ensure that it is assigned to the correct workflows. State the keyword using the **ParticipantHandlerKeyword** business object constant. The assignment Workflow handlers dynamically pick up the keywords to designate that people assigned to that new participant type are to be assigned to the task or signoff.

1. If you have not already done so, **create a custom template project** to hold your data model changes.

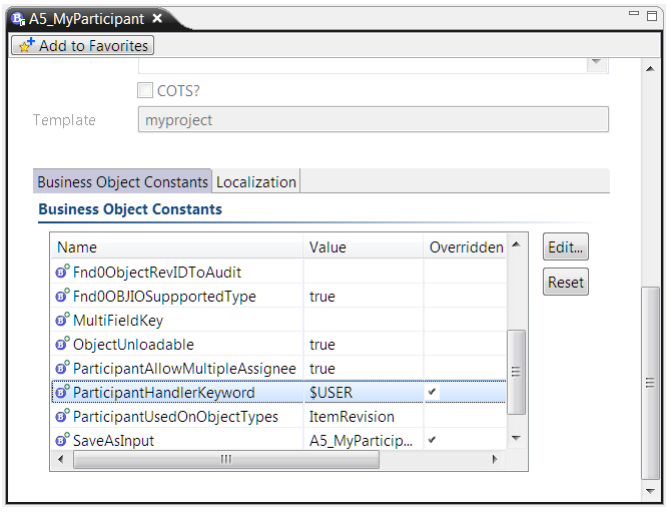
2. In the **Business Objects** folder, search for the **Participant** business object.

3. Create a child of the **Participant** business object by right-clicking the business object and choosing **New Business Object**.

4. Perform the following steps to assign handler keywords to the new participant business object type:

   a. Open the new business object type.

   b. On the **Main** tab in the **Business Object Constants** table, select the **ParticipantHandlerKeyword** business object constant.



   c. Click the **Edit** button to the right of the **Business Object Constants** table.

© 2025 Siemens

d. In the **Business Object Constant** dialog box, type the handler keywords to use for this participant. For example, type **$USER**, **$GROUP**, or **$ROLE**

e. Use the **ParticipantUsedOnObjectTypes** business object constant to define the item revision business object types that the participant can be used be used with, and use the **ParticipantAllowMultipleAssignee** business object constant to allow the participant to be one of multiple assignees.

> Note:
>
> You can set the **WRKFLW_task_assignee_dynamic_participant_sync** preference to **true** to automatically assign the dynamic participant on the change when a user reassigns a task.
>
> For example, if User 1, the change analyst, forwards a task to User 2, the change analyst role is reassigned to User 2. In the same way, if User 1 forwards tasks to User 2 while User 1 is out of the office, all incoming tasks are assigned to User 2 during that time. The change analyst role is reassigned to User 2 in all the changes associated with the task.

f. A dynamic participant, **Cm0ChangeContributors**, is available for the **ChangeItemRevision** preference. The **fnd0IsParticipant** operation is also available, verifying that the provided user is present as the given participant type of the item revision.

Modify participants based on condition

Participants assignments can be controlled using BMIDE. This process contains the following steps:

Step 1: Create a new BMIDE constant in the format that follows:

*<Participant Type>***AssignableCondition**

Step 2: Create a new BMIDE condition.

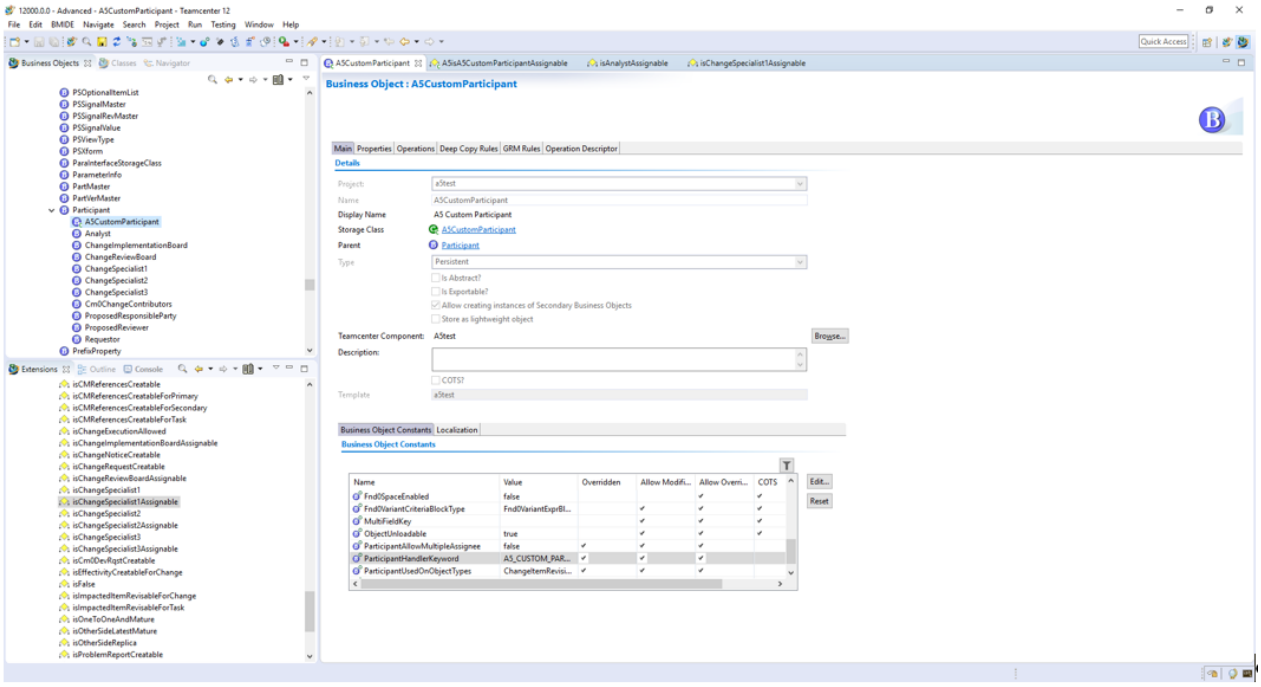
For example, **isCustomParticipantAssignable**.

Step 3: Add the new BMIDE constant created in Step 1 to **Participant type** and provide its value as new BMIDE condition created in Step 2.

The following example illustrates how to create a BMIDE constant for the Custom Participant type.

1. Create a subclass under **Participant**.

© 2025 Siemens



2. Update the following BO constant:

   ParticipantAllowMultipleAssignee = false

   ParticipantHandlerKeyword = add your key; for example **A5_CUSTOM_PARTICIPANT**

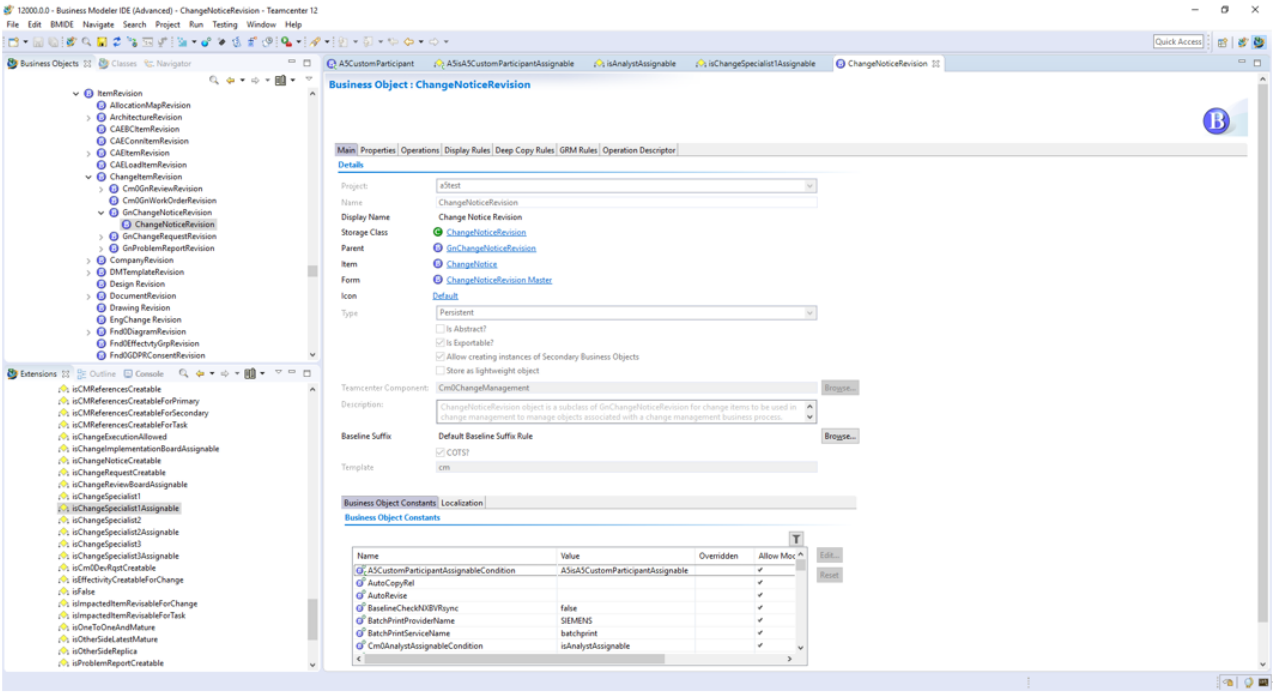
   ParticipantUserOnObjectTypes = ChangeItemRevision

3. Create a BO constant where you want to make **Assign the custom Participant** assignable. For example:

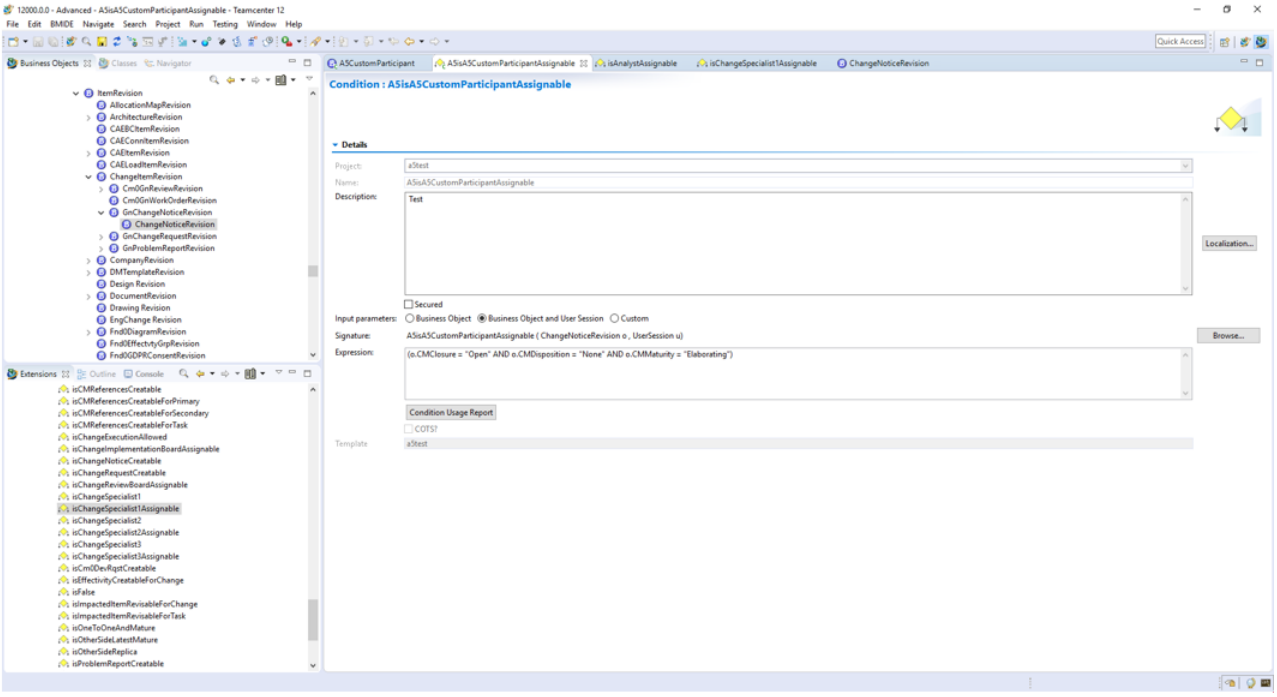
   A5CustomParticipantAssignableCondition = A5isA5CustomParticipantAssignable

   > Warning:
   >
   > The BO Constant must follow the following format, otherwise it will not be evaluated:
   >
   > *Custom Participant Type* + **AssignableCondition**

© 2025 Siemens

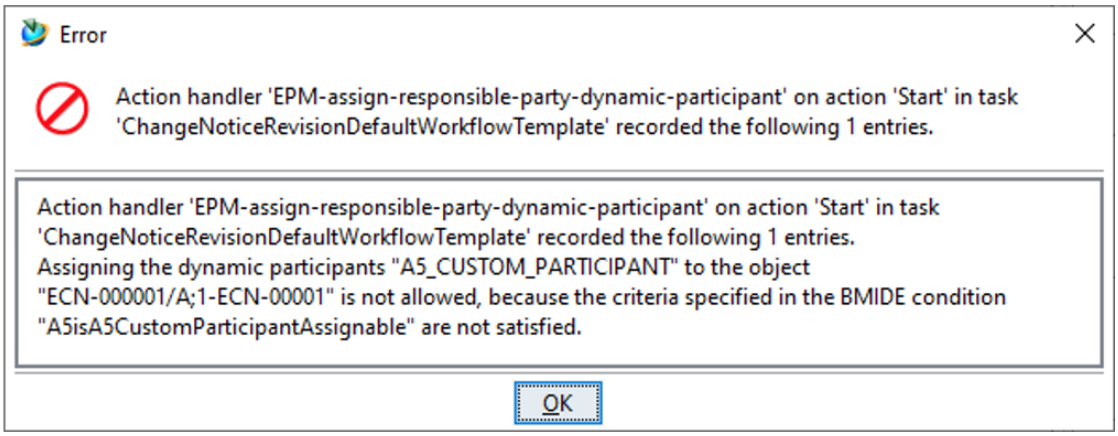


4. Create the condition to be evaluated when adding participants.



If the condition is evaluated as false, the following message displays:

© 2025 Siemens



Register custom workflow handlers

You can register custom workflow action and rule handlers in the Business Modeler IDE. After registration, the shared library is loaded at logon and the handlers are available in Workflow Designer.

To register the custom handlers, create a custom **extension rule** and execute it as a base action on the **BMF_SESSION_register_epm_handlers** operation on the **Session** business object. Then call your custom workflow handlers in the code used by the extension.

1. If you have not already done so, **create a custom template project** to hold your data model changes.

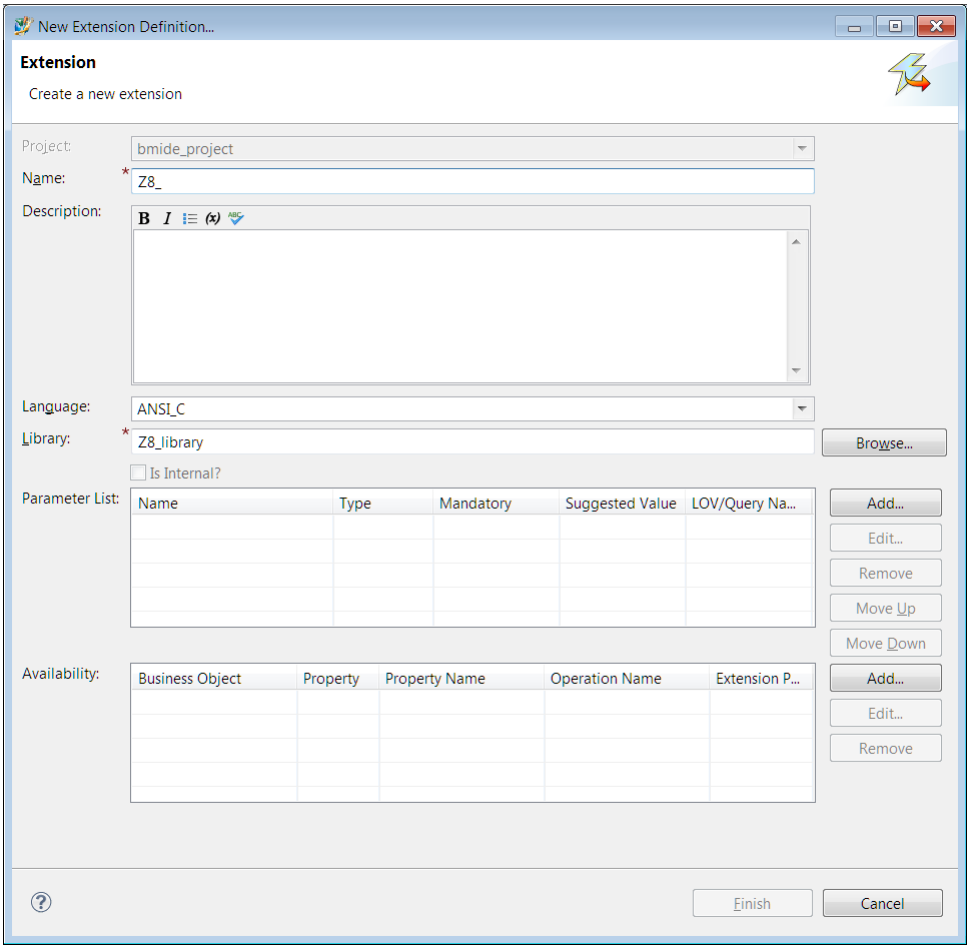
2. Create a library to hold the custom workflow handlers. Open the **Extensions\Code Generation** folders, right-click the **Libraries** folder, and click the **New Library** button.

3. Create the custom extension rule.

    a. Open the **Extensions\Rules** folders, right-click the **Extensions** folder, and choose **New Extension Definition**.

    The New Extension Definition wizard runs.

© 2025 Siemens



b. In the **Extension** dialog box, type the name of your custom extension in the **Name** box and click the **Add** button to the right of the **Availability** table.

> Note:
>
> Choose whatever name you want for your custom extension. It is preceded by your project's naming prefix, for example, **Z8_** as shown in the following example.

c. In the **Extension availability** dialog box, perform the following steps:

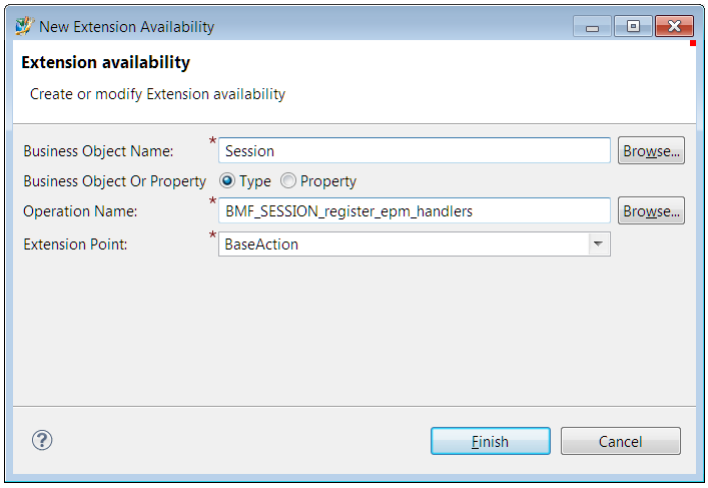
A. Click the **Browse** button to the right of the **Business Object Name** box and select the **Session** business object.

B. Click the **Browse** button to the right of the **Operation Name** box and select the **BMF_SESSION_register_epm_handlers** operation.

