Borland Together 2008

Borland Together Modeling Guide

Borland®

Micro Focus
575 Anton Blvd., Suite 510
Costa Mesa, CA 92626

June 2011
PDF

Getting Started

Concepts

Concepts

Procedures

Procedures

Reference

Getting Started

Getting Started with Together

This section contains an introduction to modeling with **Borland Together**. The sample projects and Cheat Sheets are designed to help you explore Together features while working with projects. Some of the special features include: BPMN modeling, patterns, generating project documentation, reverse engineering and so forth.

In This Section

Together Overview

Welcome to Borland® Together®, the award-winning, design-driven environment for modeling applications. Together includes features such as support for Unified Modeling Language (UML) 2.0, Object Constraint Language (OCL), patterns, Quality Assurance audits and metrics, source code generation, IBM Rational Rose (MDL) format import, XMI format import and export, and automated documentation generation.

Borland® Together® is a visual modeling platform designed to support architects; Java, C# and C++ developers; UML™ and DSL designers; business process analysts; and data modelers in the accelerated delivery of high-quality software applications.

Together® helps companies manage the complexity of today's software world by communicating ideas clearly, utilizing automation for efficiency, and allowing organizations to leverage industry and internal standards. Together improves business agility and lowers maintenance costs through the delivery of a platform-independent solution for domain-specific languages (DSLs). The unique DSL Toolkit is designed to help organizations that have needs for more specific solutions than Unified Modeling Language (UML)™ models by allowing project teams to create, customize and deploy models within their own business domain and tailored to their own specific needs. DSLs mprove the usability of modeling, eliminate unnecessary overhead, and optimize communication and efficiency among project teams. Together allows companies to achieve the right mix of leveraging industry experience embodied in standards and the freedom to tailor or invent what is needed.

Together benefits include the following:

- Leverage UML, BPMN, and ER modeling activities by generating Java, C++, C#, BPEL, and SQL DDL.
- Jump-start modeling activities by reverse-engineering Java, C++, BPEL, and SQL DDL.
- Increase productivity and quality by automating design and code reviews with audits and metrics at the model and code level.
- Improve communication with fully customizable template-based document generation that can assemble content from all model types and requirements.
- Leverage Model-Driven Architecture™(MDA) features including OMG's Query View Transformation (QVT) used in model-to-model transformations and support for OCL 2.0 with syntax highlighting, validation, code sense, refactoring, debugging and expression evaluation.
- Integrate modeling and design activities and artifacts with Application Lifecycle Management (ALM) tools and processes.

The following resources offer additional assistance, information, and services:

- For information on how to use this Help system, see Help on Help in the Related Concepts.
- Borland Together Home Page
- Borland Together Documentation
- Borland Together Support
- Borland Product Support
- Borland Services
- Borland University

If your Internet access is limited by network security, or if your computer is protected by a personal firewall, the Web-based links in this Help system might not function properly.

Related Concepts

Help on Help
Together Documentation Set

Related Reference

Together Glossary
Together Keyboard Shortcuts

Together Documentation Set

The Together documentation set consists of the following items:

Item	Description	Location
Release Notes (ReadMe)	Late-breaking information including: Last minute notes System requirements Installing and starting Together Known issues and limitations	Borland Together Release Notes
Setting Up Licensing for Borland Together.	Together licensing setup.	Setting Up Licensing for Borland Together
Online help	General, context, and dynamic help for Together including the following comprehensive information most relevant to the user: — Conceptual topics — Getting Started and Concepts — Procedural topics — Working with Projects, Creating and using profiles, Working with diagrams, Working with different types of modeling, Refactoring procedures, Using OCL, Working with patterns, Quality assurance, and Documentation generation procedures. — Reference topics — dialog boxes, wizards and GUI elements	Together main menu: **Help ▸ Help Contents**
Cheat Sheets	Interactive tutorials that help you start using basic product features. Each cheat sheet helps you complete a single task. A list of Together cheat sheets is available in the Sample projects and cheat sheets topic.	The item on Together main menu: **Help ▸ Cheat Sheets**

Related Concepts

Sample Projects and Cheat Sheets
Help on Help

Sample Projects and Cheat Sheets

Together ships with sample projects and cheat sheets that help you get acquainted with Together and its features.

The sample projects are available under **File** ▸ **New** ▸ **Example**.

Cheat sheets provided with Together are basically interactive tutorials that help you to start using some of the Together features. Each cheat sheet helps you complete some task. For more information about cheat sheets refer to Eclipse Workbench User Guide.

The cheat sheets are available under **Help** ▸ **Cheat Sheets**.

Help on Help

Together allows you to view various help topics that will assist you while you are completing your tasks.

Together Online Help

Together online Help includes conceptual overviews, procedural how-to's, and reference information, which allow you to navigate from general to more specific information as needed.

Tip: When you use a link to navigate from one topic to another topic, the context of the Help topic you are viewing might not be obvious. To find the context of a topic within the **Contents** pane, click the **Show in Table of Contents** button on the toolbar of the Eclipse Help viewer.

Concepts

Concepts introduce the main features and methods that will help you learn and understand Together techniques.

At the end of most conceptual topics, you will find links to related, more detailed information.

How-To Procedures

The how-to procedures provide step-by-step instructions.

All procedures are listed under **Procedures** in the **Contents** pane of the Help window.

Reference Topics

The reference topics provide detailed information on subjects such as configuration options, GUI elements, dialog boxes, and wizards references.

All of the reference topics are listed under the **Reference** section in the **Contents** pane of the Help window.

Context Sensitive Help

Context sensitive Help is available throughout the interface by selecting an item and pressing F1, or the **Help** button.

Typographic Conventions Used in the Help

The following typographic conventions are used throughout Together online Help.

Typographic conventions

Convention	Used to indicate
`Monospace type`	Source code, file and folder names, and text that you must type.
Boldface	GUI elements and dialog boxes.
Italics	Book titles and to emphasize new terms.
KEYCAPS	Keyboard keys, for example, the CTRL or ENTER key.

Related Concepts

Together Documentation Set

Tour of Together

Together changes the user interface according to how you want to work with Together by providing several Together perspectives to customize the Together-user experience. In Together you can choose one of the following Together perspectives:

- Modeling including Business Process Modeling Notation (BPMN), BPMN Simulation, Model Driven Architecture (MDA), Patterns and Templates, and TogetherQA group views
- DSL Toolkit
- Data Modeling
- CaliberRM
- RequisitePro

Views and Editors Associated with Each Together Perspective

The views associated with each Together Perspective vary according to the perspective selected. The views that make up each Together Perspective are described below.

Together Modeling

The Modeling perspective is the default perspective for Together. The Modeling perspective provides the following views:

View	Description
Add linked results	Shows results of applying the **Add Linked** command.
Code Generation Log	Displays a log of the code generation process.
Generate Implementation Log	Displays the log of the generating implementation code for a sequence diagram.
Generate Sequence Diagram 1.4 Log	Displays the log of the generating a sequence diagram in a UML 1.4 project.
Generate Sequence Diagram 2.0 Log	Displays the log of the generating a sequence diagram in a UML 2.0 project.
Model Audits	Displays the results of the model audits you run.
Model Bookmarks	Lists bookmarked model elements.
Model Metrics	Displays the results of the model metrics you run.
Model Navigator	Provides the logical representation of the model of your project: namespaces (packages) and diagram nodes.
Model Package Explorer	Displays the UML content for all open projects
Diagram Editor	Displays created and opened diagrams. When you use multiple diagrams, the diagram editor provides a tab for each diagram.
Properties	Displays the properties for a selected element. The properties for each element are usually divided into different categories.
Profile Constraints	Lists available profile constraints.
Profile Validation	Displays results of the profile validation process.
BPMN Validation	Lists all BPMN diagram-related errors, including diagram errors, export, and simulation errors.
BPMN Simulation	Provides simulation run progress information and tools to control the simulation process.
Metamodel Browser	Lists metamodels that can be selected as a source or a target of a Queries/Views/Transformations (QVT) transformation.

OCL Expression	Enables you to quickly evaluate OCL expressions in the explicitly specified context (a Together or Eclipse Modeling Framework [EMF] model element), or in the context of the current selection.
Last Validation Results	Displays results of the latest validation of a pattern definition.
Pattern Explorer	Enables you to logically organize patterns (using virtual trees, folders and shortcuts), and manage recognized instances of patterns.
Pattern Registry	Defines the virtual hierarchy of patterns.
Templates	Displays currently available source code templates.
Audit	Displays the results of the source code audits you run.
Metric	Displays the results of the source code metrics you run.

DSL Toolkit

The DSL Toolkit perspective provides the following views and editors:

View/Editor	Description
DSL Explorer	Provides DSL project navigation, dragging and dropping of resources and templates with model refactoring, adding and importing artifacts, and generate and validate actions.
Metamodel Explorer	Lists metamodels that can be selected as a source or target of a QVT transformation.
Generic Template Browser	Lists currently available templates.
DSL Editor	Used for editing DSL projects.
Domain Model Editor	Used for editing domain models.
Diagram Definition Editor	Used to manage general configuration properties and generation actions, advanced properties, the tooling model, the mapping model, and all of the models involved with a diagram.
Figure Gallery Editor	Used to configure figure gallery details, provide a tree view of the figure gallery model, and provide a composite viewer and editor for all models involved in a figure gallery.
Report Definition Editor	Used to define a report.
Dynamic Templates	Used to browse templates used for model code generation and to copy (override) template files into the folder specified in the Dynamic Templates Path so that you can customize the templates.
Operational QVT Traces	Used to inspect the results of a Model-To-Model QVT transformation.
Problems	Displays compilation errors.
Outline	Displays outline of the structure of the currently active file in the editor area.
Properties	Used to view and edit the properties of the currently selected item in the DSL Explorer.

Data Modeling

The Data Modeling perspective provides the following views:

View	Description
Navigation	Contains **Model Navigator** and **Navigator** tabs, by default.
Diagram Editor	Displays created and opened diagrams. When you use multiple diagrams, the diagram editor provides a tab for each diagram.
Properties	Displays the properties for a selected element. The properties for each element are usually divided into four categories: Description, a textual description of the element; Hyperlinks, links to other elements or external files and documents; Properties, UML properties; Requirement, requirement information.
Tasks	Shows tasks (reminders) that you either created or generated during the build process.

DDL Preview	Data Description Language (DDL) expressions viewer.

CaliberRM

The CaliberRM has the following views.

View	Description
CaliberRM Navigator	Allows you to connect to and browse multiple CaliberRM repositories located on different servers over the network.
Synchronizer	Allows you to review and synchronize changes made to traced requirements or external vendor objects.
CaliberRM Traces	Displays information about the requirements and external vendor objects traced to and from the requirement selected in the CaliberRM Navigator view.
Traceability Matrix	Displays all the trace links for the selected requirement, baseline or project in a single matrix view.
Requirement Grid	Displays the summary information for a set of requirements.
Properties	Displays property and attribute names and values for the requirements, traced objects, requirement types, baselines, projects and server connections.

Note: For more information about CaliberRM, refer to the CaliberRM plugin help.

RequisitePro

The RequisitePro has the following views.

View	Description
RequisitePro Navigator	Allows you to connect to and browse RequisitePro repositories.
RequisitePro Traces	Displays information about the requirements and external vendor objects traced to and from the requirement selected in the RequisitePro Navigator view.
RequisitePro Discussion	Provides the ability for users to discuss requirements by displaying the existing discussions and allowing users to post replies.

Unlike Together 2006 R2, Together 2008 does not include integrations with requirement management products such as RequisitePro. These integrations should be available separately from Borland.

Note: For complete information about RequisitePro, refer to the RequisitePro documentation.

Together provides menu items on the main menu with Together-specific commands, in addition to the views associated with each perspective.

Project Menu

Menu Item	Description
Documentation ▸ Generate HTML	Opens the Generate HTML Documentation dialog box.
Documentation ▸ Generate Using Template	Opens the Generate Documentation Using Template dialog box.

Model Menu

Menu Item	Description	
Run Model Metrics	Runs model metrics for the selected elements.	
Run Model Audits	Runs model audits for the selected elements.	
Compare With	Each Other as Model Elements	Compares two or three selected model elements against each other and shows differences in a separate view.
	Local Version	Compares a shared resource with a version stored on your disk.
Profile	Uninstall Profile	Uninstalls the selected profile.
	Open Profile Definition	Opens the profile definition project.
	Deploy profile	Starts the creating profile plug-in process.
	Run Profile Constraints	Runs profile-specific audits.
	Convert properties	Converts profile-specific properties of the projects created in the previous version of Together to the new format. For more information see Converting Profile-Specific Properties topic in the Procedures section.
	Preferences	Opens the Profile preferences in the Modeling node.
Apply Transformation	Provides QVT, eXtensible Stylesheet Language (XSL), and Model to Text transformation specific commands.	

Diagram Menu

The Diagram menu includes commands relevant to working with the diagram currently opened in the **Diagram** editor. The commands include, but are not limited to, layout and align patterns, different levels of zoom, switching grid and rulers, hiding and showing elements, and so forth.

Related Concepts

Together Capabilities Activation

Related Procedures

Activating Together Capabilities
Choosing a Together Perspective

Related Reference

Components of the Together User Interface

Concepts

Concepts

This section provides an overview of the features provided by Together.

In This Section

Project Documentation
This part describes the documentation generation facility and documentation template basics.

Together Basics

This section provides information about Together features.

In This Section

Together Project Overview

Work in Together is done in the context of a **project**. A project is a logical structure that holds all resources required for your work. All projects located in the selected Workspace are listed in the Model Navigator.

You can set up project properties when the project is being created, and modify them further, using the **Properties** dialog box.

The following is a list of projects that can be created in Together.

- **BPMN from Together 2006 Business Process Project** helps you import Business Project Modeling Notation (BPMN) projects created in Together 2006 for Eclipse.
- **Business Process Modeling Project** enables you to create end-to-end business processes.
- **C++ Modeling project** is a UML 2.0 source code modeling project.
- **Data Modeling project** provides a complete data modeling solution.
- **IDL Modeling project** is a UML 2.0 source code modeling projects.
- **Java Modeling project** is either UML 1.4 or UML 2.0 source code modeling project.
- **Java Modeling projects from Java projects** creates a Java source code modeling project from pure Java project.
- **MDA Transformation project** is a customized Eclipse plug-in project that enables you to develop various transformations in Together.
- **Pattern Definition project** is a profiled UML 2.0 modeling project that allows you to create new patterns.
- **Profile Definition project** is a profiled modeling project that allows you to create new profiles.
- **UML 1.4 project** is a design project with no source code support.
- **UML 2.0 from UML 1.4** converts both Java modeling and design projects from UML 1.4 to UML 2.0 specification.
- **UML 2.0 project** is a design project with no source code support.

Note: The project settings are initially specified on project creation. Further, you can update properties for the existing project.

Related Procedures

Together Projects

Related Reference

Together Projects

Package Overview

The notion of a package has two facets: logical and physical.

- Logically, a model consists of one or more packages. A package is a model element used to group elements, and provides a namespace for the grouped elements. A package can contain packageable elements (the elements that can be directly owned by a package) and the other packages. A model itself is a package.
- Physically, a package is a folder containing the files that store diagrams and model elements.

Contents of a package can be displayed on a special type of the Class Diagram that is synchronized with the package contents (that is, all the classifiers directly owned by this package automatically appear on the package diagram). This diagram is essential for source code projects. Each package contains the single package diagram that is created automatically and cannot be added explicitly.

The root package of a project (Model) is usually referenced as the **default** package. The package diagram of this package is called the default diagram. This diagram is created and opened just after the modeling project creation.

By default, all properties of the package diagram, both visual and semantical, are preserved in the `default.txvpck` diagram file. You can enable split package diagram persistence, which requires turning the default setting off. To do this, right-click the project in the Model Navigator, choose **Properties**, and make sure the **Store package properties in package diagram files** option is not checked. With this option off, only diagram-specific information (visual information, such as layout) is retained in the `default.txvpck` diagram file, while settings that you treat as package properties (semantical information, such as descriptions and custom properties) are moved from the `default.txvpck` file into the `default.txaPackage` file. This allows you to track your package changes using version control.

Related Concepts

Containment Metamodel
Package and logical class diagrams

Related Procedures

Working with a Package

Together Diagram Overview

Each modeling project contains a set of diagrams that are graphical representations of parts of the model. Diagrams contain graphical elements (nodes connected by paths) that represent model elements.

Each diagram belongs to a certain diagram type (for example, UML 2.0 Class Diagram). The diagram type defines the typical contents of the diagram (the kind of elements that are usually placed on this diagram) and the notation used to represent the model elements. For example, a Class in a UML 2.0 project can be added to the Class Diagram and to the Composite Structure Diagram and will have different representations there. Each diagram has the specific Palette and context menu that allow you to create the model elements specific to this diagram type. These tools can be customized.

Diagrams exist within the context of a project. You have to create or open a project before creating a new diagram.

The set of available diagram types depends on the type of project. For example, in a BPMN project, the only available diagram is a BPMN diagram. In a UML 2.0 project you have a set of standard UML diagrams defined in UML2.0 specification. Along with the design diagrams that are explicitly created by the user, Together models have the so-called Package diagrams. These diagrams have the ClassDiagram type, but they are generated automatically for each package and show its contents.

Some diagrams are source-generating. These are: class diagrams and sequence and collaboration diagrams. The contents of such diagrams are synchronized with the source code. Click a class or interface symbol on the diagram to open the source code in the editor. Class and interface source code opens the respective class (or interface) in a special tab in the editor, marked with the source class name. If the class is read-only, the tab is also marked with the lock icon. Selecting a member within a class or interface symbol automatically navigates to the appropriate line of the source code in the editor.

Related Concepts

Diagram Format

Related Procedures

Creating a Diagram

Related Reference

Tool Palette

Diagram Format

The diagrams created with Together are stored in XML-based files with the extension `*.txv<diagram_type>`. For example, the file `<name>.txvcls` corresponds to a class diagram. Design elements are stored either in the package files (`default.txaPackage`) or in separate XML-based format files with the extension `.txa<element_type>`, depending on your choice when creating a project in the **New Project** wizard.

The XML-based diagram format is common for the entire product line of Borland Together modeling products (Borland Together ControlCenter, Borland Together Edition for Microsoft Visual Studio .NET, Borland Enterprise Studio for Java, Borland Together Architect, and Borland Together Designer 2005), which makes the diagrams compatible across the product line. You can copy and reuse diagrams created in the different products.

The legacy text diagram format (`df` diagram files) used in TogetherControlCenter6.2 and previous versions is not supported now. As such, the UML diagram files created with the text-based format should be converted to an XML-based format, using the **Import Together Project** wizard.

Related Concepts

Together Interoperability and Migration

Related Procedures

Interoperability and Migration

Containment Metamodel

Together handles the logical and physical containment of design elements as follows:

- Design elements are created as children of packages.
- All elements shown on diagrams are shortcuts (or references) to actual model elements, therefore when you create a new element on a diagram, Together creates this element in the package and adds its shortcut to the diagram.
- Clipboard actions operate with references, if the source and target containers of the action are diagrams.
- You can optionally create design elements in separate files (standalone design elements) or in one file (filemates).
- You can optionally split package diagram persistence so that diagram-specific property settings (visual information, such as layout) are retained in the `default.txvpck` diagram file, while settings that you treat as package properties (semantical information, such as descriptions and custom properties) are moved from the `default.txvpck` file into the `default.txaPackage` file. This allows you to track your package changes using version control.

Related Concepts

Model Shortcut Overview

Model Element Overview

Each model in a modeling project is a set of entities that are instances of metaclasses of the metamodel chosen for the project. These instances are the Model Elements.

Each model element has a set of properties and a notation defined for its metaclass. For example, when you create a UML 2.0 project, every element created in this project instantiates a metaclass from the UML 2.0 metamodel (that is, each actor on a use case diagram in a UML 2.0 project is an instance of usecases/Actor, and each component is an instance of components/Component).

The model elements that have the graphical notation and that can be explicitly placed on diagrams are nodes and links.

In Together, model elements of the same metaclass may be either design or source code ones depending on the container project type. The model language (design, Java, C++ and IDL) may affect the set and allowed values of element properties. It also defines the model element storage; for example, an element of uml20/classes/class metaclass may be stored in the `*.txa*` file if it is design one or in `*.java, *.h, *.cpp`, and similar files when it is source code.

Related Concepts

Together Diagram Overview
Model Shortcut Overview
Containment Metamodel

Related Procedures

Populating Together Diagrams

Related Reference

Tool Palette

Model Shortcut Overview

A **shortcut** is a representation of a model element placed on a diagram. One can create multiple shortcuts to the same element on different model diagrams. The modifications of the element itself can be made from any diagram containing its shortcut and are propagated to all its shortcuts. The modifications of shortcut view properties made from any diagram do not affect the representations of this element on other diagrams. A shortcut can be removed from a diagram without removing the element from the model.

You can create shortcuts to the elements within the same project. To create a shortcut to an element from another workspace project, add this project to the Model Path of the current project.

The small special symbol appears over a node to indicate a shortcut. For the package diagrams, it appears only if this node belongs to a different namespace or package.

Select a shortcut on your diagram and choose **Select in Model Navigator** on the context menu to navigate to the source element in the Model Navigator.

Related Procedures

Creating a Shortcut
Establishing cross-project references

Roundtrip Engineering Overview

One of the main Together features is the simultaneous roundtrip engineering, which is the ability to immediately synchronize diagrams with their source code.

Roundtrip engineering is the combination of:

- Reverse engineering (drawing models from code)
- Forward engineering (generating code from visual models)

Simultaneous roundtrip engineering means that when you change a code-generating diagram, Together immediately updates the corresponding source code, and when you change the code, Together updates the visual model. This way diagrams are always synchronized with the source code that implements them. You can customize forward and reverse engineering and/or source code formatting. This feature only applies to diagram types that generate source code: class, sequence, and collaboration diagrams.

Together supports source code forward and reverse engineering with the following languages:

- Java 5
- Java 6 (syntax only, not new libraries or technologies)
- C++ (GNU and MS dialects)
- CORBA IDL 2.6

Refer to the Language Support section for details of the supported features and limitations.

Tip: To set up Java 5 support under Unix/Linux platform, see Getting Started with Eclipse and J2SE 5.0 topic in the Getting Started section of the Java Development User Guide.

Related Concepts

Language Support

Related Procedures

Opening a Diagram Element in the Source Code Editor

Related Reference

LiveSource Rules

Language Support

Together supports Java, C++ (GNU and MS dialects), and CORBA IDL. 2.6. Most of the Together features work for Java. Support for other languages is more limited. The limitations stem from the lack of object orientation for some languages and the inapplicability of some of the features to different languages.

Basic functionality	provides parsing of the syntactical constructs that map directly to UML objects (classes, interfaces, methods, and so on). As of this writing, Together offers basic functionality for Java, C++, and CORBA IDL.
Deep Parsing	functionality that handles syntactical constructs within the method bodies, initialization of variables, and so on. For example, deep parsing enables Together to generate sequence diagrams from methods, perform audits and metrics.

The table below provides summary information on the features for the supported languages and brief notes about language-specific properties.

Feature	Java	C++	CORBA IDL
Basic functionality	yes	yes	yes
Deep parsing	yes	yes	n/a
Textual templates	yes	yes	yes
Properties	yes	no	no
Syntax highlight	yes	yes	yes
Formatter	yes	yes	no
Metrics	Full set	Limited set	no
Audits	Full set	Limited set	no
Documentation generation	yes	yes	yes
IDE functionality (Refactoring)	via JDT	via CDT	no

You can find detailed language-specific information in the Reference.

Related Concepts

Roundtrip Engineering Overview

Related Reference

C++ Projects
IDL Language-Specific Information

Generating Source Code Based on Model

Together enables you to generate source code based on a language-neutral design project.

About source code generation

You can generate source code from the Class Diagrams of your UML 1.4 or 2.0 design project.

Name mapping

You can force Together to generate different names for your model elements in the source code. This feature is especially useful, if your model names are not English. You can use names in other languages on your diagrams, but keep names in Latin alphabet in your code. Name mapping is supported for Java target projects only.

If you enable this feature, the file `codegen_java_map.xml` is created in the model support folder of the source design project. You can edit it with any XML or text editor. This file contains a mapping table, where each entry (model element) has two names: one for the source design project (attribute `name`), and another one for the destination implementation project (attribute `alias`).

Related Concepts

Roundtrip Engineering Overview

Related Procedures

Generating Source Code from Design Project

Model Hyperlinking Overview

You can create hyperlinks from diagrams or model elements to other system artifacts and browse directly to them.

Why use hyperlinking?

Use hyperlinks for the following purposes:

- Link diagrams that are generalities or overviews to specifics and details.
- Link diagrams or elements to external documentation.

Create a hyperlink from an existing diagram or one of its elements to any other diagram or model element, or create a new diagram that will be hyperlinked to the current element.

You can also create hyperlinks from your diagrams to external documents such as files or URLs.

Hyperlink types

You can create hyperlinks to:

- An existing diagram or diagram element from any project in the workspace.
- A resource in the workspace.
- An external document (file or URL)

Browse-through sequence

Use case diagrams typically represent the context of a system and system requirements. Usually, you begin at a high level and specify the main use cases of the system. Next, you determine the main system use cases at a more granular level. As an example, a "Conduct Business" use case can have another level of detail that includes use cases such as "Enter Customers" and "Enter Sales". Once you have achieved the desired level of granularity, it is useful to have a convenient method of expanding or contracting the use cases to grasp the scope and relationships of the system's use case views.

The hyperlinking feature of Together allows you to create browse-through sequences comprised of any number of use cases or any other diagrams. By browsing the hyperlink sequence, you can follow the relationships between the use case diagrams.

Together does not confine hyperlinking to such sequences, however. You can use hyperlinking to link diagrams and elements based on your requirements. For example, you can create a hierarchical browse-through sequence of use case diagrams, creating hyperlinks within the diagrams that follow a specific actor through all use cases that reference the actor.

Related Procedures

Hyperlinking Diagrams
Creating a Browse-Through Sequence of Diagrams

Model Annotation Overview

The tools Palette for UML diagram elements displays note and note link buttons for all UML diagrams. Use these elements to place annotation nodes and their links on the diagram.

Notes can be free floating or you can draw a note link to some other element to show that a note pertains specifically to it.

You can attach a note link to another link.

The text of notes linked to class diagram elements does not appear in the source code.

Related Procedures

Annotating a Diagram

Together Capabilities Activation

Together provides many capabilities in areas such as BIRT, development, DSL development activities, and modeling. You can specify the capabilities that should be enabled. This simplifies the Together user interface and helps improve results and productivity.

For example, if the only modeling capability needed is UML 2.0 modeling, you can enable UML 2.0 modeling and disable other types of modeling, so that menus, menu items, and wizards for all modeling capabilities except for UML 2.0 modeling are not available.

If you need a menu, menu item, or wizard that is not available, make sure that the appropriate capability is enabled in the **Advanced Capabilities Settings** dialog.

Together Capability Categories

The Together capabilities are grouped in the following categories:

- **DSL Development**
- **Model to Model Transformations**
- **Model to Text Transformation**
- **Model Workflow**
- **Modeling**
- **Modeling Tools**
- **Reporting**
- **UML Modeling**

Together Modeling Capabilities

The following list shows the Together capabilities and their default status (enabled or disabled) in Together Modeling (classic Together modeling).

The **DSL Development** category contains the following Together capabilities:

- **Diagram Definition** (disabled by default)
- **Domain Modeling** (disabled by default)
- **DSL Project** (disabled by default)

The **Model to Model Transformations** category contains the following Together capabilities:

- **Operational Mapping Language (QVT)** (enabled by default)
- **Operational QVT Debugging** (enabled by default)

The **Model to Text Transformations** category contains the following Together capabilities:

- **Template Authoring** (enabled by default)
- **Template Exploring** (enabled by default)
- **Template Exploring (Legacy)** (disabled by default)

The **Model Workflow** category contains the following Together capabilities:

- **Workflow Definition** (enabled by default)
- **Workflow Execution** (enabled by default)

The **Modeling** category contains the following Together capabilities:

- **Business Process Modeling** (enabled by default)
- **C++ Modeling** (enabled by default)
- **Data Modeling** (enabled by default)
- **Documentation** (enabled by default)
- **IDL Modeling** (enabled by default)
- **Java Modeling** (enabled by default)
- **Manage element persistence** (disabled by default)
- **MDL and MDX Imports** (enabled by default)
- **Model QA** (enabled by default)
- **Modeling profiles** (enabled by default)
- **Patterns** (enabled by default)
- **Source code QA** (enabled by default)
- **Together Project Import** (enabled by default)
- **Together QVT** (enabled by default)
- **UML 1.4** (enabled by default)
- **UML 2.0** (enabled by default)
- **XMI Import/Export** (enabled by default)
- **XSL** (enabled by default)

The **Model Tools** category contains the following Together capabilities:

- **Hyperlinks and Requirement Traces (Early Access)** (enabled by default)
- **Model Refactoring (Early Access)** (enabled by default)

The **Team** category contains the following Together capabilities:

- **CVS Support for Modeling** (disabled by default)

The **Reporting** category contains the following Together capabilities:

- **Model Reporting** (enabled by default)
- **Report Definition** (enabled by default)

The **UML Modeling** category (disabled by default) contains the following Together capabilities:

- **UML2 Diagramming** (disabled by default)
- **UML2 Model Development** (disabled by default)

Related Procedures

Activating Together Capabilities

Together Interoperability and Migration

Together supports the possibility to exchange models created in the different products of Together product line and in the other modeling tools.

Interoperability

Interoperability is supported in the following ways:

- Together opens projects created with the other tools of Together product line. So doing, Together considers and processes the project roots and diagram formats.
- For the models created in the other tools, use the various types of import and export, such as XMI, MDL or MDX.
- Also, transformations enable the users to exchange model information. Refer to the concept section "Model Transformation Support" (listed under Related Information below) for details.

Migration from the legacyTogether products

Having created a number of projects in TCC/TA 1.x and in the other Together products, the user might want to migrate these projects to the new version, preserving the useful features of the legacy projects.

Reusing legacy projects

Reusing the legacy `*.tpr, *.tpx` and `*.jpx` projects in Together is an important interoperability goal. However, this task faces a number of problems related to the differences between the products, which are summarized in the following table:

Legacy Projects	New Projects
Support multiple modeling roots.	All modeling information is stored in a single folder.
It is possible to specify package prefix for a root.	The notion of package prefix does not exist.
Support two diagram formats (DF format and TXV format)	Supports TXV format only.
Old containment metamodel stores diagrams and model elements together.	New containment metamodel separates the diagram information from the model elements.

Together resolves these problems by means of a new migration tool implemented as **Import Together Project Wizard**, which takes a legacy project as input and produces one or more Together Eclipse projects.

The resulting projects meet the following common requirements:

- Folder structure of the resulting project is created considering the package prefixes if any.
- All diagrams are converted from the old containment metamodel to the new containment metamodel. If a model root contains diagrams in `DF` format, these diagrams are converted to `TXV` format. If a model root contains diagrams in both `DF` and `TXV` formats, then only `TXV` diagrams are considered.
- Optionally, you can create the resulting project with the design elements stored in different files. In this case, the model elements of the source project are converted to standalone design elements.
- UML 2.0 projects created in Together Designer/Developer 2005 are converted taking into account the changes in UML 2.0 specification support (converting State Machine and Activity diagrams).

Reusing artifacts

Due to different platform, Together does not support complete migration of the legacy custom artifacts. You can reuse legacy documentation templates, but custom audits, metrics, patterns and diagrams, created in TCC/TA 1.x, are not compatible with Together.

Instead Together provides the possibility to create your own artifacts and extensions using its functionality.

Modules	You can create modules using Eclipse API and Together EMF API. Use Eclipse API for IDE—related parts, and Together EMF API for working with models.
Java-based patterns	These patterns are not supported in Together. However, Together supports creating design and source code patterns and templates. See the related concepts.
Audits and Metrics	You cannot reuse audits and metrics from TCC, but can create source code audits and metrics of your own. Refer to the subsection “Using API for creating your Audits and Metrics” in the Audit and Metric Sample Project topic (listed under Related Information below).
Custom diagrams and custom properties	Use profiles to customize diagrams and define custom properties. Refer to the Profile Definition Project overview (listed under Related Information below).

Related Concepts

Model Transformation Support
Diagram Format
Profile Definition Project
Patterns and Templates

Related Procedures

Reusing documentation templates from TCC/TA 1.x
Importing Legacy Projects

Related Reference

Audit and Metric Sample Project
Import Together Project Wizard
Audit and Metric Sample Project

Modeling Overview

The topics in this section provide an overview of modeling, and information on UML diagrams and supported technologies.

In This Section

Together Modeling

Together modeling provides a concise, easily communicated picture of a system that is to be created and deployed. While you can work directly in a model itself, there are separate views available to make it easier to view, understand, and manipulate the model.

“A model is a simplified representation of a system or phenomenon, as in the sciences or economics, with any hypotheses required to describe the system or explain the phenomenon, often mathematically.” (Dictionary.com Unabridged (v 1.1). Random House, Inc. 29 Aug. 2007)

A model is a means to communicate complex ideas, and can be simplified by ignoring certain details. A model can be a plan or an “as-is” view. A model is an abstraction that makes it easier to communicate about complex things.

To maximize the benefits provided by Together, you should be familiar with the following concepts:

- Benefits of modeling
- Common modeling problems
- Together Models, views and users
- Model transformations

Benefits of modeling

Proper analysis requires that everyone has a common, complete, and accurate understanding of the problem, while proper design requires that everyone has a common, complete, and accurate understanding of the solution. These requirements apply throughout the project lifecycle. Any communication breakdown in these areas can result in project delay, increased costs, and failure.

Modeling can be of benefit in all areas of software and process development. The benefits of modeling include the following:

- Streamline and improve requirements analysis and validation
- Build agile applications using UML models that leverage industry-proven design methods, component- and service-oriented architectures, and model-driven development practices
- Provide an environment that is a step beyond high-level programming languages, allowing developers to move away from lower-level complexity and write higher-quality code
- Minimize the time and effort needed to define, understand, and create systems, applications, and processes
- Reduce the risk of project delays and failures
- Reduce costs by allowing reuse for multiple projects
- Enhance communication across the project lifecycle and among distributed teams
- Help communicate and integrate business and development requirements

Modeling allows you to shift the view of programs, applications, and processes from the system or machine view to the problem domain view, and also to automate the translation from human to system or machine language.

Analysts use modeling to help agree on and document the task that needs to be accomplished. Modeling helps avoid ambiguity and provides a “big picture” view of the task.

Architects use modeling to perform the following tasks:

- Design the overall application architecture. This includes mapping the design to the requirements, developing and communicating the design, documenting the design and architecture, ensuring the quality of the design, and providing architectural views.

- Leverage reuse of frameworks, libraries, and design patterns.
- Engineer a system that can be produced within business constraints (time and cost), staffing constraints (knowledge, skills sets, and headcount), process constraints (quality and predictability), and technology constraints (tools and existing systems).
- Engineer a system that can withstand change during the system's lifecycle, including new features, changing requirements, and the updating of IT infrastructure.

Developers use modeling to perform the following tasks:

- Explore implementation options
- Provide strong refactoring support to improve code design without affecting functionality
- Improve code reviews by adhering to best practices and ensuring that code can be efficiently maintained, modified, and reused
- Document the system, including understanding large code bases, dependencies, and interfaces

A large set of requirements can be difficult to understand as a whole. Models provide constructs to help organize ideas, a common language for all team members, visualizations to help clarify complex ideas, and standards to facilitate precise communication and provide focus for key abstractions.

Model-driven development can be used for many tasks, including the following:

- Repetitive and redundant source code
- Framework implementation
- Design patterns
- Configuration files
- Build labels
- Deployment variations (for example, Development, Test, and Production)
- Automation of deployment
- Targeting of complex distributed environments
- Test plans and test scripts
- Test automation
- Test verification and validation

Common problems in analysis and design

Several problems are frequently encountered during the analysis and design process. These include the following:

- It is difficult to communicate consistently and without ambiguity with text-only analysis and design.
- The notation and meaning of ad-hoc diagrams may not be apparent to all team members.
- Models cannot be connected to requirements with drawing tools.
- Traceability needs to be established and models must meet business requirements. These are manual processes with drawing tools, and as such, these processes are time-consuming and error-prone.
- Models need to be persisted for collaborative teamwork, with change management practices that are consistent with source code change control practices.