C. Click the arrow in the **Extension Point** box and select **BaseAction**.

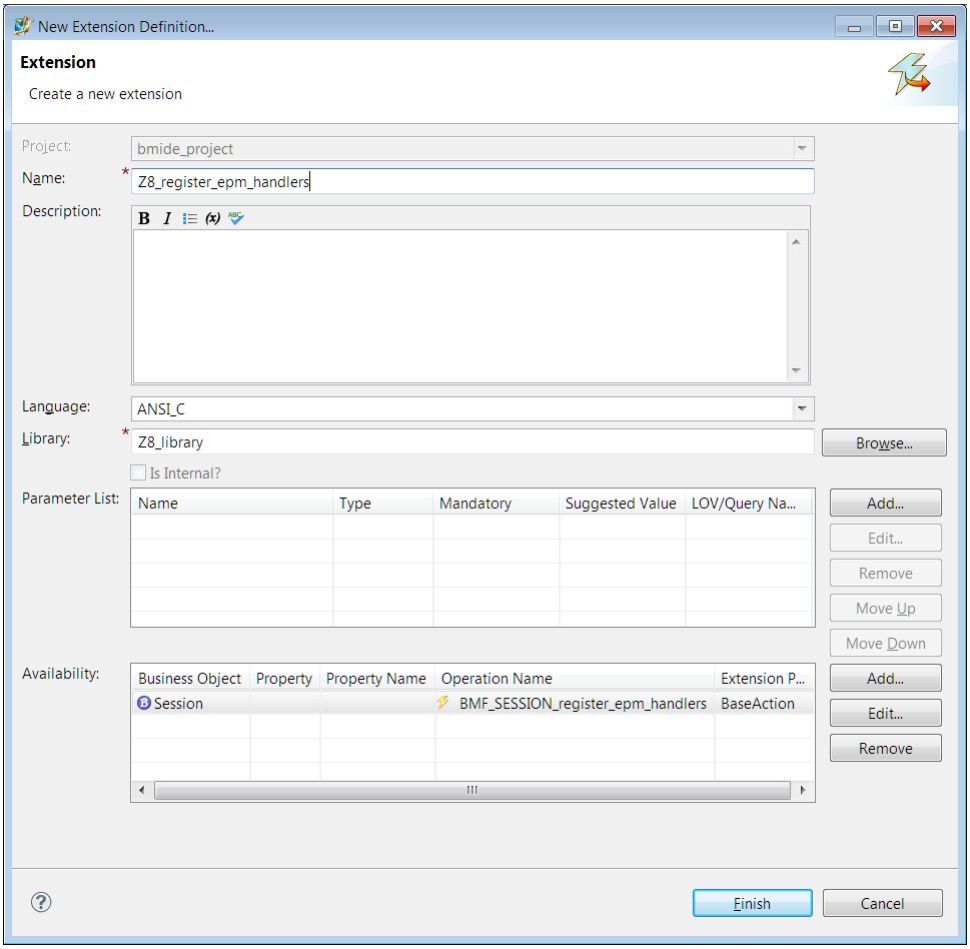
D. Click **Finish**.

© 2025 Siemens



d. Notice that the extension is made available on the **BMF_SESSION_register_epm_handlers** operation on the **Session** business object.

Click **Finish**.



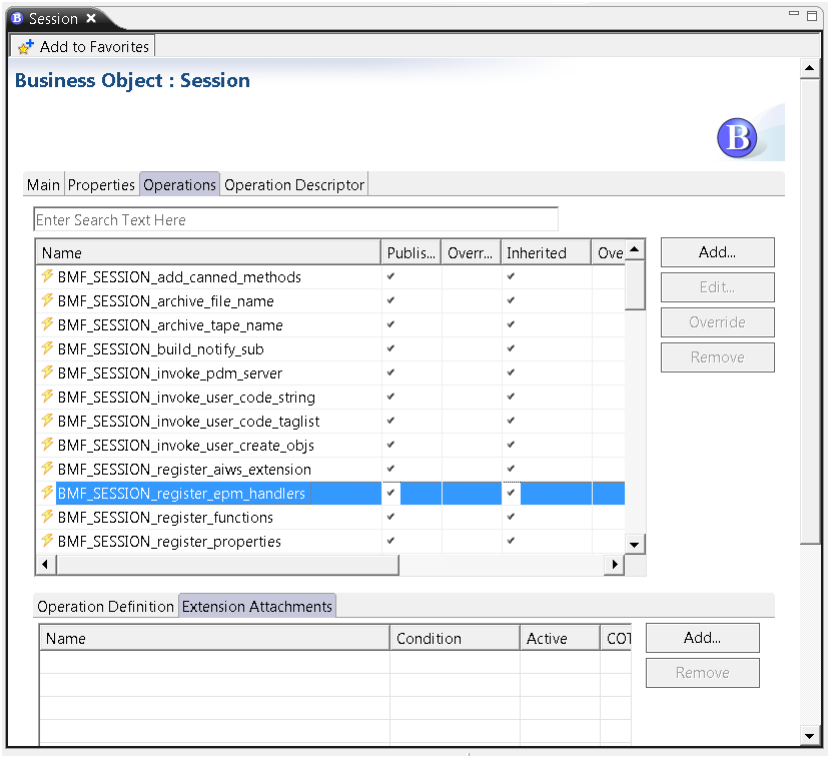
© 2025 Siemens

4. Add your custom extension as a base action on the **BMF_SESSION_register_epm_handlers** operation.

   a. Open the **Session** business object and click the **Operations** tab.

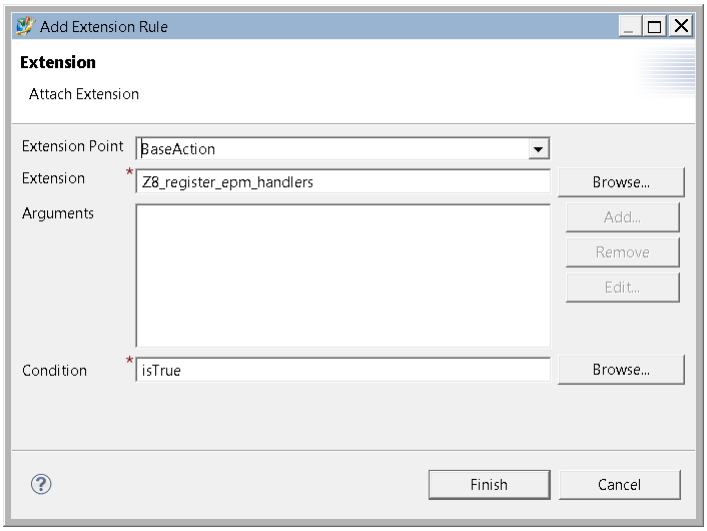
   b. Select the **BMF_SESSION_register_epm_handlers** operation.

   c. In the **Extension Attachments** tab, click the **Add** button to the right of the table.



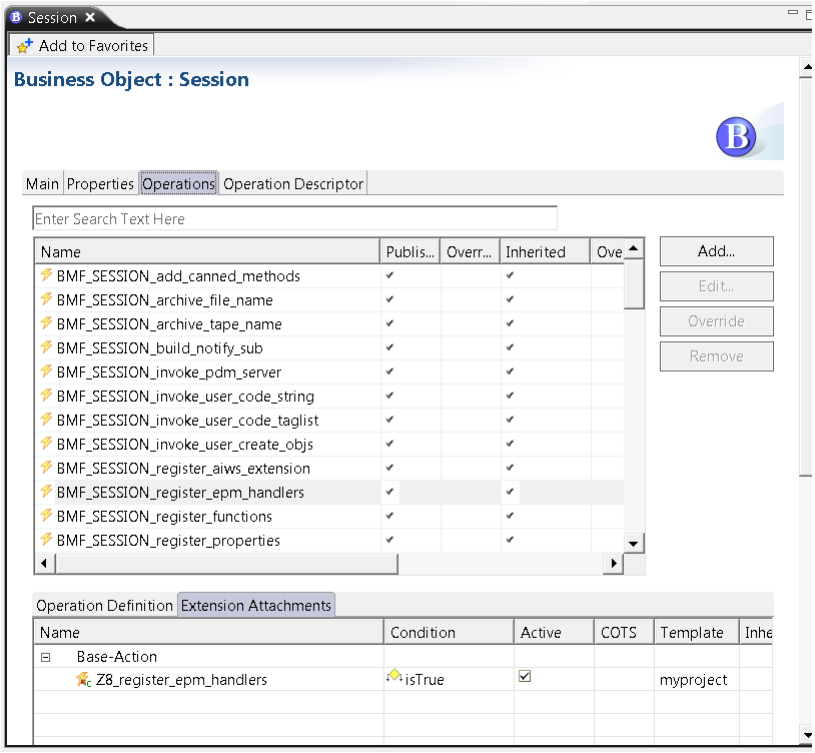
   d. Click the **Browse** button to the right of the **Extension** box and select the custom extension you created earlier.

© 2025 Siemens



e. Click **Finish**.

The custom extension is added as a base action on the **BMF_SESSION_register_epm_handlers** operation.



f. To save your changes to the template, on the menu bar, choose **BMIDE→Save Data Model**.

5. Write the code that takes place in Teamcenter when the extension is called. This code registers the workflow action handler and rule handler.

© 2025 Siemens

a. Ensure that your project is set up for coding so you can generate extension code.

b. Open the **Advanced** perspective by choosing **Window→Open Perspective→Other→Advanced**.

c. In the **Extensions** view of the **Advanced** perspective, under the **Rules\Extensions** folders, right-click the new extension you created and choose **Generate extension code**.

The extension boilerplate code is generated into an *extension-name***.c** C++ file and an *extension-name***.h** header file. To see these files, open the project in the **Navigator** view and browse to the **src\server***library* directory.

> Note:
>
> You may need to right-click in the view and choose **Refresh** to see the files that were generated.

d. Write your extension code in the new *extension-name***.c** and *extension-name***.h** files. You can use the standard methods of writing code for Teamcenter.

Place the following into the code to register your custom workflow handlers:

```
int Z8_register_epm_handlers( METHOD_message_t *msg, va_list args )
{
   EPM_register_action_handler("Z8-custom-action-handler",
"This is a custom action handler", Z8_customActionHandler);
   EPM_register_rule_handler("Z8-custom-rule-handler",
"This is a custom rule handler", Z8_customRuleHandler);

    return ifail;
}
```

When you follow this example, replace the text in bold with the names of your own extension and handlers.

e. Build your libraries.

6. **Package the template** and use Teamcenter Environment Manager to install the packaged template to your server.

7. Log on to Teamcenter.

The registration runs at Teamcenter logon because the registration operation is set as a base action on the **Session** business object. After registration, the shared library is loaded at logon and the handlers are available in Workflow Designer.

© 2025 Siemens

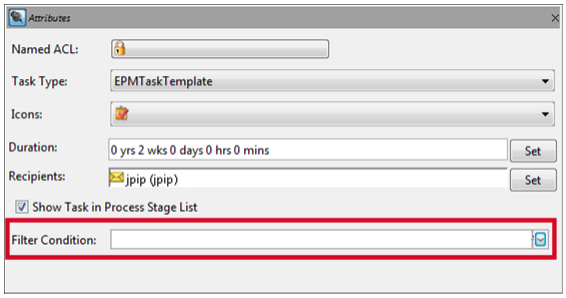
Use conditions to filter workflow template availability

You can create Business Modeler IDE conditions for use in filtering availability of workflow templates. Conditions can include versatile criteria for filtering, including:

- Session group, role, and user.
- Target object type
- Target project and target release status.
- Custom criteria, both session-specific and target-specific, that a Teamcenter administrator can create.

> Note:
>
> If there are no target objects selected, all templates are displayed during the workflow creation process.

In the rich client, in the **Attributes** dialog box for a workflow task, the **Filter Condition** attribute lets you select a condition to evaluate template filtering.



Conditions appear in the **Filter Condition** list if they meet the following requirements:

- The condition name meets the naming requirement as configured in the **Fnd0FilterCondition** dynamic LOV.

  As shipped, the naming requirement is that the condition name contains **WF**.

- The condition contains the following parameters:

  WorkspaceObject o

  ImanType t

  UserSession u

Workflow template filters affect:

© 2025 Siemens

- The **Process Template** choices displayed by the **New Process Dialog** dialog box.
- The **Process Template List** choices displayed by the **New Item** dialog box **Define Workflow Information** page.

When a workflow process is being created for a selected object, its **WorkspaceObject** parameter is used for condition evaluation to get a filtered list of workflow templates. While creating a new Item, as the object is not yet created, filter condition evaluation can use object **ImanType** parameter to get the list of filtered workflow templates. The **UserSession** parameter is used to evaluate user session values such as user, group, and role.

Example: Filter workflow templates while initiating workflow process for existing DocumentRevision objects

1. Use the following condition to filter workflow templates while initiating workflow process for a document revision.

```
Fnd0DocRevSubmitOnlyWF ( WorkspaceObject o , ImanType t ,
UserSession u)
```

2. Define the following expression:

```
(o != null) AND (o.object_type = " DocumentRevision")
```

The condition expression validates when the object submitted is not null and the object type is **DocumentRevision**.

> Note:
>
> This condition does not work for subtypes of **DocumentRevision**.
>
> Also this condition requires the object to evaluate the condition, so using this condition does not filter the templates from the **New Item** dialog box **Define Workflow Information** page.

Example: Filter workflow template for DocumentRevision and its subtypes

1. Use the following condition when any user can submit a **DocumentRevision** object or its subtype objects to a specific workflow template.

```
Fnd0DocRevSubTypesWF (WorkspaceObject o , ImanType t , UserSession u)
```

2. Define the following expression:

```
((o != null) AND u.fnd0ConditionHelper.fnd0isSubTypeOf
(o, "DocumentRevision")) OR ((t != null) AND
u.fnd0ConditionHelper.fnd0isSubTypeOf (t, "DocumentRevision"))
```


© 2025 Siemens

- The condition expression can validate when the object submitted is not null and the object type is a subtype of **DocumentRevision** using the **fnd0isSubTypeOf** function on the **fnd0ConditionHelper** class.

```
((o != null) AND u.fnd0ConditionHelper.fnd0isSubTypeOf
(o, "DocumentRevision"))
```

> Note:
>
> The **Fnd0ConditionHelper** class is a common placeholder that provides generic operations that can be used by condition expressions. See other available operations for the **fnd0ConditionHelper** business object in Business Modeler IDE. This expression clause is evaluated when the user tries to submit an existing **DocumentRevision** object to the workflow.

- Next the expression checks whether the given type is not null:

```
((t != null) AND u.fnd0ConditionHelper.fnd0isSubTypeOf
(t, "DocumentRevision"))
```

  The condition expression can validate when the given type is subtype of **DocumentRevision** using the **fnd0isSubTypeOf** function of the **fnd0ConditionHelper** class.

Example: Filter workflow templates for document revision and its subtypes when the session user belongs to the Engineering group.

1. Use the following condition when any user from **Engineering** group can submit a **DocumentRevision** or its subtypes to a specific workflow template. This example condition uses a nested condition that allows the reuse of existing conditions to write complex expressions.

```
Fnd0DocRevSubTypes_EngrGroupWF ( WorkspaceObject o , ImanType t ,
UserSession u)
```

2. Define the following expression:

```
(Condition::Fnd0DocRevSubTypesWF (o, t, u) = true) AND
(u.fnd0ConditionHelper.fnd0isSubGroupOf (u.group, "Engineering"))
```

- The expression first checks if the given object or type is a subtype of **DocumentRevision**.

```
((o != null) AND u.fnd0ConditionHelper.fnd0isSubTypeOf
(o, "DocumentRevision"))
```

  This expression reuses the existing condition described above which validates whether the given object or type is a subtype of **DocumentRevision**.

© 2025 Siemens

- Next the expression checks whether the user is a member of the **Engineering** group.

```
(u.fnd0ConditionHelper.fnd0isSubGroupOf (u.group, "Engineering"))
```

The condition expression can validate when the user is a member of the **Engineering** group or its subgroups. The expression uses the **fnd0isSubGroupOf** function on the **fnd0ConditionHelper** class to validate user membership.

Example: Filter workflow templates for any objects belonging to a specific project

1. Use the following condition for configuring the workflow template for any business objects from a specific project:

```
Fnd0_F35_ProjectDataWF (WorkspaceObject o , ImanType t , UserSession
u)
```

2. Define the following expression:

```
Function::INLIST ("F35", o.project_list, "project_name")
```

The expression checks whether the object submitted is assigned to a specific project and validates if the submitted object's **project_list** property contains a project with the name **F35**.

Create a custom form that supports setting security classification in a workflow

Using the Business Modeler IDE, you can create a custom form containing security classification properties that can be used in a workflow to set classification on an object. The workflow administrator creates a workflow that copies the classification value from the custom form to the target object.

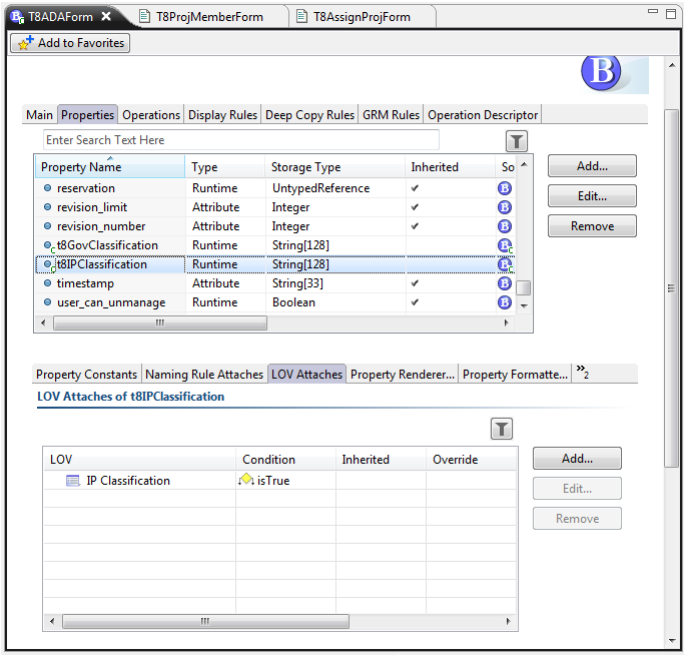
Perform the following procedure in the Business Modeler IDE to create the custom form containing security classification properties:

1. Create a custom form business object by right-clicking the **Form** business object or one of its children and choosing **New Business Object**.

2. Create a property (string type) for government classification and attach the **Fnd0GovClassification** list of values to it.

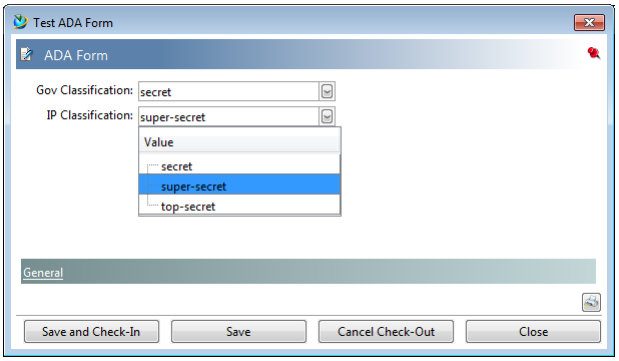
3. Create a property (string type) for IP classification and attach the **IP Classification** list of values to it.

© 2025 Siemens



4. Save the template, package the template, and install it to the Teamcenter server.

5. In the rich client, choose **File→New Form** to create an instance of the custom form. Confirm that the lists of values are attached to the properties.



6. The workflow designer configures the task template to:

   a. Display the custom form using the **EPM-display-form** handler.

   b. Create and relate the custom form to the EPM task using the **EPM-create-form** handler.

   c. Copy the classification value from the custom form to the target object using the **EPM-set-property** handler.

© 2025 Siemens

Create a custom form that supports assigning project members in a workflow

Using the Business Modeler IDE, you can create a custom form that can be used to assign members to a project using the **PROJ-assign-members** workflow handler. In this workflow, you can specify the projects and the members using handler arguments only, using properties on a form attached to the workflow template, and using a combination of handler arguments and form properties.

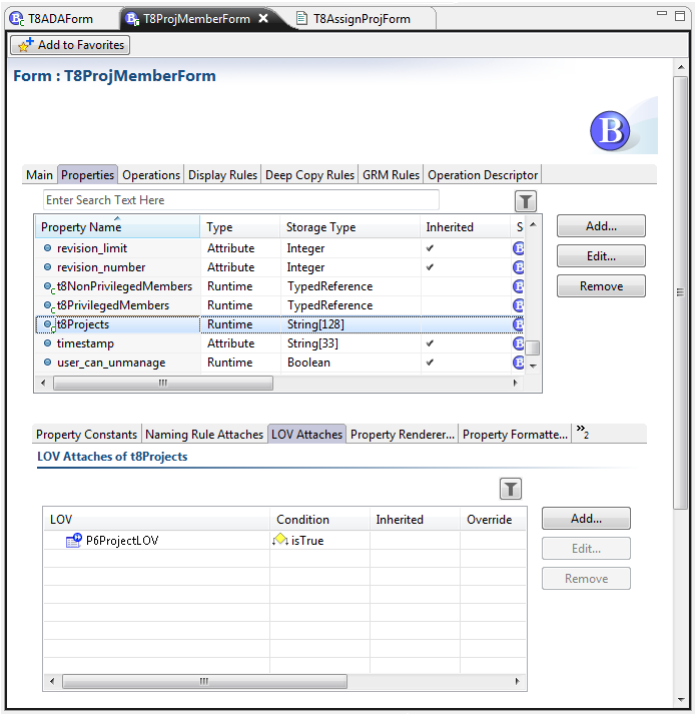
Perform the following procedure in the Business Modeler IDE to create a custom form that can be used with this workflow to assign members to projects:

1. Create a custom form business object by right-clicking the **Form** business object or one of its children and choosing **New Business Object**.

2. Create a property that is an array and a typed reference attribute type for non-privileged members. Use **GroupMember** for the reference business object. Attach the **Fnd0DynLOVGroupMember** list of values to the property. This list of values is a dynamic LOV that gathers all the available group members.

3. Create a property that is an array and a typed reference attribute type for privileged members. Use **GroupMember** for the reference business object. Attach the **Fnd0DynLOVGroupMember** list of values to the property.

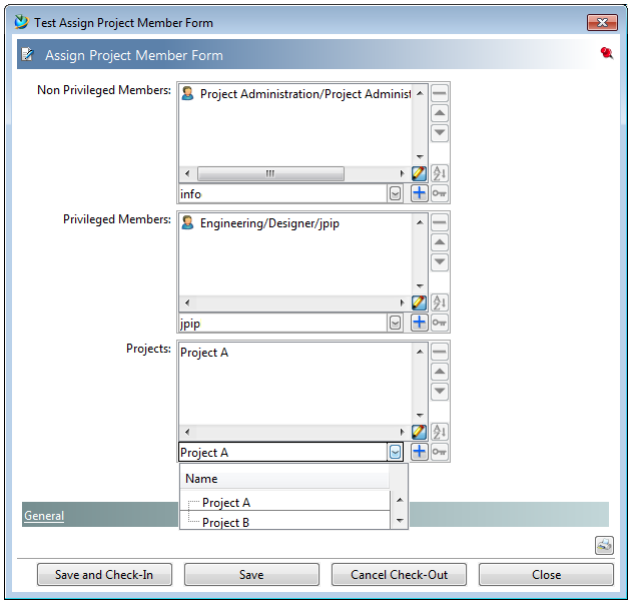
4. Create an array-type property (string type) for projects and attach a custom dynamic LOV that gathers the user's projects that the user is a member of.



5. Save the template, package the template, and install it to the Teamcenter server.

© 2025 Siemens

6. In the rich client, choose **File→New Form** to create an instance of the custom form. Confirm that the lists of values are attached to the properties.



7. The workflow administrator uses the **PROJ-assign-members** workflow handler to specify the projects and the members using the properties on the custom form attached to the workflow template.

Create a custom form that supports assigning projects in a workflow

Using the Business Modeler IDE, you can create a custom form that can be used to assign projects using the **PROJ-update-assigned-projects** workflow handler. This handler workflow assigns projects to workflow targets. Projects can be specified directly in handler arguments, indirectly using properties of a form, or both.

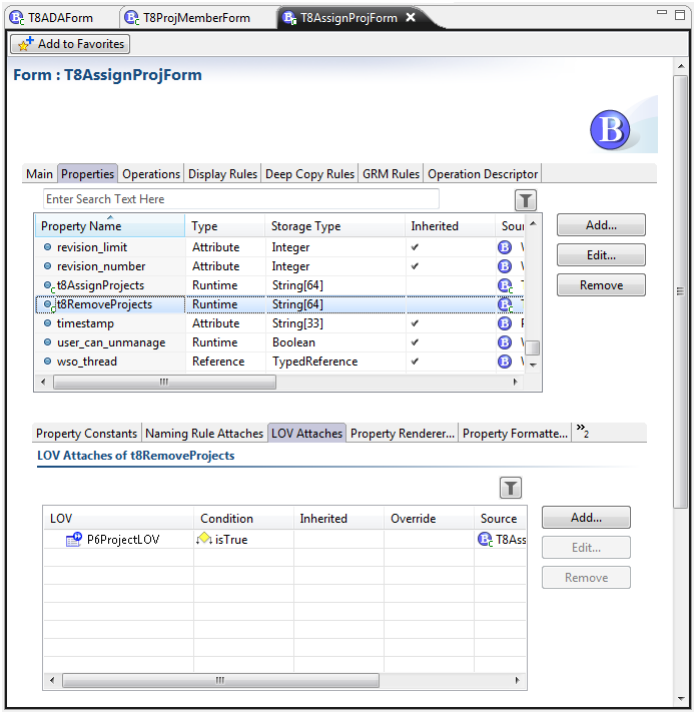
Perform the following procedure in the Business Modeler IDE to create a custom form that can be used to assign projects:

1. **Create a custom form business object** by right-clicking the **Form** business object or one of its children and choosing **New Business Object**.

2. Create an array-type property for assigning projects (string type with 64-character length) and attach a **dynamic LOV that gathers the user's projects**.

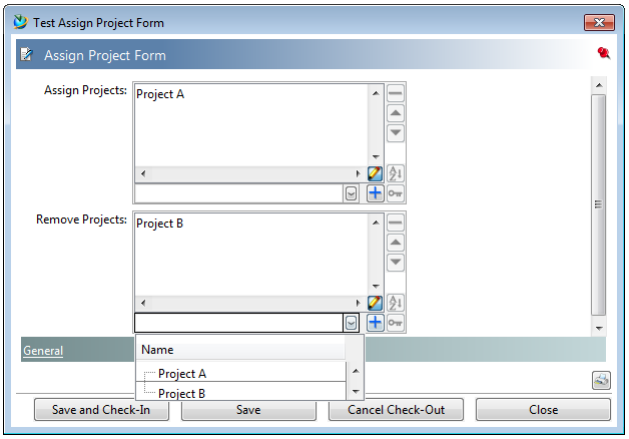
3. Create an array-type property for removing projects (string type with 64-character length) and attach a **dynamic LOV that gathers the user's projects**.

© 2025 Siemens



4. Save the template, package the template, and install it to the Teamcenter server.

5. In the rich client, choose **File→New Form** to create an instance of the custom form. Confirm that the lists of values are attached to the properties.



6. The workflow administrator uses the **PROJ-update-assigned-projects** workflow handler to specify the projects using the properties on the custom form attached to the workflow template.

Create a dynamic LOV to display project IDs

As an administrator, you can create a dynamic list of values (LOV) that displays a user's projects. When attached to a property, it allows users to select a project from the list of projects they are a member of.

© 2025 Siemens

1. Choose one of these methods:

   - On the menu bar, choose **BMIDE→New Model Element**, type **Dynamic LOV** in the **Wizards** box, and click **Next**.

   - Open the **Extensions\LOV** folders, right-click the **Dynamic LOV** folder, and choose **New Dynamic LOV**.

2. In the **New Dynamic LOV** dialog box, provide the name and description for the list of values.

3. In the **Data Type** box leave the value as **String** and for **Usage** leave the selection as **Exhaustive**.

4. Click the **Browse** button to the right of the **Type** box and select **Fnd0UserProjectsLOV**.

5. Click the **Browse** button to the right of the **Query Type** box and select **TC_Project**.

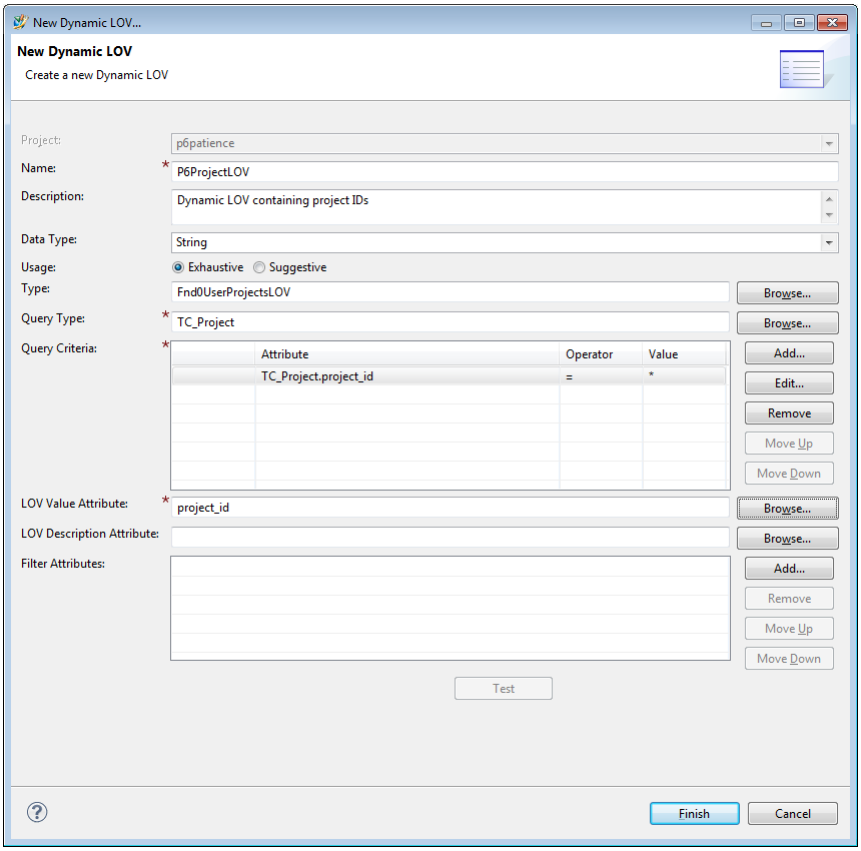
6. Click the **Add** button to the right of the **Query Criteria** table and select **project_id**.

7. In the row added to the **Query Criteria** table, type ***** in the **Value** cell.

8. Click the **Add** button to the right of the **LOV Value Attribute** box and select **project_id**.

   The dialog box should appear similar to the following example.

© 2025 Siemens



9. Click **Finish**.

10. Attach the new dynamic LOV to a property.

    When an end user clicks the arrow on the property in the user interface, the LOV table displays the project IDs.

Add custom decision labels

You can change the decision labels that are displayed in a workflow review or acknowledge task. For example, **Approve** and **Reject** are the default labels displayed to users for a review task. You can use the Business Modeler IDE to change the labels. For example, you can change the labels to **Accept** and **Deny**.

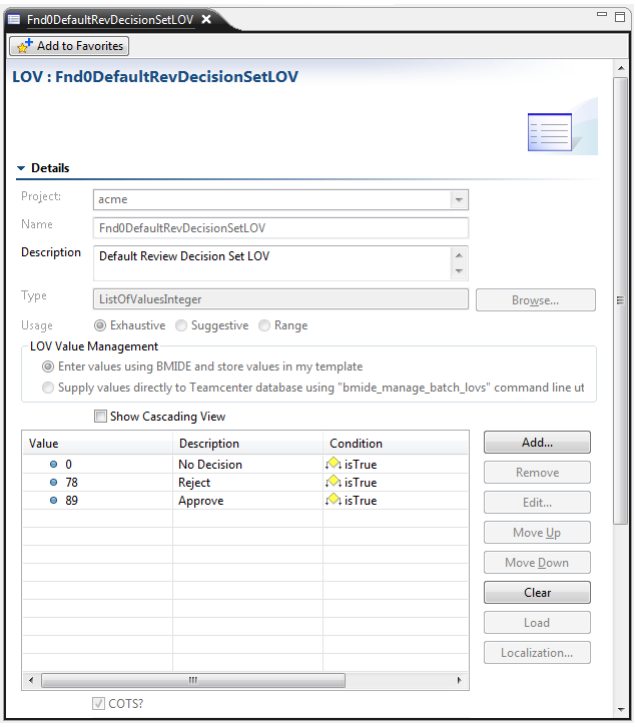
To change the labels for a review or acknowledge task, create a custom decision set LOV and add it to the appropriate master decision sets LOV (**Fnd0ReviewDecisionSetsLOV** for the review task and **Fnd0AcknowledgeDecisionSetsLOV** for the acknowledge task).

> Tip:
>
> The **Fnd0DefaultRevDecisionSetLOV** list of values contains the default review labels: **Approve**, **Reject**, and **No Decision**. It is added to the **Fnd0ReviewDecisionSetsLOV** list of values.

© 2025 Siemens

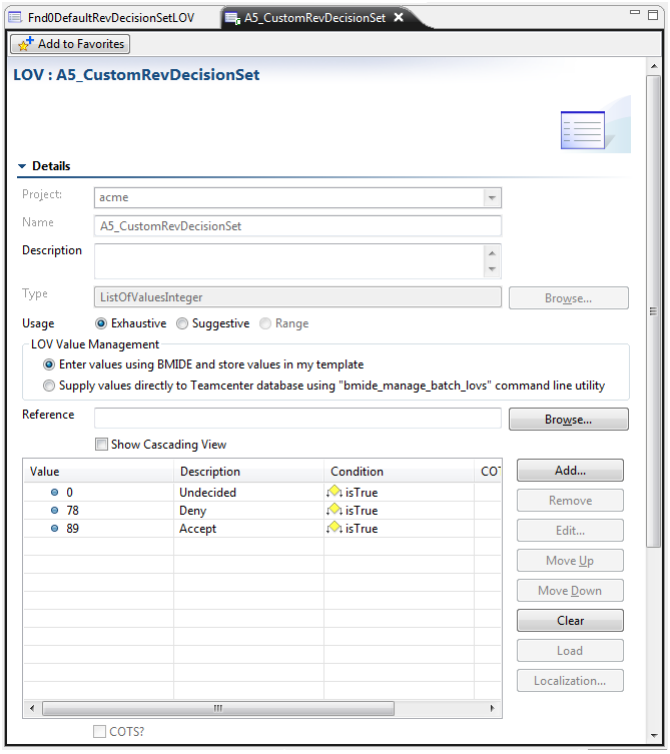
The **Fnd0DefaultAckDecisionSetLOV** list of values contains the default acknowledgement labels: **Acknowledge** and **No Decision**. It is added to the **Fnd0AcknowledgeDecisionSetsLOV** list of values.

1. To change the labels for review tasks, first view the **Fnd0DefaultRevDecisionSetLOV** list of values. Each label has a numeric value. That value corresponds to the action the workflow takes when the user selects that label. When you create your own custom labels, you must use the same numeric values so that the actions still occur when using the new labels.

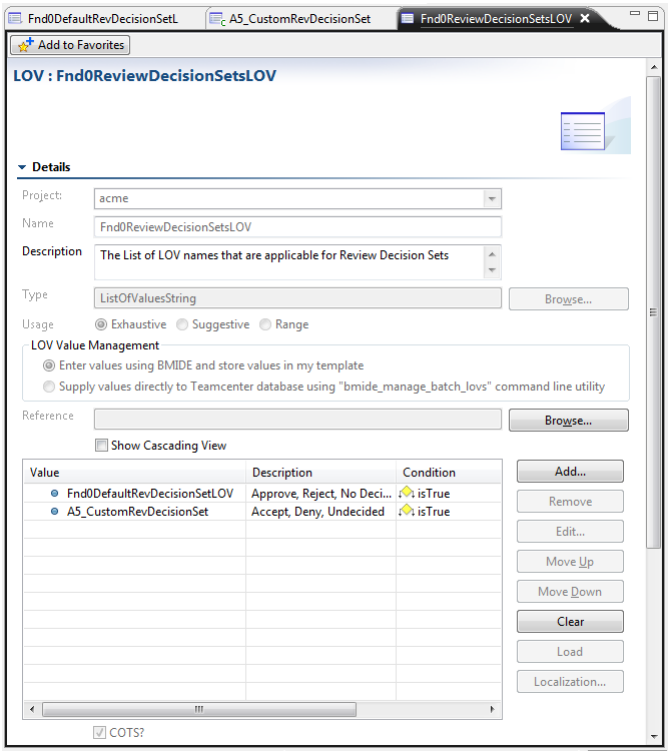


2. Create a custom review decision set LOV containing your labels. In the **Type** box, ensure that you select the **ListOfValuesInteger** type.

© 2025 Siemens



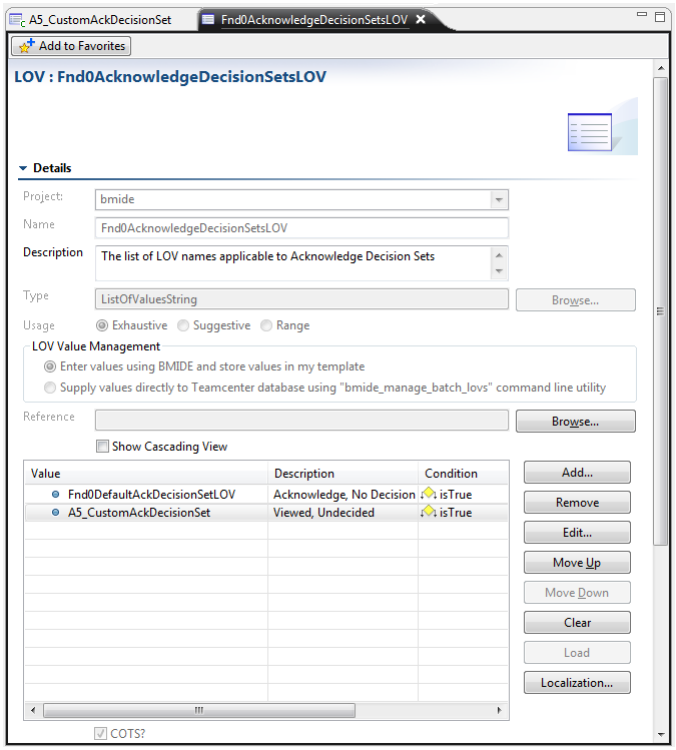
3. Add the custom review decision set LOV to the **Fnd0ReviewDecisionSetsLOV** list of values.



© 2025 Siemens

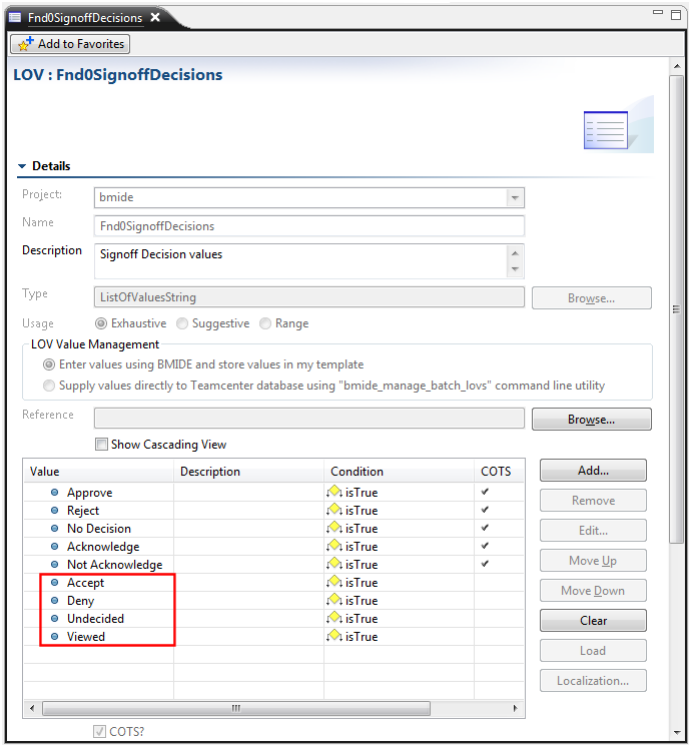
> Tip:
>
> You can add multiple custom decision set LOVs to this master decision sets LOV.

4. Perform similar steps for the acknowledgement task. For example, create a custom decision set LOV containing your acknowledgement labels and ensure that it is a **ListOfValuesInteger** type. Then add the custom acknowledgement decision set LOV to the **Fnd0AcknowledgeDecisionSetsLOV** list of values.



5. To support the custom decision labels in audit logs, update the **Fnd0SignoffDecisions** LOV with the custom decision labels.