- The ability to create documentation from models is necessary. Using models to generate documentation includes not only diagrams, but also the properties that are associated with the model elements.
- Model consistency must be ensured. A well-designed modeling tool helps users learn the model rules and enforces these rules.
- An accurate representation is needed. Models can be used to automate design and implementation and can feed other parts of the development lifecycle.

Models and views

There are several views available for a Together model. The model views should not be confused with the model itself. You can create and modify a model in any view and all of your changes are propagated to the model itself and all model views. A model view provides a window to the model itself, with the changes made to the model itself. All changes to the model, regardless of the view in which the changes were made, are synchronized in all of the model views. Users can choose the view that is best suited to their role and needs, and users can use the view of their choice to view the latest iteration of a model and make any necessary changes to the model. For example, if you are viewing a model diagram and make any changes, your changes are saved in the model itself when you save your changes. If you or another user subsequently view the model in the tree view, your saved changes are displayed in this view.

Any view, such as the diagram view or the tree view, is simply a view of a model, and is not a model in and of itself. A view provides a representation of the entire scope of the underlying model. You can use any view to view and modify a model. Together allows you to build a single unified model that can be viewed and modified in different views and validated with OCL audits.

Domain Model Diagrams and UML Class Diagrams

A domain model diagram provides a view of the parts or terms that comprise a project. A domain model defines a system or process in unambiguous terms so that everyone on a team can understand, define, and refine a system or process. A domain model defines the scope and provides a model on which to develop and refine a system or process.

A UML class diagram provides a view of the classes, class attributes, and class relationships for a system. A UML class diagram can be used for purposes ranging from defining requirements to creating a detailed design. A UML class diagram is used to define classes, interfaces, relationships, and inheritances.

Model Transformations

You can use model transformations to convert a model that conforms to a particular metamodel to a model that conforms to another metamodel. You can specify multiple source models and multiple target models. A model transformation is itself a model, in that the model transformation conforms to a metamodel. A model transformation can produce new artifacts or modify existing artifacts.

Related Concepts

UML Modeling Overview
Model Transformation Support

Related Procedures

Creating a Model-To-Model Transformation

UML Modeling Overview

Effective modeling with Together simplifies the development stage of your **project**. Smooth integration to Together provides developers with easy transition from **models** to source code.

The primary objective of modeling is to organize and visualize the structure and components of software intensive systems. **Models** visually represent requirements, subsystems, logical and physical elements, and structural and behavioral patterns.

While contemporary software practices stress the importance of developing models, Together extends the benefits inherent to modeling by fully synchronizing **diagrams** and source code.

In This Section

Supported UML Specifications
Describes supported UML specifications.

UML 2.0 Diagrams
Gives a general notion of UML 2.0 diagrams supported by Together.

UML 1.4 Diagrams
Gives a general notion of UML 1.4 diagrams supported by Together.

Supported UML Specifications

The Object Management Group’s Unified Modeling Language (UML) is a graphical language for visualizing, specifying, constructing, and documenting the artifacts of distributed object systems.

Together supports UML to help you specify, visualize, and document models of your software systems, including their structure and design.

Refer to UML documentation for the detailed information about UML semantics and notation. The *UML (version): Superstructure* document defines the user level constructs required for UML. It is complemented by the *UML (version): Infrastructure* document which defines the foundational language constructs required for UML. The two complementary specifications constitute a complete specification for the UML modeling language.

UML 1.4 and UML 2.x

The set of available diagrams depends on your project type.

Design projects and Java projects support both UML 1.4 and UML 2.x specifications. C++ and IDL projects support only UML 2.x.

Note: Because several of the features that UML 2.0 provides (such as documentation functionality and metrics) are not yet implemented in UML 2.1, the UML 2.1 capabilities are disabled by default. To turn them on, select **Window ▸ Preferences... ▸ General ▸ Capabilities**. Click **Advanced...** and select the **UML2 Diagramming** node under the **UML Modeling** feature.

The version of UML is selected when a project is created. It cannot be changed later.

UML In Color

“**UML In Color**” is an optional profile to support the **modeling in color** methodology. Color modeling makes it possible to analyze a problem domain and easily spot certain classes during analysis. Together supports the use of the four main groups of the color-modeling stereotypes:

- Role
- Moment-interval, Mi-detail
- Party, Place, Thing
- Description

When applying a stereotype to one of the diagram elements listed above, the view of the associated diagram element changes on the diagram. The stereotype field displays directly above the name field for the element, and the color of the element depends on the stereotype chosen. For each of these stereotypes you can choose a specific color to make your model more understandable at a glance. Note that the other stereotypes do not have associated colors.

See also *"Java Modeling in Color with UML: Enterprise Components and Process" by Coad, Lefebvre and De Luca*.

Related Concepts

UML Modeling Overview

UML 2.0 Diagrams

Together provides support for the most frequently needed diagrams and notations defined by the UML 2.0.

In This Section

UML 2.0 Activity Diagram Definition

Definition

The activity diagram enables you to model the system behavior, including the sequence and conditions of execution of the actions. Actions are the basic units of the system behavior.

An Activity diagram enables you to group and ungroup actions. If an action can be broken into a sequence of other actions, you can create an activity to represent them.

In UML 2.0, activities consist of actions. An action represents a single step within an activity, that is, one that is not further decomposed within the activity. An activity represents a behavior which is composed of individual elements that are actions. An action is an executable activity node that is the fundamental unit of executable functionality in an activity, as opposed to control and data flow among actions. The execution of an action represents some transformation or processing in the modeled system, be it a computer system or otherwise.

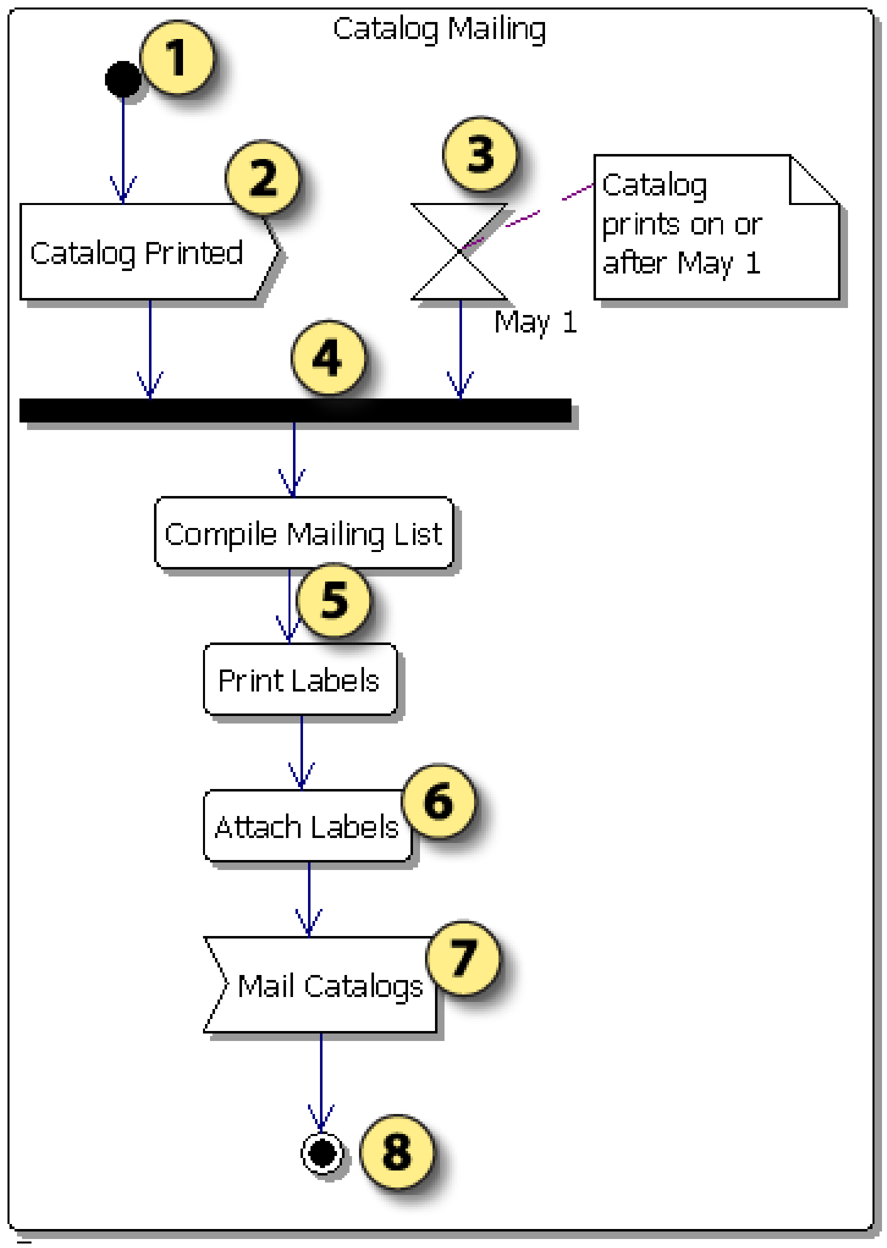
The semantics of activities is based on token flow. By flow, we mean that the execution of one node affects and is affected by the execution of other nodes, and such dependencies are represented by edges in the activity diagram. Data and control flows are different in UML 2.0.

A control flow may have multiple sources (it joins several concurrent actions) or it may have multiple targets (it forks into several concurrent actions).

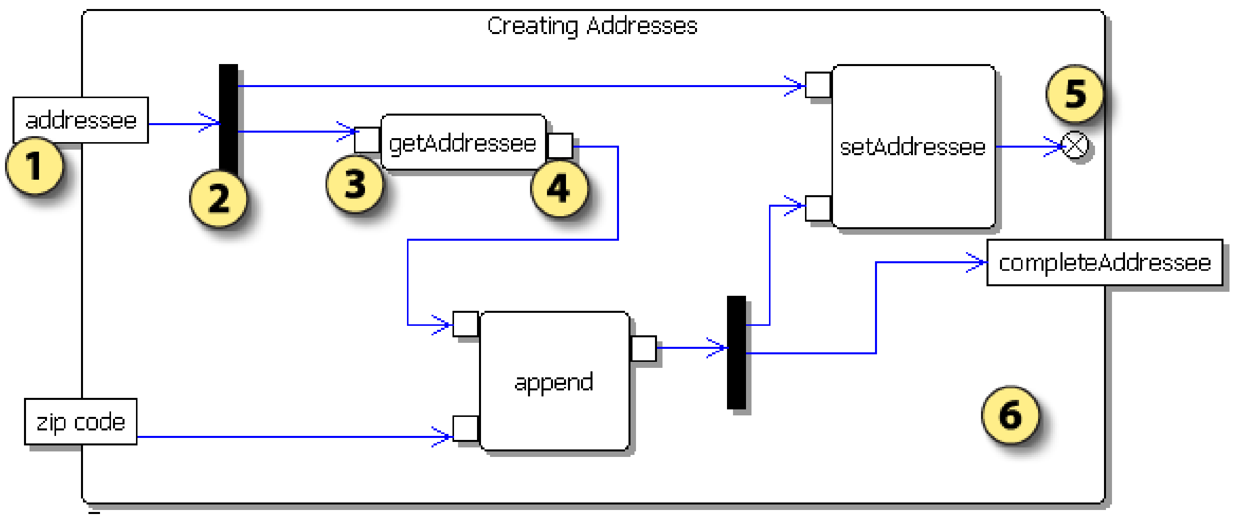
Each flow within an activity can have its own termination, which is denoted by a flow final node. The flow final node means that a certain flow within an activity is complete. Note that the flow final may not have any outgoing links.

Using decisions and merges, you can manage multiple outgoing and incoming control flows.

Sample Diagram



1	Initial State	5	Control Flow
2	Send Signal Action	6	Action
3	Accept Time Event Action	7	Accept Event Action
4	Join	8	Final State



1	Activity Parameter	4	Output Pin
2	Fork	5	Flow Final
3	Input Pin	6	Activity

Related Procedures

UML 2.0 Activity Diagrams Procedures

Related Reference

UML 2.0 Activity Diagrams

UML 2.0 Class Diagram Definition

UML 2.0 Class diagrams feature the same capabilities as the UML 1.4 diagrams.

The UML 2.0 class diagrams offer new diagram elements such as ports, provided and required interfaces.

According to the UML 2.0 specification, an instance specification can instantiate one or more classifiers. You can use classes, interfaces, or components as a classifier.

Interfaces

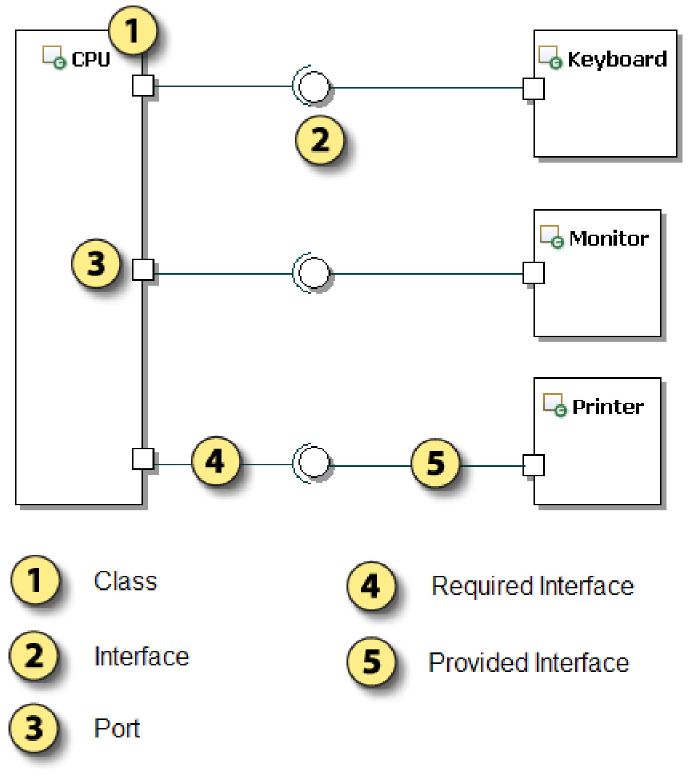
A class implements an interface via the same generalization/implementation link, as in UML 1.4 class diagram. In addition to the implementation interfaces, there are provided and required interfaces. Interfaces can be represented in class diagrams as rectangles or as circles. For the sake of clarity of your diagrams, you can show or conceal interfaces.

UML 2.0 class diagram supports the ball-and socket notation for the provided and required interfaces. Choose Show as circle command on the context menu of the interface to obtain a lollipop between the client class and the supplier interface.

Tip: Applying a provided interface link between a class and an interface creates a regular generalization/ implementation link. To create provided interface, apply the provided interface link to a port on the client class.

Sample Diagram

The figure below shows a class diagram with some of the new elements.



Special Note for the LiveSource Projects

Using UML 2.0 class diagrams in the LiveSource projects is limited with certain restrictions, and are similar to UML 1.4 class diagrams.

Related Procedures

UML 2.0 Class Diagrams Procedures

Related Reference

UML 2.0 Class Diagrams

UML 2.0 Use Case Diagram Definition

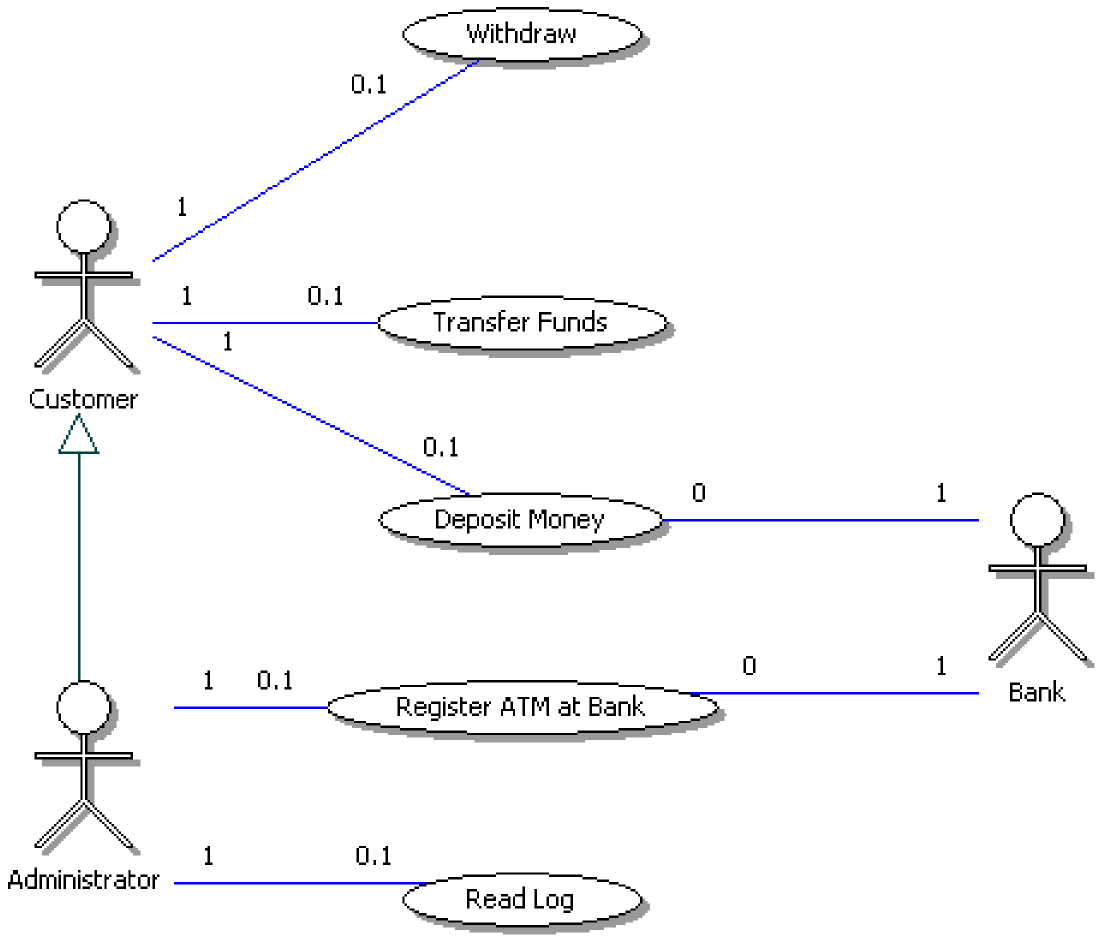
Diagram courtesy of the Unified Modeling Language: *Superstructure version 2.0. August 2003. p. 536*.

Definition

Use case diagram describes required usages of a system, or what a system is supposed to do. The key concepts that take part in a use case diagram are actors, use cases, and subjects. A subject represents a system under consideration with which the actors and other subjects interact. The required behavior of the subject is described by the use cases.

Sample Diagram

The following diagram shows an example of actors and use cases for an ATM system.



Related Procedures

UML 2.0 Use Case Diagrams Procedures

Related Reference

UML 2.0 Use Case Diagrams

UML 2.0 Component Diagram Definition

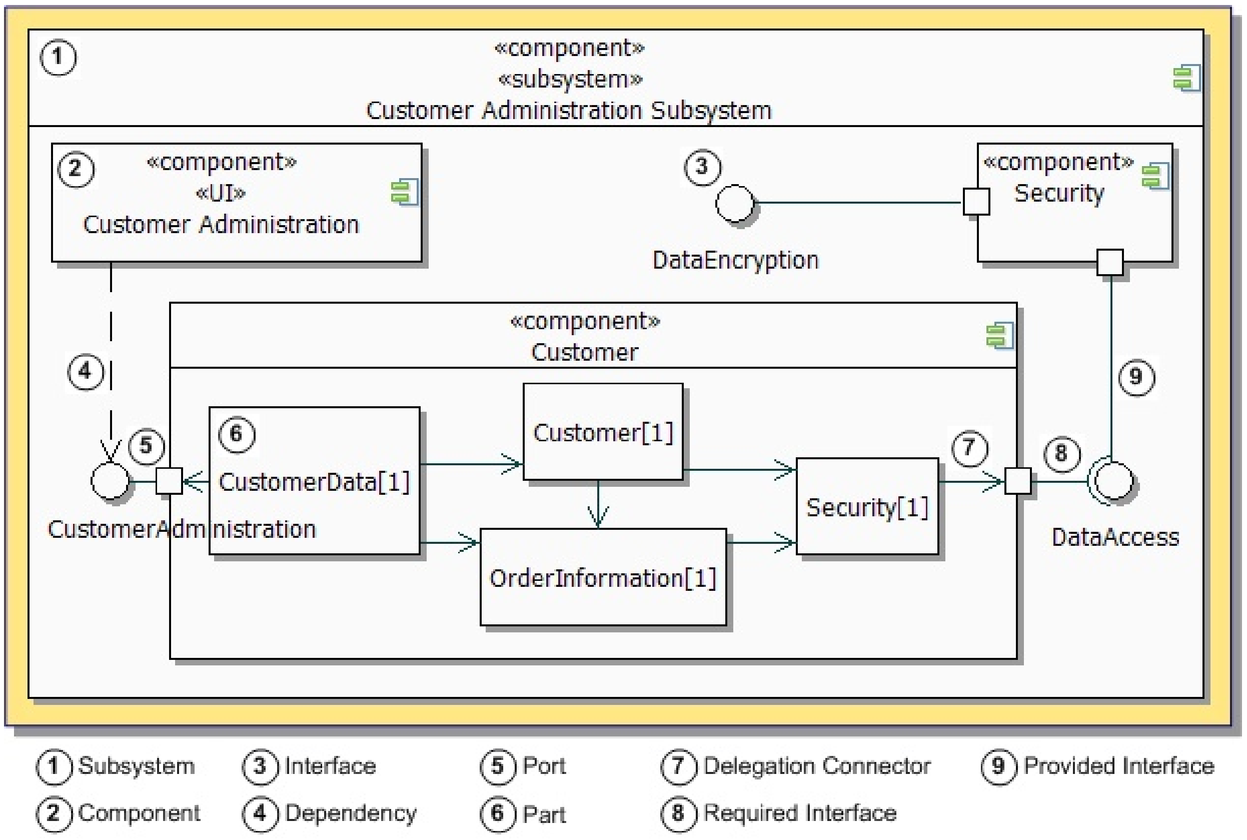
This topic describes the UML 2.0 Component Diagram.

Definition

According to the UML 2.0 specification, a component diagram can contain instance specifications. An instance specification can be defined by one or more classifiers. You can use classes, interfaces, or components as a classifiers. You can instantiate a classifier using the **Object Inspector Properties Window**, or the in-place editor.

Sample Diagram

The following component diagram specifies a set of constructs that can be used to define software systems of arbitrary size and complexity.



Related Procedures

UML 2.0 Component Diagrams Procedures

Related Reference

UML 2.0 Component Diagrams

UML 2.0 Composite Structure Diagram Definition

Diagram courtesy of the Unified Modeling Language: *Superstructure version 2.0. August 2003. p. 178*.

Definition

Composite structure diagrams depict the internal structure of a classifier, including its interaction points to the other parts of the system. It shows the configuration of parts that jointly perform the behavior of the containing classifier.

A collaboration describes a structure of collaborating parts (roles). A collaboration is attached to an operation or a classifier through a Collaboration Use.

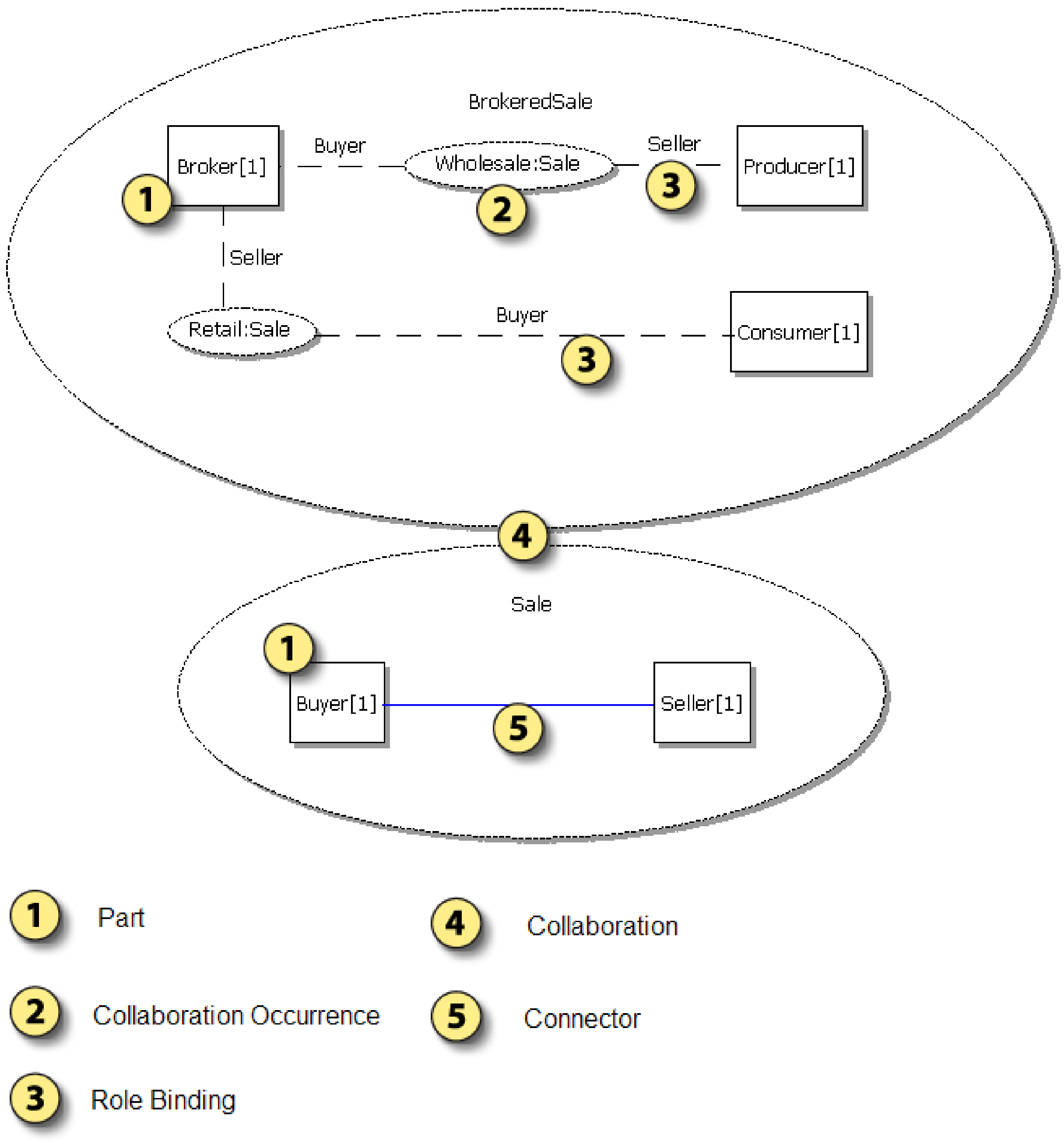
Classes and collaborations in the Composite Structure diagram can have internal structure and ports. Internal structure is represented by a set of interconnected parts (roles) within the containing class or collaboration. Participants of a collaboration or a class are linked by the connectors.

A port can appear either on a contained part, or on the boundary of the class.

The contained parts can be included by reference. Referenced parts are represented by the dotted rectangles.

Composite Structure diagram supports the ball-and-socket notation for the provided and required interfaces. Interfaces can be shown or hidden in the diagram as needed.

Sample Diagram



Related Procedures

UML 2.0 Composite Structure Diagrams Procedures

Related Reference

UML 2.0 Composite Structure Diagrams

UML 2.0 Deployment Diagram Definition

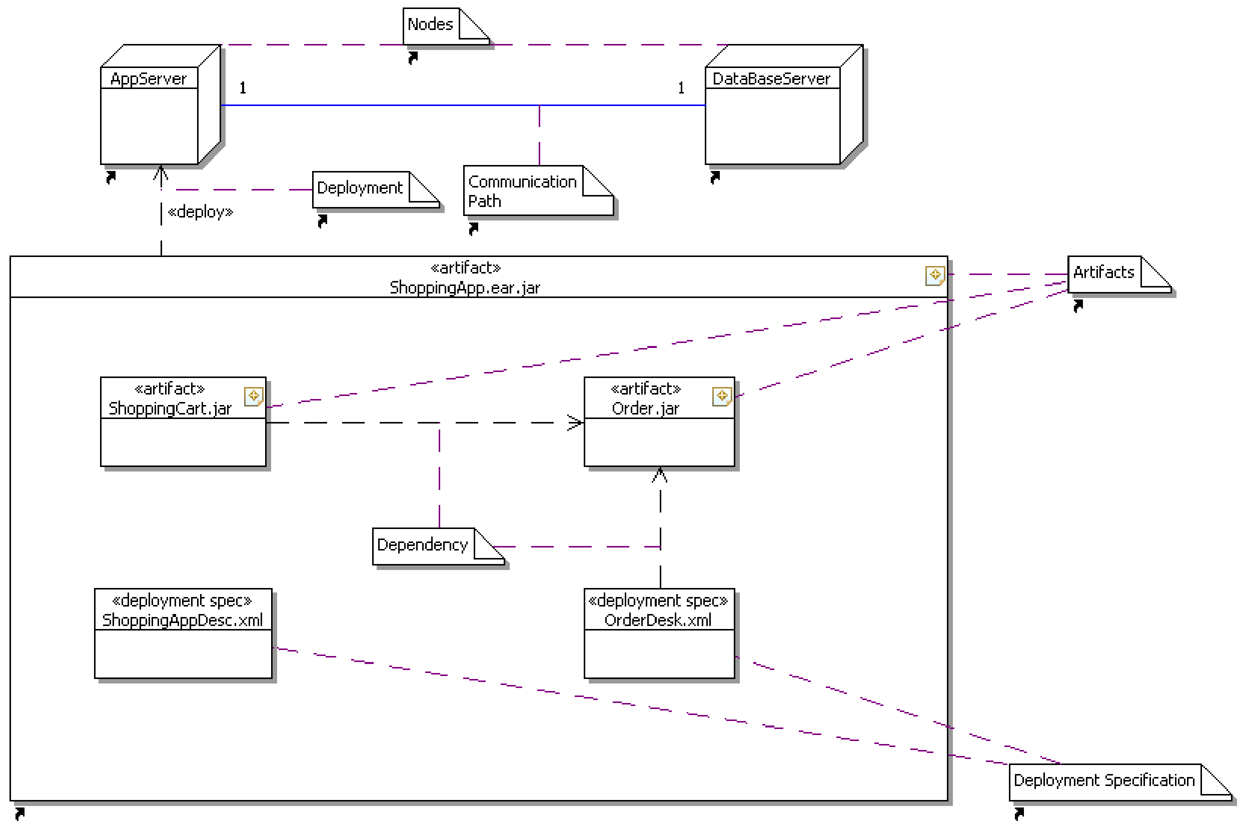
This topic describes the UML 2.0 Deployment Diagram.

Definition

The deployment diagram specifies a set of constructs that can be used to define the execution architecture of systems that represent the assignment of software artifacts to nodes. Nodes are connected through communication paths to create network systems of arbitrary complexity. Nodes are typically defined in a nested manner, and represent either hardware devices or software execution environments. Artifacts represent concrete elements in the physical world that are the result of a development process.

Diagram courtesy of the Unified Modeling Language: *Superstructure version 2.0. August 2003. pp. 207, 212.*

Sample Diagram



Related Procedures

UML 2.0 Deployment Diagrams Procedures

Related Reference

UML 2.0 Deployment Diagrams

UML 2.0 State Machine Diagram Definition

States are the basic units of the state machines. In UML 2.0 states can have substates.

Execution of the diagram begins with the Initial node and finishes with Final or Terminate node or nodes. Refer to UML 2.0 Specification for more information about these elements.

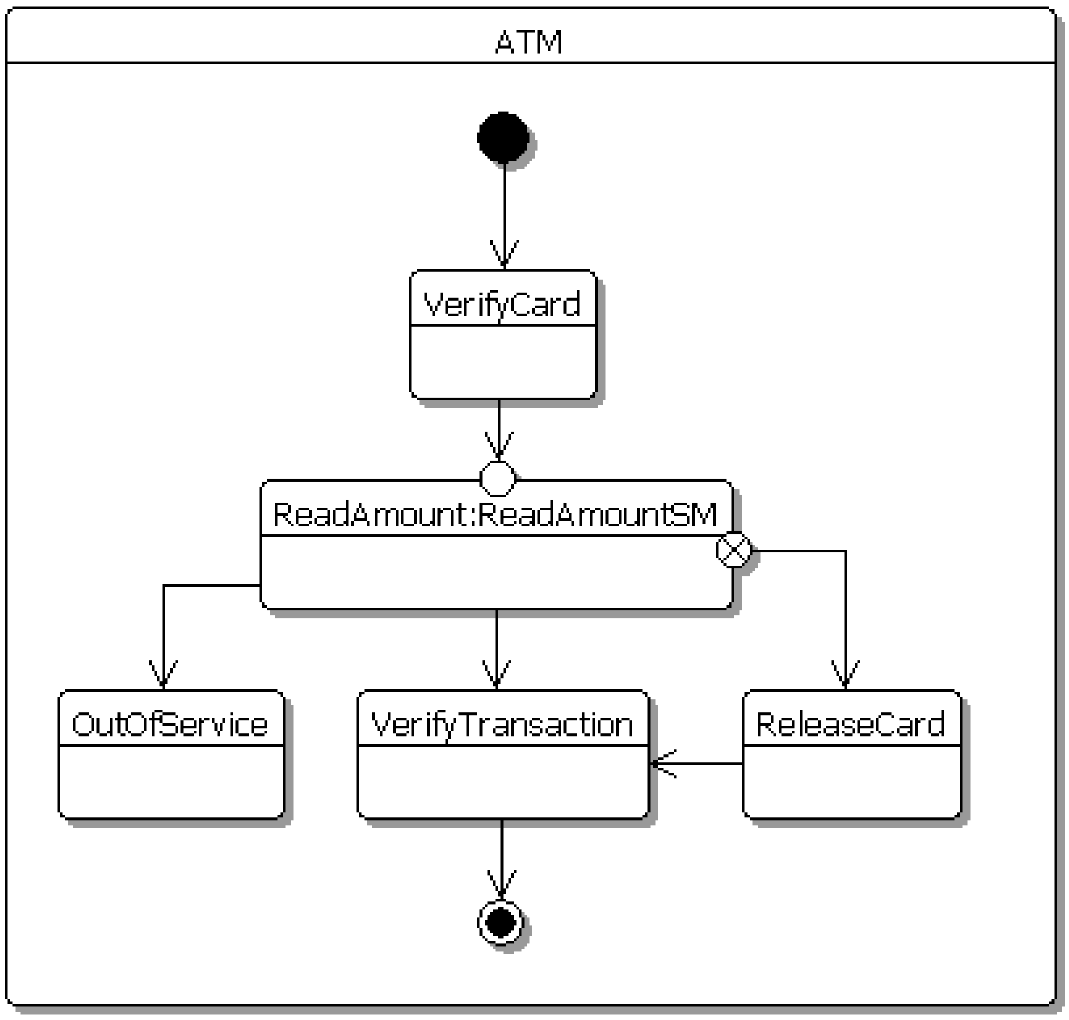
Definition

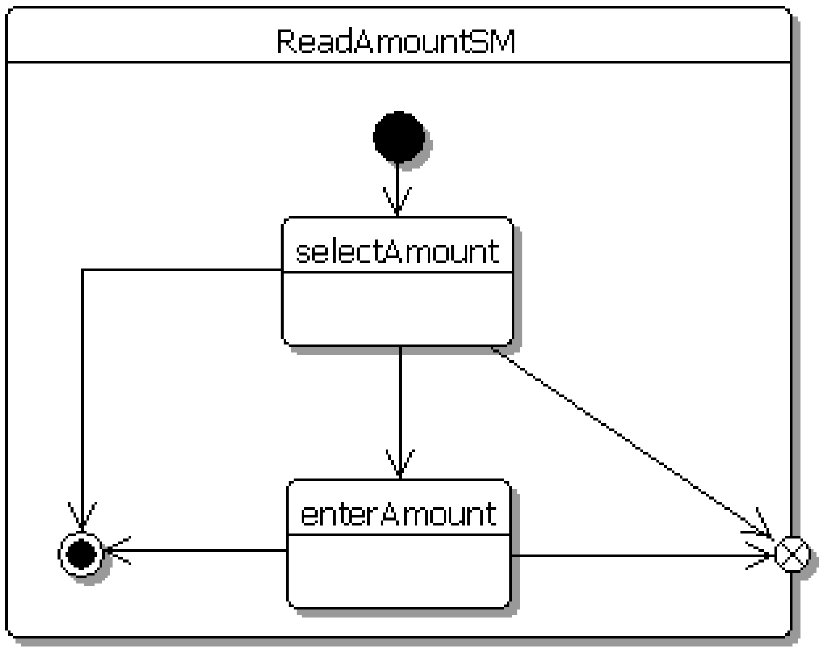
State Machine diagrams describe the logic behavior of the system, a part of the system, or the usage protocol of it.

On these diagrams you show the possible states of the objects and the transitions that cause a change in state.

State Machine diagrams in UML 2.0 are different in many aspects compared to Statechart diagrams in UML 1.4.

Sample Diagram





Related Procedures

UML 2.0 State Machine Diagrams Procedures

Related Reference

UML 2.0 State Machine Diagrams

Interaction (Sequence and Communication) Diagrams

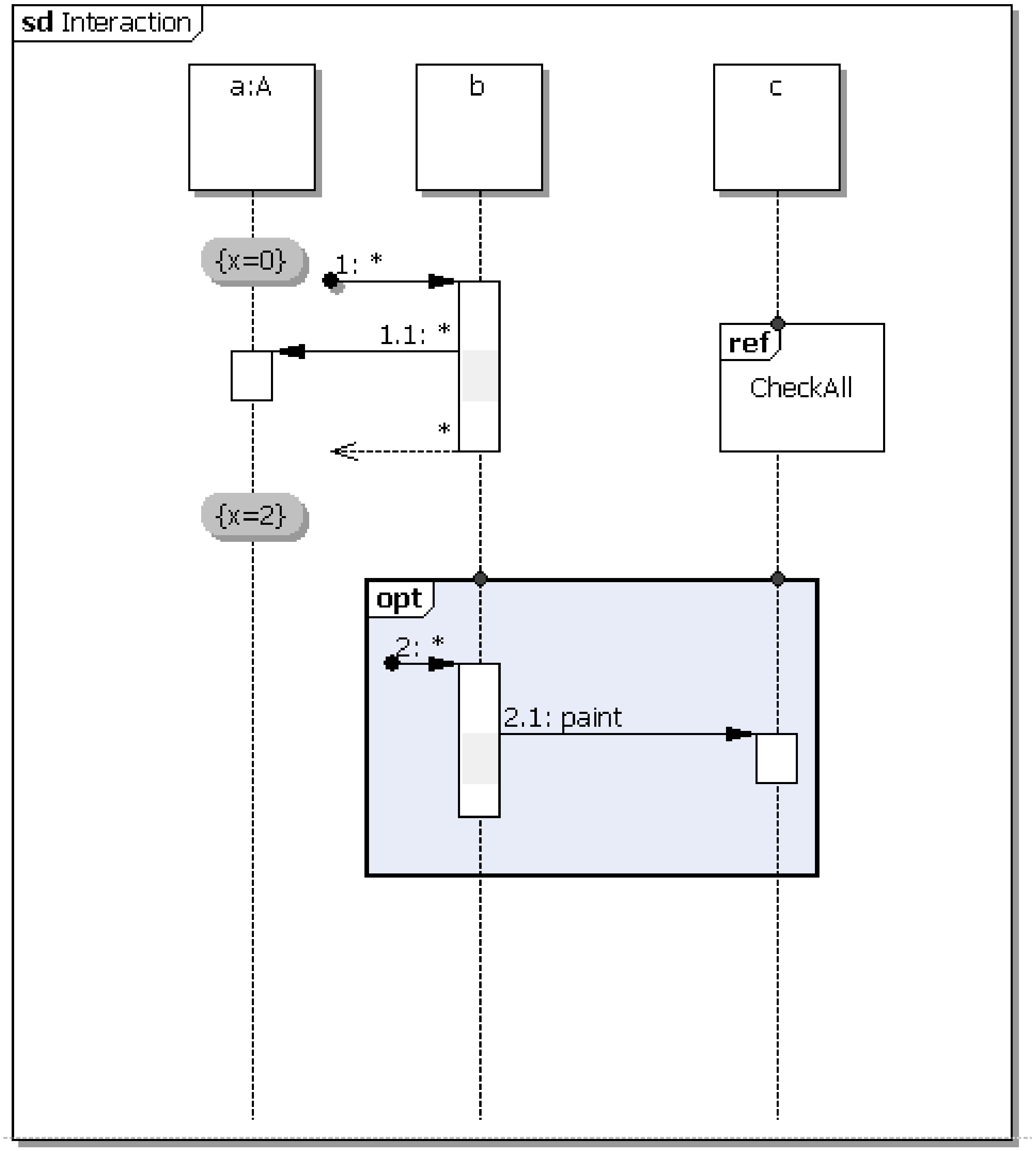
Using Together you can create interactions for the detailed description and analysis of inter-process communications. Interactions can be visually represented in your Together projects by means of the two most common interaction diagrams: Sequence and Communication. On the other hand, interactions can exist in projects without visual representation.

Whenever an interaction diagram is created, the corresponding interaction entity is added to the project. Interactions are represented as nodes in the Model Navigator and can be placed inside classes and use cases.

You can view an interaction in two ways: as a sequence diagram, or as a communication diagram. An interaction diagram contains a reference to the underlying interaction.

Unlike UML 1.4, it is not possible to switch a diagram that already exists from sequence to communication and vice versa. However, it is possible to create a sequence diagram and a communication diagram based on the same interaction.

Sequence diagram can contain shortcuts to other diagram elements. However, you cannot create shortcuts to the elements nested in Interactions.



Related Procedures

[UML 2.0 Interaction Diagrams Procedures]

Related Reference

[UML 2.0 Interaction Diagrams]

UML 1.4 Diagrams

Together provides support for the most frequently needed diagrams and notations defined by the UML 1.4.

In This Section

UML 1.4 Class Diagram Definition

Using Together, you can create language-neutral class diagrams in design projects, or language-specific class diagrams in implementation projects. For implementation projects, all diagram elements are immediately synchronized with the source code.

Definition

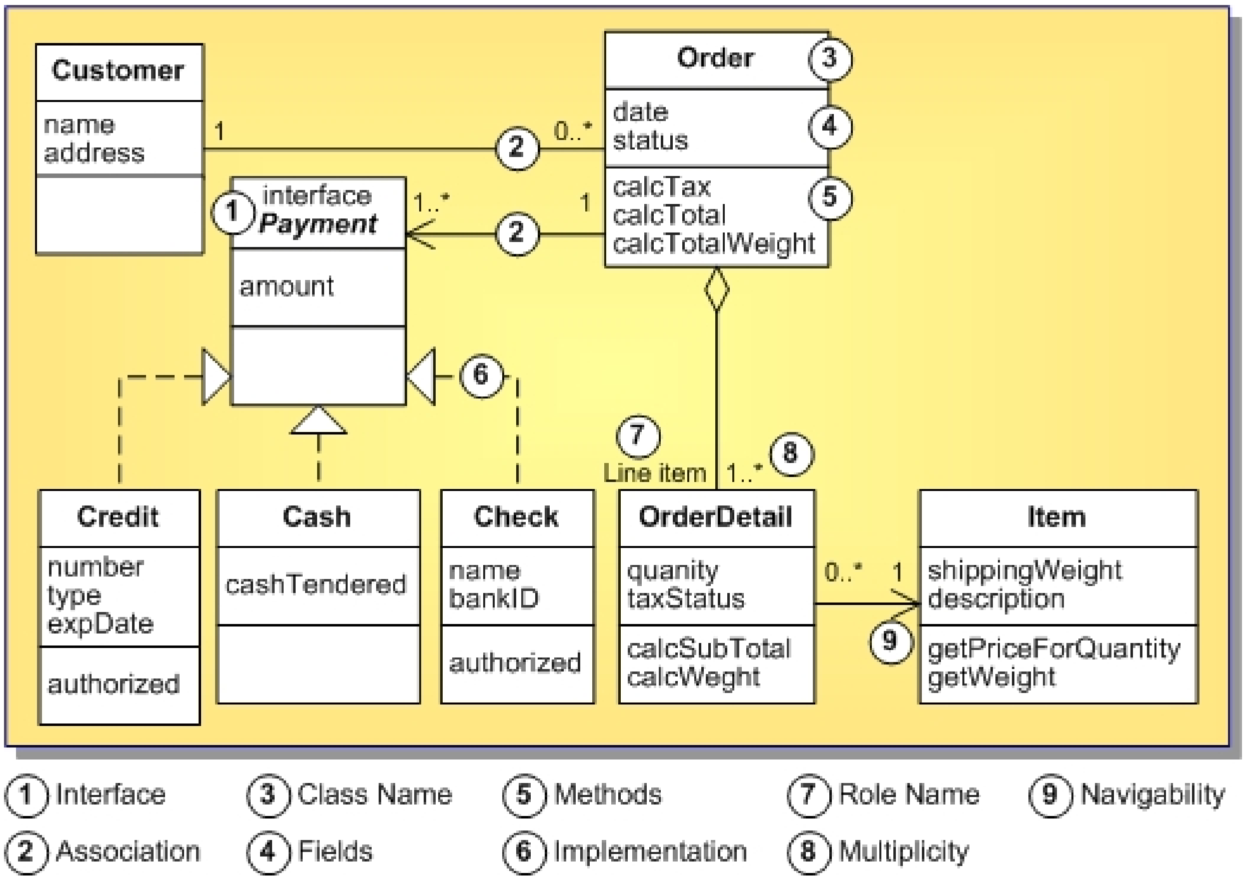
A class diagram provides an overview of a system by showing its classes and the relationships among them. Class diagrams are static: they display what interacts but not what happens during the interaction.

UML class notation is a rectangle divided into three parts: class name, fields, and methods. Names of abstract classes and interfaces are in italics. Relationships between classes are the connecting links.

In Together, the rectangle is further divided with separate partitions for properties and inner classes.

Sample Diagram

The following class diagram models a customer order from a retail catalog. The central class is the `Order`. Associated with it are the `Customer` making the purchase and the `Payment`. There are three types of payments: `Cash`, `Check`, or `Credit`. The order contains `OrderDetails` (line items), each with its associated Item.



There are three kinds of relationships used in this example:

- Association: For example, an `OrderDetail` is a line item of each `Order`.
- Aggregation: In this diagram, `Order` has a collection of `OrderDetails`.

- Implementation: `Payment` is an interface for `Cash`, `Check`, and `Credit`.

Related Procedures

UML 1.4 Class Diagrams Procedures

Related Reference

UML 1.4 Class Diagrams

Package and logical class diagrams

The two types of class diagrams used in Together are package diagrams and logical class diagrams.

Package diagrams	These diagrams are stored as XML files in the Model folder of the project with the file extension `.txvpck` (for UML 1.4 projects), or `.txvClassDiagram20` (for UML 2.0 projects) Together creates a default package diagram for a project and for each subdirectory under the project root. The default project diagram is named **default**. The default package diagrams have `default.txvpck` and `default.txvClassDiagram20` names respectively.
Logical class diagrams	These diagrams are stored as XML files with the file extension `.txvcls` (for UML 1.4 projects), or `.txvClassDiagram20` (for UML 2.0 projects).

Related Concepts

UML 1.4 Class Diagram Definition
UML 2.0 Class Diagram Definition

UML 1.4 Sequence Diagram Definition

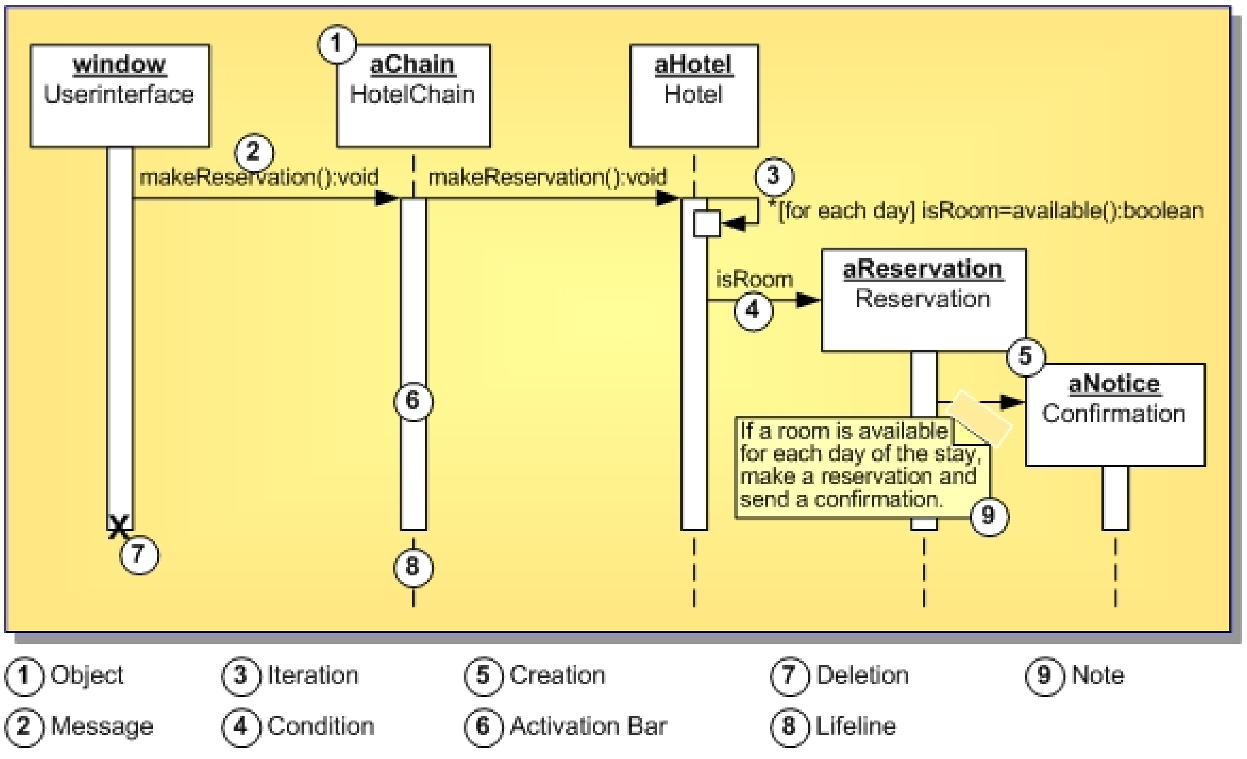
Class diagrams are static model views. In contrast, interaction diagrams are dynamic, describing how objects collaborate.

Definition

A sequence diagram is an interaction diagram that details how operations are carried out: what messages are sent and when. Sequence diagrams are organized according to time. The time progresses as you go down the page. The objects involved in the operation are listed from left to right according to when they take part in the message sequence.

Sample Diagram

Following is a Sequence Diagram for making a hotel reservation. The object initiating the sequence of messages is a Reservation window (the UserInterface).



The `UserInterface` sends a `makeReservation()` message to a `HotelChain`. The `HotelChain` then sends a `makeReservation()` message to a `Hotel`. If the Hotel has available rooms, then it makes a `Reservation` and a `Confirmation`.

Each vertical dotted line is a lifeline, representing the time that an object exists. Each arrow is a message call. An arrow goes from the sender to the top of the activation bar of the message on the receiver's lifeline. The activation bar represents the duration of execution of the message.

In this diagram, the `Hotel` issues a self call to determine if a room is available. If so, then the `Hotel` creates a `Reservation` and a `Confirmation`. The asterisk on the self call means iteration (to make sure there is available room for each day of the stay in the hotel). The expression in square brackets, `[ ]`, is a condition.

The diagram has a clarifying note, which is text inside a dog-eared rectangle. Notes can be included in any kind of UML diagram.

Related Procedures

UML 1.4 Interaction Diagrams Procedures

Related Reference

UML 1.4 Interaction Diagrams

UML 1.4 Collaboration Diagram Definition

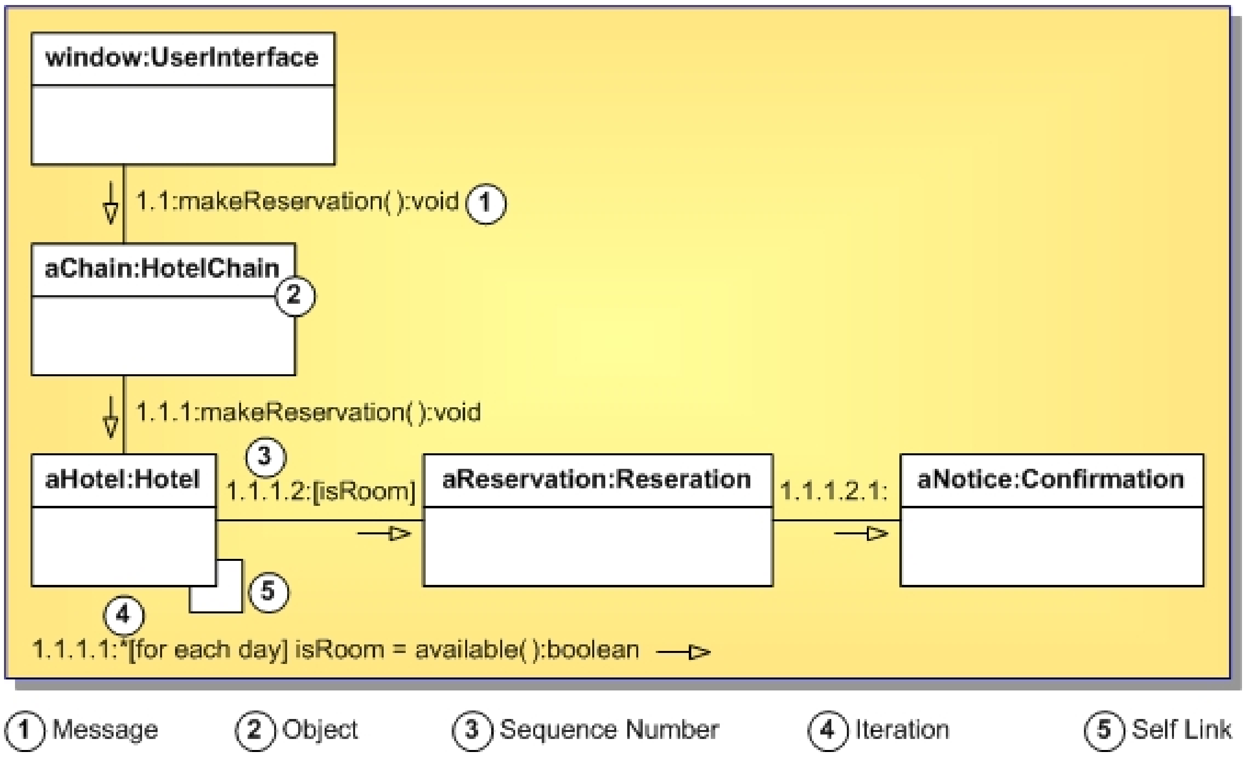
Class diagrams are static model views. In contrast, interaction diagrams are dynamic, describing how objects collaborate.

Definition

Like sequence diagrams, collaboration diagrams are also interaction diagrams. Collaboration diagrams convey the same information as sequence diagrams, but focus on object roles instead of the times that messages are sent.

Sample Diagram

Each message in a collaboration diagram has a sequence number. The top-level message is numbered 1. Messages at the same level (sent during the same call) have the same decimal prefix but suffixes of 1, 2, etc. according to when they occur.



Related Procedures

UML 1.4 Interaction Diagrams Procedures

Related Reference

UML 1.4 Interaction Diagrams

UML 1.4 Use Case Diagram Definition

Use case diagrams are helpful in three areas:

- Determining features (requirements): New use cases often generate new requirements as the system is analyzed and the design takes shape.
- Communicating with clients: Notational simplicity makes use case diagrams a good way for developers to communicate with clients.
- Generating test cases: The collection of scenarios for a use case may suggest a suite of test cases for those scenarios.

Definition

Use Case Diagram describes what a system does from the viewpoint of an external observer. The emphasis is on what a system does rather than how.

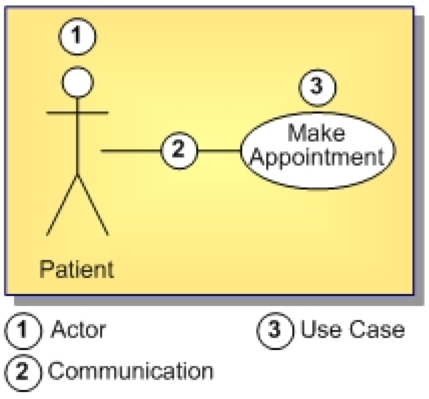
Use Case Diagrams are closely connected to scenarios. A scenario is an example of what happens when someone interacts with the system.

Sample Diagram

Following is a scenario for a medical clinic:

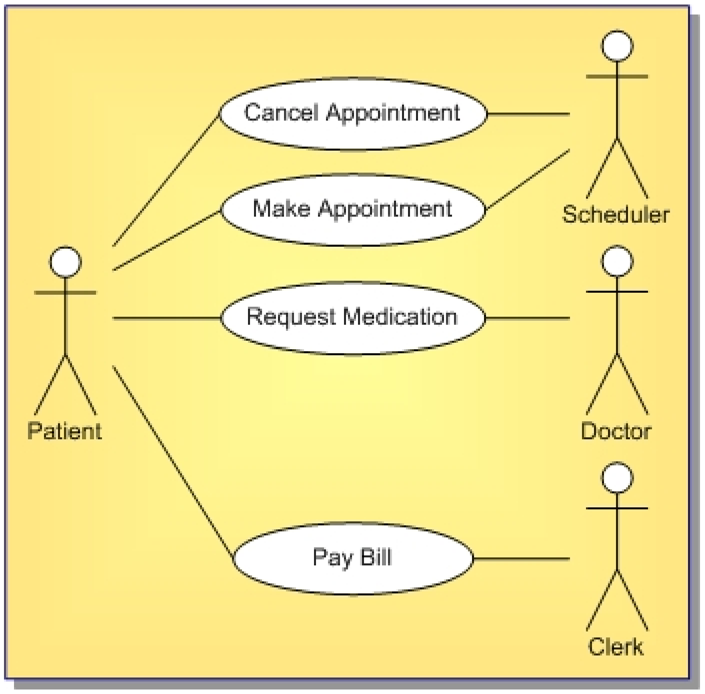
A patient calls the clinic to make an appointment for a yearly checkup. The receptionist finds the nearest empty time slot in the appointment book and schedules the appointment for that time slot.

A use case is a summary of scenarios for a single task or goal. An actor is who or what initiates the events involved in that task. Actors are simply roles that people or objects play. The following diagram is the `Make Appointment` use case for the medical clinic. The actor is a `Patient`. The connection between actor and use case is a communication association (or communication for short).



Actors are stick figures. Use cases are ovals. Communications are lines that link actors to use cases.

A use case diagram is a collection of actors, use cases, and their communications. Following is an example of the use case Make Appointment as part of a diagram with four actors and four use cases. Notice that a single use case can have multiple actors.



Related Procedures

UML 1.4 Use Case Diagrams Procedures

Related Reference

UML 1.4 Use Case Diagrams

UML 1.4 Statechart Diagram Definition

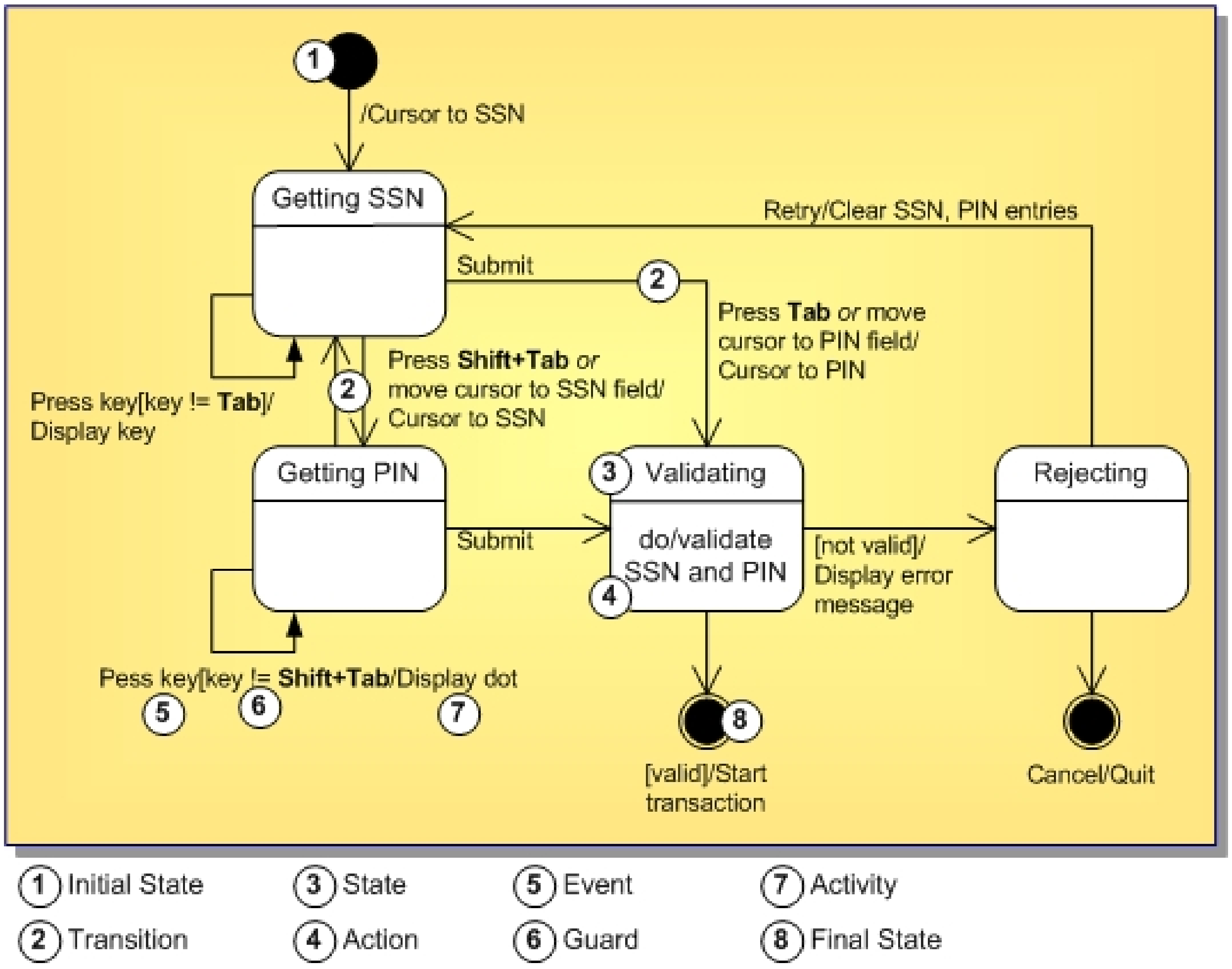
This topic describes the UML 1.4 Statechart Diagram.

Definition

Objects have behaviors and states. The state of an object depends on its current activity or condition. A statechart diagram shows the possible states of the object and the transitions that cause a change in state.

Sample Diagram

The following diagram models the login part of an online banking system. Logging in consists of entering a valid social security number and personal ID number, then submitting the information for validation. Logging in can be factored into four non-overlapping states: `Getting SSN`, `Getting PIN`, `Validating`, and `Rejecting`. Each state provides a complete set of transitions that determines the subsequent state.



1
/Cursor to SSN
Getting SSN
Retry/Clear SSN, PIN entries
Submit
2
Press Tab or move cursor to PIN field/ Cursor to PIN
Press key[key != Tab]/ Display key
2
Press Shift+Tab or move cursor to SSN field/ Cursor to SSN
Getting PIN
3 Validating
Rejecting
Submit
do/validate SSN and PIN
4
[not valid]/ Display error message
Pess key[key != Shift+Tab/Display dot
5
6
7
8
[valid]/Start transaction
Cancel/Quit
1 Initial State
3 State
5 Event
7 Activity
2 Transition
4 Action
6 Guard
8 Final State

States are depicted as rounded rectangles. Transitions are arrows from one state to another. Events or conditions that trigger transitions are written next to the arrows. This diagram has two self-transitions: `Getting SSN` and `Getting PIN`. The initial state (shown as a black circle) is a dummy to start the action. Final states are also dummy states that terminate the action.

The action that occurs as a result of an event or condition is expressed in the form `/action`. While in its Validating state, the object does not wait for an outside event to trigger a transition. Instead, it performs an activity. The result of that activity determines its subsequent state.

Related Procedures

UML 1.4 Statechart Diagrams Procedures

Related Reference

UML 1.4 Statechart Diagrams

UML 1.4 Activity Diagram Definition

This topic describes the UML 1.4 Activity Diagram.

Definition

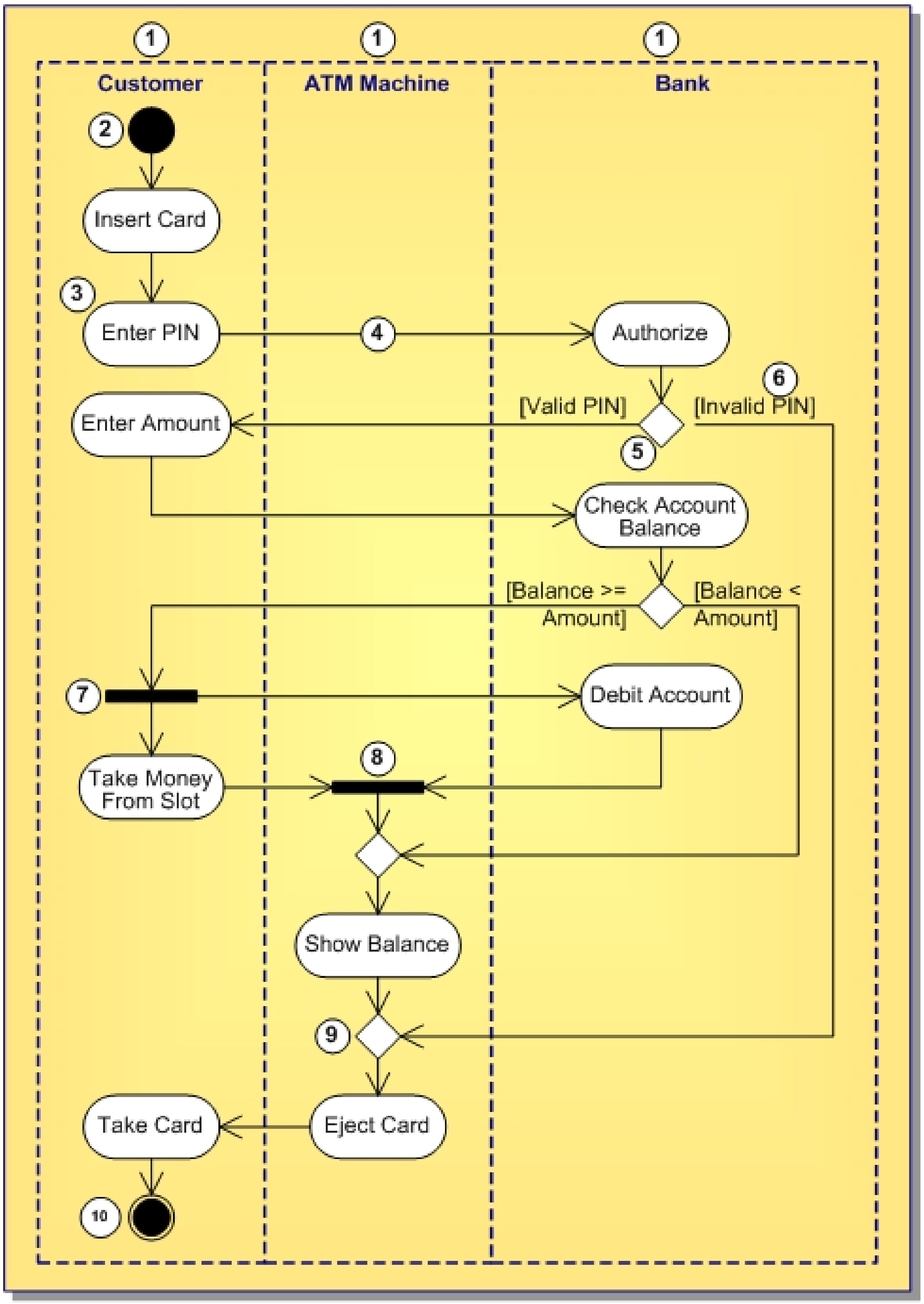
Activity diagrams enable you to model system dynamics. An activity diagram is a flowchart that describes the flow of control from one activity to the next. You can show sequential and/or concurrent steps of a process, model business workflows, model the flow control of an operation, or the flow of an object as it passes though different states at different points in a process. Unlike interaction diagrams (such as sequence and collaboration) that emphasize the flow of control between objects, activity diagrams emphasize the flow of control between activities. Activity diagrams and statechart diagrams are related. While a statechart diagram focuses attention on an object undergoing a process (or on a process as an object), an activity diagram focuses on the flow of activities involved in a single process. The activity diagram shows the how those activities depend on one another.

Activity diagrams can be divided into object swimlanes that determine which object is responsible for an activity. A single transition comes out of each activity, connecting it to the next activity. A transition can branch into two or more mutually exclusive transitions. Guard expressions (inside [ ]) label the transitions coming out of a branch. A branch and its subsequent merge marking the end of the branch appear in the diagram as hollow diamonds. A transition may fork into two or more parallel activities. The fork and the subsequent join of the threads coming out of the fork appear in the diagram as solid bars.

Sample Diagram

The Activity Diagram below uses the following process: "Withdraw money from a bank account through an ATM."

The three involved classes (people, and so on) of the activity are Customer, ATM, and Bank. The process begins at the black start circle at the top and ends at the concentric white/black stop circle at the bottom. The activities are shown as rounded rectangles.



1 Swimlane	3 Activity	5 Branch	7 Fork	9 Merge
2 Start	4 Transisiton	6 Guard Expression	8 Join	10 End

Related Procedures

UML 1.4 Activity Diagrams Procedures

Related Reference

UML 1.4 Activity Diagrams

UML 1.4 Component Diagram Definition

Both component and deployment diagrams depict the physical architecture of a computer-based system. Component diagrams show the dependencies and interactions between software components.

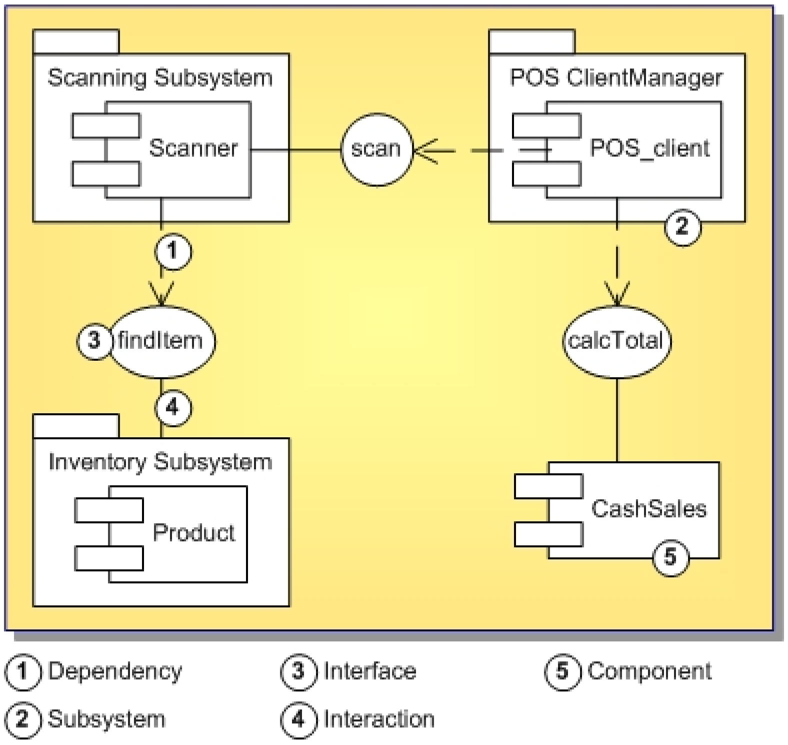
Definition

A component is a container of logical elements and represents things that participate in the execution of a system. Component also uses the services of other components through one of its interfaces.