© 2025 Siemens

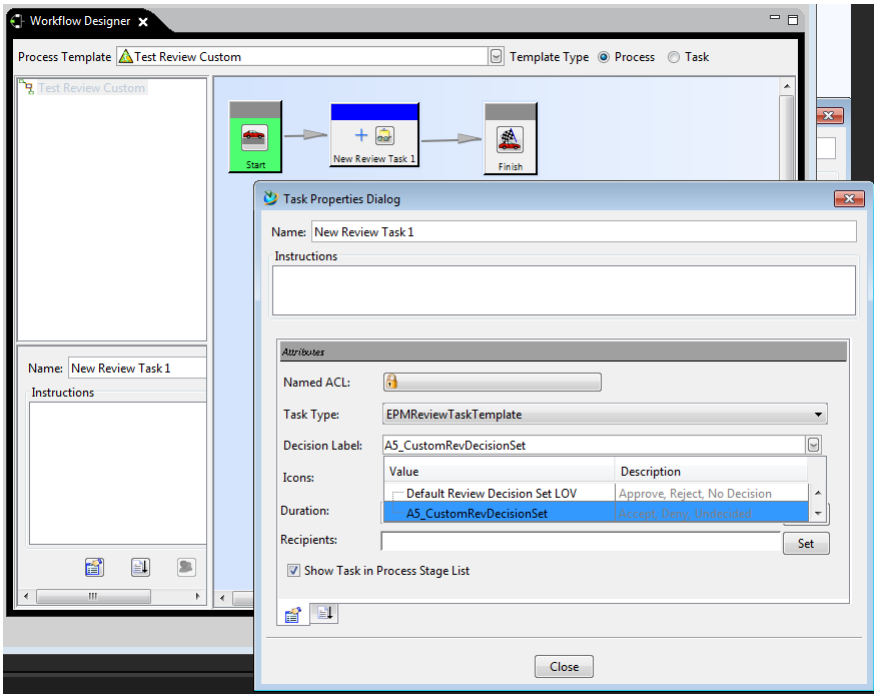


6. To save your changes to the template, on the menu bar, choose **BMIDE→Save Data Model**. Then **package the template** and use Teamcenter Environment Manager to install the packaged template to your server.

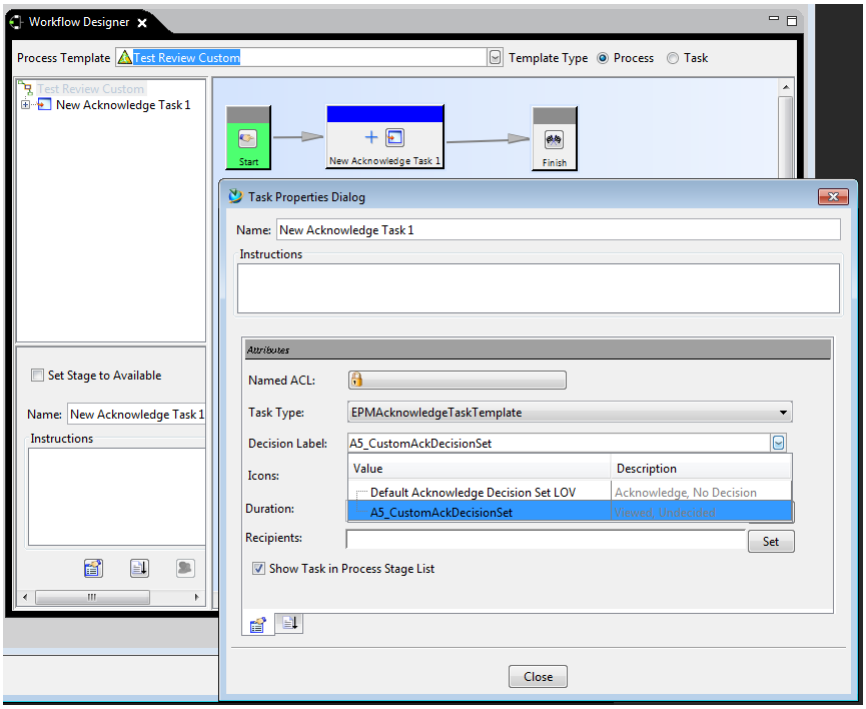
7. In Workflow Designer, the workflow administrator opens the **Task Properties** dialog box and clicks the arrow in the **Decision Label** box to select the decision set LOV to use. The dropdown list is populated based on the contents of the **Fnd0ReviewDecisionSetsLOV** and **Fnd0AcknowledgeDecisionSetsLOV** lists of values.

   Following is an example of the decision labels available on the review task.

© 2025 Siemens



Following is an example of the decision labels available on the acknowledge task.



© 2025 Siemens

15. Troubleshooting the Business Modeler IDE

BMIDE related log files

The following log files can help you identify issues related to running the Business Modeler IDE.

This log file type	Records this type of data	Can be found here
Error	Business Modeler IDE errors	*workspace***.metadata\.log** To find the *workspace* location, in the **Advanced** perspective choose **File**>**Switch Workspace**.
Deployment	Deployment data	**deploy.log**, **deploy_lang.log**, and **client_meta_cache.log** files at the following location: *workspace\project-name\output\deploy\server-connection-profile-name\date*
Console	Errors displayed in the console	**console.log** in the following location: *workspace\project-name\output\console.log*

Deployment errors

Check the deployment log

If you encounter errors deploying data model to a server, first check the deployment log. The **deploy.log** file is located in the **Project Files\output\deploy***server-connection-profile-name\date* folder.

Check your deployment setup

Problem

Deployment of a template from within the Business Modeler IDE fails.

Possible Cause

Connection from the Business Modeler IDE to Teamcenter may be configured incorrectly or cached data might be interfering with deployment.

Possible Solutions

© 2025 Siemens

Perform the following checks to make sure your environment is set up correctly for the BMIDE deployment type:

- Ensure that the Teamcenter server is running and that you can connect to it from another application running in the same tier mode as does the BMIDE. The BMIDE can be run in either two-tier (Teamcenter client communication system) or four-tier (HTTP or HTTPS) mode:

Two-tier -- Teamcenter client communication system (TCCS)	If you connect to the server using TCCS, ensure that the TCCS is set up properly. Run the *tccs-installation-location***\tccs_Teamcenter client communication system_installation\Change Teamcenter client communication system Installation** file. To ensure that there are no two-tier connection issues, if you have two-tier rich client installed, then launch the two-tier rich client and log on to the Teamcenter server.
Four-tier -- HTTP or HTTPS	To ensure that there are no four-tier connection issues, if you have a four-tier rich client installed, then launch the four-tier rich client and log on to the Teamcenter server.

- Ensure that you have a **server connection profile** set up. Server connection profiles define the Teamcenter servers to connect to. You must create a profile to deploy your extensions.

- Ensure that you do not have **TC_DATA** set as a system variable. If it is set, it may point to an incorrect **TC_DATA** location.

- If you are running in two tier mode, ensure that you have an FSC set up on your machine. This is required because Business Modeler IDE uses transient volumes to transfer files during deployment.

  1. Ensure that the **FMS_HOME** environment variable is pointing to the correct location.

  2. Verify that the transient volumes variables in the **fmsmaster.xml** file are pointing to the correct location.

  3. Check the FCC status by running the **fccstat -status** command in the Teamcenter command prompt.

     If the command shows that FCC is offline, run the **fccstat -restart** command to restart FCC, and then run **fccstat -status** again to ensure that FCC is running.

- Run the Business Modeler IDE launch batch file (*installation-location***\bmide\client\bmide.bat**) with the **-clean** option. This removes cached data that might be interfering with deployment.

Cannot connect to the server error

Problem

© 2025 Siemens

The deployment hangs.

Possible cause

The server is not running.

Solution

In the two-tier architecture, before starting the Business Modeler IDE to deploy your extensions, start the server. In the TC_ROOT directory, run the following:

```
tccs\bin\fccstat.exe -restart
```

Class is referenced error

Problem

When you deploy, you get an error stating that deployment of a class fails because the class is referenced.

Possible cause

Some changes to classes cannot be deployed to the database if the affected classes are referenced in the server session. For example, if you make changes to **WorkspaceObject** class and try to deploy, you get an error saying that the class is referenced. The reason is that when you start the server session it uses the ITK level logon that initializes all the ITK modules. This results in loading the **Home** folder of the logged in user. Because **Home** folder is an instance of **Folder** class, which is a subclass of **WorkspaceObject**, the system does not allow the changes to the **WorkspaceObject** class.

Solution

Create a template and **use Teamcenter Environment Manager (TEM) to install your changes to the database**.

Incompatible argument error

Problem

When you deploy, you get the following error:

```
An internal error occurred during: "Deploying to Teamcenter Server".
(class: org/jacorb/orb/Delegate, method: getReference signature:
(Lorg/jacorb/poa/POA;)Lorg/omg/CORBA/portable/ObjectImpl;)
Incompatible object
argument for function call
```


© 2025 Siemens

Possible cause

You **installed the Business Modeler IDE to an existing Eclipse environment**, and the **config.ini** file was not updated with an OSGI setting that resolves a boot delegation problem.

Solution

In the *ECLIPSE_HOME*/**configuration** directory, open the **config.ini** file and add the following:

```
org.osgi.framework.bootdelegation=*
```

This property is a list of packages that require delegation to the boot classpath, bypassing the controlled class loading mechanism.

Deployment fails when business object names do not contain USASCII7 characters

Problem

Deployment of business objects that have non-USASCII7 characters in the business object name causes deployment to fail.

Possible cause

Business object (type) names and class names entered in the Business Modeler IDE while performing data model extensions must be USASCII7 characters only. This prevents any template install, upgrade, or deployment issues.

Solution

For any existing business object (type) names or existing class names that do not follow the USASCII7 format, you can use the **change_type_name** utility to rename the type name to a valid USASCII7 name.

Time zone error

Problem

Live update fails when you perform a live update with a time zone setting that is different from the time zone setting previously in use during a standard template deployment.

Possible cause

The failure occurs because the values change for all date format dependent elements. This affects elements such as **TcClass** and **TcType** that have certain properties of date type.

© 2025 Siemens

Solution

Whenever you change the time zone of your server host, you must **perform a full update** of your template using the **Full Update** option in Teamcenter Environment Manager (TEM). This is necessary so that any subsequent live updates proceed without any errors.

Incorporate Latest Live Update Changes error

Problem

After using the **Incorporate Latest Live Update Changes** command, the action on the Merge wizard does not match the expected action. For example, although an LOV value is deleted, after **incorporating the latest live update changes**, the Merge wizard shows that the LOV value change is **Add no Action** rather than **Delete**.

Possible cause

After incorporating the latest live update changes into the standard template, the template is not deployed. Instead, changes are made to the live update project to those same elements that were merged.

Solution

Before choosing **Incorporate Latest Live Update Changes**:

1. Clear the **Allow Live Updates** check box on the **Live Update** preference.
2. Choose **BMIDE→Live Update→Incorporate Latest Live Update Changes**.
3. Deploy the project.
4. Select the **Allow Live Updates** check box on the **Live Update** preference.

Business Modeler IDE has slow performance, an out-of-memory error, or does not launch

Problem

The Business Modeler IDE runs slowly, displays an out-of-memory error, or does not launch.

Possible cause

The Business Modeler IDE has too many projects open, or not enough virtual machine memory is being allocated.

© 2025 Siemens

Solution

If you have many projects opened in the Business Modeler IDE, close all the projects that you are not working on. You should keep only one or two projects open at a time.

Set the virtual memory allocated to the Business Modeler IDE to a higher level in both the **BusinessModelerIDE.ini** file and the **BMIDE_SCRIPT_ARGS** environment variable. For example, use the **-Xmx1024M** value to allocate 1 GB of RAM to the Business Modeler IDE.

> Note:
>
> If you perform live updates, you must have a minimum of 2 GB of RAM on the system running the Business Modeler IDE to allow for other processes.

> Caution:
>
> If you set the **Xmx** value to a higher value than the RAM your system has, you may get the following error when you launch the Business Modeler IDE:
>
> ```
> Could not create the Java virtual machine.
> ```
>
> Set the **Xmx** value to a lower setting that your system supports, in both the **BusinessModelerIDE.ini** file and the **BMIDE_SCRIPT_ARGS** environment variable.

If these measures do not solve the problem, try the following:

- If you are running the Business Modeler IDE in an Eclipse environment, run the following command to increase virtual memory to 1 GB:

  ```
  eclipse.exe -vmargs -Xmx1024M
  ```

- Try closing some other applications. For example, many development machines have the index search engine enabled in the services panel. Try disabling the search engine.

Could not create the Java virtual machine error

Problem

The Business Modeler IDE does not run at startup, and shows the following error:

```
Could not create the Java virtual machine
```

Possible cause

More memory is being allocated to the Business Modeler IDE than is available on the computer.

© 2025 Siemens

Solution

Set the virtual memory allocated to the Business Modeler IDE to a level lower. Set the **Xmx** value to a setting that your system supports, in both the **BMIDE_SCRIPT_ARGS** environment variable and the **BusinessModelerIDE.ini** file.

Workspace is locked error

Problem

You see the following error message when you launch the Business Modeler IDE, and you do not have the Business Modeler IDE or Eclipse already running:

```
Could not launch the product
because the associated workspace
is currently in use.
```

Possible cause

The Eclipse mechanism locked the workspace to prevent from launching the Business Modeler IDE more than once.

Solution

Check to make sure you do not have the Business Modeler IDE or Eclipse already running, and remove **.lock** file that is located in the following location:

workspace`\.metadata\.lock`

To find the *workspace* location, in the **Advanced** perspective choose **File→Switch Workspace**.

Type name collision error

Problem

You see an error message in the **Console** view when you launch the Business Modeler IDE that is similar to the following:

`Model Error:` *file*`.xml Line:` *number* `Column:` *number*
`A Business Object is already defined with the name "`*name*`". Choose another name.`

Possible cause

© 2025 Siemens

A data model object in your custom template has the same name as one in a COTS template installed to the Business Modeler IDE.

Solution

You must **rename your custom data model object** so that it no longer collides with the identically named object in the COTS template.

If it is a business object, in the **Business Objects** folder, right-click the custom business object you want to rename and choose **Rename**. Then run the **change_type_name** utility to change the name of the business object in the database.

Backup and recovery following failed template deployment

Problem

You used Teamcenter Environment Manager (TEM) to deploy a template update, but the template package was bad and the deploy operation failed. To restore the environment to the state that it was in prior to the failure, you restored the database, volumes, and *TC_DATA***model** directory. To try to ensure that template files don't conflict, you also renamed your *TC_ROOT***install***template* folder (for example, **mytemplate_old**).

After you correct the template package and once again attempt to deploy it using TEM, the deployment fails again.

Cause

The environment was not fully restored. Specifically in this case, TEM cannot identify that the template folder in the *TC_ROOT***install** directory was renamed.

Solution

Fully restore the Teamcenter environment. Specifically in this case, roll back the *TC_ROOT***install***template* folder.

> Tip:
>
> Before any model-changing action, create a Teamcenter backup of all relevant data so that you can restore the environment to the state it was in prior to the model change.
>
> You should also maintain a regular **Business Modeler IDE project backup** apart from the Teamcenter database.

© 2025 Siemens

Localized property with default value changes its master locale setting after transfer to a remote site

Problem

When creating a Teamcenter object instance using a business object type that has a localized property with a default value, the value of the property is automatically populated with the default value. However, the localization VLA (variable length array) associated with this property is not populated with the master locale and the value. Therefore, when this object is transferred to the remote site that has a different master locale, the master locale of this localized property is changed to the remote site master locale.

Solution

There are two workarounds to fix this problem.

When creating a custom localized property with default value in the Business Modeler IDE:

1. Do not define the default value in the property creation dialog box. Simply create the property first.
2. After the new property is created, modify its **InitialValue** property constant and add the initial value in the **Modify Property Constant** dialog box.
3. Package and deploy the change.

If you already defined the localized property with the default value from the property creation dialog box in the Business Modeler IDE:

1. Before transferring the object to the remote site, open the **Edit Properties** dialog box in the rich client.
2. In the **Edit Properties** dialog box, click the **Localization** button next to the localized attribute with the default value.
3. In the **Localization** button dialog box, click the **OK** button to dismiss the dialog box.
4. Close the **Edit Properties** dialog box and transfer the object to the remote site.

BASE-10001: ENCODING_VALIDATION_ERROR

Problem

When loading, deploying, or upgrading a template, you receive the following error message:

© 2025 Siemens

```
BASE-10001: "ENCODING_VALIDATION_ERROR"
```

Possible cause

The template contains one or more characters that do not belong to the character set expected by the Business Modeler IDE.

Solution

The solution differs depending on whether the error occurs during template **loading**, or during template **deployment or upgrade**.

- **Template loading**

Error occurs when a template is	Example
Loaded	When a new project is created in the Business Modeler IDE or an existing project is opened or imported to the Business Modeler IDE.
Reloaded	When a user performs the **Reload Data Model** action on a project in the Business Modeler IDE.
Migrated	When the Business Modeler IDE detects that the template belongs to an older version and runs the Migration Wizard.

To resolve this error:

1. Look at the file in which the error occurs and determine its locale. If the file is a language file (for example, **custom_template_en_US.xml**), the locale of this file can be determined from the file name. For example, if the file name is **custom_template_en_US.xml**, the locale is **en_US** (that is, English).

   Language files are typically located in the *PROJECT_HOME***/extensions/lang** folder. If the file is a template file (for example, **default.xml**), then the locale of this file can be determined by the value of the **SiteMasterLanguage** global constant. This value can be checked from the **Global Constants Editor** in the Business Modeler IDE.

2. Check the character set mapping:

   a. Browse to the text server location, which is defined by the **TC_MSG_ROOT** environment variable. Typically, this is located at *TC_ROOT***/lang/textserver** directory.

   b. Browse to the **no_translation** folder within the **textserver** folder.

   c. Open the **textsrv_text.xml** file in a text editor. Within this file search for the following section:

© 2025 Siemens

```
<-- SECTION DEFINING THE SMALLEST OR CUSTOM ENCODING FOR EACH LOCALE:
THIS IS USED FOR BMIDE LOCALIZATION VALIDATION -->
```

This section defines the character set mapping for each locale. For example:

```
<key id="locale_validation_encoding_en_US">us-ascii</key>
```

This implies that for the English locale (**en_US**), the characters should belong to the US-ASCII character set. It is advisable that you conform to this restriction. However, if you feel that you need to use a higher encoding for this locale, edit the mapping as follows:

```
<key id="locale_validation_encoding_en_US">iso-8859-1</key>
```

This implies that for the English locale (**en_US**), the characters should belong to the ISO-8859-1 character set. Instead of **iso-8859-1**, you can replace your own character set.

> Note:
>
> Your server host and database must be configured for the specified character set.

d. Restart the Business Modeler IDE.

- **Template deployment or upgrade**

Error occurs when a template is	Example
Deployed	When a template is deployed from the Business Modeler IDE or installed using Teamcenter Environment Manager (TEM).
Upgraded	When a template is upgraded from an older version to a newer version.

The database cannot accept characters that do not belong to the specified character set. The error message specifies the character set against which the validation was performed.

To resolve this error, do either of the following:

- Fix the template by entering only characters that are allowed by the specified character set.
- Fix the database encoding to accept characters belonging to your required character set.

© 2025 Siemens

Correct a transposed interdependent LOV attachment

Problem

In BMIDE versions before Teamcenter 9.1, it was possible that the software erroneously transposed attachment of LOV levels to properties. For example, suppose a template contains two hierarchical LOV definitions, one for Safety and another for Hygiene. Both LOVs are attached interdependently to a business object, but the second level of the LOVs are mistakenly transposed in the following way:

Business object property	LOV attachment (transposed level 2)
Safety Property 1	Safety LOV level 1
Safety Property 2	Hygiene LOV level 2
Hygiene Property 1	Hygiene LOV level 1
Hygiene Property 2	Safety LOV level 2

After deployment, the interdependent LOV does not work as intended.

Solution

To fix the problem, do the following:

1. Open the template and delete the LOV attachments to **PropertyB** and **PropertyD**.
2. Save and deploy the template.
3. Add the attachments at the required places.

Business object property	LOV attachment (corrected)
Safety Property 1	Safety LOV level 1
Safety Property 2	Safety LOV level 2
Hygiene Property 1	Hygiene LOV level 1
Hygiene Property 2	Hygiene LOV level 2

4. Save and deploy the template.

Note:

The *FND-10001: MULTIPLE_INTERDEPENDENT_LOV_ATTACHMENT_ERROR* error message results from a different problem.

© 2025 Siemens

FND-10001: MULTIPLE_INTERDEPENDENT_LOV_ATTACHMENT_ERROR

Problem

You receive the following error message:

```
Multiple interdependent LOV Attachments exist in LOV LOV name for
type business object at level level. No parent property information,
however, is present.
```

Possible cause

The template being loaded was created in a version of BMIDE before Teamcenter 9.1 that allowed incomplete XML coding of an interdependent LOV, such that the property to which a parent level LOV is attached is not identified. The template being loaded contains a hierarchical LOV that has been attached multiple times to a single business object as an interdependent LOV. When the same cascading LOV is attached interdependently multiple times to an object, the property from which to get a parent LOV value that controls the child level list is indeterminate.

If a template containing such an incomplete XML coding of an interdependent LOV is loaded in a later version of BMIDE that identifies the error, then BMIDE displays an error message, but allows the template to load. Human intervention is required to correct the error.

The Business Modeler IDE can detect this error in either of the following cases:

- When a template is loaded, for example, when an existing project is opened or imported in the Business Modeler IDE.
- When a template is upgraded, for example, when the Business Modeler IDE detects that the template belongs to an older version and runs the Template Project Upgrade wizard.

Consider the **MyItem** business object with the following properties: **UserState**, **UserCity**, **SpouseState**, and **SpouseCity**. The same cascading State>City LOV has been interdependently attached twice as follows:

Business object property	Interdependent LOV level attachment
UserState	State
UserCity	City
SpouseState	State
SpouseCity	City

© 2025 Siemens

The XML code for the City attachments does not identify the property from which to get the parent value. In this case, the warning is shown, and one of the interdependent attachments (**UserState > UserCity** or **SpouseState > SpouseCity**) is not loaded in the Business Modeler IDE.

Solution

Delete and then recreate the structure, and then save the data model.

FND10002: INVALID_CONDITION_FOR_DEEP_COPY_RULE_ERROR

Problem

You receive the following error message:

```
FND10002: INVALID_CONDITION_FOR_DEEP_COPY_RULE_ERROR
```

Possible cause

The signature of the condition specified in the **deep copy rule** does not conform to the standard for deep copy rules. The supported signatures are:

- The default condition is **isTrue** or **isFalse**.
- The condition has three input parameters in this order: **DeepCopyRule**, **POM_object** (or its subtype), and target business object (or its supertype).
- The condition has four input parameters in this order: **DeepCopyRule**, **POM_object** (or its subtype), target business object (or its supertype), and **UserSession**.

Solution

To resolve this error, manually change the condition signature in the template extension file. You can also remove the deep copy rule from the XML file and re-create it after loading the project with the valid condition.

FND10003: THE_ELEMENT_HAS_BEEN_REMOVED_FROM_ITS_DEPENDENT_TEMPLATE

Problem

You receive the following error message:

```
FND10003: THE_ELEMENT_HAS_BEEN_REMOVED_FROM_ITS_DEPENDENT_TEMPLATE
```

Possible cause

© 2025 Siemens

A custom template changes a COTS element from a dependent template, and then the COTS template removes or modifies that element. For example, this problem occurs if a custom template changes the localization value of a property defined in its dependent template, and in the next version of the dependent template, the name of that property is changed or the property is removed.

Solution

Manually remove the modified element from the XML file of the custom template and reload the data model.

FND10004: CANNOT_SET_LOCALIZABLE_CONSTANT_ATTACHMENT_ERROR

Problem

You receive the following error message:

```
FND10004: CANNOT_SET_LOCALIZABLE_CONSTANT_ATTACHMENT_ERROR
```

Possible cause

The **Localizable** property constant is set to true on a string property that has an initial value set. You can no longer set the **Localizable** property constant on string properties with an initial value.

Solution

It is not advisable to set the **Localizable** property constant to true when you have an initial value set. This could cause problems in a Multi-Site environment. However, this should not pose a problem if you do not use a Multi-Site environment.

Applying condition rules for custom change objects after a patch or upgrade

Problem

After upgrading or patching Teamcenter, existing custom condition rules that govern the creation of custom subclasses of change items are no longer applied.

Possible cause

Change item classes delivered in Teamcenter with Change Manager include **Problem Report** (PR), **Enterprise Change Request** (ECR), **Enterprise Deviation Request** (EDR), and **Engineering Change Notice** (ECN). Using the Business Modeler IDE, you can create custom subclasses of these classes. Each custom subclass may have a condition rule associated with it that determines whether an object of

© 2025 Siemens

that class can be created. If such custom subclass condition rules exist prior to upgrading or patching Teamcenter, changes to the custom Business Modeler IDE template are necessary after the installation. Otherwise, the default condition rule behavior for the parent classes applies. If no custom condition rules for custom change classes exist prior to installation, no action is required.

Changes in Teamcenter to the application of custom change class condition rules allow for greater flexibility and ease of use when specifying condition rules. Prior to Teamcenter 10.1.7, condition rules for custom change classes were named according to the custom class with which they were associated or they would not work. For example, the condition rule to control the creation of **A5_ChangeNotice** (a subclass of ECN) had to be named **A5_isA5_ChangeNoticeCreatable** in order to function. Also, prior to Teamcenter 10.1.7, the condition rule signature was required to accept only **User Session** as a parameter. Now, the condition rule takes **Workspace Object** and **User Session** as parameters, with **Workspace Object** representing a target object selected on the user interface. This additional parameter permits the condition rule to constrain the list of change types displayed for a create change in context operation on the basis of the type of object currently selected.

Solution

Create an intermediate condition rule accepting two parameters that can call the existing custom condition rule with only a **User Session** parameter so that the existing custom condition rule still applies. Alternatively, take advantage of the additional **Workspace Object** parameter by replacing the existing custom condition rule with a new rule that accepts **Workspace Object** and **User Session** parameters.

Four business object constants available in the Change Management Business Modeler IDE template contain the name of the default condition rule for the corresponding change item class:

- The **Cm0ChangeItemCreCondition** business object constant attached to **ProblemReport** points to the condition rule named **Cm0isProblemReportCreatable**, which calls the **isProblemReportCreatable** condition rule.
- The **Cm0ChangeItemCreCondition** business object constant attached to **ChangeRequest** points to the condition rule named **Cm0isChangeRequestCreatable**, which calls the **isChangeRequestCreatable** condition rule.
- The **Cm0ChangeItemCreCondition** business object constant attached to **ChangeNotice** points to the condition rule named **Cm0isChangeNoticeCreatable**, which calls the **isChangeNoticeCreatable** condition rule.
- The **Cm0ChangeItemCreCondition** business object constant attached to **Cm0DevRqst** points to the condition rule named **Cm0isCm0DevRqstCreatable**, which calls the **isCm0DevRqstCreatable** condition rule.

For each existing custom condition rule of the form **{prefix}is{custom class}Creatable**, use the Business Modeler IDE to adjust the custom condition rules as described below, and thereby allow existing custom condition rules to function properly to create custom change subclasses.

© 2025 Siemens

1. Create a new condition rule that takes **Workspace Object** and **User Session** as parameters and set the condition expression to call the original condition rule. The new condition rule may use any name which conforms to the general Business Modeler IDE rules for condition rule names.

2. Override the value of the corresponding **Cm0ChangeItemCreCondition** business object constant by setting it to the name of the new condition rule.

Example:

If an **A5_ChangeNotice** custom subclass exists with **ChangeNotice** as its parent, and there is an existing custom subclass condition rule named **A5_isA5_ChangeNoticeCreatable**, then do the following:

1. Create a new condition rule **A5_Tc1122ChangeNoticeIsCreatable** with **Workspace Object(o)** and **User Session(u)** as parameters and set the condition expression to **Condition::A5_isA5_ChangeNoticeCreatable(u)**.

2. Set the value of the **Cm0ChangeItemCreCondition** business object constant to **A5_Tc1122ChangeNoticeIsCreatable**.

Alternatively, exploit the **Workspace Object** condition rule parameter to control the list of change types presented for a create change operation in the context of a selected object. In the example, instead of simply calling the existing custom condition rule in step 1, the old condition rule can be discarded and the new rule written to exploit the **Workspace Object** parameter.

Example:

The following condition rule expression is designed to only allow changes of the associated custom subclass to be created when the selected object type starts with **Design** and the user's role starts with **Design** and the selected object type starts with **Requirement** and the user's role is **Simulation Administrator**.

```
((o != NULL AND o.object_type = "Design*" AND u.role_name = "Design*" )
OR (o != NULL AND o.object_type = "Requirement*"
AND u.role_name = "Simulation Administrator*" )) OR o = NULL
```


© 2025 Siemens

© 2025 Siemens

A. Business Modeler IDE reference

Business Modeler IDE utilities

Business Modeler IDE utilities

Business Modeler IDE utilities allow you to perform a number of database actions related to your Business Modeler IDE work. A number of command line utilities are available to work with Business Modeler IDE data model.

Utilities are located in the **bin** directory where the Teamcenter server is installed, and each utility can be initiated from the Teamcenter Command Prompt by entering its name and optional parameters. Internal utility documentation can be displayed using a `-h` argument.

Business Modeler IDE utilities are also documented in the utilities reference.

Obsolete utilities and APIs

In Engineering Process Management, some utilities and ITK APIs allowed you to change the data model directly. However, the data model is now maintained separately through use of XML templates and is updated only through the Business Modeler IDE deployment or custom template installation. Therefore, those utilities and APIs that allowed you to change the data model directly are now obsolete.

Some of the obsolete utilities are:

apply_naming_rule
business_rules_dtdxml2plmxml
create_change_types
deepcopyrules_migration
grm
import_export_business_rules
install_bmf_rules
install_lovs
install_typedisplay_rules
install_types
make_datasettype
migrate_alias
migrate_complex_property_rules
sb (Schema Browser)

Warning:

The Business Modeler IDE is the mechanism for maintaining and deploying a data model and replaces these obsolete utilities. Obsolete utilities and APIs *must not* be used to add custom

© 2025 Siemens

classes, attributes, types, LOVs, rules, and so on, to the database. The only exception to this is if a Siemens Digital Industries Software representative asks you to use the utility.

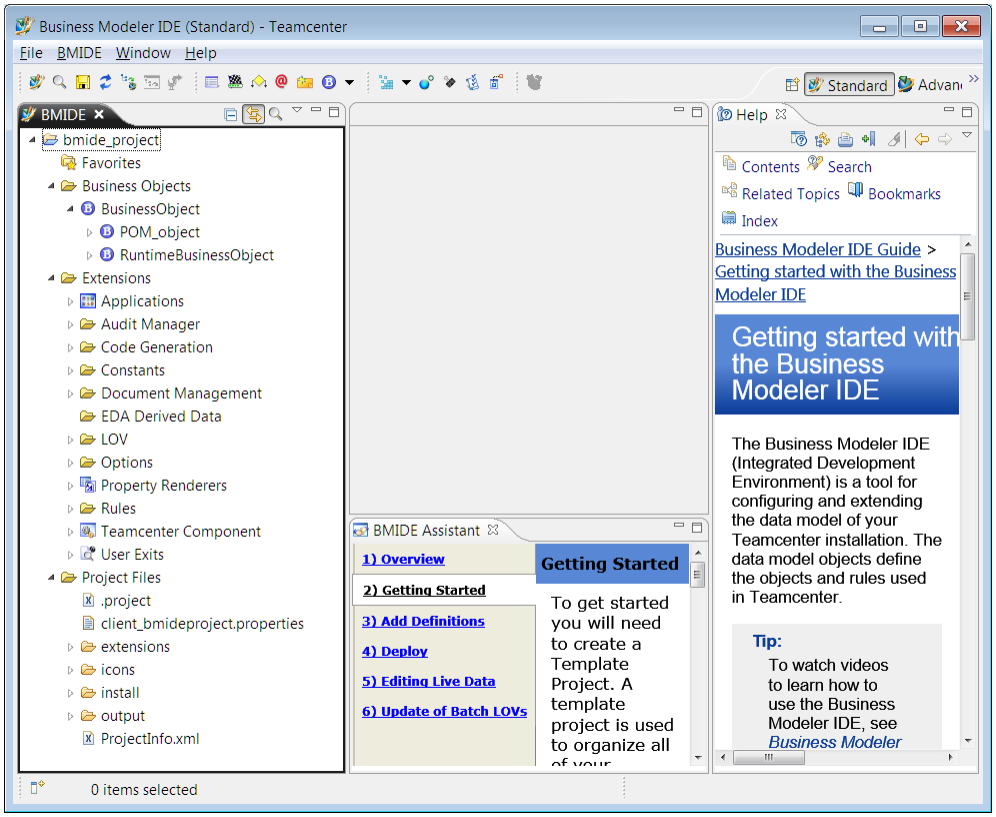
Business Modeler IDE perspectives, views, and editors

Standard perspective

Overview of the Standard perspective

The **Standard** perspective provides a simplified user interface. It contains the **BMIDE** view, which provides a centralized location for favorites, data model elements, and project files.

To access the **Standard** perspective, choose **Window→Open Perspective→Other→Standard**.



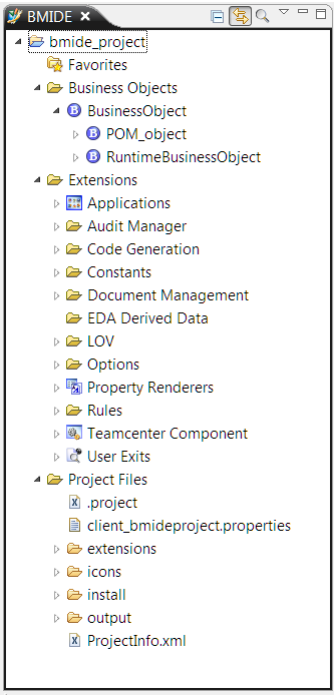
Views are the tabbed panes that appear in the user interface. Most of these views are found in the Business Modeler IDE user interface perspectives; however, some can only be accessed from the **Show View** menu. To access all the Business Modeler IDE views, choose **Window→Show View→Other→Business Modeler IDE**.

© 2025 Siemens

BMIDE view

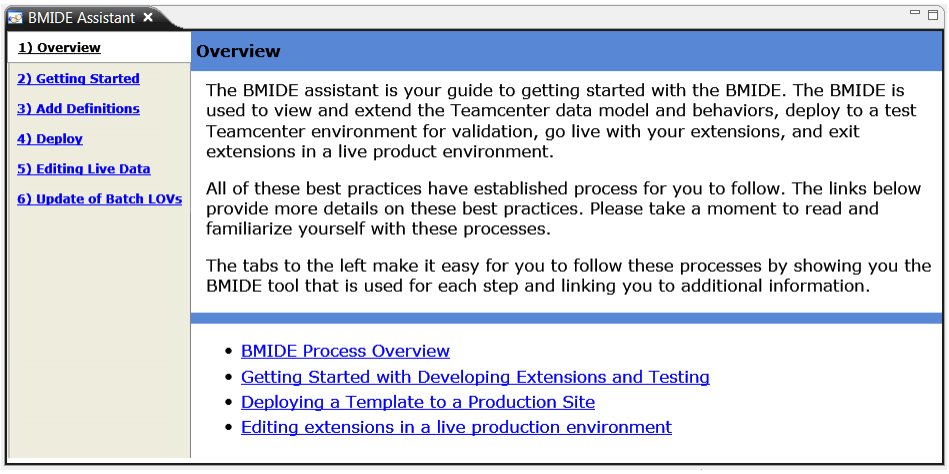
The **BMIDE** view provides a single view for favorites, data model elements, and project files. It is the main view in the **Standard** perspective.

To open this view, open the **Standard** perspective, or on the menu bar, choose **Window→Show View→BMIDE**.



BMIDE Assistant view

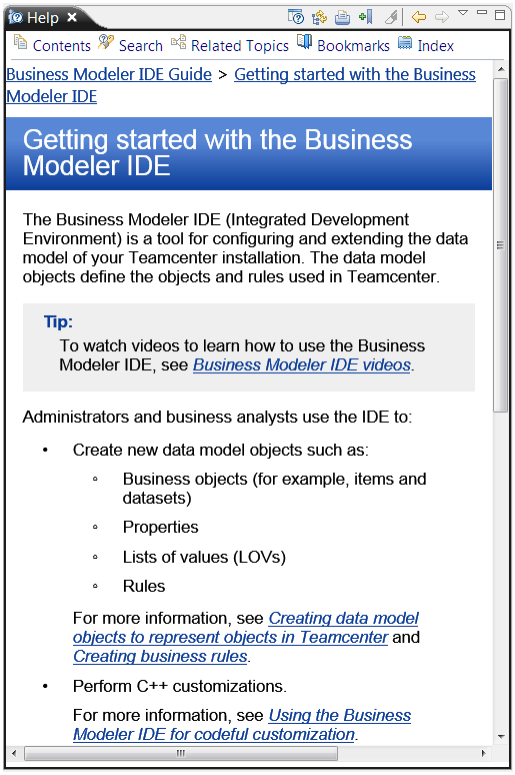
The **BMIDE Assistant** view helps new users get started using the Business Modeler IDE.



© 2025 Siemens

Help view in the Business Modeler IDE

The **Help** view presents online help for the Business Modeler IDE. You can launch this view by pressing the F1 key or clicking the question mark button ? in the lower left corner of any dialog box.



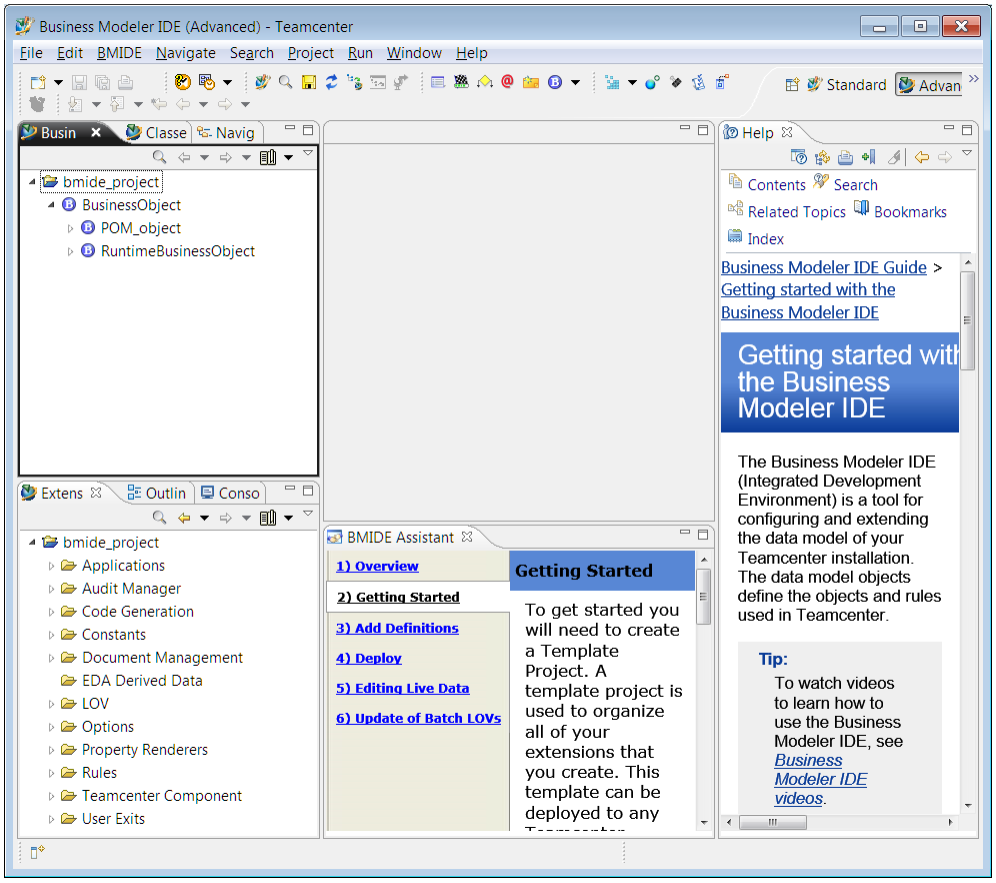
Advanced perspective

Overview of the Advanced perspective

The **Advanced** perspective allows you to work with many aspects of the Teamcenter data model and functional behavior.

To access the **Advanced** perspective, choose **Window→Open Perspective→Other→Advanced**.

© 2025 Siemens



Views are the tabbed panes that appear in the user interface. Most of these views are found in the Business Modeler IDE user interface perspectives; however, some can only be accessed from the **Show View** menu. To access all the Business Modeler IDE views, choose **Window→Show View→Other→Business Modeler IDE**.

This is a good time to bring up the Business Modeler IDE and show them the user interface.

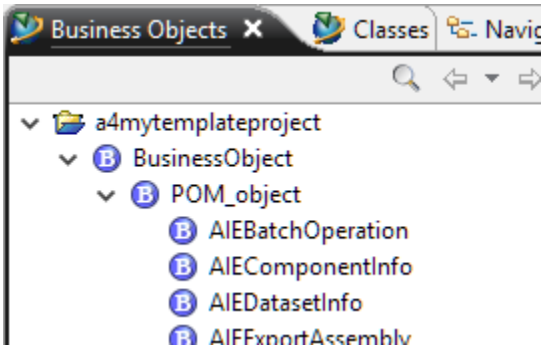
Business Objects view

Business objects are the fundamental objects used to model business data. Create business objects to represent product parts, documents, change processes, and so on.