Components are typically used to visualize logical packages of source code (work product components), binary code (deployment components), or executable files (execution components).

Sample Diagram

Following is a component diagram that shows the dependencies and interactions between software components for a cash register program.



Related Procedures

[UML 1.4 Component Diagrams Procedures]

Related Reference

[UML 1.4 Component Diagrams]

UML 1.4 Deployment Diagram Definition

Both Component and Deployment Diagrams depict the physical architecture of a computer-based system.

Deployment Diagrams are made up of a graph of nodes connected by communication associations to show the physical configuration of the software and hardware.

Components are physical units of packaging in software, including:

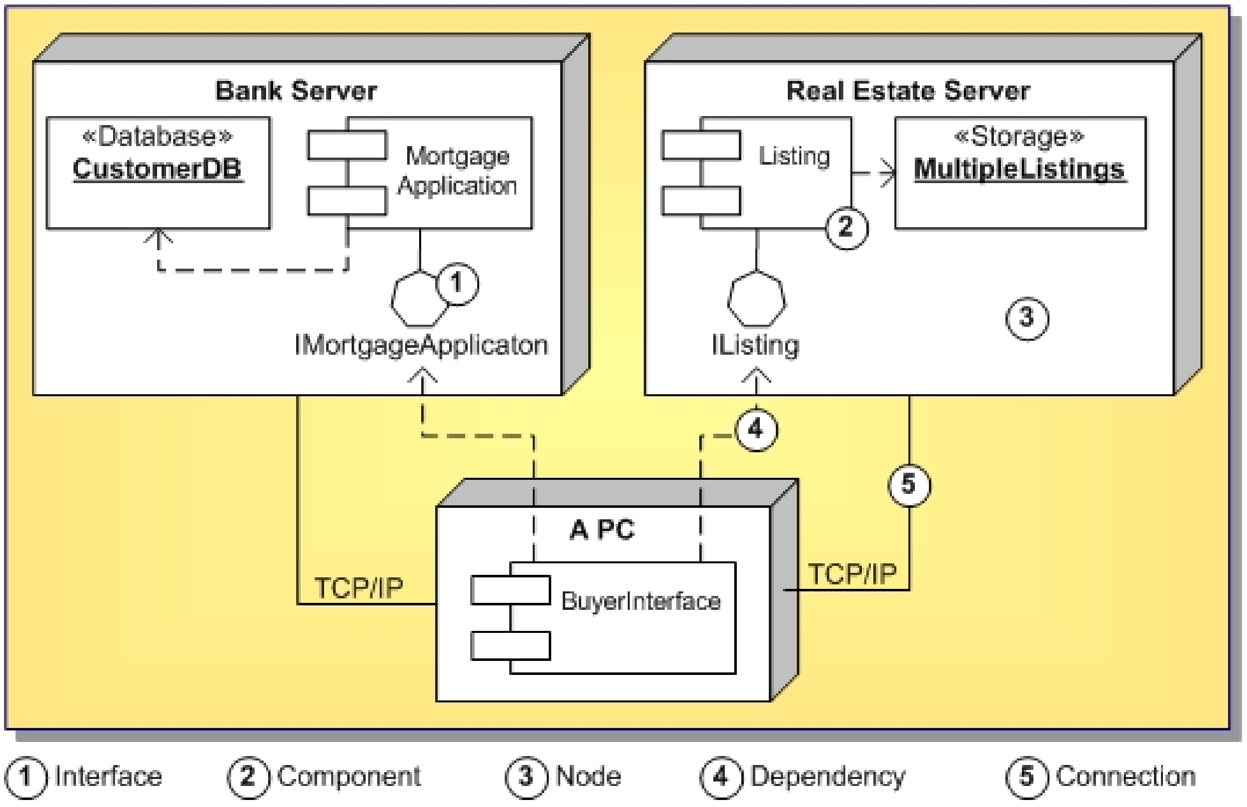
- External libraries
- Operating systems
- Virtual machines

Definition

The physical hardware is made up of nodes. Each component belongs on a node. Components are shown as rectangles with two tabs at the upper left.

Sample Diagram

Following is a Deployment Diagram that shows the relationships of software and hardware components for a real estate transaction.



Related Procedures

UML 1.4 Deployment Diagrams Procedures

Related Reference

UML 1.4 Deployment Diagrams

Business Process Modeling

Overview

Business Process Modeling Notation (BPMN) covers many types of business process modeling with various detail levels and enables you to create end-to-end business processes. After you create a diagram in the BPMN project, you can export the diagram to BPEL and WSDL files. You also can create a BPMN project from imported BPEL and WSDL files.

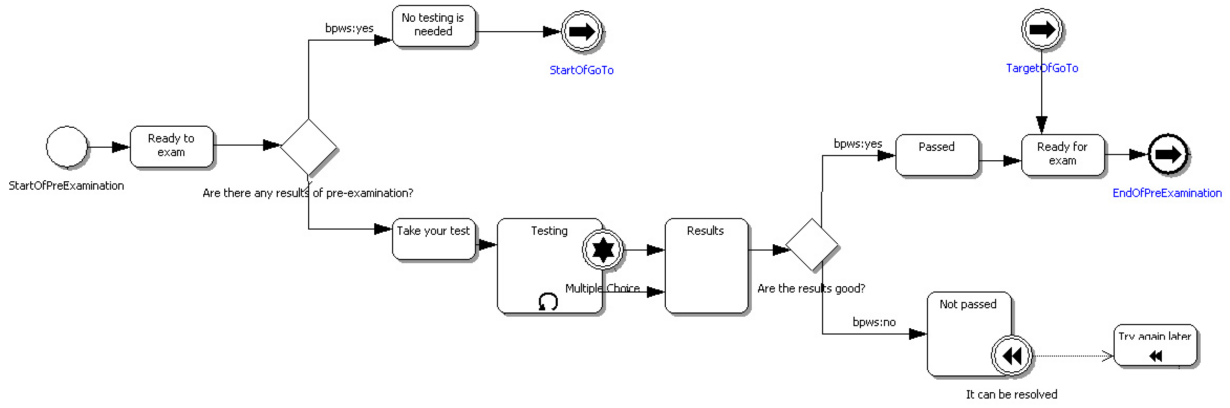
Together enables you to perform a simulated run of the designed business process specifying simulation parameters in the run configuration or using default parameters. During the simulation, Together calculates tasks execution duration, execution cost and other parameters. When simulation is finished you can open a report with statistical data on the selected business process.

By default, a business process modeling project is created with the enabled BPEL Modeling profile. This profile adds properties necessary to create a BPEL file. You also can specify general options of business process modeling (including the default profile).

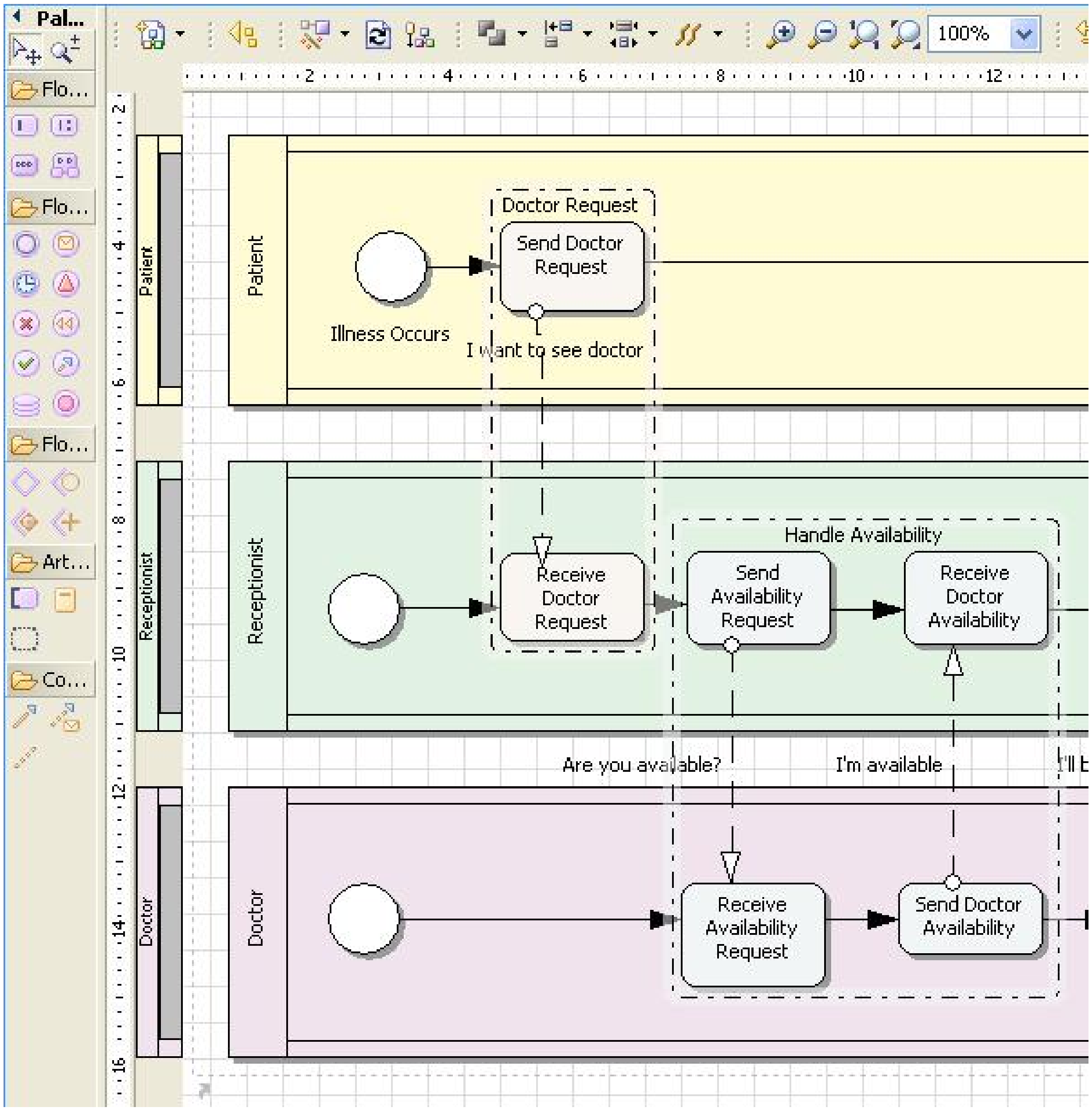
When you create a BPMN diagram, it is created with a default pool. You can use the diagram immediately for designing your process. A business process project can also contain the following elements that are invisible on the diagram but can be seen in the **Model navigator**:

- Message
- Event Detail
- Rule
- Transaction
- Assignment
- Web Service
- Property
- Property Set
- Process
- Participant
- Input Set
- Output Set

The following diagram is an example of the BPMN diagram.



A diagram in the Business Process Modeling project can contain projection bars that mirror pools and lanes from the diagram. The projection bars remain visible when the lanes are too long and the diagram have to be scrolled. You can use the projection bars to select pools or lanes.



The Group element allows you to easily differentiate between sections of a BPMN diagram. You can easily divide a BPMN diagram into logical parts using the Group element. A Group permanently keeps track of content, resizes on element move, colors elements with selected color, etc.

Reusing BPMN Projects Created in Together 2006

To reuse BPMN projects created in Together 2006 for Eclipse, use **BPMN Project from Together 2006 Business Process Modeling** project.

Note: You can open BPMN projects created in Together 2006 for Eclipse but they open as read-only and not accessible via API.

Optional Install

Since Together 2008 Release 3, the Business Process Modeling feature set is not required to be installed. When not present, the corresponding parts of the user interface and functionality are not available. Refer to the installation instructions in the Release Notes document for additional info about the product installation process.

Note: It is not recommended to omit installing the Business Process Modeling feature set if there are existing Business Process Modeling projects in the workspaces you plan to reuse or import.

Related Procedures

Performing Business Process Simulation

Data Modeling

Topics in this section provide a brief overview of data modeling in Together.

In This Section

Data Modeling Overview

Together provides a complete data modeling solution. With Together you can perform the following tasks

- Design logical models
- Design physical models
- Import DDL/SQL script to existing project
- Export logical model to physical model
- Export physical model to DDL/SQL script
- Import Data Model from Database to physical model
- Import logical models from Together Designer 2005

Related Concepts

Logical and Physical Data Models

Related Procedures

Data Modeling Procedures

Related Reference

Data Modeling Reference

Logical and Physical Data Models

A data model, which represents the business data, consists of both the logical and physical design. A logical model is developed prior to the physical model and allows you to define how the information to be stored in the database is organized. Thus, a logical model can be regarded as a blueprint that clearly defines data structures and relationships between them.

The physical design addresses the technical implementation of the logical data model and shows how the information is stored in a particular database. The physical model is bound to the target database server.

Because data modeling is a complicated process, Together enables you to separate the development of the logical and physical models.

- **Logical models** are designed in UML 2.0 modeling projects with the help of ER Logical Diagram Profile. The concept of entities and relationships in logical data modeling maps to the concept of classes and associations in the UML 2.0 class diagram. When you enable this profile for a project, Together provides a set of ER Logical Elements in the Palette, which you can use to create your logical model.
- **Physical models** are designed in Data Modeling projects.

Related Concepts

UML Profiles
Together Project Overview

Related Procedures

Data Modeling Procedures

Model Transformation Support

Together provides a complete set of Model Driven Architecture (MDA) capabilities based on the specially developed Together Model Transformation Framework (TMF). The framework implements some of the most important concepts underlying the Meta Object Facility (MOF) 2.0 Queries/Views/Transformations (QVT) specification and is based on the Eclipse Modeling Framework (EMF).

In the scope of QVT, model transformation relates to MOF models. Together Model Transformation Framework relates to EMF and Together models. Together models are accessible via an EMF API implemented as a set of lightweight wrappers placed around Together model elements. The framework provides an imperative QVT language for defining mappings between models and a transformation engine for interpreting the mapping definitions and queries.

In Together, you can create, run and debug transformations within the project environment. When your transformation is ready, you can apply it to models or model elements. The compiled transformation is deployed within the Eclipse environment as a standard Java plug-in that you can share with users in your team. Currently, Together supports the following transformation types:

- Model-To-Model transformation. Transforms a Together or EMF model into another Together or EMF model. Model-To-Model transformations produce the target model and an auxiliary trace file with detailed information about every transformation step performed. The target model opens in the corresponding model editor, the trace file opens in the **Trace** view.
- Model-To-Text transformation. Transforms a Together or EMF model into an arbitrary text output using java.
- XSL/OCL transformation. Transforms a Together or EMF model into an arbitrary text output using an XSL/OCL transformation script. The XSL/OCL script uses the OCL language. XSL uses the XPath language.
- Composite transformation is the Ant-based MDA transformation, which allows you to automatically (using Ant tasks) apply multiple MDA transformations to the specified models, and in the specified order. For Model-To-Model transformations, you can create transformation chains, where the output model of the preceding transformation is passed (via Ant properties) to the input of the next transformation in the chain.

By using QVT Model-To-Model transformations, you can transform your Computation Independent Models (CIMs) into Platform Independent Models (PIMs), and then to Platform Specific Models (PSMs). By using Model-To-Text and XSL/OCL transformations, you can generate code from your PSMs.

The framework comes with a set of tools that helps you write, run, and debug transformations. For Model-To-Model transformations, the **QVT Editor** provides basic QVT editing features (including code sensitive editing, syntax checking and highlighting).

The **Eclipse Debugger for QVT** allows you to trace the execution of your QVT code step-by-step. The debugger supports breakpoints (including StepOver, StepInto, and StepOut features), watches, and the **Variables** view. The **Trace** view allows you to inspect the result of your transformation when it is completed.

For XSL/OCL transformations, Together provides a powerful and highly customizable **XSL/OCL Editor** that supports XSL/OCL code sense, syntax highlighting, XSL structure outline, and error checking. The **XSL Debugger**, which runs in the **XSL/OCL Debugging** perspective, supports breakpoints (including StepOver, StepInto, StepOut and StepReturn features).

Together also provides a number of sample projects for each type of transformation.

Note: There are two different implementations of the OCL and QVT engines. The first version available since **Together 2006** refers to the OMG ptc/05-11-01 QVT Specification. Historically it is the primary engine to use with Together models. The other engine **Operational QVT** deployed in **Together 2008** refers to the substantially revised OMG formal/08-04-03 QVT Specification and originally best suited for the **DSL Toolkit**. Since **Together 2008 R2 SP1**, Operational QVT is adopted for use with Together models for both read and write. Legacy QVT transformations developed in the context of the MDA Transformation project, Operational QVT - in context of Operational QVT project.

Overview of the Operational QVT engine is provided in topic **Reference ▸ MDA ▸ QVTO Language**.

The source artifacts (`.qvt` for legacy Together QVT and `.qvto` for Operational QVT) may need adjustment when migrating between these engines because these engines have differences in syntax and behavior. Please refer section **Reference ▸ MDA ▸ QVTO Migration Notes** for migration guidelines. Note that the guide above operates only with pure EMF models so the specific aspects of accessing Together models via EMP API, known from working with Together QVT experience, should be taken into account. Please contact support team in case of difficulties.

Normally there should be no problems using both QVT engines simultaneously, although the UI may look a bit overloaded. It is recommended that you turn corresponding capabilities on or off as needed (select **Window ▸ Preferences**, and then select the **General** node and the **Capabilities** node).

Related Procedures

Creating an MDA Transformation Project
Creating a Model-To-Model Transformation
Creating Model-To-Text Transformations
Creating an XSL Transformation
Creating an Example MDA Transformation Project

Related Reference

QVTO Language
QVT Operational Migration Notes
MDA Example Projects

UML Profiles

Together includes several pre-installed profiles and allows you to create your own profile definitions using Profile Definition project.

In This Section

UML Profiles Basics

UML is a standard modeling language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems. While the general modeling concepts of UML are quite suitable for the majority of developers, in some situations, a further extension of these concepts is useful to allow a more refined rendering of domain-specific concepts and techniques. UML extension mechanisms address the definition of additional semantics of model elements that cannot be expressed directly using UML constructs. This technique is known as *UML Profiling*.

Profiles provide mechanisms that allow metaclasses from existing metamodels to be extended so they can be adapted for different purposes. All kinds of model elements can get stereotypes and tagged values that are defined in the profile applied to the model.

The UML standard provides refinement mechanisms for profile creation, such as stereotypes, tagged values, constraints, and notation icons that collectively specialize and tailor the UML for a specific domain or process. These elements can be used to adapt the UML semantics without changing the UML metamodel. This means that you can interpret the semantics of a profile in the context of the UML specification.

Related Concepts

Stereotype

Profile Definition Project

A Profile Definition project is a profiled modeling project that allows you to create new profile definitions.

One Profile Definition project corresponds to a single profile. Inside the Profile Definition project you can use some packages to locate different elements. For example, you can put all enumerations to one package, all palette contributions to another package, and all stereotypes to a third one. All the elements will be deployed to the same profile.

The Profile Definition adds the following elements to the class diagram Tools Palette:

- Stereotype
- Palette Contribution
- Extension
- Contribution

Tip: Stereotype and Palette Contribution elements are also added to the diagram context menu: **New ▸ Profile Definition**.

Note: Metaclasses referenced in a profile must be taken from the corresponding target UML metamodel that was selected when creating the project.

Related Concepts

Interoperability and Migration

Related Procedures

Together Profiles

Related Reference

EMF API for Together Profiles

Supported Metamodels

In Together, you can create profiles to extend the following metamodels:

- BPMN
- ER Physical
- UML 1.4
- UML 2.0

Stereotypes, tagged values and OCL constraints declared in the profile must refer to the selected metamodel.

Related Concepts

Interoperability and Migration

Related Procedures

Together Profiles

Related Reference

EMF API for Together Profiles

Stereotype

A stereotype contains properties that extend a linked metaclass, and enables the use of platform- or domain-specific terminology or notation in addition to the ones used for the extended metaclass.

A model element that has a stereotype is a special kind of element that conforms to a rigid specification, defined for this stereotype in the profile definition.

The following important issues should be taken into consideration:

- A stereotype is created as a Class20 element in your project with the <stereotype> tag.
- A stereotype extends a metaclass through an Extension link.
- When an instance of a stereotype is created in the target diagram, it gets the attributes (tagged values) of the stereotype in question.
- The following types of attributes (tagged values) of a stereotype are valid: Primitive, Enumeration, Metaclass. Attributes of all the other types are ignored during deployment.
- Tagged values of a stereotype can be defined in two ways: as attributes of the valid types and as association links drawn from a stereotype element to the valid attribute types.
- If a profile defines attributes with a default value for a given type that is different from the default value in the underlying implementation, some side effects should be noted. For example, the default value for a Boolean property is `false` and is not persisted in the model instance. If a Boolean property is set to `true` as the default value in a profile definition and a user sets an instance value to `false`, its value is not persisted but interpreted as the default value (`true`) when read back in. Similarly, instance values changed to empty/null will not be persisted and will likewise be interpreted as the default value when read back in.
- If a tagged value of a stereotype is defined as an outgoing association, its *name* and *multiplicity* properties have specific uses. The name property is set to the supplier role of the association link. If the supplier role is not defined, the association name is used instead. For multiplicity values other than `1` or `0 to 1`, the multivalued attributes are created.
- A stereotype can extend another stereotype via Generalization and inherit its attributes and viewmap. Note that the child stereotype inherits extensions from the parent stereotype and the parent viewmap (unless the child stereotype defines its own viewmap).
- A stereotype has the extended metaclass property, which is defined in the Profile Definition node of the Properties View.
- An abstract metaclass can be chosen as an extended metaclass. In this case, the new element will not appear in any toolbar, but all the elements of the child metaclasses will get this stereotype in the list of predefined stereotypes.
- If, after profile deployment, an element belonging to the type of the extended metaclass is used in the target diagram, the extending stereotypes are added to the list of predefined stereotypes for this metaclass.

Related Procedures

Creating Stereotypes

Palette Contribution

A Palette Contribution enables you to add creation tools for stereotypes and pure metaclasses to the selected Diagram Tools Palette.

A contributed stereotype is associated to a Palette Contribution by means of a Contribution link. It is also possible to associate a Palette Contribution with a shortcut to a metaclass from the metamodel. This allows you to customize the palette by adding some elements to the tool bars of different diagrams. For example, if you want to have the ability to create classes from the component20 diagram toolbar, you can associate the shortcut to `uml20::classes::Class` with a Palette Contribution in your profile definition.

A Palette Contribution can extend another Palette Contribution; when this is done, the child Palette Contribution inherits all the parent diagrams, contributed stereotypes and pure metaclasses.

Related Procedures

Creating Palette Contributions

Extension Link

An extension link indicates that the properties of a parent metaclass are extended with the new properties through a stereotype. An extension link is drawn from a stereotype to a metaclass shortcut whose properties are extended.

Contribution Link

A contribution link connects a Palette Contribution element with a Stereotype.

Modeling for EJB

The Enterprise JavaBeans (EJB) architecture is a component architecture for the development and deployment of component-based distributed business applications. Together is shipped with EJB profiles that allow you to create a specific UML model for EJB.

Together provides the following EJB Profiles:

Profile Name	Description
Standard EJB	Using the Standard EJB profile you can generate deployment descriptors that are EJB 2.0 compatible. This profile should always be selected for any work with EJB because it contains the basic EJB elements.
Standard EJB (ver 2.1)	Provides some additional elements. Also changes properties of some standard elements to comply with EJB version 2.1.
Weblogic EJB Extension	Provides WebLogic specific elements. Also changes properties of some other elements to generate deployment descriptors that are compatible with BEA WebLogic 8.1.

EJB modeling can be thought of as a three-stage process:

1. Modeling in UML using the EJB profile
2. Generating Deployment descriptors while exporting to the Java project
3. Editing the Java project and deploying it to the application server (for example, BEA WebLogic)

Using Together, you can create an EJB model and export it to the Java project.

Note: Because WebLogic 8.1 supports the EJB 2.0 specification, do not enable WebLogic and EJB Standard profile (ver. 2.1) simultaneously.

Related Concepts

UML Profiles Basics
Profile Definition Project

Model Compare and Merge

Together provides a comprehensive solution for comparing and merging models in your project.

EMF and UML Models Compare

Together supports two-way and three-way comparison of EMF or UML models, or model elements of the similar type in a tree view.

In a two-way compare, the compared models are called *Left* and *Right*. Model Compare/Merge traverses the compared models, going level by level down the containment tree. On each level, objects are matched using ID features that you set in the **ID Features** page of the **Preferences** dialog box (**Window** ▶ **Preferences** ▶ **Modeling** ▶ **EMF Model Compare** ▶ **ID Features**). After that, Model Compare/Merge compares values of attributes and non-containment references.

You can export the compare results to an EMF XMI file.

Shared Models Compare

Together provides integration with version control systems and allows two-way and three-way comparison and merge of shared (version controlled) models.

When comparing shared models, the *Left* model represents the local version while the *Right* model represents the remote version. In a three-way compare, the third model is called *Ancestor*. It represents a common ancestor version of the two versions taken from VCS.

Together utilizes standard Eclipse synchronization APIs and is able to compare models stored in any version control system that supports the Eclipse **Synchronize** view.

Comparing and merging shared models requires one (for two-way comparison) or two (for three-way comparison) remote versions of the compared model.

Together copies your local model to a temporary project, then applies changes reported by the repository provider, and then displays these changes in the **Synchronize** view. Temporary models are read-only, and Together uses a modal **Model Compare** dialog box, instead of the standard **Compare** editor.

Merging Models

The merging capability enables you to transfer elements from one model to another.

Related Procedures

Comparing and Merging Models

Related Reference

Model Compare/Merge

Template Elements and Generics Overview

Template elements

Together supports templates, as defined in the UML 2.0 superstructure specification. This support provides the ability to show templates, template signature, parameters, and template bindings in the UML 2.0 diagram.

A templateable element may contain a template signature which specifies the formal template parameters. A templateable element that contains a template signature is a template.

A template signature displays in a diagram as a rectangle in the top-right corner of the owing element. In the Properties View of a template element, the `isTemplate` property is set to `true`.

A template binding represents a relationship between a templateable element and a template. A template binding specifies the substitutions of actual parameters for the formal parameters of the template.

Generics

In the LiveSource projects, Together supports generic language constructs that describe specialization of templates for a certain type. Such constructs display in a diagram as special entities. For C++ projects, this possibility is enabled by default; for Java projects, generics are enabled by means of a special setting in the Project Properties dialog.

Consider the following example:

```
template<class T> Class A; // defines a template A with the parameter type T
class B;
A<int> *a // on the diagram A<int> will display an entity as a specialization of a template
A for the type <int>
A<float> *f // displays another entity for the floating type
```

Related Procedures

Creating Template Elements

Model Import and Export Overview

You can share model information with other systems by importing and exporting model information, or by sharing project files:

Feature	Description
Exporting diagrams to images	You can save diagrams in several formats, including: Bitmap image (`BMP`) Enhanced windows metafile (`EMF`) Graphics interchange (`GIF`) JPEG file interchange (`JPG`) Scalable Vector Graphics (`SVG`)
Importing IBM Rational Rose (MDL) models	It is possible to convert models designed in IBM Rational Rose 2003 to the format of Together. The following file formats are supported: `.mdl, .ptl, .cat`, and `.sub`.
Importing from MDX	Together enables you to create projects around an IBM® Rational® XDE `.mdx` file.
Importing from XMI Exporting to XMI	XMI (XML Metadata Interchange) enables the exchange of metadata information. Using XMI, you can exchange models across languages and applications. For example, if you have a modeling project created with a tool other than Together, you can import it to Together as an XMI file for extension or as the basis of a new project. Likewise, you can export Together projects for use in other applications. The result in each case is a single, portable .xml file. XMI for UML 2.0 was introduced in IBM® Rational® Software Architect and allows you to exchange models that comply with UML 2.0 specification. The models are exchanged via files with an .uml2 extension. For import and export, Together supports the following UML versions/platforms: • XMI for UML 1.3 (Unisys Extension) • XMI for UML 1.3 (with Unisys Extension recommended for Together ControlCenter) • XMI for UML 1.3 (with Unisys Extension recommended for Rose) • XMI for UML 1.4 (OMG) • XMI for UML 2.0 • XMI for UML 2.0 compliant with OMG standard (XMI created without usage of some non-OMG-standard tags such as eAnnotations) To import a project from Together ControlCenter, first use Together ControlCenter and export the project to UML 1.3 (Unisys and Together Extensions) and then import it into Together. In addition, always use XMI for UML 1.3 (with Unisys Extension, recommended for TCC) when exporting a Together project to be used in Together ControlCenter. XMI

	export and import makes it possible to reuse multi-root projects.
Importing from other versions of Together Sharing with other versions of Together	You can reuse models created in other editions and versions of **Borland Together**. This feature is known as **interoperability**. TVS projects and projects created in Together Editions prior to version 7.0 cannot be imported to Together.
Export a Quality Assurance metric chart to image	Create a chart and then export it to image.

Related Concepts

Together Interoperability and Migration

Related Procedures

Exporting a Diagram to an Image
Importing a Project in IBM Rational Rose (MDL) Format
Importing a Project in an IBM Rational Rose MDX Model
Importing a Project in XMI Format
Exporting a Project to XMI Format
XMI Export and Import of the Models with Cross-Project References
Creating a Metrics Chart

OCL Support

This section provides an overview of OCL in Together.

In This Section

About OCL Support in Together

This topic describes support for Object Constraint Language.

About OCL

The Object Constraint Language (OCL) is a formal language that describes expressions on UML models. OCL expressions specify operations or actions that, when executed, alter the state of the system. UML modelers can use OCL to specify application-specific constraints in their models. UML modelers also can use OCL to specify queries on the UML model, which are completely programming language independent. For more information about OCL, refer to the OCL 2.0 specification.

Together allows you to use all the capabilities of OCL 2.0 to work with your model:

- Add the OCL constraints to the types defined in your model, providing them as constraint notes linked to the context elements on the diagram. The constraint text opens in a powerful editor that provides syntax highlighting, errors validation, and code completion functionality.
- Generate Java code from your model, optionally generating the code for OCL expressions used in the model.
- Use OCL as a query language operating with types defined in the metamodel. You can perform a Search In Model by OCL query, write and run Model Audits and Metrics, and use OCL expressions in the documentation templates for the documentation generator.

Note: Portions of this product include the Object Constraint Language Library, courtesy of Kent University, United Kingdom. See http://www.cs.kent.ac.uk/projects/ocl/

Supported Diagram Types

OCL supports the diagrams listed in the following table.

Diagram types with OCL support

Diagram type	UML version	Support provided
All diagram types	2.0	Object constraints. The default language of constraints depends on the context element type and project type.
Interaction (Sequence and Communication)	2.0	State invariant constraints for lifelines and constraints for the operands of the combined fragments as OCL expressions. Pre- and post- condition for the Interaction. These elements are realized as inner constraint elements available via element's Properties.
State Machine	2.0	Guard conditions of transitions as OCL expressions. Pre- and post- conditions of a StateMachine, and a StateInvariant for a State. These elements are realized as inner constraints available via element's Properties.
Activity	2.0	Pre- and post- conditions for activity; local pre- and post- conditions for an action.

Related Concepts

UML Modeling Overview

Related Procedures

Together Object Constraint Language (OCL)

Related Reference

Diagram View
EMF API for Together Profiles

OCL Constraints and Expressions

As the OMG's specification describes it, OCL is a formal language used to describe expressions on UML models. These expressions typically specify invariant conditions that must hold for the system being modeled or queries over objects described in a model.

The buttons on the diagram **Palette** allow you to create **OCL constraints** as design elements on diagrams, and link these constraints with the desired **context**. The **OCL Expressions** view provides an OCL editor that lets you develop and validate OCL expressions. Any OCL constraint contains an **OCL expression**.

OCL support for constraints provides syntax and error highlighting in the **OCL Editor** view. The text of the constraint is validated when the constraint is linked to its context.

Related Concepts

UML Modeling Overview

Related Procedures

Together Object Constraint Language (OCL)

Related Reference

Diagram View

OCL on Non-Class Diagrams

Constraints on non-class diagrams fall into two categories:

- Inner constraints
- External constraints

The OCL editor provides syntax highlighting, errors validation and code completion functionality.

Inner constraints

The following properties are defined by creating nested constraints inside the elements. Generally, each property is a property tab that contains two properties—language (OCL/text) and body (constraint body).

- `guard` in transition/internal transition
- `precondition` and `postcondition` in StateMachine, Activity, Interaction
- `local precondition` and `local postcondition` in Action
- `state invariant` in State
- `Condition` in Extends on a Use Case diagram
- `state invariant` and `Interaction constraint` on a Sequence diagram
- `body, precondition, postcondition,` and `ownedRule` in Operation

The OCL editor is available for the constraints inside the elements. You can expand the element node in the Model Navigator and open a constraint in the editor.

Tip: For the properties of elements that can have class as their context, the OCL context is set automatically.

Constraint	Context
Element with defined constraint	Correct context for the constraint.
StateMachine	Operation selected as a Specification association of this StateMachine (a class is selected for the context property and specification is a method of the selected class).
Activity	Specification of the activity.
Action	Specification of the activity that contains this action.
State and Transition	Class selected in the context property of the StateMachine that contains this State or Transition.
Operation	Operation itself. Note that the operation has a valid OCL context only if it is owned by Class or Interface. Operations owned by other classifiers get no OCL context, and their constraints should have text as a constraint language.
Interaction	Context of this Interaction or Specification if it is defined.
State invariant and Interaction constraint	Class selected as type of the Lifeline that contains this element. It can be either a class directly selected as the type of the Lifeline or part selected as the Lifeline representation.
Extends	Context cannot be specified and constraint can only be defined as text (`language=text`).

Tip: If a context for a constraint with `language=ocl` is not specified or cannot be specified, such constraints are shown as invalid.

External constraints

In addition to the inner constraints described above, all the elements on the non-class diagrams that are Types (various classifiers) can be furnished with external OCL constraints.

An external constraint is created as a Constraint element linked to the constrained element by the context link.

Unlike inner constraints, the external constraints always use the linked type as a context. Thus, the StateMachine constraints may have the context of the assigned specification if it is an inner precondition, or the context of the StateMachine itself if it is the external linked constraint.

Related Concepts

UML Modeling Overview

Related Procedures

Together Object Constraint Language (OCL)

Related Reference

Diagram View

Patterns and Templates

This section describes patterns and templates in Together.

In This Section

Patterns and Templates Overview

Patterns provide software developers with powerful reuse facilities. Rather than trying to tackle each design problem from the very outset, you can use the predefined patterns supplied with Together. The hierarchy of patterns is defined in the **Pattern Registry**. You can manage and logically arrange your patterns using the **Pattern Organizer**.

Patterns and templates are pluggable extensions for Together enabling you to:

- Create new and frequently used elements
- Modify existing elements
- Implement source code constructions and solutions in your project.

Code templates

Together supports code templates to provide backward compatibility with previous versions of Together. You can use your legacy source code templates to create elements in the source-code projects. Code templates are text files with the extension specific for Java that use macros to be substituted with real values when the templates are applied. Therefore, code templates can be regarded as forms ready for "filling in" for a specific instance. A code template consists of a template file containing source code, and a properties file that contains macro descriptions and their default values. Templates in Together are mostly used in Java Modeling projects.