The **Business Objects** view displays the hierarchy of business objects on your server. Business objects are shown in a tree where the top-level **POM_object** business object is at the top and its children are directly below. The **RuntimeBusinessObject** folder contains business objects that have no parents and are used in run-time processes in Teamcenter.

You can use preferences to set the **WorkspaceObject** business object as the top-level object or to hide the **RuntimeBusinessObject** folder.

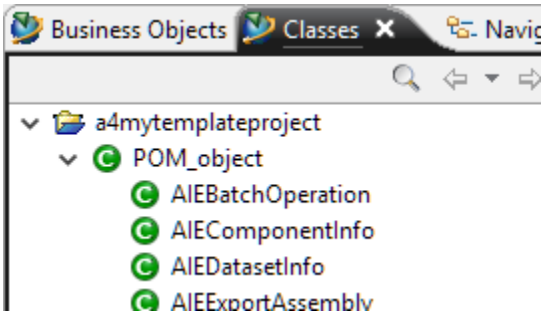
© 2025 Siemens



Classes view

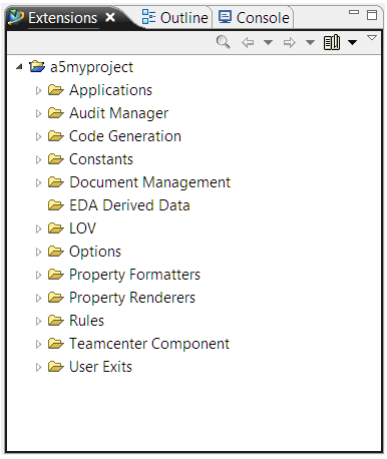
Classes are the persistent representations of the data model schema, and provide attributes to business objects. Classes are also known as the *persistent object model (POM)*.

The **Classes** view displays the classes on your server. Classes are shown in a tree where the top-level **POM_object** class is at the top and its children are directly below.



Extensions view

The **Extensions** view displays extensions to the data model.



© 2025 Siemens

The **Extensions** view contains the following extensions:

Applications	Applications are used on classes to indicate what rich client application can add or edit attributes on that class.
Code Generation	This folder contains objects used for software development, such as data types, libraries, and releases.
Constants	Constants are reusable values. They provide consistent definitions that can be used throughout the system.
Document Management	Document Management objects define how documents are handled in Teamcenter.
EDA Derived Data	EDA derived data objects define how data is derived from an electronic CAD (ECAD) design. These objects are used by Teamcenter EDA, an application that integrates Teamcenter with ECAD design applications, such as Cadence and Siemens EDA.
LOV	Lists of values (LOVs) are pick lists of values. They are commonly accessed by end users from a menu at the end of a text box.
Options	Options are miscellaneous objects such as change requests, notes, and so on.
Property Formatters	Property formatters are display definitions that dictate how properties are shown in the user interface.
Property Renderers	Property renderers are configurations to render the overlays on business object icons.
Rules	Rules define the business rules that govern how objects are handled, including how they are named, what actions can be undertaken on them, and so on. Creating rules is also known as business modeling.
Teamcenter Component	**Teamcenter Component** objects set up rules that use conditions to evaluate how business objects are used in Teamcenter.
User exits	User exits are used for adding base action extension rules. User exits are places in the server where you can add additional behavior by attaching an extension to the user exit. To work with user exits, right-click a user exit and choose **Open Extension Rule**.

Business Modeler IDE editors

Audit Manager editors

Introduction to Audit Manager editors

The **Extensions→Audit Manager** folder holds objects used to **configure Audit Manager**. When you right-click an Audit Manager object and choose **Open**, an editor appears that allows you to work on characteristics of the object.

© 2025 Siemens

Audit Definition editor

The **Audit Definition** editor displays characteristics of the selected audit definition. An audit definition specifies the information that must be captured when an event occurs to a particular kind of object. Before creating an audit definition, you must ensure that an event mapping has been created for the business object type and the event specified in the audit definition.

1. To access the **Audit Definition** editor, open the **Extensions→Audit Manager→Audit Definitions** folders, right-click an audit definition, and choose **Open**.

2. In the **Audit Definition** editor, you can change some aspects of custom audit definitions. You can add or remove audit extensions, and select the **Is Active?**, **Track Old Values?**, or **Audit On Property Change Only?** check boxes.

3. Click the **Primary Audit Properties** tab or the **Secondary Audit Properties** tab to add or remove properties to the audit log definition.

Event Type Mapping editor

The **Event Type Mapping** editor displays characteristics of the selected event mapping. An event type mapping declares that you want to receive an audit log for a certain event on a certain kind of object. An event mapping must be created for a business object type and event before you used that business object and event type in an audit definition.

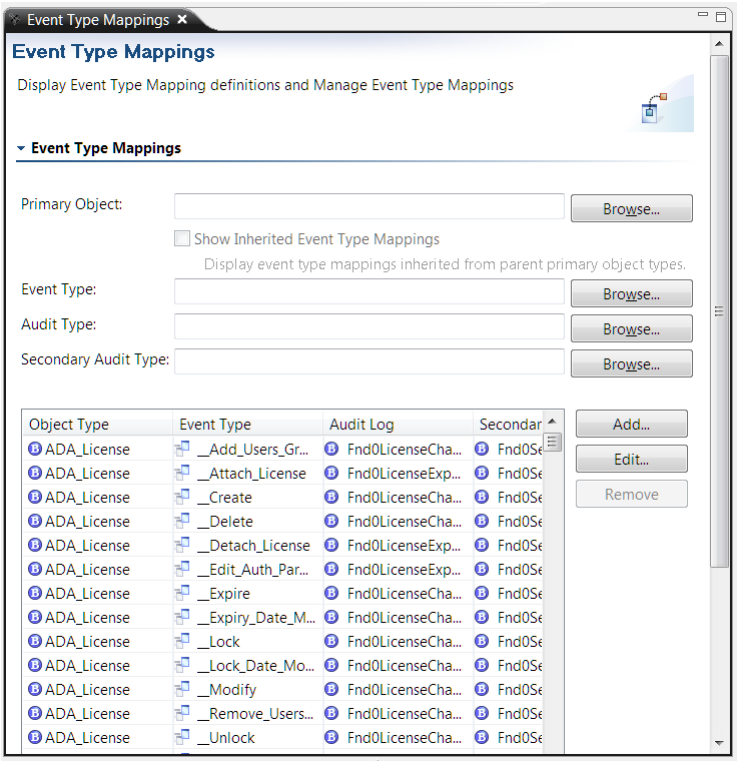
> Note:
>
> Event mapping is inherited by child business object types. For example, instances of the **Part** business object type inherit the mapping from the **Item** business object type.

1. To access the **Event Type Mapping** editor, open the **Extensions→Audit Manager→Event Type Mappings** folders, right-click an event type mapping, and choose **Open**.

2. In the **Event Type Mapping** editor, you can change some aspects of custom event type mappings. You can change the values of the **Audit Type** and the **Secondary Audit Type** boxes, and select the **Subscribable?** check box. You can also select the **Auditbale?** check box if the event type mapping is not referenced any audit definitions.

In addition to this editor, there is another editor you can use to work with even type mappings. Right-click the **Event Type Mappings** folder and choose **Open Event Type Mappings**. This editor allows you to browse, add, or edit Audit Manager objects.

© 2025 Siemens



Event Type editor

The **Event Type** editor displays characteristics of the selected event. An event type is an action that occurs to an object in Teamcenter, for example, when an item is checked out. Teamcenter records audit logs when certain events occur on certain type of objects. The audit definition object specifies the event type for which audit logs are to be written.

1. To access the **Event Type** editor, open the **Extensions→Audit Manager→Event Types** folders, right-click an event type, and choose **Open**.

2. In the **Event Type** editor, you can change some aspects of custom event types. You can change the values of the **Display Name** and **Description** boxes, and you can change the values in the **Localization** table.

Business Object editor

Introduction to the Business Object editor

When you open a business object, the **Business Object** editor displays tabs that enable you to work on different characteristics of the business object.

The Business Modeler IDE is built on an Eclipse framework and therefore the **Business Object** editor can only have 30 to 35 tabs open. If a **Business Object** editor tab does not need to be displayed, close the tab before exceeding the memory limit and causing the Business Modeler IDE to crash. To optimize

© 2025 Siemens

the **Business Object** editor, make sure that you have **allocated appropriate memory** for the Business Modeler IDE.

Dataset editors

Dataset tabs

When you open a child of the **Dataset** business object, the following dataset-specific tabs appear:

- **Dataset Properties**

  Defines the applications (tools) that can be used to edit and view the dataset and the constants on the business object.

- **References**

  Defines the type of application file that the dataset represents.

- **ToolActions**

  Defines the applications (tools) that can be used to perform actions on the dataset such as open or print.

These tabs enable you to work on different characteristics of the dataset business object, and are in addition to the standard business object tabs.

Dataset Properties tab

The **Dataset Properties** tab displays the applications (tools) that can be used to edit and view a dataset, as well as the constants on the business object. These tools are initially defined when the **dataset business object is created**.

1. To access the **Dataset Properties** tab, open a child of the **Dataset** business object and click the **Dataset Properties** tab.

2. In the **Dataset Properties** tab, click the **Add** button to the right of the **Tools for Edit** table to add a tool that can be used to edit the dataset.

3. Click the **Add** button to the right of the **Tools for View** table to add a tool that can be used to view the dataset.

References tab

The **References** tab displays the type of application file that the dataset represents. These references are initially defined when the **dataset business object is created**.

© 2025 Siemens

1. To access the **References** tab, open a child of the **Dataset** business object and click the **References** tab.

2. In the **References** tab, click the **Add** button to the right of the **References** table to add more file types that can be associated with the dataset.

3. Use the **Move Up** and **Move Down** buttons to change the priority that the file type associations have.

ToolActions tab

The **ToolActions** tab displays the applications (tools) that can be used to perform actions on the dataset, such as open or print. These tool actions are initially defined when the **dataset business object is created**.

1. To access the **ToolActions** tab, open a child of the **Dataset** business object and click the **ToolActions** tab.

2. In the **ToolActions** tab, click the **Add** button to the right of the **Tool Action** table to select the tools you can use to perform actions on the dataset.

Main tab in the Business Object editor

The **Main** tab in the **Business Object** editor displays basic information about the selected business object.

1. To access the **Main** tab, open a business object and click the **Main** tab.

2. You can type in the **Display Name** box to change the name of the object displayed in the Teamcenter user interface. This also changes the value in the **Localization** table at the bottom of the page.

3. You can click links to open referenced objects:

   - Click the **Parent** link to open the parent business object.

   - For **Item** business objects, click the **Item Revision** link to open the item revision associated with the item and click the **Form** link to open the form business object that holds the business object properties.

   - Click the **Storage Class** link to open the class that stores attributes for the business object.

4. Click the **Edit** button to the right of the **Business Object Constants** table to change the value of **business object constants**. Business object constants provide default values to business objects.

5. Click buttons to the right of the **Localization** table to change the name of the object displayed in different languages in the Teamcenter user interface.

© 2025 Siemens

Properties tab

The **Properties** tab shows available **properties** on business objects.

1. To access the **Properties** tab, open a business object and click the **Properties** tab.

2. In the **Properties** tab, click the **Add** button to the right of the properties table to add a property to the business object.

3. To change a property constant value, select a constant on the **Property Constants** tab and click the **Edit** button.

4. To **add a naming rule to a property**, select a valid property on the properties table (such as **item_id** or **object_name**) and click the **Add** button on the **Naming Rule Attaches** tab.

5. To **attach a list of values (LOV) to a property**, select a property on the properties tab and click the **Add** button on the **LOV Attaches** tab.

6. To **change the name of the property** displayed in the Teamcenter user interface, click buttons on the **Localization** tab.

7. To attach a property rendering configuration to a property, click the **Attach** button on the **Property Renderer Attaches** tab. A property rendering configuration can control the **icon** to display on a business object based on the property's value.

8. To add an operation to a property, click the **Add** button on the **Property Operations** tab.

Property Bulk Loaders tab

Use the **Property Bulk Loaders** tab to add a bulk loader to a business object.

1. In the BMIDE, open a business object and click the **Property Bulk Loaders** tab.

2. Click the **Add** button.

   Assign a name and library for the bulk loader, then select the properties to be loaded by the new bulk loader function.

3. Click **Finish**.

To see the details of a bulk loader, select the bulk loader and click the **Property Bulk Loader Definition** tab.

For more information about bulk loading properties, see Create a custom bulk loader.

© 2025 Siemens

Operation Descriptor tab

The **Operation Descriptor** tab displays the metadata of the properties on the selected business object.

You can use the **CreateInput** operation on this tab to choose the properties that are seen in dialog boxes when the user creates items. For example, when a My Teamcenter user chooses **File→New→Item** to create a new **Item** object, properties that are checked as visible and required for the **CreateInput** operation for the **Item** business object are visible and required in the creation dialog boxes.

You can also use the **SaveAsInput** operation on this tab to choose the properties that are seen in dialog boxes when the user performs a **Save As** operation on an item. For example, when a My Teamcenter user selects an item instance and chooses **File→Save As**, properties that are selected as visible and required for the **SaveAsInput** operation for the **Item** business object are visible and required in the dialog boxes.

1. To access the **Operation Descriptor** tab, open a business object and click the **Operation Descriptor** tab.

2. In the **Operation** box, select **CreateInput** to add properties to the creation dialog boxes, or select **SaveAsInput** to select properties on the **Save As** dialog boxes.

3. Select a property on the table and click the **Edit** button to make that property visible or required on creation dialog boxes.

   The **Operation Input Property Descriptor** section on the tab provides details of the selected property.

4. Click the **Add** button to the right of the table to add a new property to the list of visible and required properties on this business object.

Display Rules tab

The **Display Rules** tab displays the groups that cannot view a business object type in menus in the Teamcenter user interface. Because display rules are primarily used to hide business objects from creation (**File→New**) menus, display rules restrict those who can create the business object type.

Business object display rules only apply to the following business objects and their children: **Alias**, **Dataset**, **Folder**, **Form**, **Identifier**, and **Item**.

You can also use the Command Suppression application to suppress the display of menus and commands for certain groups.

1. To access the **Display Rules** tab, open the relevant business object or one of its children and click the **Display Rules** tab.

2. To **add a new display rule**, click the **Add** button to the right of the table.

© 2025 Siemens

Deep Copy Rules tab

Use the **Deep Copy Rules** tab to add, modify, or remove deep copy rules. Deep copy rules define whether objects belonging to a business object instance can be copied when a user performs a save as or revise operation on that instance. Deep copy rules can be created for any business object.

> Note:
>
> Although deep copy rules defined at a parent business object are inherited by the children, they are not editable at the child level. The **Edit** and **Remove** buttons are disabled on the **Deep Copy Rules** tab for the child business objects.

1. To access the **Deep Copy Rules** tab, open a business object and click the **Deep Copy Rules** tab.

2. To **add a new deep copy rule**, click the **Add** button to the right of the table.

3. To change rules, use the **Edit** button. You can modify an existing COTS non-secure rule or a custom-defined rule. Inherited rules cannot be modified.

   > Note:
   >
   > Although deep copy rules defined at a parent business object are inherited by the children, they are not editable at the child level. The **Edit** button is disabled on the **Deep Copy Rules** tab for the child business objects.

4. To delete rules, use the **Remove** button. You can remove custom-defined rules. You cannot remove COTS rules or inherited rules.

Alternate ID Rules tab

The **Alternate ID Rules** tab displays details about the selected alternate ID rule. Alternate identifiers store information (such as part numbers and attributes) about the same part from different perspectives.

1. To access the **Alternate ID Rules** tab, open the **Item** business object or one of its children and click the **Alternate ID Rules** tab.

2. You can filter the rules to appear using the following options in the box above the **Master Alternate ID Rules** table:

   - **Show All**

     Displays all the rules.

   - **As Many**

     Displays the rules that do not have constraints.

© 2025 Siemens

- **One Of**

  Displays only the rules that match the cardinality shown in the constraints box. To look for available cardinality values, click the **Browse** button.

  The cardinality values are obtained from the **Rule** box when you create an alternate ID rule.

3. In the **Alternate ID Rules** tab, click the **Add** button to the right of the **Master Alternate ID Rules** table to **create a new alternate ID rule**.

GRM Rules editor

Use the **GRM Rules** editor to add, modify, or remove Generic Relationship Management (GRM) rules. A GRM rule applies constraints on the relationship between two business objects.

1. To access the **GRM Rules** editor, on the menu bar, choose **BMIDE→Editors→GRM Rules Editor**.

2. To search for existing GRM rules by primary object, secondary object, or relation, type strings in the **Primary**, **Secondary**, or **Relation** boxes, or click the **Browse** buttons. The GRM rules table displays the results of the search.

3. To **create a GRM rule** between two objects, click the **Add** button.

Operations tab of the Business Object editor

1. To access the **Operations** tab in the **Business Object** editor, open a business object and click the **Operations** tab.

   The **Operations** tab shows available operations on business objects.

2. To create a new operation on a business object, click the **Operations** tab and then click the **Add** button.

   > Note:
   >
   > To create a new operation on a property, select a property on the **Properties** tab, click the **Property Operations** tab, and click the **Add** button.

3. To make an operation overridable by another operation, select it and click the **Override** button.

4. To see the details of an operation, select the operation and click the **Operation Definition** tab.

5. To see the extension points defined for an operation, select the operation and click the **Extensions Attachments** tab. Extension points include:

   - **Pre-Condition**

© 2025 Siemens

Places limits before an action. For example, use this to limit individual users by their work context to create only a certain item type. Preconditions are executed first in an operation dispatch. If any of the preconditions fails, the operation is aborted. Typical examples of preconditions are naming rules.

- **Pre-Action**

  Executes code before an action. For example, use this to add user information to the session prior to translation. Preactions are executed after preconditions and before the base action. If any of the preactions fail, the operation is aborted. A typical example is an initial value rule that needs to set an initial value before the save base action is invoked.

- **Post-Action**

  Executes code after an action. For example, use this to automatically start an item in a workflow. If any of the postactions fail, the operation is aborted.

- **Base-Action**

  Executes code for an action. The base action is the actual operation implementation, and cannot be replaced. **Base-Action** is used only for user exits operations.

Class editor

The **Class** editor displays information on the selected class.

1. To access the **Class** editor, in the **Advanced** perspective, right-click a class in the **Classes** view, and choose **Open**.

2. To add an attribute to a custom class, in the **Class** editor, click the **Add** button to the right of the **Attributes** table.

Code Generation editors

Introduction to Code Generation editors

The **Extensions→Code Generation** folder is used for software development. When you right-click a code generation object and choose **Open**, an editor appears that allows you to work on characteristics of the object.

© 2025 Siemens

Data types editors

Introduction to data types editors

In software development, *data types* are different kinds of data, for example, characters, numbers (double and float), symbols (Boolean), dates, and so on. To see available data types, open the **Extensions→Code Generation→Data Types** folder.

When you right-click a data type object and choose **Open**, an editor appears that allows you to work on characteristics of the data type object.

External Data Type editor

The **External Data Type** editor displays characteristics of the selected external data type. *Data types* define what kind of data can be used in code, and *external data types* are standard data types, as well as custom defined data types which can be imported into the Business Modeler IDE.

- To access the **External Data Type** editor, open the **Extensions→Code Generation→Data Types→External Data Type** folders, right-click an external data type, and choose **Open**.

Primitive Data Type editor

The **Primitive Data Type** editor displays characteristics of the selected primitive data type. *Data types* define what kind of data can be used in code, and *primitive data types* are the commonly used types such as Boolean, character, double, float, and short.

- To access the **Primitive Data Type** editor, open the **Extensions→Code Generation→Data Types→Primitive Data Type** folders, right-click a primitive data type, and choose **Open**.

Template Data Type editor

The **Template Data Type** editor displays characteristics of the selected template data type. *Data types* define what kind of data can be used in code, and *template data types* allow code to be written without consideration of the data type with which it will eventually be used.

- To access the **Template Data Type** editor, open the **Extensions→Code Generation→Data Types→Template Data Type** folders, right-click a template data type, and choose **Open**.

Library editor

The **Library** editor displays information about a library. A *library* is used in software development and is a collection of files that includes programs, helper code, and data.

1. To access the **Library** editor, open the **Extensions→Code Generation→Libraries** folders, right-click a library, and choose **Open**.

© 2025 Siemens

2. In the **Library** editor, for custom libraries, you can click the **Add** button to the right of the **Dependent Libraries** box to add a library that the selected library is dependent on.

Release editor

The **Release** editor displays details about the selected release object. A *release* is a distribution of a software product. If your organization makes software, you can create release objects to represent the software releases.

1. To access the **Release** editor, open the **Extensions→Code Generation→Releases** folders, right-click a release, and choose **Open**.

2. In the **Release** editor, for custom releases, you can click the down arrow in the **Type** box to change the release from major, to minor, to patch. You can also click the **Browse** button to the right of the **Prior Release** box to specify the release just before this release.

Services editors

Introduction to services editors

A *service* is a collection of Teamcenter actions (service operations) that all contribute to the same area of functionality. To see available services, open the **Extensions→Code Generation→Services** folder. The **Services** folder contains service libraries, and each library contains services.

When you right-click a service library or service and choose **Open**, an editor appears that allows you work on characteristics of the object.

Service Library editor

The **Service Library** editor displays characteristics of the selected service library. A *service library* is a collection of files used by a service and its operations that includes programs, helper code, and data. Services are grouped under service libraries. You must create a service library before you create a service.

1. To access the **Service Library** editor, open the **Extensions→Code Generation→Services** folders, right-click a service library, and choose **Open**.

2. In the **Service Library** editor, you can change some aspects of custom service libraries. You can change the description, and you can click the **Add** button to the right of the **Dependent Libraries** pane to add libraries that the current library depends on.

Main tab in the Service editor

The **Main** tab of the **Service** editor displays characteristics of the selected service. A *service* is a collection of Teamcenter actions (service operations) that all contribute to the same area of functionality.

© 2025 Siemens

- To access the **Main** tab of the **Service** editor, open the **Extensions→Code Generation→Services** folders, expand a service library, right-click a service, and choose **Open**.

Data Types tab

The **Data Types** tab in the **Service** editor displays data types on the selected service. *Service data types* are used by service operations as return data types and can also be used as parameters to operations. The data types on a service are defined for use only by that service and are not shared with other services or other libraries.

1. To access the **Data Types** tab in the **Service** editor, open the **Extensions→Code Generation→Services** folders, expand a service library, right-click a service, choose **Open**, and click the **Data Types** tab.

2. In the **Data Types** tab, you can select a data type and click **Data Type Definition** to see characteristics of the data type. You can also change the data type by clicking the **Edit** button, or create a new data type by clicking the **Add** button.

Operations tab in the Service editor

The **Operations** tab in the **Service** editor displays operations on the selected service. A *service operation* is a function, such as create, checkin, checkout, and so on. Group service operations under a service to create a collection of operations that all contribute to the same area of functionality.

1. To access the **Operations** tab in the **Service** editor, open the **Extensions→Code Generation→Services** folders, expand a service library, right-click a service, choose **Open**, and click the **Operations** tab.

2. In the **Operations** tab, you can select a operation and click **Operation Definition** to see characteristics of the operation.

3. You can also change the operation by clicking the **Edit** button, or create a new service operation by clicking the **Add** button. To deprecate the operation, click **Deprecate**.

Constants editors

Introduction to constants editors

The **Extensions→Constants** folder is used for holding different kinds of configuration point objects, known as constants. When you right-click a constant and choose **Open**, an editor appears that allows you to work on characteristics of the constant.

Business Object Constant editor

The **Business Object Constant** editor displays details about the selected **business object constant**. *Business object constants* provide default values to business objects. Because these constants are attached to business objects, they are inherited and can be overridden in the hierarchy.

© 2025 Siemens

1. Open the **Extensions→Constants→Business Object Constants** folders, right-click a constant, and choose **Open**.

2. In the **Business Object Constant** editor, you can work with custom business object constants. Click the **Add** button to the right of the **Business Object Scope** box to change which business objects the constant applies to. You can also change the custom constant's value in the **Default Value** box.

Global Constants editor

The **Global Constants** editor shows available **global constants** and allows you to edit their values. *Global constants* provide consistent definitions that can be used throughout the system. These constants have only one value, either the default value or the value you set.

1. On the menu bar, choose **BMIDE→Editors→Global Constants Editor**.

2. Select the constant in the **Global Constants** table and click the **Edit** button.

3. In the **Modify Global Constant** dialog box, there are two ways to change values, depending on whether the constant can hold a single value or multiple values:

   - Single-value constants

     Enter a new value in the **Value** box, or for true/false constants, select **true** or **false**.

   - Multiple-value constants

     Click the **Add** button to the right of the **Value** table to add constants.

4. Click **Finish**.

   The changed value displays in the **Global Constants** table. Note that a check mark appears in the **Overridden** column and the name of your project displays in the **Template** column, indicating that your project overrides the value of the constant.

You can also open an individual global constant. Open the **Extensions→Constants→Global Constants** folders, right-click a constant, and choose **Open**. In this editor, you can change the description and the default value of custom global constants.

Property Constants editor

The **Property Constant** editor displays details about the selected **property constant**. *Property constants* provide default values to business object properties. Because these constants are attached to properties, they are inherited and can be overridden in the hierarchy.

1. To access the **Property Constant** editor, open the **Extensions→Constants→Property Constants** folders, right-click a constant, and choose **Open**.

© 2025 Siemens

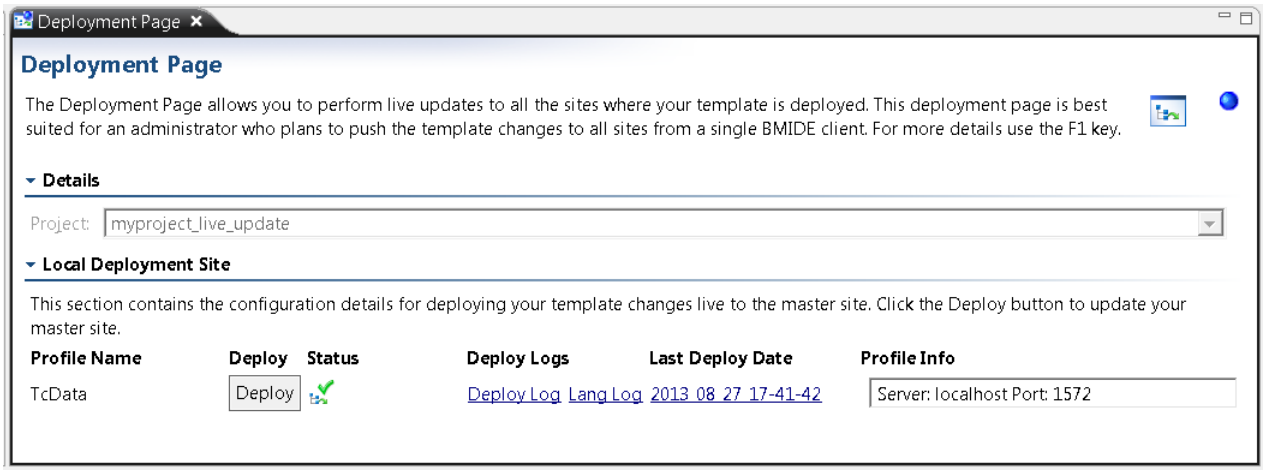
2. You can change some aspects of custom property constants. In the **Property Constant** editor, click the **Add** button to the right of the **Property Scope** box to change the properties to which the constant applies, and type in the **Default Value** box to change the constant's value.

Deployment page

The deployment page is used to **perform live updates** of nonschema data (for example, LOVs) to production servers.

To connect to multiple sites, you must **configure FMS**.

1. After you finish testing and are ready to deploy to the production servers, choose **BMIDE→Live Update→Deployment Page**.



2. Click the **Deploy** button to deploy the template to production servers.

3. To verify the live update, log on to a production server and confirm that you can create instances of your newly revised data model.

> Note:
>
> To **see the data changes**, end users must log off and log on again and wait for servers to cycle through the changes.

Document management editors

Introduction to document management editors

The **Extensions→Document Management** folder allows an administrator to create objects that specify how documents are handled in Teamcenter. When you right-click a document management object and choose **Open**, an editor appears that allows you to work on characteristics of the object.

© 2025 Siemens

Dispatcher service config editor

Introduction to the Dispatcher Service Config editor

The **Extensions→Document Management→Dispatcher Service Config** folder is used for holding dispatcher service configuration objects. A *dispatcher service configuration* is an object that defines the visualization file format that a dataset file is translated into. For example, it may specify that CAD design files are translated into 3D visualization files.

When you right-click a dispatcher service configuration object and choose **Open**, an editor appears that allows you to work on characteristics of the object.

Dispatcher Service Config Creation Info tab

The **Dispatcher Service Config Creation Info** tab displays details about the selected dispatcher service configuration object. A dispatcher service configuration defines the visualization file format that a dataset file is translated into in Teamcenter.

1. To access the **Dispatcher Service Config Creation Info** tab, open the **Extensions→Document Management→Dispatcher Service Config** folder, right-click a dispatcher service configuration object, choose **Open**, and click the **Dispatcher Service Config Creation Info** tab.

2. In the **Dispatcher Service Config Creation Info** tab, you can revise details about the selected dispatcher service configuration. You can click the **Add** button to add arguments to the dispatcher service.

Dispatcher Service Config Relation Info tab

The **Dispatcher Service Config Relation Info** tab on the dispatcher service configuration editor displays details about the source dataset type to be translated, and the target (derived) dataset type it is to be translated to. A *dispatcher service configuration* defines the visualization file format that a dataset file is translated into in Teamcenter.

1. To access the **Dispatcher Service Config Relation Info** tab, open the **Extensions→Document Management→Dispatcher Service Config** folder, right-click a dispatcher service configuration object, choose **Open**, and click the **Dispatcher Service Config Relation Info** tab.

2. In the **Dispatcher Service Config Relation Info** tab, you can revise details about the source and target (derived) dataset types for a custom dispatcher service configuration.

IRDC editor

In the Business Modeler IDE, the **Extensions** > **Document Management** > **IRDC** folder is used for holding item revision definition configuration (IRDC) objects. An *IRDC* defines how item revisions are handled at specific times in the life cycle, such as at item creation, checkin, checkout, save as, and revise.

© 2025 Siemens

When you right-click an IRDC object and choose **Open**, an editor appears that allows you to work on characteristics of the object.

This tab	Contains this information
Base Criteria Info	Details about the selected item revision definition configuration (IRDC) object.
IRDC Dataset Criteria Info	Specifies the source and target (derived) dataset types for visualization translation for this item revision. Example: You may want Microsoft Word document datasets (**MSWord** business object) to get translated into PDFs (**CrfOutputPdf** business object). You can also set up the visualization for datasets by creating dispatcher service configurations.
IRDC Dataset Naming Page Info	Specifies how the item revision dataset is named, as well as how the translated (derived) visualization dataset file is named at checkin.
IRDC Checkin Page Info	Specifies what happens when the item revision is checked in, such as whether to generate visualization files or to index dataset files.
IRDC Deep Copy Rules Page Info	Configures deep copy rules for the item revision. Deep copy rules govern how item revisions are copied during save as and revise operations. This tab works the same as the **Deep Copy Rules tab**.
IRDC Markup Page Info	Configures markup rules for the selected IRDC item revision.

Print Configuration editor

Introduction to the Print Configuration editor

The **Extensions→Document Management→Print Configuration** folder is used for holding print configuration objects. A *print configuration* is an object that defines batch print settings.

When you right-click a print configuration object and choose **Open**, an editor appears that allows you to work on characteristics of the object.

Dispatcher Service Information tab

The **Dispatcher Service Information** tab displays details about the selected print configuration object. A print configuration defines the batch print settings for documents.

© 2025 Siemens

1. To access the **Dispatcher Service Information** tab, open the **Extensions→Document Management→Print Configuration** folder, right-click a print configuration object, choose **Open**, and click the **Dispatcher Service Information** tab.

2. In the **Dispatcher Service Information** tab, you can revise details about the selected print configuration. You can click the **Add** button to add arguments to the print configuration.

Printing Specification tab

The **Printing Specification** tab displays printing details about the selected print configuration object. A print configuration defines the batch print settings for documents.

1. To access the **Printing Specification** tab, open the **Extensions→Document Management→Print Configuration** folder, right-click a print configuration object, choose **Open**, and click the **Printing Specification** tab.

2. In the **Printing Specification** tab, you can revise printing details about the selected print configuration.

    a. Click **Browse** to the right of the **Printer Name** box to select the networked printer where this print configuration will be printed.

    b. Click the **Add** button to the right of the **Paper Sizes** table to select the available paper sizes on the printer to make available for print batch jobs.

    c. Click **Add** to the right of the **Supported Datasets** table to select the document types that can be printed.

    d. Select the **Stamps Supported** check box to allow stamps on the printed documents such as the date or a watermark. If object types are printed for which stamps are configured, stamps are automatically placed on the printed documents.

    > Note:
    >
    > The **Stamps Supported** field is not yet supported for Teamcenter Document Management

System Stamp Configuration editor

Introduction to the System Stamp Configuration editor

The **Extensions→Document Management→System Stamp Configuration** folder is used for holding system stamp configuration objects. A *system stamp configuration* is an object that defines the system stamp on documents when batch printing.

When you right-click a system stamp configuration object and choose **Open**, an editor appears that allows you to work on characteristics of the object.

© 2025 Siemens

Base Information tab

The **Base Information** tab displays details about the selected system stamp configuration object. A system stamp configuration defines the system stamp on documents when batch printing.

1. To access the **Base Information** tab, open the **Extensions→Document Management→System Stamp Configuration** folder, right-click a system stamp configuration object, choose **Open**, and click the **Base Information** tab.

2. In the **Base Information** tab, you can revise details about the selected system stamp configuration.

Stamp Information tab

The **Stamp Information** tab allows you to change the stamp and watermark to place on batch printed documents.

1. To access the **Stamp Information** tab, open the **Extensions→Document Management→System Stamp Configuration** folder, right-click a system stamp configuration object, choose **Open**, and click the **Stamp Information** tab.

2. In the **Stamp Information** tab, you can revise details about the stamp and watermark settings.

Teamcenter Component editor

Use the **Teamcenter Component** editor to group business objects and conditions together that belong to the same function.

1. To access the **Teamcenter Component** editor, open the **Extensions→Teamcenter Component** folder, right-click a **Teamcenter Component** object, and choose **Open**.

2. To enable a custom **Teamcenter Component** object for use with verification rules, select the **Enabled for Verification Rule** ☑ check box.

LOV editor

The **LOV** editor displays details about the selected **list of values**. Lists of values (LOVs) are pick lists of data entry items that are accessed by Teamcenter users from a menu at the end of a data entry box.

1. To access the **LOV** editor, open the **Extensions→LOV** folder, right-click an LOV in the **Batch LOV**, **Classic LOV**, or **Dynamic LOV** folder, and choose **Open**.

2. For classic LOVs, in the **LOV** editor, click the **Add** button to the right of the value table to **add more values to the LOV**.

3. To **attach the LOV to a property**, click the **Attach** button to the right of the **LOV Attachments** table.

© 2025 Siemens

Options editors

Introduction to options editors

The **Extensions→Options** folder holds objects that modify different aspects of business objects. When you right-click an object in a subfolder under the **Options folder** and choose **Open**, an editor appears that allows you to work on characteristics of the object.

ID Context editor

The **ID Context** editor displays details about the selected ID context. An **ID context** defines when you use unique item IDs.

1. To access the **ID Context** editor, open the **Extensions→Options→ID Context** folder, right-click an ID context, and choose **Open**.

2. You can type in the **Display Name** box to change the name of the object displayed in the Teamcenter user interface. This also changes the value in the **Localization** table at the bottom of the page.

3. Type in the **Description** box to change the description of the ID context.

4. Click buttons to the right of the **Localization** table to change the name of the object displayed in different languages in the Teamcenter user interface.

Note Type editor

The **Note Type** editor displays details about the selected **note type**. A note type is an object associated with a product structure occurrence in a Structure Manager bill of materials (BOM).

1. To access the **Note Type** editor, open the **Extensions→Options→List of Note Types** folder, right-click a note, and choose **Open**.

2. You can type in the **Display Name** box to change the name of the object displayed in the Teamcenter user interface. This also changes the value in the **Localization** table at the bottom of the page.

3. Select the **Attach Value List** check box if you want to attach a value to a custom note from a list of values (LOV), and then click the **Browse** button to the right of the **LOV** box to select the list of values, and click the **Browse** button to the right of the **Default Value** box to select the value.

4. Click buttons to the right of the **Localization** table to change the name of the object displayed in different languages in the Teamcenter user interface.

© 2025 Siemens

Occurrence Type editor

The **Occurrence Type** editor displays details about the selected **occurrence type**. An occurrence type is used to distinguish how items occur in a product structure.

1. To access the **Occurrence Type** editor, open the **Extensions→Options→List of Occurrence Types** folder, right-click an occurrence type, and choose **Open**.

2. You can type in the **Display Name** box to change the name of the object displayed in the Teamcenter user interface. This also changes the value in the **Localization** table at the bottom of the page.

3. Click buttons to the right of the **Localization** table to change the name of the object displayed in different languages in the Teamcenter user interface.

Status editor

The **Status** editor displays details about the selected **status**. A status is applied to an object after it goes through a workflow. Typical statuses are **Pending** and **Approved**.

1. To access the **Status** editor, open the **Extensions→Options→Status** folder, right-click a status, and choose **Open**.

2. You can type in the **Display Name** box to change the name of the object displayed in the Teamcenter user interface. This also changes the value in the **Localization** table at the bottom of the page.

3. Click buttons to the right of the **Localization** table to change the name of the object displayed in different languages in the Teamcenter user interface.

Storage Media editor

The **Storage Media** editor displays details about the selected **storage media** object. *Storage media* is a storage device category such as a hard disk or optical device.

1. To access the **Storage Media** editor, open the **Extensions→Options→Storage Media** folder, right-click a storage media object, and choose **Open**.

2. You can change the settings in the **Media Type**, **Logical Device**, and **Description** boxes.

Tool editor

Introduction to the Tool editor

The **Extensions→Options→Tool** folder is used for holding **tool** objects. A tool represents a software application, such as Microsoft Word or Adobe Acrobat. You associate a tool with a type of dataset so you

© 2025 Siemens

can launch the dataset file from Teamcenter. When you right-click a tool and choose **Open**, an editor appears that allows you to work on characteristics of the tool.

New Tool Type tab

The **New Tool Type** tab displays details about the selected tool. A tool represents a software application, such as Microsoft Word or Adobe Acrobat.

1. To access the **New Tool Type** tab, open the **Extensions→Options→Tool** folder, right-click a tool, choose **Open**, and click the **New Tool Type** tab.

2. You can type in the **Name** box to change the name of the object displayed in the Teamcenter user interface.

3. You can revise values in the following boxes: **MIME/TYPE**, **Shell/Symbol**, **Vendor Name**, **Revision**, and **Release Date**.

Tool Input/Output tab

The **Tool Input/Output** tab displays details about the type of data the tool accepts or outputs, for example, ASCII or binary.

1. To access the **Tool Input/Output** tab, open the **Extensions→Options→Tool** folder, right-click a tool, choose **Open**, and click the **Tool Input/Output** tab.

2. Click the **Add** button to the right of the **Input** table to specify the type of data the tool accepts.

3. Click the **Add** button to the right of the **Output** table to specify the type of data the tool outputs.

Tool Markup Info tab

The **Tool Markup Info** tab displays details about the view and markup capabilities of the **tool**.

- To access the **Tool Markup Info** tab, open the **Extensions→Options→Tool** folder, right-click a tool, choose **Open**, and click the **Tool Markup Info** tab.

View Type editor

The **View Type** editor displays details about the selected **view type**. A *view type* is a BOM view revision (BVR) category.

1. To access the **View Type** editor, open the **Extensions→Options→List of View Types** folder, right-click a view, and choose **Open**.

© 2025 Siemens

2. You can type in the **Display Name** box to change the name of the object displayed in the Teamcenter user interface. This also changes the value in the **Localization** table at the bottom of the page.

3. Click buttons to the right of the **Localization** table to change the name of the object displayed in different languages in the Teamcenter user interface.

Property Formatter editor

The **Property Formatter** editor displays details about the selected **property formatter**. *Property formatters* are display definitions that dictate how properties are shown in the Active Workspace user interface.

To access the **Property Formatter** editor, in the **BMIDE** view, open the **Extensions**>**Property Formatters** folder, right-click a property formatter object, and choose **Open**.

- If it is a custom property formatter, to provide a new format, to the right of the **Format Definition** click **Edit**.

- To **attach the property formatter to a property**, on the **Property Formatters Attachments** tab, click **Attach**.

- To add text to be used in different languages, on the **Localizations** tab, click **Add**.