Patterns

Each pattern describes of a set of model elements, relations between them, and constraints applied to those elements. Patterns are represented by special modeling projects covering all the aspects of patterns. Patterns, in general, are independent of any programming or markup language. You can use them to create or modify any type of element. However, concrete patterns are designed to work with elements of a specific type. Use **Pattern Registry** to manage patterns.

Note: Together is shipped with some predefined patterns that cannot be deleted or otherwise edited.

Pattern instances

Pattern instances appear as a result of recognition of the existing model or creating new instances (along with model elements playing pattern roles) in the model. Pattern instances contain information about the pattern name and the role of each participant. They are shown in the **Pattern Explorer** view and under the Patterns node in the **Model Navigator**.

When applied to a diagram, such patterns create their entities and are presented on the diagram itself, with the links to the created entities. Such patterns enable further modification by means of adding new participants (new pattern part). All patterns that appear in the **Pattern Explorer** are represented in the project model in the form of entities with metaclass “pattern”. Visually, pattern instances are displayed as ovals (like collaboration occurrences). Pattern entities have children links to pattern participants, which allow viewmap links on diagrams from pattern instances to pattern participants. Actions on pattern instances in the model are the same as in pattern explorer.

During the lifetime of the pattern instance, the model can change (some elements from the instance may be deleted, others may be changed so that they no longer satisfy the pattern definition) and the pattern instance can become invalid. This is why you need to perform pattern instance validation regularly.

Related Concepts

Pattern Definition Project
Templates

Related Procedures

Patterns and Templates

Related Reference

Patterns and Templates
Pattern Registry
Pattern Explorer

Pattern Definition Project

Using a pattern support subsystem in Together, you can easily work with patterns via pattern definitions. You can use well-known, predefined patterns. You can also define new ones and delete, rename, or edit existing ones. Using Together, you can manage pattern instances by recognizing patterns in an existing model, creating elements by pattern, creating new participants for particular roles for an existing pattern instance, and so on.

A pattern definition project is a profiled UML 2.0 modeling project with the following modifications that distinguish it from a pure UML 2.0 project.

The following elements are allowed in a pattern definition project:

- Instance specifications
- Slots
- Pattern definition links (derived from a Kernel Association class, able to connect instance specifications)
- Constraints
- Value specifications
- Pattern constraint links (derived from a Binary link class, aimed to define constraint parameters)
- Class diagrams
- All other elements are prohibited

The following extensions are added to the allowed metaclasses:

- Instance Specification—able to aggregate pattern definition links.
- Slot—the following new properties are added: `Use for recognition` (Boolean) — Controls whether to use this property on recognition. `Use for generation` (Boolean) — Controls whether to set this property on creating elements by pattern. `Is configurable` (Boolean) — When set to true, indicates that by using the “create by pattern” wizard, the user can modify the value of this property to be set on element creation. This property should be false if `Use for generation` property is set to false.
- Constraint—able to aggregate pattern constraint link.
- Class diagram—`patternPartWizardDefinition` (Boolean) — When set to true, the new “create pattern part” wizard will be generated for the instances of this pattern.

When you create new pattern instances from existing project elements, the creation process uses participants from your current selection and enables you to modify the pattern properties. You can easily view and modify properties of pattern instances using the standard Properties view. Any change that you make to a pattern property applies immediately to the pattern participants (via refactoring).

All pattern definitions are stored in the `com.borlang.tg.patterns\patterns` subfolder of the Together plugins folder. Each pattern definition is an archive file packed by zip utilities provided by the Java Development Kit (JDK). Pattern definitions contain compilation results suitable for recognition and completion engines and for the whole definition project so that definition editing can be done. Folder and shortcuts structure are stored in the `pattern.registry` file in the same location.

Related Procedures

Creating Pattern Definition

Related Reference

Create Pattern from Elements

Pattern recognition

Pattern recognition identifies pattern instances from existing elements in the project. The identification process determines pattern participants and parameters. You can start the pattern recognition process from the project's context menu to perform pattern recognition, validation, and problem reporting.

Related Procedures

Recognizing Patterns

Templates

A template allows you to quickly and automatically create code, insert code, or transform existing code. There are several different types of templates:

- Package: For modifying/creating specific groups of classes and members
- Class: For modifying existing classes or creating a new class
- Link: For modifying existing links or creating a new links on the Class diagram

You can use template extensions to create template instances. Template instances are managed by the template manager, which also gives you the ability to manage existing template instances, or create new ones. A template instance operates on existing elements using an associated template source. The template source contains a template-specific specification of elements and constructions that are applied on the target elements. For example, an instance of the Java class template uses its template source to specify imports, fields, methods, and inner-types that are created when the template is applied to a target Java class.

You can create new template instances using the template wizard. The current selection of elements is analyzed and then an appropriate template source is created using the data from the selected elements. For example, you can create a new Java class template from an existing class. The wizard analyzes the selection and extracts imports, fields, and methods from the selection. It then creates the template source. The template source is then associated with the new template instance.

Quality Assurance

Quality Assurance in Together provides teams and managers with measures of the quality of their project. As with any Quality Control, the team should understand what is measured, and why. Although audits and metrics are similar in that they both analyze your project, they serve different purposes. Audits and metrics are run as separate processes. Because the results of these two processes are different in nature, Together provides different features for interpreting and organizing the results.

In This Section

Code Audits

Together provides a wide variety of audits, ranging from design issues to naming conventions, along with descriptions of what each audit looks for and how to fix violations. The process of running audits begins with your selecting the specific rules to which your source code should conform. Together generates an audit report that displays only the violations of those rules. You can examine each violation and decide whether to correct the source code. You can create, save, and reuse sets of audits to run. Together ships with a predefined saved audit set and you can create your own custom sets of audits to use.

Problem Detection Audits

For most violations, the audit report generated by Together indicates the line of code that causes the violation. For some audits, however, such a line number is inappropriate. These are called problem detection audits. An example is the Misplaced Class audit, in which the package of the class is deemed inappropriate because of the dependency between the class and a different package. For problem detection audits, Together uses one or more of detection metrics to analyze the code to determine audit violations.

Together audit reports show problem detection audits along with the other, line-oriented audits.

Bad Smell Audits

Together includes a group of audits known as "Bad Smell Audits" that detect some issues or convention violations in source code (misplaced classes, attributes and methods, wrong inheritance usage), which require some code refactoring.

Related Concepts

Quality Assurance

Related Procedures

Running Source Code Audits

Model Audits

Together supports a wide range of model audits. The list of available model audits can be viewed in the **Preferences** dialog box. You can define, save, and reuse sets of model audits. Model audits are OCL queries that produce Boolean results and that operate in the context of existing metamodels. You can also employ additional OCL operations provided for the Borland metamodel, specified in the **OCL operations** and **OCL library operations** tabs in the **Preferences** dialog box.

Together also contains a set of sample audits (the ideas of most of them are taken from Ambler and Fowler books). These audits can be used as examples for custom rules creation. For a description of the predefined model audits provided in Together, refer to “Model Audits and Metrics Descriptions.”

After you run model audits, the results are displayed in the Model Audits View. The view provides detailed descriptions for all found errors and you can navigate to the corresponding problem element from this view by double-clicking the error message.

Related Concepts

Quality Assurance
OCL

Related Procedures

Running Model Audits and Metrics

Related Reference

Model Audits and Metrics Descriptions
QA Model

Code Metrics

Metrics evaluate object model complexity and quantify your code. It is up to you to examine the results and decide whether they are acceptable. Metrics results can highlight parts of code that need to be redesigned, or they can be used for creating reports and for comparing the overall impact of changes in a project.

Together provides a wide variety of metrics, ranging from lines of code to comment ratio. When you run metrics in Together, you first select which metrics are important for your project. You can use metrics results that Together generates to determine which code needs to be redesigned, or you can use the results to create reports and compare the overall impact of changes in a project. Together makes it easy to run metrics, view the results, and interpret the findings.

Related Concepts

Quality Assurance

Related Procedures

Running Source Code Metrics

Model Metrics

Together supports a wide range of model metrics. The list of available model metrics can be viewed in the **Modeling ▸ QA Model** node of the **Preferences** dialog box. You can define, save, and reuse sets of model metrics. Model metrics are OCL queries that produce Integer results and that operate in the context of existing metamodels. You can also employ additional OCL operations provided for the Borland metamodel, specified in the **OCL operations** and **OCL library operations** tabs in the **Modeling ▸ OCL** page.

For a description of the predefined model metrics provided in Together, refer to “Model Audits and Metrics Descriptions.”

After you run model metrics, the results are displayed in the **Model Metrics** view. You can navigate to the corresponding elements listed in the **Model Metrics** view by double clicking the element name.

Related Concepts

Quality Assurance

Related Procedures

Running Model Audits and Metrics
Using OCL in Model Audits and Metrics

Related Reference

Model Audits and Metrics Descriptions
QA Source

Metrics Graphical Representation

Metrics results can also be viewed graphically. Two graphic views allow you to summarize metrics results: bar charts and Kiviat charts. Both charts are invoked from the context menu of the table. Use the Kiviat chart for rows and the bar chart for columns.

Bar Chart

The bar chart displays the results of a selected metric for all packages, classes, and/or operations.

The bar color reflects conformance to the limiting values of the metric in reference:

- Green represents values that fall within the permissible range.
- Red represents values that exceed the upper limit.
- Blue represents values that are lower than the minimal permissible value.
- A thin vertical red line represents the upper limit and a thin vertical blue line represents the lower limit.

Kiviat Chart

The Kiviat chart demonstrates the analysis results of the currently selected class or package for all the metrics that have predefined limiting values. The metrics results are arranged along the axes that originate from the center of the graph.

Each axis has a logarithmic scale with the logarithmic base being the axis metric upper limit so that all upper limit values are equidistant from the center. In this way, limits and values are displayed using the following notation:

- Upper limits are represented by a red circle. Any points outside the red circle violate the upper limit.
- Lower limits are represented by blue shading, showing that any points inside the blue area violate the lower limit. Note that blue shading does not show up in areas of the graph with lower limits of 1 or 0.

Tip: To see the value of an individual data point on the Kiviat graph, hover your mouse pointer over it to display a popup.

- The actual metrics show up in the form of a star with metric values drawn as points.
- Green points represent acceptable values.
- Blue points represent values below the lower limit.
- Red points represent values exceeding the upper limit.
- Scale marks are displayed as clockwise directional ticks perpendicular to the Kiviat ray.
- Lower limit labels are displayed as counterclockwise directional blue ticks perpendicular to the Kiviat ray.

Related Concepts

Quality Assurance

Related Procedures

Running Source Code Metrics
Running Model Audits and Metrics
Creating a Metrics Chart

Exporting and Importing Audits and Metrics

Introduces import and export functionality for audits and metrics.

Source Code Audits and Metrics

Once you identified the QA rules the team needs to use, you can create specific QA Sets for source code audits and metrics in Together. These QA sets can be saved to your local file system, or you can save them in the project. Saving them with the project makes them easier to distribute to your team via the version control system (VCS).

The **C++** and **Java QA Source** pages of the **Preferences** dialog box display the set of all available audits and metrics for C++ and Java source code projects respectively. When you open a project, a default subset is active. Active audits and metrics are indicated by check marks. You can select necessary audits and metrics and save the selected set for future use. The quality assurance sets are saved with a `.qa` extension.

Model Audits and Metrics

You can import and export model metrics and audits all at once, including a set of named OCL queries on metamodels, and other settings. Model audits and metrics can be saved to files with `.ModelMetrics` and `.ModelAudits` extensions. When importing such a file, you completely replace your currently defined model audits or metrics.

Related Procedures

Exporting and Importing Model Audits/Metrics
Creating and Using Code QA Sets

Refactoring Overview

Together leverages refactoring operations provided by the platform.

When refactoring is applied to source code, the changes propagate to the model. For example, when classes or operations are renamed by means of refactoring, the hyperlinks to the renamed elements are preserved.

Refactoring is available for the model elements in Together projects by means of context menus. Refer to JDT documentation for information on Java refactoring and to CDT documentation for information on C++ refactoring.

Requirements Management

Requirements Management allows you to create and manage traces between Together diagram elements and Borland CaliberRM or Rational RequisitePro requirements.

Traceability is supported via CaliberRM and RequisitePro Integrations for Together, respectively. You can find more information about working with specific requirements in online help provided with both integrations.

Together provides the following requirements management possibilities:

- Create and delete traces between requirements and Together diagram elements.
- Create requirements based on use case.
- Manage traces between requirements and model elements in the Element Traces view.
- Synchronize traces that are out of date using the Trace Synchronizer view.
- Navigate easily between traced elements and related requirements.

Note: Together 2007 and later versions of the product do not include integrations with requirements management products. These integrations are available separately on demand. Corresponding parts of user interface and functionality are not available when integration is not installed.

Related Procedures

Opening Requirements Views
Creating Traces from Requirements to Model Elements
Deleting Traces
Creating Requirements Based on Use Case
Viewing Element Traces
Synchronizing Traces
Navigating from Model Elements to Requirements

Related Reference

Element Traces View
Trace Synchronizer View

Version Control in Together

This topic provides an overview of version control features in Together.

Overview

Together uses a file-based approach to store models. This provides openness and choice when selecting a version control system to manage your models.

Together supports several version control systems that can be integrated in Eclipse. They include but are not limited to CVS, StarTeam, and ClearCase. Version control in Together enables several users to work with one modeling project.

Together leverages the functionality provided by the Version Control System client and maps these menus from the file resources level (as provided by the Version Control System provider) to model elements.
Together provides context menus to work with CVS, StarTeam, and ClearCase version control systems.

Visual team status indicators for items are also displayed in the model navigator.

Note: The Together teamwork-related functionality that is provided depends on how well the specific Version Control System integrates with the basic Eclipse team support flow and UI. Therefore, some Version Control System features may be not available or may work differently for a given Version Control System.

Your version control system should be set up so that only one user can work with a shared model at a time. In case several users edit the model at a time, use the model compare and merge functionality of Together. You can compare the structure of your models and merge inconsistencies if necessary. Alternatively, you can revert to the saved version of the model.

The model merge tooling provides a hierarchy comparison of two models with annotations to show what has been added, what is missing, and what has been changed. However, the model merge does not provide a comparison of changes in the layout, sizing, or ordering of model elements.

Note: For more information about each version control system, refer to the appropriate program documentation.

Note: Together 2007 and later versions of the product do not include integrations with version control systems (StarTeam, ClearCase). These integrations are available separately. Eclipse clients for corresponding versions control systems are not bundled with the Together product and must be additionally installed to the same Eclipse instance as Together.

Models, views, and files

It is important to be aware of the relationship between models, views, and the files used to store models and views in Together.

Task-aware features

Task-aware features of Together make it easier to work with modeling resources under version control by automatically finding all the resources that need to be checked out for the modeling resource to be modified. To enable task-aware features, use the following Team options for locking files and managing modeling resources:

Path to option	Options
Preferences ▸ Team ▸ Star Team ▸ File ▸ Locking	**Clear file lock on check-in** and **Mark unlocked working files read-only** This option also exists on a project level.

Preferences ▸ Team ▸ Modeling resources	AutoCheckout modeling resource on edit.

While editing some model elements, a lock dialog box appears, which lists all resources that need to be locked. By choosing **Yes** you apply a locking mechanism to the selected resources and they become writable. After modification, when you are checking in the files, they will become unlocked and read-only. Each time you try to edit a read-only file that is under version control, a dialog box appears suggesting that you lock the file for editing. To automatically lock files on checkout, check the **auto lock** check box in the dialog box or use the **Lock files on modification** in the Preferences ▸ Team ▸ Star Team ▸ File ▸ Locking page (this is a StarTeam Eclipse client option).

Note: The Preferences ▸ Team ▸ Star Team options are for a Version 10.1.0.24 of StarTeam and may be different for other versions of StarTeam.

Note: The task-aware feature of Together cannot manage read-only files located inside a Package when this package is moved or renamed. Eclipse does not let you automatically check out files in this situation. Use Navigate to resources on the Team menu to select files for locking and then try to move or rename the package again.

You can also use the **Ignore default package diagrams** option (Preferences ▸ Team ▸ Modeling Resources) to specify that default package diagrams are not stored or synchronized. This option adds default.txvpck and default.txvClassDiagram20 patterns to the ignored resources.

Together Version Control Recommendations

- Use an Eclipse Team Provider.
- Check in the **.txaPackage** file.
- Do not check in the default package diagram.
- Use version control locking at the package level.
- Consider the version control implications if you rename any diagrams or packages.
- Consult an expert if you need to resolve any merge conflicts.

Support for CVS edit/unedit commands

Together provides a number of additional variants of edit/unedit commands for diagrams and packages in the context menu Team. These commands are available when the Capability Team ▸ CVS support for Modeling is enabled. Commands help to perform version control operation on not the individual resources but the entire set of the related resources.

For Diagram the available choices are:

Command	Description
Edit/Unedit View	Affects only the file of selected diagram (*.txv*)
Edit/Unedit View and Model	Affects the file of selected diagram and also files (*.txv*/*.txa*) containing elements shown on the diagrams regardless of their package
Edit/Unedit View and package locally	Affects the file of selected diagram and also files containing elements shown on the diagrams from the same package
Edit/Unedit View and package recursively	Affects the file of selected diagram and also files containing elements shown on the diagrams from the same package and all its subpackages

For the package

Command	Description
Edit/Unedit	Affects the diagram and model files (*.txv*/*.txa*) in the selected package
Edit/Unedit Recursively	Affects the diagram and model files in the selected package and all its subpackages recursively

Related Concepts

Together Capabilities Activation

Related Procedures

Using Version Control and Teams in Together

Related Reference

Common Diagram Context Commands
Package Context Menu

Project Documentation

This part describes the documentation generation facility and documentation template basics.

Related Concepts

Documentation Generation Overview
Documentation Template

Related Procedures

Generating HTML Documentation
Creating Custom Documentation Template

Documentation Generation Overview

Together enables you to create external documentation for the open projects, or from the command line. Use the generated reports to illustrate your projects with the documentation in one of the available formats.

Documentation generation is available for all types of Together projects, including Business Process, Data Modeling, Pattern Definition or Profile Definition projects.

The generated documentation can include the results of Audits, as well as the information extracted from the integrated products (for example, from CaliberRM).

Documentation output formats

You can generate documentation in one of the following output formats:

- RTF
- HTML
- TXT
- PDF
- XSL-FO

By default, documentation is generated in HTML format.

Documentation files

All the documentation that Together generates is written to a single directory that you specify in the documentation generation dialogs. By default, this is the `out` folder of your Eclipse workspace.

The generated documentation opens in the appropriate viewer, associated with the output format.

If a report is generated from an Ecore model, the top package name is used as the model file extension. If a report is generated from a domain, the domain name is used as the model file extension.

Documentation templates

Project reports are created by applying documentation templates to Together projects. The templates contain commands to the documentation generator; the projects provide the source of project-specific data. Documentation templates are `*.tpl` text files with formatting instructions and tags for the commands.

Together comes with a set of predefined templates and also lets you create custom documentation templates, using the built-in **Documentation Template Designer**.

Custom Together templates make it possible to use styles, headers and footers from the Word documents.

Related Concepts

Organization of a Documentation Template

Related Procedures

Generating HTML Documentation
Generating Project Documentation Using Template
Generating Project Documentation from Command Line
Configuring the Documentation Generation Facility
Creating Custom Documentation Template
Using Word Documents in Documentation Templates

Related Reference

Documentation Template Designer

Documentation Template

The documentation generator uses Together projects and templates to produce project reports. You can use predefined templates that are delivered with the product, or create your own custom templates, using the Template designer. This part discusses the structure of templates, its zones, sections and controls.

In This Section

Documentation Generator Metamodel
Documentation Generator makes use of its own metamodel that defines the hierarchy of metatypes.

Organization of a Documentation Template
This topic describes the organization of a documentation template and the correspondence between the elements of a template and the generated output.

Documentation Template Sections
This topic describes sections of a documentation template.

Documentation Template Controls
Controls are the items in documentation templates that determine the contents of reports.

Multi-frame Documentation Templates
This topic describes multi-frame documentation templates structure.

Hyperlinks in Documentation
A hypertext link connects a link reference (starting point or source) to a link destination (target).

Javadoc Link References
Together supports Javadoc References (or JDRefs), which are the expressions associated with Javadoc tags.

Enable Conditions
Enable conditions are Boolean expressions for turning section processing on or off.

Documentation Generator Metamodel

The Documentation Generator (DocGen) has its own metamodel described in the metamodel `plugins\com.borland.gendoc.core_8.1.0\templates\MetaModel.mm` definition file.

It is a textual file that defines the metatypes hierarchy, how metatypes correspond to the model elements, the types of elements another element can contain, and the properties of each metatype. The beginning of each model definition file lists the properties that DocGen knows. These include DocGen-specific properties and others. Properties are defined as follows:

```
property_name = "[name of property localization key]"
```

The remainder of each model definition file contains the metatype definitions. The major fields in the definitions are as follows:

- `name:` metatype name
- `extends:` parent metatype
- `full_name:` the name displayed in the Documentation Template Designer
- `metatype_filter:` defines the correspondence between metatype and model element
- `rwi_entity:` the type of the related element in Together API
- `properties`: a list of properties available for this type; descendant metatypes inherit their properties from parent metatype
- `excluded_properties`: items listed among the properties that are not documented when using the 'all properties' scope in Property iterator
- `contained_metatypes:` metatypes that can be contained by this metatype

The `name` field for each type is always present. The existence of the other fields varies with the type. An example of a metatype definition follows:

```
<metatype>
name=NODE
extends=ELEMENT rwi_entity=node
full_name="[gendoc/gen_doc_by_template1/full_name_NODE]”
properties = { %package }
contained_metatypes = { NODE; MEMBER; LINK }
</metatype>
```

An element iterator or folder can contain nested element iterators whose type is listed among its contained metatypes, the contained metatypes of its parent, or indirectly through the contained metatypes of one of its contained metatypes. For example, an element iterator with a DIAGRAM scope can contain nested element iterators with the following scopes:

- hyperlink (inherited from ELEMENT)
- diagram reference
- diagram
- node
- link
- member (indirectly through the contained metatype, NODE)

Element properties are inherited. An element iterator can contain nested property iterators whose type is inherited from its ancestor or listed directly among its properties. For example, an element iterator with a DIAGRAM scope can contain nested property iterators for the following types of scopes: shapetype, name, documentation, annotation, hyperlink, url (inherited from ELEMENT), package, stereotype, and alias.

Related Concepts

Organization of a Documentation Template

Organization of a Documentation Template

A documentation template is a `*.tpl` text file that contains instructions to the Documentation Generator. Project reports are created by applying documentation templates to Together projects.

In this section you will learn about:

- Zones of a template
- Body of a template, and its representation in a generated report
- Root object metatype
- Current model elements

Zones of a template

Documentation templates consist of headers, footers, and body sections. The Documentation Template Designer divides templates into five major zones:

- Page header
- Report header
- Body
- Report footer
- Page footer

The zones are horizontal bands that go across two panes. The *scope pane*, which is on the left, reveals the template structure. The *details pane* on the right shows the contents of the zones, which include commands to the DocGen engine. Context menus for each zone are different in the scope pane and in the details pane.

Headers and footers are at the top and the bottom of the Designer window. The report header and the report footer apply only once per document. Page headers and footers apply once per page for RTF documentation; they are ignored for HTML and text documentation.

The body zone of a template contains the commands that produce the body of the generated report. DocGen builds a report into horizontal regions. Each region in the report corresponds to a section in the template that determines the data for that region and how that data should appear.

Body of a template

The body of a documentation template is organized into a hierarchy of sections. Some sections in the body are nested inside others. Some sections have siblings. Sections that are not nested within any others are children of the root. The scope pane reveals the tree structure, indenting each section according to its level in the tree.

Root object metatype

Every section in the body of a template has a section scope. Scopes are based on metatypes that correspond to the different types of model elements. The section scope of the body zone corresponds to the root object metatype.

The model itself is considered to be a special metatype, which is the default root metatype for a new template.

Current model element

Documentation Generator uses a dynamic current model element to go through a template and access specific project information. The type of the current element is the metatype for the section that the engine is currently processing. The value of the current element changes according to when the processing for the section takes place.

The body of a report is created starting from the root element, going in a “depth-first” fashion. In other words, processing starts with the first root section, visiting it along with any of its nested subsections before continuing to the next root section. This pattern is recursive: visit the sub-tree rooted at a section before going to the next sibling section. For each sibling of a section, DocGen begins its processing with the same current element.

Related Concepts

Documentation Generation Overview
Documentation Template Sections
Documentation Template Controls

Related Procedures

Creating Custom Documentation Template

Related Reference

Documentation Template Designer
Area Properties

Documentation Template Sections

The body of a newly created template consists of a generic element iterator and a static section nested within. It provides a minimal base for constructing the tree of sections. Every new section must be a sibling or a child of an existing section.

There are six different types of body zone sections:

- Static sections
- Element iterators
- Element property iterators
- Folder sections
- Calls to stock sections
- Calls to template sections

Static section

Static sections contain the commands to the Documentation Generator for getting project data.

Of all kinds of body sections, only static sections contain controls for producing actual output. Headers and footers can also contain controls. Folders and iterators, which cannot directly contain controls, must have at least one static section nested somewhere within.

You can edit properties of a static section. Refer to the *Static section* reference for details.

Element iterators

Element iterators provide a way of looping through elements of a model. Each element iterator has its own metatype, which must be consistent with the metatype of the iterator's parent's.

If you want an iterator to be able to access an entire model, choose Package as the metatype.

In an element iteration section, a new current element is calculated according to the current element of the parent section and the metatype of the iterator. Documentation Generator loops through an element iteration section using each possible new element as the current element for that iteration. The properties of an element iterator affect the way a new current element is calculated and how it changes during iterations. If no elements are encountered corresponding to the iterator's metatype, no documentation is produced.

Element iterators can have headers and footers. If the section execution does not result in output, then the iterator's headers and footers are ignored.

Scope options determine which elements of the model this iterator will document. Each iterator works over the sub-tree of the model that is rooted at the current element (the element that starts the iteration).

You can edit properties of an element iterator. Refer to the *Element Iterator* reference for details.

Element property iterators

Element property iterators are for looping through the properties of model elements.

Element iterators traverse model elements. Element property iterators traverse element properties instead of elements.

A property iterator can reside inside an element iterator, folder, or property iterator. A property iterator must contain at least one static section, folder section, or call to a stock section or template. A property iterator may also contain an element iterator, or another property iterator.

A property iterator is described by its properties, such as its iteration scope and sorting. Refer to the *Property Iterator* reference for details.

Folder sections

Folder sections group other sections together. A folder has at least one nested section, and it may have a header or footer. In that sense, folders are similar to element iterators, except that DocGen executes folders only once.

Folders inherit their metatypes from their parents. The sections nested within a folder must be consistent with its metatype. Folders provide a way to put section-level properties on their contents. This includes enabling conditions for toggling its processing on and off.

Folders can have headers and footers. If the sections in a folder do not result in output, then the folder's headers and footers are ignored.

A folder section is described by its properties, such as its output style and enable condition. Refer to the *Folder section* reference for details.

Calls to stock sections

Stock sections are reusable folders or iterators that reside in the template's collection of stock sections. They are not shared among different templates. When a call to a stock section is processed, it is the same as if the called stock section were simply embedded at the position of the call.

Stock sections are especially convenient for frequently used constructs. You can insert a call to a stock section from any section whose metatype is consistent with the metatype of the stock section. Stock sections may contain calls to other stock sections, as well recursive calls to themselves.

You can edit properties of a call to stock section. Refer to the *Call to stock section* reference for details.

Calls to template sections

With a call to a template, DocGen can produce documentation using a different template without terminating the current one.

When a template is called, the current element of the calling template becomes the root element of the called template. A calling template can pass additional information to the called template through template parameters.

Calls to templates make it possible to construct a library for generating documentation for particular model elements (class, actor, use case, and so on).

You can edit the properties of a call to a template section. Refer to the *Call to template section* reference for details.

Related Concepts

Documentation Generator Metamodel

Related Procedures

Creating Custom Documentation Template
Creating Sections

Related Reference

Static Section Properties
Element Iterator Properties
Property Iterator Properties
Folder Section Properties
Call to Stock Section Properties

Documentation Template Controls

Of the six kinds of body sections, only static sections contain controls for producing actual output. Headers and footers can also contain controls. Folders and iterators, which cannot directly contain controls, must have at least one static section nested somewhere within.

When you insert a new control, the Documentation Template Designer displays a dialog box for setting the control's properties. The template shows each control as a shaded rectangle in the details pane. You can change the properties of a control after it is created.

The controls described in this section include:

- Label
- Image
- Panel
- Formula
- Data
- Include Text

Label, Image, and Panel Controls

The simplest kinds of controls are labels, panels, and images.

Label

A **label** generates static text that is independent of its containing section. The text does not depend on the metatype of the section or where the section belongs in the template. Placing identical labels in a header and a static section results in the same output as long as the header and static section are not skipped. Label properties include the label's text, style (font, color, and border), and if and how to hyperlink the output.

Image

Depending on its type, an **image** can be external to the project or it can be a project diagram. You can put an image control in a static section to include an image of a diagram in the generated document. Documentation Generator, while processing the section, will create an image only if the current model element represents a model diagram.

Panel

A **panel** is simply a container for other controls. Panels are convenient for grouping controls together to provide a uniform style and precise alignment. You can set the panel's background color, border, and style, and the parameters that will be passed to the controls within the panel.

Data Controls

Data controls provide the major mechanism of placing data from a project into a report. When a data control is processed, the actual data are obtained from the current model element.

The source of information for a data control can be one of the following:

Element Property	A property of the current element. The Data Control dialog box displays a list of every property belonging to the metatype of the current model element.

Generator’s Variable	A variable used by DocGen. You can use this in report headers or footers to insert the project name or the date and time the report is created.
Document Field	A field of the report such as page number or bookmark. You can select Document Field to insert page numbers and number of pages into page headers or footers. The Document Field list is empty for report headers and footers.

Formula and Text Controls

Formulae provide a way to place data into a report that DocGen calculates when it processes the control. You must enter the formula that DocGen evaluates to calculate that output. Both formula controls and text controls rely on such formulas.

Formula controls

A formula is an expression that Documentation Generator can evaluate to a string. The expression can be a combination of string literals, DG variables, and OCL or legacy RWI functions.

DG variables are special variables that are available to DocGen at runtime when it is producing a report. DG variables include items such as current element, the date and time, and template parameters. Find the complete list of DG variables, OCL functions and legacy RWI functions in the section *Documentation Generator and Template Designer Reference*.

Supported formula types are Legacy and OCL. The syntax depends on the selected formula type, as shown in the following table.

Supported formulae type	Syntax
Legacy	single quotes for string literals; + for string concatenation -> for calls to functions via pointers
OCL	OCL

The following examples demonstrate the usage of formulae expressions for the different formulae types.

Example 1:

From a section with a **class** metatype, put Package followed by the name of the containing package into the report:

Syntax	Formulae expression
Legacy	`"Package " + getContainingPackage() -> getProperty("$name")`

OCL	`context uml14::kernel::classes::Class`
	
	`'Package '.concat(self.getContainingPackage().name)`

Example 2

From a section with a **generic class** metatype, put Interface in the report if the current element is an interface and put Class if it is not.

Syntax	Formulae expression
Legacy	`if (hasProperty("$interface"), "Interface", "Class")`

OCL	`context uml14::kernel::classes::Class`
	
	`if self.interface then 'Interface' else 'Class' endif`

Include Text controls

Include Text controls are used for copying text from other files into a template. When you insert an **Include Text** control, you must enter an expression for the location of the text file. The expression can be hard-coded as a string literal, or it can use a formula as described above. **Include Text** controls have formatting properties identical to those for formula and label controls.

Related Concepts

Documentation Template Sections
Hyperlinks in Documentation

Related Procedures

Creating Controls
Hyperlinking Documentation

Related Reference

Control Properties

Multi-frame Documentation Templates

Multi-frame HTML documentation divides project reports into frames to give multiple views within the same browser window. Multi-frame HTML documentation consists of two kinds of HTML files:

- A collection of HTML files to define the content for each frame
- A frameset file to specify the layout of frames

A frameset template consists of two major parts. One part describes the frameset file that can be defined through the template properties. The other part, which is the body of the frameset template, contains calls to the templates that provide the contents of the frames.

The body of a frameset template is similar to the body of an ordinary document template. A frameset template body can contain any number of iteration sections (element iterators and property iterators), folder sections, and stock section calls. However, static sections and headers and footers for folder sections and iterators are prohibited. Calls to template sections replace static sections to produce the actual output.

The section properties of a call to template determine how the output for a template call can be used. With multi-frame HTML documentation, calls to template sections typically generate separate files that can be loaded into a frame of the resulting HTML project documentation.

When the Documentation Generator processes a frameset template, it produces the frameset HTML file and the separate HTML files for the frame content. The Documentation Generator begins processing a frameset template at its body. When it encounters a call to a template section, the engine suspends the current template execution, loads the called template, and processes it to produce a separate HTML document. The root element for the called template is the current model element of the calling template. After the called template's processing is completed, the Documentation Generator resumes executing the calling template. After the body of the frameset template has completed processing, the Documentation Generator produces the special HTML frameset file. This file corresponds to the frameset structure specified in the template properties. The name of the frameset file matches the name of the frameset template. It is the starting point of the generated documentation.

Related Concepts

Documentation Generation Overview

Related Procedures

A Typical Scenario of Creating a Template for Multi-Frame Documentation

Related Reference

Frameset Template Properties

Hyperlinks in Documentation

A hypertext link connects a link reference (starting point or source) to a link destination (target). The link reference is a text or image in the HTML document. The link destination is a file (usually an HTML document or an anchor in an HTML document). Document templates support both references and targets. Link references are properties of controls. Link targets are properties of static sections, headers, and footers.

Any generated output that contains an anchor or bookmark can be a link target. Documentation templates have facilities for inserting anchors at the “main documentation” of model elements.

It is occasionally necessary to provide link references to several different documents (or locations in HTML files) created with the same model element. For example, along with the main documentation file created for a package, there could be a different HTML document that simply lists all classes in the package. If this listing document were in a separate “navigation” pane, it would serve as an index for the package. Clicking the package on a diagram (or in some more general text) could load that listing document in the navigation frame. The Documentation Template Designer enables you to target different documentation locations generated by the same model element.

Link references in multi-frame documentation can have multiple targets. Clicking on such a reference could simultaneously load two different documents in two different frames. For example, suppose a diagram element represents a package. Clicking on this element could load the image of the package diagram in one frame and the main (textual) documentation for the package in another. Such link references are named *compound*.

Related Procedures

Hyperlinking Documentation

Related Reference

Control Properties

Javadoc Link References

Javadoc References (or JDRefs) are the expressions associated with Javadoc tags such as `{@link}` and `@see.` You can use them to create link references inside documentation text `({@link})` as well as with some other documenting tags. The Documentation Generator can convert JDRefs into real hypertext links. Each JDRef should conform to the rules described in the standard Javadoc documentation. There are three types of Javadoc references.

- An element reference refers to an element of the model (such as method, class, or package). The general form of an element reference is `package.class#member` label, where `package.class#member` is the referenced model element and label is optional text to be displayed with the link. (If label is omitted, the name of the referenced element is displayed.) The Documentation Generator can convert each element reference into a hyperlink to the main documentation of the element.
- URL reference represents a link to a relative or absolute URL. The general form of an URL reference is `<a href="URL#value">label</a>`
- Text reference has the form "string" (a text string in double-quotes). A text reference is simply information that does not represent a hyperlink.

A JDRef appears in one of two forms:

- inside `{@link}` tags embedded in documentation text. The JDRef is the value of the `$doc` property and other Javadoc element’s properties.
- as the value of some Javadoc element’s properties such as `see`.

The Documentation Template Designer provides conversions for both cases. You need to specify the conversion in the properties of the control.

Related Procedures

Creating Javadoc Link References (Advanced)

Related Reference

Control Properties

Enable Conditions

Enable conditions are Boolean expressions for turning section processing on or off. They are created using the OCL or legacy notation.

An enable condition is evaluated before stepping into this section, so the properties of the metatype of a section are not available to the Documentation Generator at the moment of expression evaluating. Enable conditions typically have subexpressions that are calls to special DG functions returning DG options and template parameters. (See the list of DG functions and variables in the *Documentation Generator and Template Designer Reference*.) They can also use the properties of the upper-level section metatype. The results can be joined together with logical operators under the usual precedence rules. The following table shows two examples of the enable conditions in the Legacy and OCL notation:

Legacy	OCL

`getDGVariable('reportScope') != 'current_diagram'`	`context OclAny`
	`getDGVariable('reportScope') <> 'current_diagram'`

`getContainingNode() -> hasProperty ("$interface")`	`context uml::kernel::Element`
	`self.getContainingNode().oclAsType (uml14::kernel::classes::Class).interface`

Related Concepts

OCL Support

Related Procedures

Creating Sections

Related Reference

DG functions in Formulae Expressions
Folder Section Properties
Static Section Properties

Procedures

Procedures

This section provides how-to information for the various areas of software development supported by Together.

In This Section

Together Documentation Templates Procedures
This section provides how-to information on creating and editing custom documentation templates using the Documentation Template Designer.

Interoperability and Migration
Provides how-to information on exchanging model information between the various products of the Together product line.

Getting Started Procedures

This section provides how-to information on configuring Together, working with projects, and more.

In This Section

Using Example Projects
How to use sample projects in Together.

Activating Together Capabilities

To enable or disable all Together capabilities in a category

1. Select Window ▸ Preferences to display the **Preferences** dialog.
2. Expand the **General** item in the tree view and select **Capabilities**. You can enable or disable complete categories in the **Capabilities** area.
3. Click **OK**.

To enable or disable individual Together capabilities

1. Select Window ▸ Preferences to display the **Preferences** dialog.
2. Expand the **General** item in the tree view and select **Capabilities**.
3. Click the **Advanced** button to display the **Advanced Capabilities Settings** dialog. You can expand categories in the **Capabilities** tree view to enable or disable specific Together capabilities. You can also click **Restore Defaults** to restore the default Together capabilities settings, **Enable All** to enable all Together capabilities settings, or **Disable All** to disable all Together capabilities settings.
4. Click **OK**.

Related Concepts

Together Capabilities Activation

Adding a Single Model Element to a Diagram

You can create a single node element using the diagram Palette, or the **New** command of the diagram context menu.

To create a single model element

1. Open a target diagram in the Diagram Editor .
2. On the Palette, click the icon for the element you want to place on the diagram. The button stays down.

   Tip: Icons are identified with tooltips.

3. Click the diagram background in the place where you want to create the new element. This creates the new element and activates the in-place editor for its name.

Tip: Alternatively, you can right-click the diagram background in the Diagram Editor , or the diagram node in the Model Navigator, and choose **New** on the context menu. The submenu displays all of the basic elements that can be added to the current diagram and the **Shortcuts** command.

Related Procedures

Adding Multiple Elements to a Diagram
Creating a Simple Link

Bookmarking Model Elements

Together allows you to bookmark model elements. Bookmarked elements are listed in the **Model Bookmarks** view.

To add or remove a bookmark

1. To add a bookmark, right-click an element on the diagram editor and choose **Model Bookmarks** ▸ **Add Bookmark**.
2. To remove a bookmark, right-click an element on the diagram editor and choose **Model Bookmarks** ▸ **Remove Bookmark**

   Alternatively, you can use the context menu of the **Model Bookmarks** view to remove a bookmark.

To navigate to a bookmarked element

1. Open the **Model Bookmarks** view.
2. Right-click a bookmark and choose either **Show in Model Navigator** or **Select on Diagram**.

Note: Double-click a bookmark in the **Model Bookmarks** view to select the element on the diagram.

Related Reference

Model Bookmarks View

Choosing a Together Perspective

Together changes the user interface according to how you want to work with Together by providing several perspectives. By default, Together starts with the Modeling perspective.

Note: The way you have Together configured influences which perspectives and views you can choose from. For example, clicking the **Take me to the DSL workbench** link from the Welcome page displays only the DSL Workbench's default perspectives. In order to display a list of all the perspectives, check the **Show all** check box in the **Open Perspective** dialog box. An additional **Confirm Enablement** dialog box might appear that requires you to enable any necessary capabilities.

To choose a Together Perspective

1. On the main menu, choose **Window** ▸ **Open Perspective** ▸ **Other**. The **Select Perspective** dialog box opens.
2. Select one of the Together Perspectives from the list and click **OK**.

After you select a perspective, Together automatically customizes the interface to provide ready access to only the relevant elements of the interface, and to show only the information in the model that best supports the chosen perspective. Interface elements and/or model information that are not generally relevant to the perspective are hidden. You can still access hidden information by changing the relevant configuration options and restoring hidden panes manually, but you may find it easier to just switch perspectives.

Related Concepts

Together Capabilities Activation
Tour of Together

Related Procedures

Activating Together Capabilities

Configuring Together Preferences on the Workspace and Diagram Levels

You can flexibly change configuration of Together. Use the **Preferences** dialog box to tune modeling features to best fit your requirements.

The **Preferences** dialog window provides a number of diagram customization settings. You can configure the appearance and layout of the diagrams, specify font properties, member format, and level of detail on the diagram and workspace levels.

To configure Together settings on the workspace level

1. On the main menu, choose **Window** ▸ **Preferences**.
2. In the **Preferences** dialog window, expand the **Modeling** category.
3. Click the desired subcategory.
4. Edit configuration options as required.
5. Click **OK** to apply changes and close the dialog window.

You can configure certain diagram-specific options (Diagram, Layout, View management and Print) on the diagram level.

To enable configuration changes on the diagram level

1. On the main menu, choose **Diagram** ▸ **Preferences**.
2. Set the checkbox **Enable diagram-specific settings**.
3. Click the desired subcategory (Diagram, Layout, View management and Print).
4. Edit configuration options as required.
5. Click **OK** to apply changes and close the dialog window.

To disable configuration changes on the diagram level

1. On the main menu, choose **Diagram** ▸ **Preferences**.
2. Clear the checkbox **Enable diagram-specific settings**.
3. Click **OK** to apply changes and close the dialog window.

Related Reference

Together Preferences

Creating a Browse-Through Sequence of Diagrams

You can link entire diagrams at one level of detail to the next diagram up or down in a sequence of increasing granularity, or you can link from key use cases or actors to the next diagram.

To create a browse-through sequence

1. Open the main diagram of the sequence you are going to create.
2. Select the source model element, or right-click the diagram background to link the entire diagram.

   Tip: It is recommended that you use some common approach for all links in your sequence.

3. Create a hyperlink to the next diagram or model element you would like to participate in the sequence. The titles of source and destination model elements turn blue.
4. Open the destination diagram.
5. Repeat steps 3–5 for all parts of your sequence.
6. Optionally, create hyperlinks in the reverse motion.

Related Concepts

Model Hyperlinking Overview

Related Reference

UML 1.4 Use Case Diagrams
UML 2.0 Use Case Diagrams

Creating a Diagram

Diagrams exist within the context of a project. Create or open a project before creating any new diagrams.

To create a new diagram from the Model Package Explorer

1. In the Model Package Explorer view, right-click on a package or the project root.
2. From the context menu, select **New Diagram**. The New Diagram Wizard displays. See **To create a diagram using the New Diagram Wizard** below for more information.

Alternatively, right-click the default diagram of a source package, select **New Diagram**, and choose the diagram type from the submenu.

To create a new diagram from the Model Navigator

1. In the Model Navigator, right-click on a package or the project root.
2. From the context menu, select **New Diagram** and choose the diagram type from the submenu.

To create a new diagram using the Diagram Editor toolbar

1. Click the arrow to the right of the **New Diagram** icon on the diagram editor toolbar.
2. Choose the diagram type from the submenu.

   or

   1. Click directly on the **New Diagram** icon. The UML Diagram dialog opens.
   2. In the resulting dialog, select a diagram type from the drop down list. Select the package where the new diagram will be created. Click **Browse** to choose a package. Enter a name for the new diagram.
   3. Click **Finish**.

To create a new diagram using the New Diagram Wizard

1. Select **File** ▶ **New** ▶ **Diagram**.
2. Specify the properties for the new diagram as follows:
   - **Location:** By default, the new diagram is created in the package selected before the wizard displays.
   - **Type:** Use the drop down list to select a diagram type. By default, the Class Diagram is selected.
   - **Name:** Use the text field to type a name for the new diagram.
3. Click **Finish**.

To create a class diagram from a package diagram

1. On the package diagram, select classes that you would like to display in a separate class diagram.
2. On the main menu, select **Model** ▶ **Generate Class Diagram**.

Note: This action is available only when several classes are selected.

Creating a Project

Together provides several projects that you can work with. The projects in Together are created in the same manner. While creating a project, you will specify different options depending on the type of project.

To create a Together Project

1. Select **File** ► **New** ► **Project** on the main menu. The **New Project** wizard displays.
2. Expand the **Modeling** node in the tree view list, and select the type of project you want to create. Click **Next**.
3. Follow the wizard steps to specify necessary options for a new project and click **Finish** to complete the wizard.

Related Concepts

Together Project Overview

Related Procedures

Together Projects

Related Reference

Together Projects

Creating a Shortcut

You can create a shortcut to a model element from the current project or from the projects connected by cross-projects references, by using three methods:

- By choosing **New Shortcut** on the Diagram Editor context menu
- By dragging and dropping a shortcut from the Model Navigator
- By choosing **Add as Shortcut** on the Model Navigator context menu

To create a shortcut by using the Shortcuts dialog window

1. Right-click the diagram background.
2. Choose **New** ▸ **Shortcuts** on the context menu.

   Tip: Use the CTRL+SHIFT+N keyboard shortcut

3. In the **Shortcuts** dialog window, choose the required element from the tree view of available contents.

   Note: If the project has cross-project references to the other projects in the workspace, the contents of these projects is available for being added as a shortcut.

4. Click **Add** to place the selected element to the list of the existing or ready-to-add elements.
5. When the list of ready-to-add elements is complete, click **OK**.

To create a shortcut by using drag-and-drop

1. Select the element in the Model Navigator.
2. Drag-and-drop the element onto the diagram.

To create a shortcut by using the Model Navigator context menu

1. Open the diagram where the shortcut will be added.
2. In the Model Navigator, select the element to be added to the current diagram as a shortcut.
3. Right-click the element in the Model Navigator and choose **Add as Shortcut** on the context menu.

Related Concepts

Model Shortcut Overview

Related Procedures

Establishing cross-project references
Adding a Single Model Element to a Diagram

Creating a Simple Link

You can create a link to another node, or a shortcut of an element of the same or another project (these projects must be of the same UML version).

To create a simple link between two nodes

1. On the diagram Palette, click the button for the type of link you want to draw in the diagram. The button stays down.
2. Click the source element.
3. Drag to the destination element and drop when the target element is highlighted.

Related Procedures

Rerouting a Link
Creating a Link with Bending Points
Creating Model Element by Pattern

Related Reference

Class Diagram Relationships

Deleting a Diagram

Warning: You cannot delete the default diagram created automatically for a package.

To delete a diagram

1. In the **Package Explorer**, select the diagram to be deleted.
2. On the context menu, choose **Delete**.
3. Confirm deletion, if required.

Related Procedures

Creating a Diagram
Closing a Diagram

Deleting Elements

All elements shown on diagrams are shortcuts to actual model elements. When deleting an element on a diagram, you have the option to delete either the shortcut from view or delete the element from the model (except classes on synchronized package diagrams). This behavior is configured in the Modeling Preferences.

To delete an element

1. Select the element on the diagram.
2. Choose **Delete** on the context menu of the element.

   Tip: Alternatively, click the DELETE key.

3. Confirm deletion, if this behavior is selected in the Modeling Preferences.

To delete an element from View

1. Select the element on the diagram.
2. Choose **Delete from View** on the context menu of the element.
3. Confirm deletion, if this behavior is selected in the Modeling Preferences.

Related Procedures

Adding a Single Model Element to a Diagram

Related Reference

Modeling Preferences

Hiding and Showing Model Elements

You can control the visibility of elements on a diagram by using the **Hide** command (available on the context menu for individual diagram elements) and the **Show/Hide** command (available on the diagram context menu).

To hide elements using the Diagram Editor

1. Open the Diagram Editor .
2. Do one of the following:
   - Select the element on the diagram, right-click and choose **Hide** on the context menu.
   - Select multiple elements on the diagram using CTRL+CLICK or by lassoing, and select **Hide** from the context menu.
   - Right-click the diagram background and choose **Hide/Show** on the context menu. The **Show Hidden** dialog box opens, as discussed below.

To show or hide diagram elements using the Show Hidden dialog box

1. Right-click the diagram and choose **Show/Hide** on the context menu. The **Show Hidden** dialog box opens.
2. Select the element(s) that you want to hide from the Diagram Elements list.
3. To add elements in the Diagram Elements list to the Hidden Elements list, do one of the following:
   - Double-click the element.
   - Click the element once and click Add.
   - Select multiple elements using CTRL+CLICK and click Add.
4. To remove items from the Hidden Elements list, do one of the following:
   - Double-click the element.
   - Click the element once and click **Remove**.
   - Select multiple elements using CTRL+CLICK and click **Remove**.
   - To remove all items from the Hidden Elements list, click **Remove All**.
5. Click **OK** to close the dialog box.

Related Procedures

Adding a Single Model Element to a Diagram

Related Reference

View Management Preferences

Opening a Diagram

You can open diagrams from the Model Navigator, Model Package Explorer, or by using the Diagram Editor toolbar.

In this section you will learn how to:

- open an existing diagram
- cancel a diagram's opening process

To open a diagram

1. In the Model Navigator view, navigate to the diagram you want to open.
2. Select the diagram node in the tree-view and do one of the following:
   - Double-click the diagram
   - Select **Open** from the context menu
   - Select **Open in Active Editor** from the context menu (this replaces the contents of any currently opened diagram)

You can use the Diagram Editor toolbar to open the parent diagram.

To terminate a diagram opening

1. Open diagram as described above.
2. In the **Diagram opening** information dialog, click **Stop**.

The Diagram Editor displays the diagram with those elements that have been loaded before termination. Under the diagram title, a message appears informing the user that the diagram contents were only partially loaded.

Note: The layout of a diagram may get adjusted automatically when opened. This happens in order to take into account changes which may have happened to the model elements shown on the diagram, or to the diagram preferences controlling the elements presentation while the diagram was closed (off-line).

For example: Assume the diagram has a class and a link coming from its bottom edge to some other class shown below. While the diagram was closed the class may have had a few members removed or the font size set to be smaller. Any of these changes would cause the height of the class to get smaller. That in turn needs the position of the link end to be adjusted. Such a change would normally cause a diagram file to be resaved.

If the corresponding diagram file is under the version control that would cause an outgoing change, which may be unwanted if the diagram was opened without the intention to edit it during the current session, but rather simply to view and close.

With Together 2008 R3 an option is provided that allows user control whether or not such changes should be saved (**Windows ▸ Preferences ▸ Modeling ▸ Diagram ▸ If layout is changed on diagram open**). **Save** is the default state, which mimics previous Together behavior: in this case the changes are silently saved. Other choice is **Ignore**. With this the changes will not be saved immediately, but will be indicated with a '*' at the name shown on the corresponding diagram editor Tab. If the diagram or underlying model has no other changes intentionally made from UI, then when it is closed the changes are dropped. Note that the '*' marker will disappear if further edits are made to the diagram as it indicates that only automatic changes have been made; once intentional changes are made the marker isn't needed as the automatic changes will be saved as well.

Related Procedures

Opening a Parent Diagram

Related Reference

Diagram Preferences

Opening a Diagram Element in the Source Code Editor

Based on the LiveSource technology, you can open elements for editing and synchronize your model with the source code.

In this section you will learn how to:

- Open a source-generating element in the Editor
- Enable synchronization between diagram and source code.

To open a source-generating element in the Editor

1. Right-click an element in the diagram.
2. On the context menu, choose **Open**.

Tip: Alternatively, press F3 or just double-click the element.

To enable synchronizing source code with the model

1. On the main menu, choose **Window** ▶ **Preferences** ▶ **Modeling** ▶ **Diagram**.
2. Check the option **Synchronize source code editor to diagram**.

Tip: Alternatively, just click **Link with Editor** button on the diagram toolbar.

If the source code editor is opened for a class and a class member is selected, the editor for this class gets focus, and the member is selected in the source.

Related Concepts

Roundtrip Engineering Overview

Printing Diagrams

To print a diagram or multiple diagrams

1. Open the diagram or select the tab in the Diagram Editor that displays the diagram.
2. Select **File** ▸ **Print** on the main menu. The **Print Diagram** dialog box is displayed.
3. Select the scope for printing.
4. If you check the option **Print whole diagram as an image**, the **Print diagram as black and white image** option becomes enabled. Make your selections.
5. Click the **Preview >>>** button to see how the diagram or diagrams look with the current print settings.
6. Click the drop-down arrow to set the **Preview Scale** factor.
7. Click the **Print Options** button to define Together Print Preferences. You can use the scroll bars to scroll around the diagram or to view other diagrams that were included in the scope.
8. Click **Print** to proceed to the standard print dialog box where you can select your printer.

Note: There is a known issue that the Java print library is not able to update the standard printer settings. Be sure to check the paper and orientation and set them, if needed, to the settings in the Together Print Preferences.

Related Procedures

Print Preferences

Reusing Existing Source Code in Modeling Projects

Together allows you to convert your existing source code to UML models. Together provides two ways to use reverse engineering:

- Convert the existing source while creating a new project
- Import the existing source code to a Java Modeling project

To import Java source code while creating a new project

1. On the main menu, choose **File** ▸ **New** ▸ **Project**.
2. Expand the **Modeling** node and select **Java Modeling Project**. Click Next
3. On the **Java Modeling Project** page, type the project's name. Click Next
4. On the **Modeling Settings** page, choose the desired metamodel (UML 2.0 or UML 1.4, UML 2.0 is default). Uncheck the **Store package properties in package diagram files** if you like them being stored in txaPackage files. Check **Create design elements in separate files** if you like to have each model element stored in its own txa* file. Click Next
5. Skip the **Profiles** page unless you like to enable one or more profiles for your project. Click Next
6. On the **Java Settings** page, **Source** tab, click the **Link Additional Source to Project** button in the upper right toolbar, use the **Browse...** button to specify the path to the existing source code folder. Click **Finish**. If you like this linked folder to be the only source folder for your project, remove the default source folder using **Remove** button.
7. Click **Finish**.

To import source code to the existing modeling project

1. Right-click a source folder of the target Java Modeling project in the **Navigator** view and select **Import...** on the context menu.
2. In the **Import** wizard, select **General** ▸ **File System** and click **Next**.
3. Browse to the folder with source code to import.
4. Select sources you want to import or click **Select all** to import the entire folder.
5. Click **Finish**.

Related Procedures

Creating a Project

Selecting Model Elements

Most manipulations with diagram elements and links involve dragging the mouse or executing context menu commands on the selected elements.

In this section you will learn how to:

- Select one or more elements
- Cancel selection

To select an element

1. Open a diagram in the Diagram Editor .
2. On the diagram Palette, click the **Select** button.
3. In the Diagram Editor , click any element or a member to select it.

To select multiple elements, do one of the following

1. Hold down the CTRL key and click each element individually, OR
2. Click the background and drag a lasso around an area to select all the elements it contains, OR
3. Press CTRL+A to select all elements on a diagram, OR
4. Right-click the diagram background and choose **Select All** on the context menu.

Note: To cancel a selection, press the ESC key.

Related Procedures

Aligning Model Elements

Related Reference

Together Keyboard Shortcuts

Using Drag-and-Drop

Drag-and-drop applies to the members as well as to the node elements. You can move or copy members (methods, fields, properties, and so on) by using drag-and-drop in the Diagram Editor or in the Model Navigator. You can also change the origin and destination for links on your diagrams using drag-and-drop.

Drag-and-drop functionality from the Model Navigator to the Diagram Editor and within the Model Navigator works as follows:

- Selecting an element in the Model Navigator and using drag-and-drop to place the element onto the diagram creates a shortcut.
- Using drag-and-drop while pressing the SHIFT key moves the element to the selected container.
- Using drag-and-drop while pressing the CTRL key copies the element to the selected container.

To move a link to a new destination

1. Select a link in the Diagram Editor .
2. Hover the cursor over the destination arrow.
3. Drag the arrow and drop it on the new destination. If the destination element is not in view, drag the link in the appropriate direction, and the diagram will scroll with you.

Tip: Follow the same instructions to move the link source to an allowable location.

Related Procedures

Selecting Model Elements
Moving Model Elements

Related Reference

Together Keyboard Shortcuts

Using Example Projects

Together comes with a set of predefined sample projects.

To use a Together Example Project

1. Select **File ▸ New ▸ Project** on the main menu. The **New Project** wizard opens.
2. Expand the **Examples** node in the tree view list, and select the project you want. Click **Next**.
3. Follow the wizard steps to specify the necessary options for a new project and click **Finish** to complete the wizard.

Tip: Alternatively, choose **File ▸ New ▸ Example** on the main menu.

Related Concepts

Together Project Overview

Related Procedures

Together Projects

Related Reference

Together Projects

Diagrams

This section describes how to create Together diagrams, customize their appearance, and populate diagrams with elements and shortcuts.

In This Section

Common Diagrams Procedures

This section describes procedures that apply to all types of diagrams.

In This Section

Annotating a Diagram

Use the following actions to annotate a diagram:

1. Draw a note
2. Draw a note link
3. Type comments

To draw a note

1. In the Diagram Editor , you can:
   - Hyperlink the note to another diagram or element.
   - Edit the text when its in-place editor is active.
   - Edit the properties of a note using Properties View.

2. In the Properties View for the note, you can:
   - Edit the text.
   - Change the foreground and background colors.
   - Change the text-only property.

To draw a note link

1. Click the **Note Link** button on the Palette.
2. In the Diagram Editor , click the source element.
3. Drag the link to the destination element.
4. Drop when the second element is highlighted.

Tip: You can use the Properties View to view both the client and supplier sides of the link.

To enter comments

1. To enter comments in the source code, use the **Comment** fields (Author, Since, Version) in the Properties View for the class.
2. You can also enter source code comments directly into the code using the Editor.

Related Concepts

Model Annotation Overview

Related Procedures

Adding a Single Model Element to a Diagram
Creating a Shortcut

Browsing a Diagram with Overview Pane

To open the Overview pane

1. Open a diagram and click the **Overview** button. The pane expands to show a thumbnail image of the current diagram.
2. Click the shaded area and drag it. This is a convenient way to scroll around the diagram.
3. Resize the **Overview** pane by clicking the upper-left corner of the pane and dragging it.
4. Close the **Overview** pane by clicking the diagram.

Related Procedures

Zooming a Diagram

Changing the Default Diagrams Directory

By default, Together diagram files are contained within the default design root folder, which is called the Model Folder.

To change the default diagrams directory

1. Right-click the project root in the Model Navigator, or Model Package Explorer view, and select **Properties**. The **Properties** dialog box displays.
2. Choose **Design root path** from the **properties** list on the left.
3. Specify the path in the **Design root path** field, and press OK.

Warning: The path name can contain only the folder name that the existing design root will be renamed to, not the path to the folder.

Closing a Diagram

To close a diagram

1. Switch to the Diagram Editor .
2. Click the cross icon to close the current view.

Tip: Alternatively, choose **File ▸ Close** on the main menu, or CTRL+W.

Note: Closing a diagram does not remove it from your project.

Related Concepts

Together Diagram Overview

Creating a Diagram

Diagrams exist within the context of a project. Create or open a project before creating any new diagrams.

To create a new diagram from the Model Package Explorer

1 In the Model Package Explorer view, right-click on a package or the project root.
2 From the context menu, select **New Diagram**. The New Diagram Wizard displays. See **To create a diagram using the New Diagram Wizard** below for more information.

Alternatively, right-click the default diagram of a source package, select **New Diagram**, and choose the diagram type from the submenu.

To create a new diagram from the Model Navigator

1 In the Model Navigator, right-click on a package or the project root.
2 From the context menu, select **New Diagram** and choose the diagram type from the submenu.

To create a new diagram using the Diagram Editor toolbar

1 Click the arrow to the right of the **New Diagram** icon on the diagram editor toolbar.
2 Choose the diagram type from the submenu.

or

   1 Click directly on the **New Diagram** icon. The UML Diagram dialog opens.
   2 In the resulting dialog, select a diagram type from the drop down list. Select the package where the new diagram will be created. Click **Browse** to choose a package. Enter a name for the new diagram.
   3 Click **Finish**.

To create a new diagram using the New Diagram Wizard

1 Select **File** ▸ **New** ▸ **Diagram**.
2 Specify the properties for the new diagram as follows:

   - **Location:** By default, the new diagram is created in the package selected before the wizard displays.
   - **Type:** Use the drop down list to select a diagram type. By default, the Class Diagram is selected.
   - **Name:** Use the text field to type a name for the new diagram.

3 Click **Finish**.

To create a class diagram from a package diagram

1 On the package diagram, select classes that you would like to display in a separate class diagram.
2 On the main menu, select **Model** ▸ **Generate Class Diagram**.

Note: This action is available only when several classes are selected.

Deleting a Diagram

Warning: You cannot delete the default diagram created automatically for a package.

To delete a diagram

1. In the **Package Explorer**, select the diagram to be deleted.
2. On the context menu, choose **Delete**.
3. Confirm deletion, if required.

Related Procedures

Creating a Diagram
Closing a Diagram

Exporting a Diagram to an Image

To export a diagram to an image

1 Place the focus on the diagram that you want to export in the Diagram Editor .

2 Choose **File** ▸ **Export** on the main menu. The **Export** wizard opens.

3 In the **Select** page of the wizard, choose **Modeling** ▸ **Image (GIF, JPEG, Bitmap, EMF, SVG)**, and click **Next**.

4 In the **Export to Image** page, specify the following settings:

- **Destination file**: enter the fully qualified name of the resulting file, or click the Browse button and navigate to the desired location.
- **Diagrams scope**: click a radio button to select the diagrams or diagram elements to be exported.
- **Format**: select the desired format from the drop-down list.
- **Scale**: enter magnification factor.
- **Export heading**: check the option to save the image together with the diagram title.
- **Open in viewer**: check the option to launch the default image viewer.

Click **Next** to preview, or **Finish** to complete the export.

Related Concepts

Model Import and Export Overview

Related Reference

Export Diagram to Image Wizard

Hyperlinking Diagrams

Select Hyperlinks from the diagram context menu to create, view, remove, and browse hyperlinks.

Use the following techniques to create a hyperlink

1. Create a hyperlink to an existing diagram or element
2. Create a hyperlink to a new diagram
3. Create a hyperlink to an external URL or file
4. Browse hyperlinks
5. Remove a hyperlink

To create a hyperlink to an existing diagram or element

1. Open an existing diagram or create a new diagram from which to create the hyperlink.
2. Select the element that you want to link to another diagram or element.
3. To link the entire diagram, click the diagram background to deselect all elements.

   Note: Do not select the actual package in the Model Navigator to create a hyperlink. Rather, expand the package node, and select the desired diagram.

4. Right-click and choose **Hyperlinks** ▸ **Edit**. The **Edit Hyperlinks** dialog window (Selection Manager) opens.
5. Select the Model Elements tab to view the pane containing a tree view of the available project contents.
6. Select the diagram or element you want from the list, and click **Add**.
7. For element selection, expand diagram nodes in the **Model Elements** tab.
8. To remove an element from the selected list, select the element and click **Remove**.
9. Click **OK** to close the dialog box and create the link.

To create a hyperlink to a new diagram

1. Open a diagram in the Diagram Editor , or select it in the Model Navigator.
2. On the context menu, choose **Hyperlinks** ▸ **New Diagram**.
3. In the **New Diagram** dialog box, select the diagram type, enter the diagram name and click OK.

To create a hyperlink to an external URL or file

1. Open an existing diagram or create a new diagram from which to create the hyperlink.
2. Select the element that you want linked to the external document.

   To link the entire diagram, click the diagram background to deselect all elements.
3. Right-click and choose **Hyperlinks** ▸ **Edit**. The **Edit Hyperlinks** dialog box opens.
4. Select the **External Documents** tab to view the Recently Used Documents list which contains a list of previously selected files or URLs.
5. To add a file to the Recently Used Documents list:
   1. Click **Browse**. The **Open file** dialog box opens.

    2 Navigate to the appropriate file and click **Open**.

6 To add a URL to the Recently Used Documents list:

    1 Click URL.
    2 In the dialog box that opens, enter the appropriate URL and click **OK**.

Tip: You can create a hyperlink to an external document by entering a relative URL path.

7 To remove an element from the selected list, select the element and click **Remove**.
8 To clear the Recently used Documents list, click **Clear**.
9 Click **OK** to close the dialog box and create the link.

To browse hyperlinks

1 To view hyperlinks to a diagram, element or external document, right-click on the diagram background or element, and choose **Hyperlinks** from the context menu. All hyperlinks created appear under the **Hyperlinks** submenu. On a diagram, all names of diagram elements that are hyperlinked are displayed in blue font. When you select a link from the submenu, the respective element appears selected in the Diagram Editor .
2 After you have defined hyperlinks for a selected diagram or element, use the context menus to browse to the linked resources.

Note: Browsing to a linked diagram opens it in the Diagram Editor or makes it the current diagram (if it is already open).

Browsing to a linked element causes its parent diagram to open or become current, and the diagram scrolls to the linked element and selects it.

To remove a hyperlink

1 Open the diagram that displays the link you want to remove.
2 Choose **Hyperlinks** ▸ **Edit** from the diagram or element context menu. The **Edit Hyperlinks** dialog box opens.
3 In the selected list on the right of the dialog, click the hyperlink that you want removed.
4 Click **Remove**.
5 Click **OK** to close the dialog box.

Note: To remove a hyperlink from a specific element, select the element first. Then choose **Hyperlinks** ▸ **Edit** on the context menu.

Related Concepts

Model Hyperlinking Overview

Opening a Diagram

You can open diagrams from the Model Navigator, Model Package Explorer, or by using the Diagram Editor toolbar.

In this section you will learn how to:

- open an existing diagram
- cancel a diagram's opening process

To open a diagram

1. In the Model Navigator view, navigate to the diagram you want to open.
2. Select the diagram node in the tree-view and do one of the following:
   - Double-click the diagram
   - Select **Open** from the context menu
   - Select **Open in Active Editor** from the context menu (this replaces the contents of any currently opened diagram)

You can use the Diagram Editor toolbar to open the parent diagram.

To terminate a diagram opening

1. Open diagram as described above.
2. In the **Diagram opening** information dialog, click **Stop**.

The Diagram Editor displays the diagram with those elements that have been loaded before termination. Under the diagram title, a message appears informing the user that the diagram contents were only partially loaded.

Note: The layout of a diagram may get adjusted automatically when opened. This happens in order to take into account changes which may have happened to the model elements shown on the diagram, or to the diagram preferences controlling the elements presentation while the diagram was closed (off-line).

For example: Assume the diagram has a class and a link coming from its bottom edge to some other class shown below. While the diagram was closed the class may have had a few members removed or the font size set to be smaller. Any of these changes would cause the height of the class to get smaller. That in turn needs the position of the link end to be adjusted. Such a change would normally cause a diagram file to be resaved.

If the corresponding diagram file is under the version control that would cause an outgoing change, which may be unwanted if the diagram was opened without the intention to edit it during the current session, but rather simply to view and close.

With Together 2008 R3 an option is provided that allows user control whether or not such changes should be saved (**Windows** ▸ **Preferences** ▸ **Modeling** ▸ **Diagram** ▸ **If layout is changed on diagram open**). **Save** is the default state, which mimics previous Together behavior: in this case the changes are silently saved. Other choice is **Ignore**. With this the changes will not be saved immediately, but will be indicated with a '*' at the name shown on the corresponding diagram editor Tab. If the diagram or underlying model has no other changes intentionally made from UI, then when it is closed the changes are dropped. Note that the '*' marker will disappear if further edits are made to the diagram as it indicates that only automatic changes have been made; once intentional changes are made the marker isn't needed as the automatic changes will be saved as well.

Related Procedures

Opening a Parent Diagram

Related Reference

Diagram Preferences

Opening a Diagram Element in the Source Code Editor

Based on the LiveSource technology, you can open elements for editing and synchronize your model with the source code.

In this section you will learn how to:

- Open a source-generating element in the Editor
- Enable synchronization between diagram and source code.

To open a source-generating element in the Editor

1. Right-click an element in the diagram.
2. On the context menu, choose **Open**.

Tip: Alternatively, press F3 or just double-click the element.

To enable synchronizing source code with the model

1. On the main menu, choose **Window** ▶ **Preferences** ▶ **Modeling** ▶ **Diagram**.
2. Check the option **Synchronize source code editor to diagram**.

Tip: Alternatively, just click **Link with Editor** button on the diagram toolbar.

If the source code editor is opened for a class and a class member is selected, the editor for this class gets focus, and the member is selected in the source.

Related Concepts

Roundtrip Engineering Overview

Opening a Parent Diagram

You can open a parent diagram from the Diagram Editor toolbar.

To open the parent diagram

1. Click the **Open Parent Diagram** button to open the parent of the active diagram.
2. If a diagram has no parent, the button is disabled.

Related Procedures

Opening a Diagram

Printing Diagram Elements

To print one or more diagram elements

1. Open the diagram or select the tab in the Diagram Editor that displays the diagram containing the diagram elements you want to print.
2. Right-click the diagram element or multiple elements, and select **Print** from the context menu. The **Print Diagram** dialog box displays.
3. When you select the **Print whole diagram as an image** option, it enables the **Print diagram as black and white image** option. Make your selections.
4. At this point you can click **Preview >>>** button to see how the selected diagram element or elements look with the current print settings. You can click the **Print Options** button to set up Together Print Preferences.
5. Click the drop-down arrow to choose the **Preview Scale** factor from the list of available scales to best fit the printed image on the page.
6. Click **Print** to proceed to the standard print dialog where you can select your printer.

Related Procedures

Print Preferences

Printing Diagrams

To print a diagram or multiple diagrams

1. Open the diagram or select the tab in the Diagram Editor that displays the diagram.
2. Select **File** ▶ **Print** on the main menu. The **Print Diagram** dialog box is displayed.
3. Select the scope for printing.
4. If you check the option **Print whole diagram as an image**, the **Print diagram as black and white image** option becomes enabled. Make your selections.
5. Click the **Preview >>>** button to see how the diagram or diagrams look with the current print settings.
6. Click the drop-down arrow to set the **Preview Scale** factor.
7. Click the **Print Options** button to define Together Print Preferences. You can use the scroll bars to scroll around the diagram or to view other diagrams that were included in the scope.
8. Click **Print** to proceed to the standard print dialog box where you can select your printer.

Note: There is a known issue that the Java print library is not able to update the standard printer settings. Be sure to check the paper and orientation and set them, if needed, to the settings in the Together Print Preferences.

Related Procedures

Print Preferences

Searching Model Elements

Together enables you to use its search facilities to locate model elements on model diagrams. This function enables you to search the current diagram or all opened diagrams for the specified string in a certain scope. You can create search strings using wildcards and regular expressions. The function is case-sensitive.

To find model elements that fall under specified criteria, perform the following steps

1. On the main menu, choose **Search** ▸ **Model**. The Search dialog box opens, with the **Model Search** tab selected.
2. Specify the search string in the **Search String** field. Check the following options if necessary:
   - **Case sensitive**: Searches for text that matches uppercase and lowercase characters
   - **Regular expression**: Enables using regular expressions.
3. In the **Search for** section, click the appropriate radio button to select the name or any other property to search for.
4. In the **Scope** section, click the appropriate radio button to select the search area. The possible options are workspace, selected resources, the current project or a predefined working set.