Rules editors

Introduction to the rules editors

The **Extensions**→**Rules** folder holds **rules** that govern the behavior of business objects. When you right-click an object in a subfolder under the **Rules** folder and choose **Open**, an editor appears that allows you to work on characteristics of the object.

Alias ID Rule editor

The **Alias ID Rule** editor displays details about the selected **alias ID rule**. Alias identifiers store part numbers and other attribute information for similar parts.

1. To access the **Alias ID Rule** editor, open the **Extensions**→**Rules**→**AliasId Rules** folder, right-click an alias ID rule, and choose **Open**.

2. Click the **Browse** button to the right of the **Identifier Context** box to change the **identifier context** on the rule.

3. Click the **Browse** button to the right of the **Identifier Type** box to select the identifier or identifier revision business object to use for this rule.

© 2025 Siemens

Application Extension Point editor

The **Application Extension Point** editor displays details about the selected **application extension point**. An *application extension point* is an decision item that can be used in a Teamcenter rich client application.

1. To access the **Application Extension Point** editor, open the **Extensions→Rules→Application Extension Points** folder, right-click a point, and choose **Open**. In the **Application Extension Point** editor, you cannot change anything on COTS points, but you can revise custom points.

2. Click the **Add** button to the right of the **Inputs** or **Outputs** table to change the inputs and outputs on the application extension point.

Application extension rule editor

The **Application Extension Rule** editor displays details about the selected **application extension rule**. An *application extension rule* determines when an application extension point is used and defines inputs and outputs. When the input is matched, the rule engine returns the output to the application that called the extension point.

1. To access the **Application Extension Rule** editor, open the **Extensions→Rules→Application Extension Point** folder, right-click an application extension rule, and choose **Open**. In the **Application Extension Rule** editor, you cannot change anything on COTS application extension rules, but you can revise custom ones.

2. Click the **Add** button to change the inputs and outputs on the custom application extension rule.

Business Context editor

The **Business Context** editor displays details about the selected business context. A *business context* defines the user groups for whom a rule applies.

1. To access the **Business Context** editor, open the **Extensions→Rules→Business Contexts** folder, right-click a context, and choose **Open**.

2. Click the **Add** button to the right of the **Accessor** table to add another group or role to the business context. The Teamcenter Repository Connection wizard prompts you to log on to a server to look up its available groups and roles. Select the group and role from the **Accessor Selection** dialog box.

Condition editor

The **Condition** editor displays details about the selected **condition**. *Conditions* are conditional statements that resolve to true or false based on the evaluation of an expression.

1. To access the **Condition** editor, open the **Extensions→Rules→Conditions** folder, right-click a condition, and choose **Open**.

© 2025 Siemens

2. The values that can be changed in this editor depends on whether the condition is COTS and if it is marked as secure:

   - If it is COTS and secure, nothing can be changed.
   - If it is COTS and not secure, only the **Expression** can be changed.
   - If it is custom, then you can select new **Input parameters** to change the **Signature**, or type new expressions in the **Expression** box. Remember that if you change the signature, it can impact areas where the condition has been applied (for example, naming rule attachments, LOV attachments, and so on).

Extension Definition editor

The **Extension Definition** editor displays details about the selected **extension rule**.

1. To access the **Extension Definition** editor, open the **Extensions→Rules→Extensions** folder, right-click an extension rule, and choose **Open**. You can change custom COTS extension rules in this editor.

2. Click the **Add** button to the right of the **Parameter List** table to add a new parameter to the rule.

3. Click the **Add** button to the right of the **Availability** table to make the extension rule available on another business object or property.

ID Generation Rule editor

The **ID Generation Rule** editor displays details about the selected ID generation rule. An *ID generation rule* is a rule used to generate an item ID with additional counters or property values appended to it as part of **intelligent part numbering**.

1. To access the **ID Generation Rule** editor, open the **Extensions→Rules→ID Generation Rule** folder, right-click an ID generation rule, and choose **Open**.

2. Click the **Add** button to the right of the **Concatenation Rules** box to add a new concatenation rule.

3. Click the **Attach** button to the right of the **ID Generation Rule Attachments** table to attach the rule to a business object property.

Naming Rule editor

The **Naming Rule** editor displays details about the selected **naming rule**. A *naming rule* defines how objects are named, including how IDs are automatically assigned when objects are created.

1. To access the **Naming Rule** editor, open the **Extensions→Rules→Naming Rules** folder, right-click a naming rule, and choose **Open**.

© 2025 Siemens

2. Click the **Add** button to the right of the **Patterns** box to **add a new naming rule pattern**.

3. Click the **Attach** button to the right of the **Naming Rule Attachments** table to attach the rule to a business object property.

Propagation Rules editor

The **Propagation Rules** editor displays details about defined **propagation rules** and enables you to modify, **add** or remove rules.

Filtering the rules display

Set the following display options to filter the propagation rules that are shown in the editor. Filter options are cumulative.

Option	Description
Direction	Filters the displayed rules based on the direction defined in the rules. Options include **Forward**, **Reverse**, and **Both**.
Property Type	Filters the displayed rules based on the **Property Type** defined for the rule. • **Relation** Limits the display to rules with **Property Type**=**Relation**. • **Reference** Limits the display to rules with **Property Type**=**ReferenceProperty**.
Destination	Enabled when the selected **Property Type** is **Relation** To limit the display to rules with a specific destination business object, enter the business object name. Because rules for business objects are inherited by child business objects, rules for parent business objects of the specified business object are included in the display.
Source	To limit the displayed rules to rules with a specific source business object, enter a business object name.
Relation / **Reference**	The name of the option depends on the selected **Property Type** filter option.

© 2025 Siemens

Option	Description
	To limit the displayed rules to a particular **Relation Type**, in **Property Type** select **Relation**, and then in the **Relation** option enter a relation type business object name. To limit the displayed rules to a particular source business object reference property, in **Property Type** select **Reference**, in **Source** enter a business object name, and then in the **Reference** option enter a property name.
Propagation Group	To limit the displayed rules to a particular **Propagation Group** enter a propagation group name.

Revision naming rule editor

The **Revision Naming Rule** editor displays details about the selected **revision naming rule**. A *revision naming rule* is a business rule that defines the naming convention and sequence for a revision property.

1. To access the **Revision Naming Rule** editor, open the **Extensions→Rules→Revision Naming Rules** folder, right-click a revision naming rule, and choose **Open**.

2. Type in the available boxes to change values on the revision naming rule.

3. Click the **Attach** button to the right of the **Revision Naming Rule Attachments** table to attach the rule to a business object property.

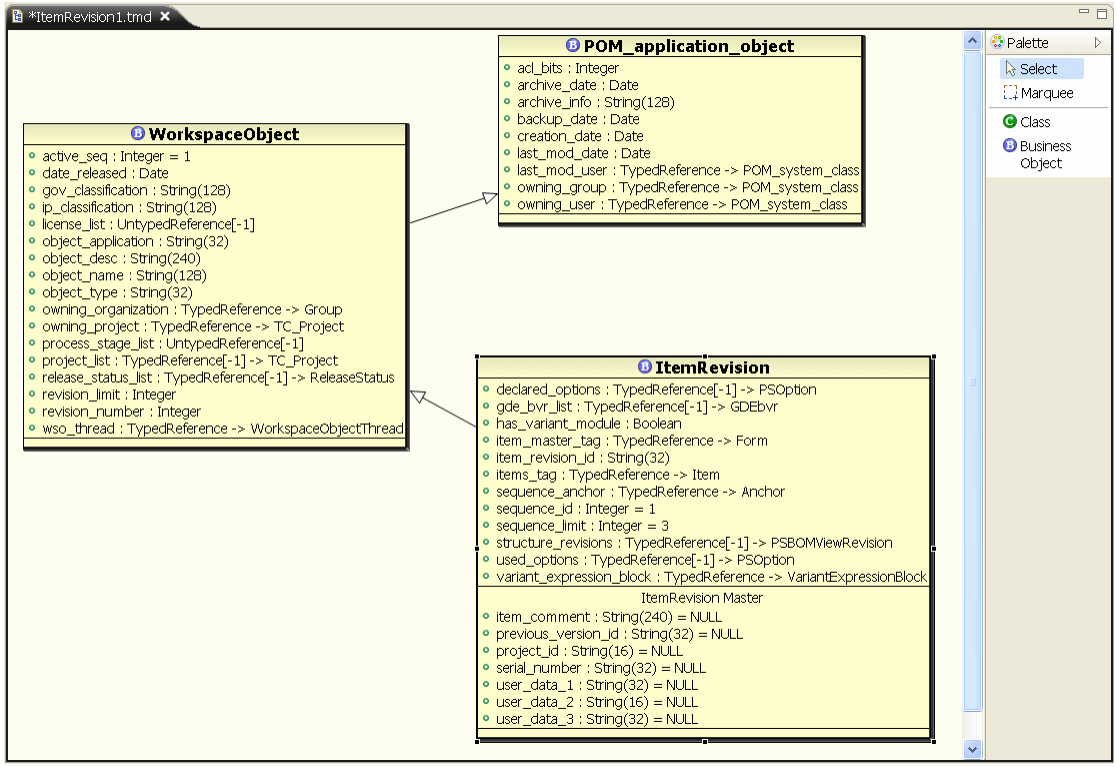
UML editor

UML editor interface

The **UML editor** shows business objects and classes in a graphical format. To access the UML editor, right-click a business object or a class and choose **Open In UML Editor**. The business object or class appears in the UML editor.

To work with the business object or class displayed in the editor, right-click the object and make selections from the menu. You can also use the palette on the right side of the editor.

© 2025 Siemens



The UML editor shows the following attribute information:

- For attributes with initial values, the initial value is shown, for example:

```
active_seq : Integer = 1
user_data_1 : String(32) = NULL
```

- For attributes that have a string storage, the string size is shown in parentheses, for example:

```
object_desc : String(240)
object_type : String(32)
```

- For attributes that are typed reference, the reference class is shown, for example:

```
owning_organization : TypedReference -> Group
```

- For attributes that have an array storage, the array size is shown in square brackets, for example:

```
License_list : UntypedReference[-1]
MyStringArray : String[44](32)
```


© 2025 Siemens

UML editor shortcut menu

The following is a list of actions you can perform from the shortcut menu when you right-click in the UML editor. To act on the business object or class, you must right-click the name of the business object or class at the top of the UML box.

Symbol	Menu command	Description
	**Undo**	Cancels the previous action.
	**Redo**	Performs the action that had been canceled with **Undo**.
	Print	Prints the UML diagram.
	Add Class	Adds a subclass to the selected class.
	Add Business Object	Adds a new subobject to the selected business object.
	Open	Opens the selected object in a new view.
	Open Extension Rules	Opens the **extension rules** that apply to the selected business object.
	Show	Displays aspects of the selected business object or class: • **Children** Displays the children of the selected business object or class. • **Parent** Displays the parent of the selected business object or class. • **Inheritance to Root** Displays the inheritance path from the selected business object or class to the root class (**POM_object**). • **Storage Class**

© 2025 Siemens

Symbol	Menu command	Description
		Displays the class that stores attributes for the selected business object or class. • **Typed Reference** Displays the typed reference when you right-click a typed reference attribute on the business object or class. A typed reference points to a Teamcenter class. • **Class** Searches for a class and displays it in the UML editor. • **Business Object** Searches for a business object and displays it in the UML editor. • **Relations** Shows the relations between the selected objects. Drag business objects into the UML editor, press the Ctrl or Shift key and click multiple business objects to select them, and right-click and choose **Show→Relations**. The **New GRM Rule wizard** runs. If you type ***** in the **Relation** box and click **Finish**, all the relations between the selected business objects are displayed.
	Hide	Hides the selected object.
	Select Filters	Sets filters to limit the types of attributes shown in the view.
	Generate Code→C++ Classes	Generates C++ source code for the selected project.

© 2025 Siemens

Note:

The remaining options on the menu are not provided by the Business Modeler IDE but are provided by the Eclipse framework.

UML editor palette

With objects displayed in the UML editor, click the arrow at the top of the **Palette** bar docked on the right of the window. The palette expands to display a series of buttons.

The following is a list of actions you can perform from the UML editor palette.

Symbol	Menu command	Description
	**Select**	Activates a cursor arrow that allows you to select individual objects in the UML editor.
	**Marquee**	Activates a **+** cursor that allows you to select a group of objects in the UML editor.
	**Class**	Dragging the **Class** button into the UML editor launches the new class wizard.
	**Business Object**	Dragging the **Business Object** button onto the UML editor launches the new business object wizard.

User Exits (Extension Rules) editor

User exits are predefined operations. The editor used for user exits is the **Extension Rules** editor. Extension rules allow you to write a custom function or method for Teamcenter in C or C++ and attach the rules to predefined hook points in Teamcenter.

To access the **Extension Rules** editor for a user exit, open the **Extensions→User Exits** folder, right-click a user exit, and choose **Open Extension Rule**. In the **Extension Rules** editor, the available operations are found in the **Operations** folder.

To see extension rules operations defined for business objects, right-click a business object, choose **Open**, and click the **Operations** tab in the resulting editor. The available operations for the business object are found in the **Operations** folder, and available operations for the properties are found in the **Properties** folder. To see the extension points defined for an operation, select the operation and click the **Extensions Attachments** link on the lower right side of the editor.

© 2025 Siemens

Verification rule editor

Use the **Verification Rule** editor to specify when **Teamcenter Component objects** can be used in Teamcenter.

1. To access the **Verification Rule** editor, choose **BMIDE→Editors→Verification Rules Editor**.

2. To **add a verification rule**, click the **Add** button to the right of the **Verification Rule** table.

Eclipse views used by the Business Modeler IDE

Introduction to Eclipse views used by the Business Modeler IDE

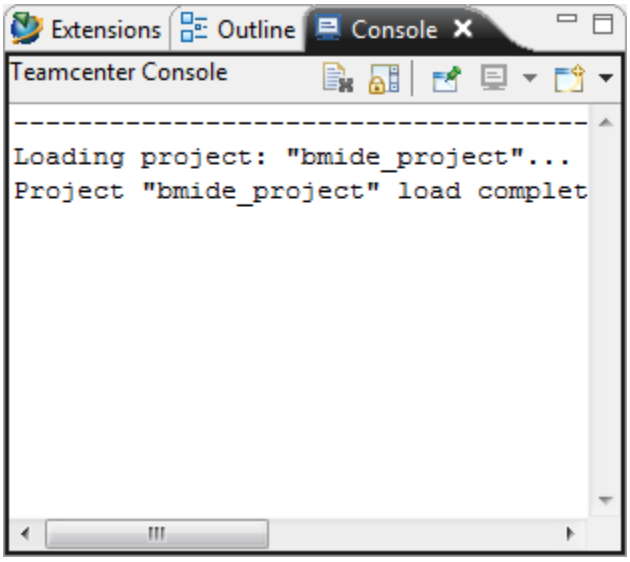
The Business Modeler IDE is built on an Eclipse framework. This section documents default Eclipse views that are used for Business Modeler IDE functions. Views are the tabbed panes that appear in the user interface.

For more documentation on Eclipse views, see the Workbench User Guide accessible by choosing **Help→Help Contents**.

Console view in the Business Modeler IDE

The **Console** view is a default Eclipse view used to display output from loading and building your projects. Watch this window for any problems or errors with the build, database update, or server startup.

Typically, an error may occur as a result of incorrectly merging a source file with a source control management (SCM). Each error displays the XML file name, the line number, and error message. All errors should be fixed before continuing to use the Business Modeler IDE.



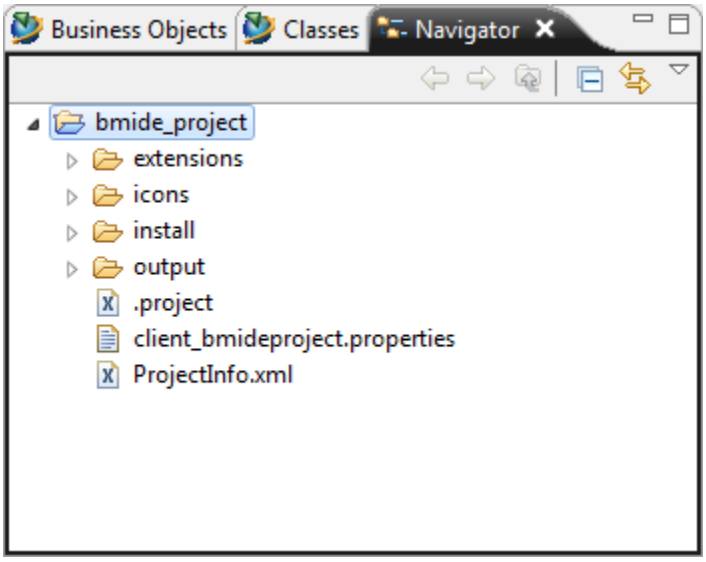
© 2025 Siemens

Navigator view in the Business Modeler IDE

The **Navigator** view is a default Eclipse view used for browsing file system objects.

The **Navigator** view in the **Advanced** perspective always shows the projects at the top level and any folders and files under them. For example, if you have a project, expand the project in the **Navigator** view and note the files it contains. You can open files in an editor by double-clicking them, or right-click them and choose **Open**.

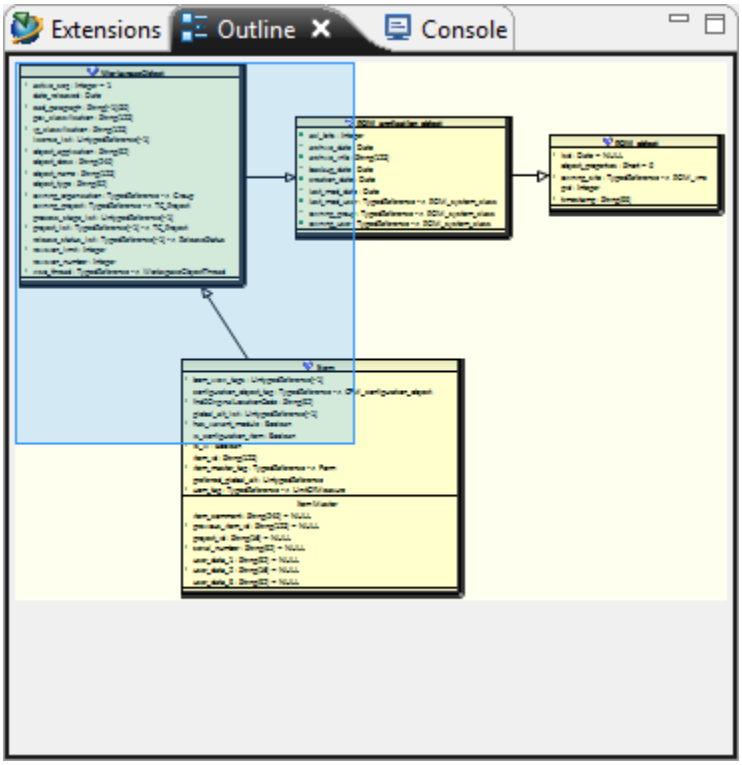
Projects typically contain a project file, and source files that contain your extensions to the data model. They can also contain **.tmd** files generated by the UML editor.



Outline view in the Business Modeler IDE

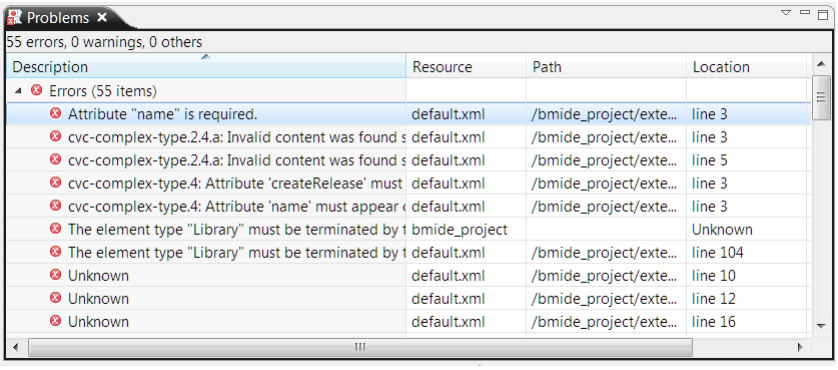
The **Outline** view is a default Eclipse view that is used for displaying a thumbnail of the contents of the UML editor. Dragging the shaded box changes what is displayed in the UML editor.

© 2025 Siemens



Problems view in the Business Modeler IDE

The **Problems** view is a default Eclipse view used for identifying problems in source files. You can click a problem in this view to open the source file where the problem exists.



© 2025 Siemens