   To select a working set, click the **Choose** button. In the **Select Working Set** dialog, choose your working set and click **OK**. If there are no available working sets, use the **New** button to create one.
5. Click **Search**.

Related Procedures

Searching Model with OCL queries

Searching Model with OCL queries

Together lets you search for models using OCL queries.

To find model elements that match the specified OCL query

1 On the main menu, choose **Search** ▸ **Model**.

The **Search** dialog box is displayed.

2 Click the **OCL Model Search** tab.

3 Specify the context for your expression in the **Context** field.

Tip: Use the drop-down list or the Content Assistant. To open the Content Assistant, click on the **Context** field and press CTRL +SPACE. Choose your element from the list.

For example, to search for all UML 2.0 classes that have the stereotype `MyStereotype`, enter:

```
uml20::classes::Class
```

4 In the **Invariant** field, type the query expression.

For example, to complete your search for all UML 2.0 classes that have the stereotype `MyStereotype`, enter:

```
self.stereotypes->includes('MyStereotype')
```

5 In the **Scope** section, click the appropriate radio button to select the search area. The possible options are workspace, selected resources, the current project or a predefined working set.

To select a working set, click the **Choose** button. In the **Select Working Set** dialog, choose your working set and click **OK**. If there are no available working sets, use the **New** button to create one.

6 Click **Search**.

A tree with the list of matching elements opens. You can navigate to the corresponding diagram from this view by double-clicking the selected element.

Related Concepts

OCL Support

Related Procedures

Searching Model Elements

Customizing Appearance of Together Diagrams

In This Section

Hiding and Showing Model Elements

You can control the visibility of elements on a diagram by using the **Hide** command (available on the context menu for individual diagram elements) and the **Show/Hide** command (available on the diagram context menu).

To hide elements using the Diagram Editor

1. Open the Diagram Editor .
2. Do one of the following:
   - Select the element on the diagram, right-click and choose **Hide** on the context menu.
   - Select multiple elements on the diagram using CTRL+CLICK or by lassoing, and select **Hide** from the context menu.
   - Right-click the diagram background and choose **Hide/Show** on the context menu. The **Show Hidden** dialog box opens, as discussed below.

To show or hide diagram elements using the Show Hidden dialog box

1. Right-click the diagram and choose **Show/Hide** on the context menu. The **Show Hidden** dialog box opens.
2. Select the element(s) that you want to hide from the Diagram Elements list.
3. To add elements in the Diagram Elements list to the Hidden Elements list, do one of the following:
   - Double-click the element.
   - Click the element once and click Add.
   - Select multiple elements using CTRL+CLICK and click Add.
4. To remove items from the Hidden Elements list, do one of the following:
   - Double-click the element.
   - Click the element once and click **Remove**.
   - Select multiple elements using CTRL+CLICK and click **Remove**.
   - To remove all items from the Hidden Elements list, click **Remove All**.
5. Click **OK** to close the dialog box.

Related Procedures

Adding a Single Model Element to a Diagram

Related Reference

View Management Preferences

Using a Class Diagram as a View

Class diagrams can also be used to create subviews of the project.

To use a class diagrams as a view

1. Create a new class diagram.
2. Create shortcuts to the original diagram to easily and quickly build subset views for easier management.

Tip: Using this feature, you can create views of distributed classes into one diagram, with Together automatically displaying any relationships that the gathered classes may have with each other.

Note: In implementation projects, changes made here also update the source code, keeping diagram and source code in sync.

Related Concepts

Roundtrip Engineering Overview

Related Reference

UML 1.4 Class Diagrams
UML 2.0 Class Diagrams

Zooming a Diagram

Use the zooming commands of the main menu, or toolbar buttons, to obtain the required magnification in the Diagram Editor .

To specify the magnification in the Diagram Editor

1. On the main menu, choose **Diagram**.
2. Choose one of the available zooming commands on the menu: **Zoom In, Zoom Out, Fit to Window, Actual Size**.

Tip: Alternatively, use the diagram Palette or keyboard shortcuts.

Related Reference

Together Keyboard Shortcuts

Populating Together Diagrams

This topic provides How-To information about creating node elements, links and members in all types of Together diagrams.

In This Section

Adding a Member to a Container

You can add members to class diagram elements (containers) by using the respective context menu for the diagram element in the Diagram Editor or Model Navigator, or by using the available shortcut keys.

To add a member to a container

1. Right-click the desired container element.
2. On the context menu, choose **New** ▸ **<Member type>**, where the Member type corresponds to the target container.

   Tip: You can also use keyboard shortcuts to add fields and methods to a container that allows such members. Click CTRL+W (for fields) and CTRL+M (for methods and functions).

3. You can edit the member using the in-place editor, Properties View, or source code editor.

Related Procedures

Adding a Single Model Element to a Diagram

Related Reference

UML 1.4 Class Diagrams Procedures
UML 2.0 Class Diagrams

Adding a Single Model Element to a Diagram

You can create a single node element using the diagram Palette, or the **New** command of the diagram context menu.

To create a single model element

1. Open a target diagram in the Diagram Editor .
2. On the Palette, click the icon for the element you want to place on the diagram. The button stays down.

   Tip: Icons are identified with tooltips.

3. Click the diagram background in the place where you want to create the new element. This creates the new element and activates the in-place editor for its name.

Tip: Alternatively, you can right-click the diagram background in the Diagram Editor , or the diagram node in the Model Navigator, and choose **New** on the context menu. The submenu displays all of the basic elements that can be added to the current diagram and the **Shortcuts** command.

Related Procedures

Adding Multiple Elements to a Diagram
Creating a Simple Link

Adding Multiple Elements to a Diagram

You can place several elements of the same type on a diagram without returning to the Palette or by using the diagram context menu. Each element will have a default name that can be edited with the in-place editor or in the Properties View.

To create multiple elements

1. Holding down the CTRL key, click the Palette button for the element you want to create (the button stays down). Release the CTRL key.
2. Click the desired location on the diagram background. The new element is placed on the diagram at the point where you click.
3. Click the next location on the diagram background. The next new element is placed on the diagram.
4. Repeat the previous step until you have the desired number of elements of that type.
5. To stop multiple element creation, click the Pointer Palette button or press the ESC key to deselect the element after closing the in-place editor of the last inserted element.

Tip: After making a selection on the Palette or doing the first of a multi-draw or multi-placement operation, you can cancel the operation by clicking the Pointer button on the Palette or by pressing the ESC key.

Related Procedures

Adding a Single Model Element to a Diagram
Creating a Simple Link

Related Reference

Together Keyboard Shortcuts

Creating a Link with Bending Points

If your diagram is densely populated, you can draw bent links between the source and target elements to avoid other elements that are in the way.

To create a link with bending points

1. Click the link button on the Palette.
2. Click the source element.
3. Drag the link line, clicking the diagram background each time you want to create a section of the link. Sections on a link lie between two blue bullets. The bullets display whenever you select the link on the diagram.

   Tip: You can cancel each section of a link pressing the BACKSPACE key.

4. Click the destination element to terminate the link.

Tip: After you have created a link, you can add bending points to it. Click on a specific point of the link, and drag it to the position you want.

Related Procedures

Rerouting a Link
Creating a Simple Link

Related Reference

Class Diagram Relationships

Creating a Shortcut

You can create a shortcut to a model element from the current project or from the projects connected by cross-projects references, by using three methods:

- By choosing **New Shortcut** on the Diagram Editor context menu
- By dragging and dropping a shortcut from the Model Navigator
- By choosing **Add as Shortcut** on the Model Navigator context menu

To create a shortcut by using the Shortcuts dialog window

1. Right-click the diagram background.
2. Choose **New** ▸ **Shortcuts** on the context menu.

   Tip: Use the CTRL+SHIFT+N keyboard shortcut

3. In the **Shortcuts** dialog window, choose the required element from the tree view of available contents.

   Note: If the project has cross-project references to the other projects in the workspace, the contents of these projects is available for being added as a shortcut.

4. Click **Add** to place the selected element to the list of the existing or ready-to-add elements.
5. When the list of ready-to-add elements is complete, click **OK**.

To create a shortcut by using drag-and-drop

1. Select the element in the Model Navigator.
2. Drag-and-drop the element onto the diagram.

To create a shortcut by using the Model Navigator context menu

1. Open the diagram where the shortcut will be added.
2. In the Model Navigator, select the element to be added to the current diagram as a shortcut.
3. Right-click the element in the Model Navigator and choose **Add as Shortcut** on the context menu.

Related Concepts

Model Shortcut Overview

Related Procedures

Establishing cross-project references
Adding a Single Model Element to a Diagram

Creating a Simple Link

You can create a link to another node, or a shortcut of an element of the same or another project (these projects must be of the same UML version).

To create a simple link between two nodes

1. On the diagram Palette, click the button for the type of link you want to draw in the diagram. The button stays down.
2. Click the source element.
3. Drag to the destination element and drop when the target element is highlighted.

Related Procedures

Rerouting a Link
Creating a Link with Bending Points
Creating Model Element by Pattern

Related Reference

Class Diagram Relationships

Creating an Inner Classifier

This section includes instructions for adding inner classifiers to classes (including Windows classes, such as Windows forms, Inherited forms, User Controls and so on), structures, and modules (collectively, containers) in implementation projects.

You can add inner classifiers to class diagram elements (containers) using the respective context menu for the diagram element in the Diagram Editor or Model Navigator. You can also select a classifier in the Palette and click the container element in the Diagram Editor to add the inner classifier to the container element.

Tip: You can use drag-and-drop or clipboard operations to remove an inner classifier from the container element.

To create an inner classifier using the context menu

1. Right-click the container element.
2. Choose **Add** ▸ **<Inner classifier type>**

To create an inner classifier using the clipboard operations

1. Use the clipboard operations to either cut or copy an existing classifier.
2. Select the container element.
3. Use the clipboard operations to paste the selected classifier into the container element.

To create an inner classifier using drag-and-drop

1. Select an existing classifier in the Diagram Editor .
2. Drag-and-drop it onto an existing container in the Diagram Editor . A border highlights the location that Together recognizes as a valid destination for the inner classifier.

Related Procedures

Adding a Single Model Element to a Diagram

Related Reference

UML 1.4 Class Diagrams
UML 2.0 Class Diagrams

Editing Together Diagrams

In This Section

Aligning Model Elements

You can automatically rearrange all or selected model elements on a diagram according to the order you specify. The following options are available:

- Top
- Bottom
- Right
- Left
- Center Horizontally
- Center Vertically

To align model elements on a diagram

1. Select several nodes or inner classifiers on a diagram.
2. On the main menu, choose **Diagram** ▸ **Align** ▸ **<option>**.

Tip: Alternatively, use the diagram Palette buttons.

Related Procedures

Laying Out a Diagram Automatically

Assigning a Stereotype to an Element

You can assign a stereotype in the diagram by using the in-place editor, or the Properties View.

To assign a stereotype by using the in-place editor

1. Double-click the stereotype name to activate the in-place editor.
2. Enter the new name.
3. Press ENTER.

To assign a stereotype by using the Properties View

1. Select an element on your diagram.
2. In the Properties View, select the **Stereotype** field.
3. Click the value editor button.
4. In the Edit Property Values dialog, click the **Add** button and enter the required stereotype.

Related Reference

Stereotype Options of UML Profile for Modeling In Color

Changing Type of an Association Link

Use the following techniques to change the type of an Association link

- ♦ Set the link type by using the Properties View
- ♦ Set the link type by using the context menu

To set the Association link type by using the Properties View

1 Select a link on the diagram.
2 Open the Properties View.
3 In the Properties View, select the **Type** field.
4 Click the drop-down arrow and select the appropriate property from the list. Your available choices are association, aggregation, or composition.

To set the Association link type by using the context menu

1 Right-click an Association link on the diagram.
2 Choose **Link Type** on the context menu.
3 Choose **Association**, **Aggregation**, or **Composition**.

Related Procedures

Creating a Simple Link

Related Reference

Class Diagram Relationships

Copying and Pasting Model Elements

The move and copy operations are performed by drag-and-drop, context menu commands, or keyboard shortcut keys.

Note: You can move or copy an entire diagram. In this case, all elements addressed on this diagram are not copied, and a new diagram contains shortcuts to these elements.

To copy and paste one or more elements

1 Select the desired element or elements.

2 To copy the selection, do any of the following:

- ♦ Right-click and choose **Copy** on the context menu
- ♦ Press CTRL+C on the keyboard

3 To paste the selection, do any of the following:

- ♦ Right-click the target location and choose **Paste** on the context menu
- ♦ Select the target location and press CTRL+V

Note: Pasting elements from one package to another also maps relationships of those elements to the target package. By default, a prompt appears warning users of this before the paste is complete. To disable this warning, select **Window** ▸ **Preferences** from the main menu, choose the **Modeling** node, and uncheck the **Show warning about relationships when elements copied** option.

Related Procedures

Adding a Single Model Element to a Diagram
Together Keyboard Shortcuts

Deleting Elements

All elements shown on diagrams are shortcuts to actual model elements. When deleting an element on a diagram, you have the option to delete either the shortcut from view or delete the element from the model (except classes on synchronized package diagrams). This behavior is configured in the Modeling Preferences.

To delete an element

1. Select the element on the diagram.
2. Choose **Delete** on the context menu of the element.

   Tip: Alternatively, click the DELETE key.

3. Confirm deletion, if this behavior is selected in the Modeling Preferences.

To delete an element from View

1. Select the element on the diagram.
2. Choose **Delete from View** on the context menu of the element.
3. Confirm deletion, if this behavior is selected in the Modeling Preferences.

Related Procedures

Adding a Single Model Element to a Diagram

Related Reference

Modeling Preferences

Laying Out a Diagram Automatically

To lay out a diagram by using one of the algorithms

1. Right-click the diagram background.
2. On the context menu, select **Layout**, and choose a command from the submenu.

   There are several Layout commands on the **Layout** submenu:

   - **Do Full Layout**: Sets the layout of all elements according to the layout algorithm defined for the current diagram.
   - **Layout for Printing**: Sets the layout of all elements using the *Together* algorithm, regardless of the option selected on any level.
   - **Route All Links**: Streamlines the links removing bending points.
   - **Optimize Sizes**: Enlarges or shrinks all elements on the diagram to the optimal size.

Note: Individual diagram elements also have the **Route Links** and **Optimize Size** layout commands on their respective context menus. The **Route Links** command streamlines the links and removes any bending points. The **Optimize Size** command enlarges or shrinks the element to the optimal size, leaving enough space for its label and any sub elements it may contain.

Tip: To enable the layout of the inner substructure in diagrams, check the **Recursive** option (**(level)** ▶ **Diagram** ▶ **Layout** ▶ **General**) in the **Options** dialog window.

To set up the diagram layout

1. On the main menu, choose **Window** ▶ **Preferences** ▶ **Modeling** ▶ **Layout**.
2. Select the desired layout for links in the **Links layout** section.
3. Choose the desired algorithm from the **Algorithm** drop-down list, and specify the algorithm-specific options (if any).
4. To enable layout of the inner substructure in diagrams, check the **Recursive** option.

You can now observe results of layout tuning when you apply one of the *Layout* commands to the diagram.

The context menu available in the Diagram Editor provides access to the automated layout optimization features in Together.

Related Procedures

Aligning Model Elements

Laying out a Diagram for Printing

Together has automated layout optimization for printing diagrams. Using automated layout for printing ensures that all diagram elements fall within page borders.

Invoke automated layout immediately before printing a diagram.

To lay out you diagram elements for printing

1. Right-click the diagram background.
2. On the context menu, choose **Layout** ▶ **Layout All for Printing**.

Tip: You can revert to your manual layout after a Layout and optimize operation by using Undo. For example, you might invoke Layout and optimize, print the diagram, then call Undo to restore your manual layout.

Related Procedures

Print Preferences

Moving Model Elements

Create your own layout by selecting and moving single or multiple diagram elements.

You can:

- Select a single element and drag it to a new position.
- Select multiple elements and change their location.
- Manually reroute links.
- Use Cut and Paste operations.

Note: If you drag an element outside the borders of the Diagram Editor , the diagram automatically scrolls to follow the dragging.

Tip: Manual layouts are saved when you close a diagram or project and are restored when you next open it. Manual layouts are not preserved when you run one of the auto-layout commands (**Do Full Layout** or **Optimize Sizes**).

To move one or more elements

1. Select the element or elements to be moved.
2. Drag-and-drop the selection to the target location.

Tip: If you have selected several model elements in certain diagrams (State Machine, Use Case, Activity or Business Process), use the heading area of one of the selected elements to drag the entire group. Any attempt to drag by an internal area of an element results in switching the Diagram Editor to the Select mode and losing the current selection. However, if you hold the mouse button down and press ESC, the new selection will be canceled and the current selection will be preserved.

Related Procedures

Selecting Model Elements

Related Reference

Together Keyboard Shortcuts

Renaming a Diagram

Warning: The automatically created package diagram cannot be renamed.

To rename a diagram

1. In the Properties View, double-click the diagram name to initiate the inline editor.
2. Enter a new name.
3. Press Enter.

To rename a diagram using the Model Navigator

1. Select the diagram in the Model Navigator.
2. Press F2 or right-click and choose **Rename** on the context menu.
3. Enter a new name.
4. Press ENTER.

Related Procedures

Creating a Diagram

Rerouting a Link

To reroute a link

1. Select a link.
2. Drag and drop the client or supplier end of the link to the destination object.
3. To change the direction of the link, click a place on the link where you want to reroute the link.
4. Drag the line. Together automatically reshapes the link the way you want.

Tip: Model elements have the **Layout ▸ Route All Links** command on diagram context menus.

Related Concepts

Model Element Overview

Related Procedures

Laying Out a Diagram Automatically

Resizing Model Elements

You can resize diagram elements automatically or manually. When new items are added to an element that has never been manually resized, the element automatically grows to enclose the new items.

To resize an element manually

1. Click an element. The selected element is highlighted with bullets.
2. Drag one of the bullets in the direction you want to expand.

When the element contents change (for example, when members are added or deleted, and the element size is too small to display all members) scroll bars are displayed to the right of compartments.

To optimize a node element size

1. Right-click an element.
2. Choose **Layout** ▶ **Optimize Size**.

To optimize the elements on an entire diagram

1. Right-click the diagram background.
2. Choose **Layout** ▶ **Optimize Size**.

Related Procedures

Laying Out a Diagram Automatically

Selecting Model Elements

Most manipulations with diagram elements and links involve dragging the mouse or executing context menu commands on the selected elements.

In this section you will learn how to:

- Select one or more elements
- Cancel selection

To select an element

1. Open a diagram in the Diagram Editor .
2. On the diagram Palette, click the **Select** button.
3. In the Diagram Editor , click any element or a member to select it.

To select multiple elements, do one of the following

1. Hold down the CTRL key and click each element individually, OR
2. Click the background and drag a lasso around an area to select all the elements it contains, OR
3. Press CTRL+A to select all elements on a diagram, OR
4. Right-click the diagram background and choose **Select All** on the context menu.

Note: To cancel a selection, press the ESC key.

Related Procedures

Aligning Model Elements

Related Reference

Together Keyboard Shortcuts

Working with Rulers Guides and Grid

Together provides means to use ruler guides in the **Diagram Editor** for aligning purposes.

To add or remove a ruler guide

1 To add a ruler guide, click either the vertical or horizontal ruler. The guide appears at the click point.

> **Note:** Alternatively, right-click a ruler and choose **Create Guide**. The guide is created at the zero point of the ruler.

2 To remove a ruler guide, click a guide on the ruler, drag it out of the ruler space until your pointer becomes a normal select shape, and release your mouse button.

After you created several guides, you can connect your elements to ruler guides. If you connect several elements to a guide, all elements move when you move the guide.

Aligning elements with ruler guides

1 Move or resize an element on the diagram to place one side of the element close to a rule guide.

2 Drop the element when the guide highlights.

3 Repeat the previous steps to connect other elements to the guide.

4 Move the guide. Notice how all connected elements move with the connected ruler guide.

> **Note:** To disconnect an element from the guide, simply move the element from the guide.

You can optionally display or hide a design grid on the diagram background and have elements “snap” to the nearest grid coordinate when you place or move them. Grid options are configured in the **Diagram** page of the **Preferences** dialog box.

To show the grid

1 Open **Preferences** dialog box.

2 Choose the **Modeling** ▸ **Diagram** category, **Rulers, Grid, and Snapping** group.

3 Adjust the options.

> **Note:** Grid display and snap are enabled by default.

Related Reference

Diagram Preferences

Together Projects

This section provides how-to information on using Together projects.

In This Section

Changing the Default Diagrams Directory
Lists the steps for changing the default directory of the diagram files.

Choosing a Together Perspective
How to choose a Together perspective.

Configuring C++ Projects
How to define C++ project structure and language-specific options.

Configuring IDL Projects
How to define IDL project structure and language-specific options.

Converting UML 1.4 Project to UML 2.0 Project
How to convert a UML 1.4 Project to a UML 2.0 project.

Creating a Project
How to create a project in Together.

Enabling UML Profiles
Describes how to enable profile support for a project.

Establishing cross-project references
Describes how to establish cross-project references between the projects located in the same workspace.

Exporting a Project to XMI Format
How to export a project to XMI format.

Exporting a Project to XMI Format Using the Command Line
How to export a project to XMI format using the command line.

Generating Source Code from Design Project
How to create a source code project from design project in Together.

Importing a Project in an IBM Rational Rose MDX Model
How to import .mdx projects.

Importing a Project in IBM Rational Rose (MDL) Format
How to import .mdl projects.

Importing a Project in IBM Rational Rose (MDL) From the Command Line
How to import .mdl projects using the command line utility.

Importing a Project in IBM Rational Rose (MDX) From the Command Line
How to import Rational projects using the command line utility.

Importing a Project in XMI Format
How to import XMI data.

Importing Java Modeling Projects Created in Together Edition for Eclipse 7.0
How to import a project created in TEC 7.0.

Importing Legacy Projects
How to import a legacy project and handle multiple project roots.

Navigating between the Tree View, Diagram, and Source Code
How to synchronize the Tree View, Diagram, and source code.

Resolving Duplicates During an XMI Import
How to resolve duplicates while importing an XMI project.

Reusing Existing Source Code in Modeling Projects
How to use existing source code in modeling projects.

Showing libraries
Describes how to show classes or packages from the standard Java libraries in a class diagram.

Troubleshooting a Model
How to troubleshoot a model.

Using Example Projects
How to use sample projects in Together.

Working with a Package
How to work with a package.

XMI Export and Import of the Models with Cross-Project References
You can import and export multi-root projects using XMI. Note that XMI imports and exports are implemented differently for UML 1.4 and Java modeling projects, and for UML 2.0 projects.

Changing the Default Diagrams Directory

By default, Together diagram files are contained within the default design root folder, which is called the Model Folder.

To change the default diagrams directory

1. Right-click the project root in the Model Navigator, or Model Package Explorer view, and select **Properties**. The **Properties** dialog box displays.
2. Choose **Design root path** from the **properties** list on the left.
3. Specify the path in the **Design root path** field, and press OK.

Warning: The path name can contain only the folder name that the existing design root will be renamed to, not the path to the folder.

Choosing a Together Perspective

Together changes the user interface according to how you want to work with Together by providing several perspectives. By default, Together starts with the Modeling perspective.

Note: The way you have Together configured influences which perspectives and views you can choose from. For example, clicking the **Take me to the DSL workbench** link from the Welcome page displays only the DSL Workbench's default perspectives. In order to display a list of all the perspectives, check the **Show all** check box in the **Open Perspective** dialog box. An additional **Confirm Enablement** dialog box might appear that requires you to enable any necessary capabilities.

To choose a Together Perspective

1. On the main menu, choose **Window** ▶ **Open Perspective** ▶ **Other**. The **Select Perspective** dialog box opens.
2. Select one of the Together Perspectives from the list and click **OK**.

After you select a perspective, Together automatically customizes the interface to provide ready access to only the relevant elements of the interface, and to show only the information in the model that best supports the chosen perspective. Interface elements and/or model information that are not generally relevant to the perspective are hidden. You can still access hidden information by changing the relevant configuration options and restoring hidden panes manually, but you may find it easier to just switch perspectives.

Related Concepts

Together Capabilities Activation
Tour of Together

Related Procedures

Activating Together Capabilities

Configuring C++ Projects

In this section, you will learn how to define the project structure and processing options:

- Access C++ project properties
- Define source path
- Define entry points
- Include search paths
- Define C++ processing settings (for example, skip standard includes option, or suffixes for the C++ files)
- Define indexer
- Enable C++ formatting
- Set up formatting options

To configure a C++ project

1. Select the desired project in the Model Navigator.
2. On the main menu, choose Project Properties.

   Tip: Alternatively, choose Properties on the context menu of the project.

   The **Properties for <project>** dialog opens. Select the **Project Properties** page.
3. In the **Project source path** tab, click the **Link Additional Source to Project** button.
4. In the **Link Additional Source** dialog, specify the linked folder location and name, and click **OK**.

   Tip: Use the context menu of the linked folder to remove it, mark it read-only, or filter it.

5. Configure parsing entry points using the Configure Entry points dialog.
6. In the **Include paths** tab, click **Add**.
7. In the **Add Include Folder** dialog, enter the folder name or click **Browse** and navigate to the folder you want to add.
8. In the **C++ Processing Settings** tab, select your C++ project options.
   - To skip standard includes, check the **Skip standard includes** option.
   - If you want to use the preinclude file, specify its name in the **Preinclude file name** field.

9. Select the **C/C++ indexer** page, and select an indexer from the list. Among the available indexers, you can choose the Borland indexer.

To enable C++ formatter

1. On the main menu, choose Window ▸ Preferences
2. Under the C/C++ category, select the Code Formatter page.
3. From the list of available formatters, select Together C++ Code Formatter.

To set up formatting options

1. Under your Together installation, expand the `plugins` folder.
2. In the `com.borland.tg.cdtintegration` plugin, open the `formatter.properties` file.
3. Use the documentation provided with the file to edit as required.

Related Reference

New project Wizard C++ Language-Specific Options
C++ Projects

Configuring IDL Projects

In this section you will learn how to define the project structure and processing options:

- Access IDL project properties
- Define source path
- Include search paths
- Define IDL processing settings

To configure an IDL project

1. Select a project in the Model Navigator.
2. On the main menu, choose **Project Properties**.

   Tip: Alternatively, choose **Properties** on the context menu of the project.

3. In the **Project source path** tab, click the **Link Additional Source to Project** button.
4. In the **Link Additional Source** dialog, specify the linked folder location and name, and click **OK**.

   Tip: Use the context menu of the linked folder to remove it, mark it read-only, or filter it.

5. In the **Include paths** tab, click **Add**.
6. In the **Add Include Folder** dialog, enter the folder name or click **Browse** and navigate to the folder.
7. In the **IDL Processing Settings** tab, select your IDL project options. Refer to the IDL Language-Specific Options section for details.

Related Reference

New project Wizard IDL Language-Specific Options
IDL Language-Specific Information

Converting UML 1.4 Project to UML 2.0 Project

This topic describes how to convert a UML 1.4 project to a UML 2.0 project. If you want to preserve any cross-project dependencies during the conversion, observe the following:

- Ensure that all dependent projects are part of the current workspace and are opened.
- Select **Window** ▸ **Preferences** ▸ **Modeling** ▸ **UML 1.4 to UML 2.0 Converter** and ensure that the **Enable referenced projects support** option is checked.
- Using the procedure that follows, convert each project separately, beginning with those that do not reference other projects. Projects that are referenced by other projects should be converted first.

To convert an existing UML 1.4 project to a UML 2.0 project

1. On the main menu, choose **File** ▸ **New** ▸ **Project**. The **New Project** wizard opens.
2. Expand the **Together** node in the tree view list, and select **UML 2.0 from 1.4 Project**. Click **Next**.
3. Type the new UML 2.0 project name and specify other project-related options. Click **Next**.
4. Select the UML 1.4 project you want to convert. If necessary, specify mappings between referenced UML 1.4 projects and existing UML 2.0 projects near the bottom section of the dialog that shows a list of all UML 1.4 projects currently referenced by the selected project. Automatic mappings are normally created during any previous conversions.
5. Click **Next**.
6. Click **Finish** to complete the wizard.

The selected UML 1.4 project is converted to a newly created UML 2.0 project.

Related Procedures

Reusing Existing Source Code in Modeling Projects
Working with a Package
Creating a Diagram

Creating a Project

Together provides several projects that you can work with. The projects in Together are created in the same manner. While creating a project, you will specify different options depending on the type of project.

To create a Together Project

1. Select **File ▸ New ▸ Project** on the main menu. The **New Project** wizard displays.
2. Expand the **Modeling** node in the tree view list, and select the type of project you want to create. Click **Next**.
3. Follow the wizard steps to specify necessary options for a new project and click **Finish** to complete the wizard.

Related Concepts

Together Project Overview

Related Procedures

Together Projects

Related Reference

Together Projects

Enabling UML Profiles

There are several ways to enable UML profiles for Together projects.

To enable UML profiles support while creating a project

1. On the main menu, choose File ▸ New ▸ Project. The **New Project** wizard opens.
2. Expand the **Modeling** node in the tree view list, and select the UML project you want to create (UML 2.0 or UML 1.4). Click **Next**.
3. Follow the wizard to the **Profiles** screen. The **Profiles** screen of the wizard lists available profiles.
4. Select one or more profiles you want to enable and click **Next** to continue creating a new project with the **New Project** wizard.

To enable UML profiles support for existing projects

1. In the Model Navigator, right-click the root project folder, and select Properties on the context menu. The **Properties for <project>** dialog box displays.
2. From the list on the left, select **UML Profiles**.
3. Select any of the UML profiles that you want to enable. More than one can be activated.
4. Click **OK**.

Note: You can also access the **Properties for <project>** dialog box through the **Model Package Explorer** view and Navigator view.

To specify the default set of UML profiles enabled for all new workspace projects

1. Choose Window ▸ Preferences on the main menu.
2. In the left pane of the **Preferences** dialog box, expand the **Modeling** node.
3. Select the **UML Profiles** node.
4. Select the profiles you want to enable for UML 1.4 and UML 2.0 projects.

Note: The selected UML profiles are automatically enabled for projects created after you changed profile preferences. Profiles support of existing projects is not changed.

Establishing cross-project references

You can establish references between the projects of a similar type within your workspace. This capability is enabled in the **Model Path** page of the **Project Properties** dialog. When cross-project referencing is enabled, the imported project is included in the original project as read-only root and becomes visible in the selection dialogs. Consequently, any changes in the referenced projects are propagated across the target project as well. For example, renaming elements in the referenced project is immediately reflected in the target project.

To enable cross-project references

1. Select the desired project in the Model Navigator.
2. On the main menu, choose Project Properties. The **Properties for <Project Name>** dialog opens.

   Tip: Right-click on the project node and choose Properties on the context menu.

3. Select **Model Path** node.
4. In the **Model Path** page, click **Add Project** button. The **Select Projects to Import** dialog opens, displaying the list of available projects in the workspace.

   Note: Only the projects of similar types are included in the list.

5. Check one or more projects in the field **Available Projects in the Workspace** and click **OK**.
6. Click **OK** to confirm your settings and close the **Project Properties** dialog.

Warning: Avoid establishing recurrent references.

Tip: The **Project references** tab of the **Project Properties** dialog is a part of Eclipse functionality, and does not have any effect on the Together cross-project references.

Related Concepts

Together Project Overview

Related Procedures

Creating a Project

Related Reference

Project Properties

Exporting a Project to XMI Format

You can export projects or sections of projects created in Together for use by other applications/languages using XMI. Together supports several XMI formats. The availability of formats depends on the types of projects currently opened in Together.

To export a project to XMI format

1. Select **File** ▸ **Export** on the main menu. The **Export** dialog box opens.
2. Under Modeling, choose **XMI File** and click **Next**.
3. In the **Export Project to XMI FIle** dialog box, specify the following:
   - Select the project to export. For UML 1.4 and Java modeling projects, you can also expand the project to select only a portion of it. You cannot proceed until a project or package is selected.
   - Select the XML and UML version you want the file to support under **Select XMI Type**. A UML 2.0 project can be exported to XMI for UML 2.0 only.
   - Select an appropriate XMI Encoding requirement in the **XMI Encoding** list.
   - Specify the destination in the **Select the export destination** field. You can include the path as well as the name of the file that will be created, or you can accept the default. For UML versions 1.3-1.4, the name consists of `<project_folder>\out\xmi\<project_name><number>.xml`. For the first file generated under this name, `<number>=1`. Thereafter, `<number>` increases by one for each file saved under the same name. Note that `.xml` is automatically added as the file extension. For UML version 2.0, the name consists of `<project_folder>\out\xmi\<project_name>.uml2.`
4. Click **Finish** to generate the XMI file.

A dialog box opens indicating that the XMI export is completed. If there are any warnings produced during XMI export, the **XMI Export** dialog box notifies you to refer to the **Task** view. To open the **Task** view, select **Window** ▸ **Show View** ▸ **Other** ▸ **Basic** ▸ **Tasks** from the main menu.

Note: For UML 2.0 projects with applied profiles or projects that contain any stereotypes or primitive types, the following files are created during the export process in addition to the model .uml2 file:

<model name>.profile.uml2 – for stereotypes and primitive types

<profile name>.profile.uml2 – for applied profiles

Related Concepts

Model Import and Export Overview

Related Procedures

XMI Export and Import of the Models with Cross-Project References

Related Reference

XMI Export Wizard

Exporting a Project to XMI Format Using the Command Line

Together provides a comand-line method for an XMI export of UML 1.4, UML 2.0, and Java Modeling projects. Use the XMIExport.cmd on the Windows platform or XMIExport.sh on the UNIX platforms.

To export a project to XMI format under Windows

1. Locate the XMIExport.cmd file in the Together installation folder.
2. Run the XMIExport.cmd file with necessary parameters.

Note: For usage instructions and command-line parameters, run `XMIExport.cmd -help` or `XMIExport.sh -help`.

Related Concepts

Model Import and Export Overview

Related Reference

XMI Export Wizard

Generating Source Code from Design Project

Together provides several projects that you can work with.

To generate source code from a design project

1. Select **File** ▸ **Export** on the main menu. The **Export** wizard is displayed.
2. Select either **Generate C++ Project** or **Generate Java Project** in the list. Click **Next**.
3. Select the modeling project you want to use for source generation. Click **Next**.
4. Specify source code generation options and click **Next**.
5. Specify a new code generation project name. Click **Finish** for a C++ Project. Click **Next** to specify Java-related options for a Java project.
6. Click **Finish** to complete the wizard.

A new code generation project is created from the selected modeling project.

Related Concepts

Together Interoperability and Migration

Importing a Project in an IBM Rational Rose MDX Model

Together enables you to create projects around an IBM® Rational® XDE .mdx file.

Note: Together design projects that are created on the basis of the imported MDX models always comply with the UML 2.0 specification.

To create a project from an MDX model

1. On the main menu, choose **File ▸ Import**. The **New Project** wizard opens.
2. Select **Project from MDX file** and click **Next**.
3. Specify the path to the MDX file you want to import or click **Browse** to locate the file. You can also specify the following:
   - Use the **Scale factor** field to specify the element dimensions coefficient. By default, the scale factor is 0.03.
   - **Preserve diagram nodes and bounds**: If this option is selected, user-defined bounds are preserved in the resulting diagrams. Otherwise the default values are applied.
4. Click **Next**.
5. Specify new project name. Click **Next**.
6. Specify the diagram to start with. Click **Next**.
7. Select one or more profiles you want to enable for this project. Click **Next**.
8. Select any referenced projects.
9. Click **Finish** to complete the wizard. A new project will be created with elements from the MDX file.

Note: If a profile was applied to the Rational XDE model while importing the MDX model to Together, the properties from this profile are imported as custom properties.

Related Concepts

Model Import and Export Overview

Related Reference

MDX Import Wizard
MDX Projects Import Options

Importing a Project in IBM Rational Rose (MDL) Format

Together enables you to create projects around IBM® Rational® Rose model files (.mdl, .ptl, .cat, .sub).

Note: You can import a set of petal and subunit files.

Warning: Together projects created on the basis of the imported MDL models always comply with the UML 1.4 specification.

To create a design project from an IBM Rational Rose (MDL) project

1. On the main menu, choose File ▸ Import. The **New Project** wizard opens.
2. Select **Project from MDL file** and click **Next**.
3. Click either **Add** or **Add Folder** to designate the MDL project path. This step specifies the name (or names) of the Rational Rose project file (or files) to be imported (several model files can be imported at once). Click **Remove** to delete the selected file or files from the Paths list. Click **Remove all** to delete all files from the Paths list.

   Note: Avoid adding a model file along with its subunit to the import list because this results in invalid project.

4. Use the **Scale factor** field to specify the element dimensions coefficient. By default, the scale factor is 0.3.
5. Specify the following options for the project:

   - **Convert Rose default colors**: If this option is selected, the default Rational Rose colors will be replaced with the default Together colors.
   - **Preserve diagram nodes and bounds**: If this option is selected, user-defined bounds are preserved in the resulting diagrams. Otherwise the default values are applied.
   - **Convert Rose actors**: This option enables you to choose mapping for the Rose actors. If the option is selected, the Rose actors are mapped to Together actors. If the option is not selected, the Rose actors are mapped to the classes with the Actor stereotype, such as Actor, Business Actor, Business Worker, or Physical Worker.
   - **Generate source code**: If this option is selected, a new Java Modeling project is created; otherwise, a Modeling project is created from imported MDL.

6. Click **Finish**.
7. When prompted, supply a name for your project and click **Finish**.
8. Follow the remaining steps in the wizard to specify options for your new project, and click **Finish** to complete the wizard.

After the import process is complete, you can view the project structure in the Model Navigator view. The `mdlimport.log` file is generated by default and lists any errors encountered during the import process.

Note: After entering a project name, you can click **Finish** without completing the remaining steps of the wizard. The project is created using the remainder of default settings.

Related Concepts

Model Import and Export Overview

Related Procedures

Generating Source Code from Design Project

Related Reference

Together Projects
MDL Projects Import Options
MDL Import Wizard

Importing a Project in IBM Rational Rose (MDL) From the Command Line

Together enables you to create projects around IBM® Rational® Rose model files (`.mdl, .ptl, .cat, .sub`) by executing the import wizard from the command line.

To execute an MDL import from the command line

1. Navigate to `plugins\com.borland.tg.mdlimport_8.1.0` in the Together installation folder.
2. Execute `'java -cp mdlimport.jar com.borland.tg.mdlimport.CmdLineImporter <parameters>'`.

For example, `Java -cp mdlimport.jar com.borland.tg.mdlimport.CmdLineImporter -d c:\myproject -project myproject -modelfile mymodel.mdl`.

Related Concepts

Model Import and Export Overview

Related Procedures

MDL Projects Import Options

Importing a Project in IBM Rational Rose (MDX) From the Command Line

Together enables you to create projects around IBM® Rational® XDE model `*.mdx` files by executing the MDL import wizard from the command line with specific parameters.

To execute an mdx import from command line

1. Navigate to `plugins\com.borland.tg.mdlimport_8.1.0` in the Together installation folder.
2. Execute `'java -cp mdlimport.jar com.borland.tg.mdlimport.CmdLineImporter <parameters>'`.

Note: For parameter values, refer to MDX Project Import Options.

Related Concepts

Model Import and Export Overview

Related Reference

MDX Projects Import Options

Importing a Project in XMI Format

You can import projects or sections of projects that were created in other modeling tools and saved in XMI format.

Note: For UML 1.4 and Java Modeling projects only, XMI 1.1/1.2 imports are supported. Attempting to import an XMI 1.0 file results in an empty project.

To import a project from an XMI file

1. Select **File** ▶ **Import** on the main menu. The **Import** dialog box opens.
2. Select **XMI File** and click **Next**.
3. In the **Import Project from XMI File** dialog box, specify the following:
   - ♦ The Together project to which your XMI data will be imported in the **Select destination project** field.
   - ♦ The full path to the .xml, .xmi, or .uml2 file you want to import in the **Select source .xmi file** field.
4. Click **Finish**.

   Note: A .xml or .xmi file can be imported to UML 1.4 and Java Modeling projects; a .uml2 file can be imported to UML 2.0 projects.

After you are notified that the import process is complete, you can view the results in the **Model Navigator**.

Note: When importing UML 2.0 models with profile files related to the model, for the models originally exported from Together for Eclipse, select model .uml2 file as a source and make sure that all the profile files are located in the same folder with the model file.

If there are any warnings produced during XMI import, the XMI Import dialog notifies you to refer to the **Task** view. To open the Task view, select **Window** ▶ **Show View** ▶ **Other** ▶ **Basic** ▶ **Tasks** from the main menu.

Related Concepts

Model Import and Export Overview

Related Procedures

XMI Export and Import of the Models with Cross-Project References

Importing Java Modeling Projects Created in Together Edition for Eclipse 7.0

You can import projects created in Together Edition for Eclipse 7.0.

The general procedure for importing a project created in Together Edition for Eclipse 7.0 consists of the following steps:

1. Importing your existing project into a workspace
2. Creating a Java modeling project from a Java project

To import an existing project from TEC 7.0

1. Select **File ▸ Import** on the main menu.
2. Select **Existing Projects into Workspace** and click **Next**.
3. In the **Import Projects** dialog box, specify the path to your project's root directory and select one or more projects you want to import.
4. Click **Finish** when you specified all necessary options.

   The new Java project is created and opened in your workspace.

Note: The name of the imported project cannot be changed during the import process. Therefore, the projects are created with the same name as the imported projects.

To create a Java modeling project from a Java project

1. Select **File ▸ New ▸ Project** on the main menu. The **New Project** wizard opens.
2. Expand the **Together** node in the tree view list and select **Java Modeling projects from Java projects**. Click **Next**.
3. Select the Java project you created from the project created in Together Edition for Eclipse. Click **Next**.
4. Specify other project-related options.
5. Click **Finish** when you specified all necessary options.

Related Concepts

Together Interoperability and Migration

Importing Legacy Projects

Together allows you to import projects from some of the previously released Together products. Considering the differences between the products, Together suggests two ways to accomplish this import. You can merge all roots of a legacy multi-rooted project into a single root, or you can create a separate project for each root of the source project.

- The **Merge option** is recommended for typical cases of when the input project has one design root and several source code roots.
- The **Separate projects option** is recommended when your input project has nonstandard configuration with several design roots, which you would like to preserve as separate projects.

To import a legacy project merging all source roots into a single project

1. Select File ▸ Import on the main menu
2. In the **Import Wizard**, select Modeling Together Project and click **Next**. The second page of the wizard opens.
3. Click **Browse** to specify the fully qualified name of the project you want to import.
4. In the **Design elements storage policy** section, choose whether the design elements of the resulting project will be stored as standalone design elements or as filemates.
5. In the **Migration type** section, select the **Merge all roots contents into the new project** option.
6. Click **Next**. The third page of the wizard opens.
7. Specify the name of the target project. The default project name is constructed from the names of the last two folders of the source project file location.
8. Click **Finish** to import the selected project.

Warning: TVS projects and projects created in Together Editions for Eclipse prior to version 7.0 cannot be imported to Together.

To create separate projects for each selected root

1. Select File ▸ Import on the main menu
2. In the **Import Wizard**, select Modeling Together Project and click **Next**. The second page of the wizard opens.
3. Click **Browse** to specify the fully qualified name of the project you want to import.
4. In the **Design elements storage policy** section, choose whether the design elements of the resulting project will be stored as standalone design elements or as filemates.
5. In the **Migration type** section, select the **Create a separate project for each root** option.
6. On the third page of the wizard, the **Root location** table displays the list of folders of the source project. Select each root from the list and define the way you want to handle the root and its contents:
   - In the **Together project name** field, specify the name of the target project for the selected root. The default name is constructed from the package prefix, if any. If there is no package prefix, the project name is created from the names of the last two folders of the root location.
   - The read-only **Content type** and **Diagram format** fields display the corresponding information for the selected root.
   - In the **Decision** field, choose the way to handle information of the selected root. If the root contains design files, you can either copy them to the target location or skip the root. If the root contains source code files,

you have the choice to copy it as is, copy and convert it to design language, or skip the root. The option **Copy and convert to design language** is the default choice for the roots that contain Java files.

- In the **Dependencies to be preserved while importing** field, you can specify whether the import handles links and references between projects created for the currently selected root and projects created for other roots. All dependencies are processed by default. However, if you are aware of any one-way dependencies between the original roots, and the selected root does not refer to any elements from other roots, uncheck those corresponding projects listed in the field to save CPU resources and complete the import faster.

7. Click **Next**. The fourth page of the wizard opens.
8. Specify the name of the master project that contains references to all projects created in the course of the migration. The default name of the master project is based on the source project name.

   Note: The master project is created to demonstrate the contents and structure of the source project. It is read-only and not intended for editing. Use the real projects to create or edit contents and establish dependencies.

9. Click **Finish** to import the selected project.

All resulting projects belong to the same type, which is defined by the properties of the source project and your choice in the **Decision** field of the **Import Wizard**. Java modeling projects are created if there is at least one Java source root for which the **Copy** option is selected. UML 1.4 modeling projects are created if there are no Java source roots, or if such roots exist but the **Decision** field is set to **Skip** or **Convert to design language**.

Related Concepts

Together Interoperability and Migration

Related Reference

Import Together Project Wizard

Navigating between the Tree View, Diagram, and Source Code

Together provides constant synchronization between different aspects of your project:

- Model hierarchy, presented in the tree view (**Model Navigator View**)
- Model graphical representation in the **Diagram Editor**
- Source code (for implementation projects)

Tip: You can also use the Refresh function of the **Model Tree View** to update the entire model, and the Refresh function of the **Diagram Editor**.

You can navigate between the Model Tree, Diagram Editor, and source code in the following directions:

1. Navigate to the **Diagram Editor** from the **Model Tree View**.
2. Navigate to a model element from the **Model Tree View** to the **Diagram Editor**.
3. Navigate from the **Diagram Editor** to the **Model Tree View**.
4. Navigate from a lifeline to its classifier in the **Model Navigator View** or a Class diagram.
5. Navigate from source code to the **Tree View**.
6. Navigate from the **Model Tree View** or **Diagram Editor** to source code (for implementation projects).

To navigate to the Diagram Editor from the Model Navigator View

1. In the **Model Navigator View**, right-click the diagram node.
2. Choose **Select on Diagram**.

Alternatively, double-click the diagram node in the **Model Navigator View**.

To navigate to a model element from the Model Navigator View to the Diagram Editor

1. Right-click a model element in the **Model Navigator View**.
2. Choose **Select on Diagram** on the context menu.

Note: Click the Link with Editor button on the Model Navigator toolbar and all elements selected in the Model Navigator will be automatically selected on diagrams.

To navigate from the Diagram Editor to the Model Navigator View

1. Right-click the selected element or diagram background in the **Diagram Editor**.
2. Choose **Select in Model Tree** on the context menu.

To navigate from a lifeline to its classifier in the Model Navigator View or a Class diagram

1. Right-click the selected lifeline on a UML 2.0 Sequence diagram in the **Diagram Editor**.
2. Choose **Select ▸ Type in Model Navigator View** to navigate to the classifier in the **Model Navigator View**,

OR

Choose **Select** ▸ **Type On Diagram** to navigate to the classifier on a Class diagram in the **Diagram Editor**.

To navigate from source code to the Model Navigator View

1. Right-click the line that contains the element you want.
2. On the context menu of the selection, choose **Select in Model Tree**.

The corresponding element is highlighted in the **Model Navigator View**.

To navigate from the Model Navigator View or Diagram Editor to source code (for implementation projects)

1. Right-click a model element or a node member.
2. Choose **Open** on the context menu.

Note: This command is available for source code-generating elements.

Click the Link with Editor button on the Model Editor toolbar and corresponding definitions will be automatically selected in the source code editor when you select model elements. Likewise, corresponding model elements will be selected when definitions are selected in the source code editor.

Related Concepts

Roundtrip Engineering Overview

Related Procedures

Troubleshooting a Model

Resolving Duplicates During an XMI Import

To resolve duplicates when importing an XMI file

1. If a duplicate package exists in the XMI file, the **Confirmation** dialog opens.
2. Select whether to rename the imported package or replace a package in the current project with the imported package.

Further behavior depends on the project type:

UML 1.4 and Java Modeling projects: When you choose rename, the name of the imported entity is automatically updated, adding a numeric value to the end. For example, if you have a project that contains a package named "problem_domain," and the imported XMI file also contains a package with the same name, choosing rename will rename the imported package "problem_domain1." Choosing replace will automatically replace the entity in the current project with the imported entity.

UML 2.0 projects: During XMI import, if an entity exists in the XMI file and it has the same name as an entity in the project, a new entity is created. For each imported package that has the same name as a package in the project, a new package is created and an incremental number is added to the package name.

Related Concepts

Model Import and Export Overview

Related Reference

XMI Export Wizard

Reusing Existing Source Code in Modeling Projects

Together allows you to convert your existing source code to UML models. Together provides two ways to use reverse engineering:

- Convert the existing source while creating a new project
- Import the existing source code to a Java Modeling project

To import Java source code while creating a new project

1. On the main menu, choose **File** ▸ **New** ▸ **Project**.
2. Expand the **Modeling** node and select **Java Modeling Project**. Click Next
3. On the **Java Modeling Project** page, type the project's name. Click Next
4. On the **Modeling Settings** page, choose the desired metamodel (UML 2.0 or UML 1.4, UML 2.0 is default). Uncheck the **Store package properties in package diagram files** if you like them being stored in txaPackage files. Check **Create design elements in separate files** if you like to have each model element stored in its own txa* file. Click Next
5. Skip the **Profiles** page unless you like to enable one or more profiles for your project. Click Next
6. On the **Java Settings** page, **Source** tab, click the **Link Additional Source to Project** button in the upper right toolbar, use the **Browse...** button to specify the path to the existing source code folder. Click **Finish**. If you like this linked folder to be the only source folder for your project, remove the default source folder using **Remove** button.
7. Click **Finish**.

To import source code to the existing modeling project

1. Right-click a source folder of the target Java Modeling project in the **Navigator** view and select **Import...** on the context menu.
2. In the **Import** wizard, select **General** ▸ **File System** and click **Next**.
3. Browse to the folder with source code to import.
4. Select sources you want to import or click **Select all** to import the entire folder.
5. Click **Finish**.

Related Procedures

Creating a Project

Showing libraries

When you create a project, you can define directories with any number of search paths whose content you want to show in diagrams. For example, you can show entities that reside in the standard Java libraries. Such resources exist for the project, but Together does not include them in the generated HTML documentation for the project.

To show classes or packages from the standard Java libraries in a class diagram

1. Open or create a class diagram.
2. Right-click on the background and choose **New > Shortcut**. The Shortcuts dialog opens displaying available model elements.
3. Under the Model Elements tab, expand the libraries node, and navigate to the resource you want to add. Click Add. Repeat until you have added all the resources you want.
4. Click OK to close the dialog.

Tip: If the resource you are looking for is not shown, it is probably not in the Java build paths defined in Project Properties. You can add resources to the Java build paths at any time by using the Navigator view (Navigator view is not the same as Model Navigator View). Right-click the project in the Navigator view, and select **Properties** from the context menu. In the dialog that displays, select **Java Build Path**. Add the appropriate paths to your project by using the different tabs listed on the Java Build Path page.

Note: The new command available from the context menu, **New**, is disabled for classes that have been added from libraries (or compiled source code) to the diagram.

Troubleshooting a Model

You can also reload your project from the source code.

Use the following techniques to troubleshoot your model:

1 Refresh a model
2 Reload a model
3 Fix a model

To refresh a model

1 Open the **Diagram View**.
2 Press F6.

To reload a model

1 Open the **Model View**.
2 Right-click the project root node and choose **Reload** on the context menu.

Note: Use the Reload command as a workaround for issues that might appear while making changes in Together that cause some elements on the diagram to stop responding. The command is also helpful if you get certain errors from Together, such as `<undefined value>`.

Tip: Usually, when these problems occur, the elements also disappear from the Together **Structure View Class View** and the corresponding source code is underlined in blue in the Together Editor. Together cannot always properly handle such elements that become broken. To restore broken elements to a normal state, edit the code in the text editor according to the recommendation shown in the Together Editor. In these cases, it is best to refresh the model using Reload to prevent further problems.

To fix a model

1 For interaction diagrams, regenerate them from the source code.
2 For all types of diagrams, check that none of the necessary elements are hidden.

Related Procedures

Navigating between the Tree View, Diagram, and Source Code

Using Example Projects

Together comes with a set of predefined sample projects.

To use a Together Example Project

1. Select **File** ▶ **New** ▶ **Project** on the main menu. The **New Project** wizard opens.
2. Expand the **Examples** node in the tree view list, and select the project you want. Click **Next**.
3. Follow the wizard steps to specify the necessary options for a new project and click **Finish** to complete the wizard.

Tip: Alternatively, choose **File** ▶ **New** ▶ **Example** on the main menu.

Related Concepts

Together Project Overview

Related Procedures

Together Projects

Related Reference

Together Projects

Working with a Package

By default, a package element on diagram displays the package contents.

You can accomplish the following tasks for a package:

1. Open a package
2. Modify package contents
3. Delete a package
4. Rename a package
5. Move a package

To open a package

1. Select the package in the Diagram Editor or in the Model Navigator.
2. Choose the **Open** or **Open in Active Editor** command on the package context menu.

Tip: Alternatively, double-click the package element on the diagram.

To modify package contents

1. To add an element, choose **New <element>** on the package context menu.

   Tip: You can use the context menu of a nested element in a package to add its fields and subelements directly without opening it in diagram.

2. To delete an element from a package, press the DELETE key.

To delete a package

1. Select the package in the Diagram Editor or in the Model Navigator.
2. Choose **Delete** on its context menu.

   Warning: Deleting a package also deletes all of its contents.

To rename a package

1. Select the package in the Diagram Editor or in the Model Navigator.
2. To rename the package, including changing its name in all of its source files, do one of the following:
   - Choose **Rename** on the context menu of the package in the Diagram Editor or in the Model Navigator.
   - Press F2 to invoke the in-place editor for the package element in the Diagram Editor or in the Model Navigator.

- ♦ Edit the Name field in the Properties View

To move a package

1. Select the package in the Diagram Editor or in the Model Navigator.
2. Drag the package and drop it to the target location.

Warning: It is not recommended to undo move operations for packages.

To split package diagram persistence

1. Right-click the project in the Model Navigator and choose **Properties**.
2. Make sure the **Store package properties in package diagram files** option is not checked (this option is on by default).

   The default setting specifies that all properties of the package diagram, both visual and semantical, are preserved in the `default.txvpck` diagram file. With this option off, only diagram-specific information (visual information, such as layout) is retained in the `default.txvpck` diagram file, while settings that you treat as package properties (semantical information, such as descriptions and custom properties) are moved from the `default.txvpck` file into the `default.txaPackage` file. This allows you to track your package changes using version control.
3. Click **OK**.

   Note: Changing this option from Project Properties dialog converts the project files. This option can also be set using the New Project Wizard.

Related Concepts

Package Overview

XMI Export and Import of the Models with Cross-Project References

You can import and export multi-root projects using XMI. Note that XMI import and export is implemented differently for UML 1.4 and Java modeling projects, and for UML 2.0 projects.

- **UML 1.4 and Java modeling projects:** When a project that contains cross-project references is exported to an XMI file, the main project root and referenced roots are exported to the same XMI file. The **Use prefix of imported root** option of the **Export Wizard** enables you to reproduce the package structure of each root in top-level packages named as the root prefixes. If the option is unchecked, all same-named packages from the different roots are merged. When an XMI file is imported, the resulting project contains all packages and elements from the main model and referenced roots.
- **UML 2.0 projects:** When a project that contains cross-project references is exported to an XMI file, `*.imports.uml2` special files are created for each referenced root. The exported XMI file contains references to these files. When an XMI file is imported, the resulting project contains the main model only. If the referenced roots still exist in the workspace, the resulting UML 2.0 model recognizes them. References to the elements from these roots can be resolved only if the unique identifiers (UINs) of the elements have not been changed since export. Note that when an element is moved, its container is changed, and this can change the UIN.

To export a UML 1.4 and Java modeling project with cross-project references

1. On the main menu, choose **File Export**.
2. On the first page of the **Export Wizard**, select XMI file under Modeling and click **Next**.
3. On the second page of the wizard:
   - Select the project to be exported;
   - Select the XMI type and encoding;
   - Specify the export destination;
   - Check the **Use prefix of imported root** option if you want to reproduce the package structure of each root in top-level packages named as the root prefixes. By default, this option is unchecked.
4. Click **Finish**.

Tip: Package prefixes of the referenced roots are never used if you perform an export via the `XMIExport.cmd` command line utility.

To export a UML 2.0 project with cross-project references

1. On the main menu, choose **File Export**.
2. On the first page of the **Export Wizard**, select XMI file under Modeling and click **Next**.
3. On the second page of the wizard:
   - Select XMI for UML 2.0 as the project to be exported
   - Specify export destination
4. Click **Finish**.

Related Concepts

Together Interoperability and Migration
Model Import and Export Overview

Related Procedures

Importing a Project in XMI Format
Exporting a Project to XMI Format

Together Profiles

Together allows you to model diagrams using several preinstalled profiles as well as profiles created with Profile Definition projects.

In This Section

A Typical User Scenario of Working With Profiles

To create, deploy and apply a new profile definition, perform the following general steps:

1. Create a Profile Definition project. While creating a Profile Definition project, specify a UML version that the profile is targeted for (UML 2.0 by default). Metaclasses referenced in a profile must be taken from the corresponding target UML metamodel.

   Creating a Project

2. Open the default class diagram of your Profile Definition project and edit the profile properties:

   Defining Profile Properties

3. Create Stereotypes:

   Creating Stereotypes

4. Edit Stereotypes (edit properties, create shortcuts to metaclasses):

   Adding Shortcuts to Metaclasses

5. Add attributes (tagged values) to the stereotypes:

   Adding Attributes to Stereotypes

6. Define view properties for the stereotypes:

   Setting Viewmap Properties for Stereotypes

7. Create Palette Contributions; fill them with the contributed stereotypes or pure metaclasses.

   Creating Palette Contributions

8. Deploy profile:

   Deploying Profiles

9. Apply profile:

   Applying Profiles

Related Concepts

UML Profiles Basics

Related Procedures

Together Profiles

Creating a Project

Together provides several projects that you can work with. The projects in Together are created in the same manner. While creating a project, you will specify different options depending on the type of project.

To create a Together Project

1 Select **File** ▸ **New** ▸ **Project** on the main menu. The **New Project** wizard displays.

2 Expand the **Modeling** node in the tree view list, and select the type of project you want to create. Click **Next**.

3 Follow the wizard steps to specify necessary options for a new project and click **Finish** to complete the wizard.

Related Concepts

Together Project Overview

Related Procedures

Together Projects

Related Reference

Together Projects

Defining Profile Properties

When a Profile Definition project is created, you can specify or edit profile properties that are accessible via the default package diagram of the Profile Definition project. These properties are:

- Textual profile description
- Namespace, which identifies the profile

To specify profile definition properties

1. Open the default package diagram of the Profile Definition project.
2. On the context menu of the diagram, choose **Properties**. The Properties View opens.
3. In the Properties View, select the **Profile Definition** tab.
4. Click the **description** field and enter the description text. Optionally, click the **Edit** button.
5. Click the **namespace** field and enter a valid string.

Related Reference

Profile Definition Properties

Creating Stereotypes

To create a Stereotype

1. Using the **Profile Definition** palette, add a **Stereotype** node to the diagram background.
2. In the Properties View, choose **Profile Definition** node.
3. In the **Extended metaclass** field, click the **Edit** button.

   In the dialog box that opens, select the desired metaclasses from the **Model Elements** pane. Use the **Add** and **Remove** buttons to make up a list of extended metaclasses. Click **OK** when you are finished.
4. If necessary, specify the required stereotype property:

   Working with Required Stereotypes
5. Define view properties.

   Tip: View properties are not available for the stereotypes that extend across multiple metaclasses.

   Setting Viewmap Properties for Stereotypes
6. Add attributes (tagged values) to the stereotype:

   Adding Attributes to Stereotypes
7. Using **Contribution link**, connect the Stereotype to the desired Palette Contribution.

Related Procedures

Working with Required Stereotypes
Setting Viewmap Properties for Stereotypes
Adding Attributes to Stereotypes

Adding Shortcuts to Metaclasses

When creating your profile definition project, you can use shortcuts to metatypes from the metamodel root. Shortcuts to metaclasses can be created in several ways, some of which are similar to adding any other shortcut to your diagram.

To use the Shortcuts dialog box

1. Right-click the diagram background and select **Shortcuts** ▸ **New**. The **Shortcuts** dialog box opens.
2. Expand the metamodels node, choose the desired metaclass, and click **Add**.
3. Use the **Add** and **Remove** buttons to make up a list of extended metaclasses.
4. Click OK when you are finished.

To use cut, copy, and paste operations

1. Cut, copy, or drag a metaclass in the Model Navigator
2. Paste and drop it to any diagram that is not a package diagram.

To use the drag-and-drop operation to create a shortcut on a package diagram, press and hold the CTRL and SHIFT keys while dragging an element from the Model Navigator to your package.

Related Procedures

Profile Definition Properties

Adding Attributes to Stereotypes

In this section you will learn how to add attributes to stereotypes, and how to define attribute properties, groupings and descriptions. After applying a profile, stereotype attributes become visible in the Properties View of the elements with this stereotype.

You can add attributes to stereotypes in one of the following ways:

- Using the Properties View
- Using outgoing association links

To add attributes (tagged values) to a stereotype using the Properties View

1. Right-click on the selected stereotype in a profile definition diagram, and choose **New** ▸ **Attribute** on the context menu. Add as many attributes as required.
2. Select an attribute in the stereotype node. Its properties are displayed in the Properties View.
3. In the **Properties** tab:
   - Choose the **name** field and enter the attribute name.
   - Click the **type** field and specify the valid type using the combobox for primitive types or the selection manager dialog for the enumerations and metaclasses.

   Note: Invalid types are ignored during profile deployment.
4. In the **Profile Definition** tab, click the **inspector group** field, and select the inspector group from the list of existing groups, or create a new one. After applying the profile, the attribute is displayed in a separate tab (group) of the Properties View.
5. In the **Description** tab, choose the **Edit** tab and enter description text. After applying the profile, this description shows up as a tooltip in the tab (group) of the Properties View.

After the attribute is added to the stereotype, the tagged value name is equal to the attribute name. Multiplicity of the tagged value is equal to the attribute's multiplicity.

Note: If a profile defines attributes with a default value for a given type that is different from the default value in the underlying implementation, some side effects should be noted. For example, the default value for a Boolean property is `false` and is not persisted in the model instance. If a Boolean property is set to `true` as the default value in a profile definition and a user sets an instance value to `false`, its value is not persisted but interpreted as the default value (`true`) when read back in. Similarly, instance values changed to empty/null will not be persisted and will likewise be interpreted as the default value when read back in.

To add attributes (tagged values) to a stereotype using outgoing association links

1. On the profile definition diagram, create shortcuts to certain types (Primitive types, Enumerations or Metaclasses). Note that invalid types are ignored during profile deployment.
2. In the Class Diagram group of the Tool Palette, select the Association link and draw it from the stereotype to a shortcut to the type that you have chosen.
3. Select the created association link. Its properties are displayed in the Properties View.
4. In the **Supplier** tab, specify the following properties: `supplier multiplicity`, `supplier role`

5 If necessary, specify the inspector group and description, as described in the steps 4 and 5 of the previous procedure.

After the attribute is added to the stereotype, the supplier role (if specified) is used as the tagged value name. If the supplier role is not specified, the link name or default name is used as a tagged value name. Multiplicity of the tagged value is equal to the supplier multiplicity. If the upper value of the supplier multiplicity is greater than 1, the attribute is treated as multivalued.

Note: When the attribute of a metaclass type or the association to a metaclass shortcut is added to a stereotype, this attribute or association receives the viewmap and icon properties specified in the Profile Definition section of the Properties View. These properties allow you to select the viewmap and icon for the link that appears when this profile-specific property is set in the target project for the element with this stereotype.

Related Reference

Profile Definition Properties

Setting Viewmap Properties for Stereotypes

You can specify a visual representation of the elements in the profile you create. Values in the icon and viewmap properties affect the way the elements are displayed on your diagram.

Note: Viewmap property values are different for stereotypes that extend links.

To set viewmap and icon properties

1. Select a stereotype rectangle. The **Properties** view displays the properties of the selected stereotype.
2. In the **Profile Definition** node of the **Properties** view, choose the extended metaclass field and click the **Edit** button.
3. In the selection manager dialog, navigate to the a metaclass and click **Add**. You can select several extended metaclasses. Click **OK** when you are finished.

   Tip: The viewmap property is available for the stereotypes that extend one metaclass only.

4. After a value for the extended metaclass property is provided, viewmap and icon properties are displayed.
5. Select the viewmap field and click the **Edit** button. This opens the **Viewmap Editor** dialog box. Note that viewmap depends on the type of the extended metaclass. There are different sets of viewmap properties for the links and nodes.
6. Specify the viewmap properties. For a stereotype that extends a node, select **color** to specify color for the element, or select **svg** to browse for an `svg` file. If you choose **svg**, you can also specify the figure from those described in the selected `svg` file and specify whether the graphical node with the selected `.svg` figure will be resizable. For a stereotype that extends a link, specify values for Source decoration, Target decoration, Foreground, Background, and Line style options.
7. Click **OK** to save the changes and close the dialog box.

Creating Palette Contributions

In this section you will learn how to create a new tool, assign target diagrams and define the tool icon.

A Palette Contribution is defined by the two basic notions:

- its target diagram
- contributed stereotypes or pure metaclasses

To create a Palette Contribution

1. Using the **Profile Definition** tool, add a Palette Contribution node to the diagram background.
2. In the Properties View, choose **Profile Definition** node.
3. In the **diagrams** field, define the diagram types where you want the new creation tools to appear. Click the **Edit** button and in the **Select Diagrams** dialog box that opens, check the desired diagram types.
4. In the **icon** field, click the **Edit** button. In the **Select Icon** dialog box that opens, navigate to the `*.gif` file you want using the **Copy From File System** button.

   Tip: This dialog lets you arrange your icons in an orderly way. Use the **Create New Directory** button to create a special folder for storing icons, and populate it with the required images. This is useful for the large shared projects.

5. Using the Contribution link, define the contributed stereotype set:
   - Linking to a stereotype defines the contributed stereotype
   - Linking to a metaclass shortcut defines the contributed pure metaclass

Related Concepts

UML Profiles Basics

Related Procedures

Together Profiles

Deploying Profiles

After you create one or more profiles, you can create profile plugins to share them with your team members.

To deploy a created profile

1. Select the Profile Definition project or any project element in the Diagram Editor or Navigator view.
2. Choose **Model ▸ Profile ▸ Deploy profile**. The **Deploy Profile** wizard opens.
3. In the **Profile Content Project Settings** page, update project and plugin settings as required and click **Next**.
4. In the **Behavior** page, define the way the new profile will be deployed. Follow the notes of the wizard. Click **Next**.
5. In the **Target Directory** page, select the directory where the plugin will be deployed. You can choose from the default location, linked folders or an external location outside of the Eclipse platform. Click **Finish**.
6. Any errors that occur during the profile validation are reported in the **Profile Validation Results** view. You can navigate from an error message to the respective profile definition element and correct the error. When you are ready, click the **Deploy** button in the view.
7. After the profile plugin is deployed, you will be prompted to restart the workbench and make the new profile available in the list of supported profiles. Click **Yes** to restart.

Note: Because the profiles are internationalized on creation, you can edit the `.properties` file inside your new profile plugin to provide any strings.

Related Procedures

Uninstalling Profiles

Applying Profiles

Profile plugins that you create can be distributed among your team members.

If you have just created a profile plugin, you need to restart Together for the changes to take effect and for the plugin to become available in the program.

To enable a created profile

1. Select the **Navigator** view tab. If this view is not open, select **Window** ▸ **Show View** ▸ **Navigator** on the main menu.
2. In the **Navigator** view, right-click the root project folder, and select **Properties** from the context menu. The **Properties** dialog box displays.
3. From the list on the left, select **UML Profiles**.
4. Select the profile you created. More than one can be activated.
5. Click **OK**.

Note: There is no binary compatibility of compiled profiles across the various operating systems and versions of Together. You can copy a deployed profile to the plugins folder on another computer if the operating system and Together version are the same.

Related Procedures

UML Profiles Basics
Profile Definition Project
A Typical User Scenario of Working With Profiles

Adding Attributes to Stereotypes

In this section you will learn how to add attributes to stereotypes, and how to define attribute properties, groupings and descriptions. After applying a profile, stereotype attributes become visible in the Properties View of the elements with this stereotype.

You can add attributes to stereotypes in one of the following ways:

- Using the Properties View
- Using outgoing association links

To add attributes (tagged values) to a stereotype using the Properties View

1. Right-click on the selected stereotype in a profile definition diagram, and choose New ▸ Attribute on the context menu. Add as many attributes as required.
2. Select an attribute in the stereotype node. Its properties are displayed in the Properties View.
3. In the **Properties** tab:
   - Choose the **name** field and enter the attribute name.
   - Click the **type** field and specify the valid type using the combobox for primitive types or the selection manager dialog for the enumerations and metaclasses.

   Note: Invalid types are ignored during profile deployment.
4. In the **Profile Definition** tab, click the **inspector group** field, and select the inspector group from the list of existing groups, or create a new one. After applying the profile, the attribute is displayed in a separate tab (group) of the Properties View.
5. In the **Description** tab, choose the **Edit** tab and enter description text. After applying the profile, this description shows up as a tooltip in the tab (group) of the Properties View.

After the attribute is added to the stereotype, the tagged value name is equal to the attribute name. Multiplicity of the tagged value is equal to the attribute's multiplicity.

Note: If a profile defines attributes with a default value for a given type that is different from the default value in the underlying implementation, some side effects should be noted. For example, the default value for a Boolean property is `false` and is not persisted in the model instance. If a Boolean property is set to `true` as the default value in a profile definition and a user sets an instance value to `false`, its value is not persisted but interpreted as the default value (`true`) when read back in. Similarly, instance values changed to empty/null will not be persisted and will likewise be interpreted as the default value when read back in.

To add attributes (tagged values) to a stereotype using outgoing association links

1. On the profile definition diagram, create shortcuts to certain types (Primitive types, Enumerations or Metaclasses). Note that invalid types are ignored during profile deployment.
2. In the Class Diagram group of the Tool Palette, select the Association link and draw it from the stereotype to a shortcut to the type that you have chosen.
3. Select the created association link. Its properties are displayed in the Properties View.
4. In the **Supplier** tab, specify the following properties: `supplier multiplicity`, `supplier role`

5 If necessary, specify the inspector group and description, as described in the steps 4 and 5 of the previous procedure.

After the attribute is added to the stereotype, the supplier role (if specified) is used as the tagged value name. If the supplier role is not specified, the link name or default name is used as a tagged value name. Multiplicity of the tagged value is equal to the supplier multiplicity. If the upper value of the supplier multiplicity is greater than 1, the attribute is treated as multivalued.

Note: When the attribute of a metaclass type or the association to a metaclass shortcut is added to a stereotype, this attribute or association receives the viewmap and icon properties specified in the Profile Definition section of the Properties View. These properties allow you to select the viewmap and icon for the link that appears when this profile-specific property is set in the target project for the element with this stereotype.

Related Reference

Profile Definition Properties

Adding Shortcuts to Metaclasses

When creating your profile definition project, you can use shortcuts to metatypes from the metamodel root. Shortcuts to metaclasses can be created in several ways, some of which are similar to adding any other shortcut to your diagram.

To use the Shortcuts dialog box

1. Right-click the diagram background and select **Shortcuts** ▶ **New**. The **Shortcuts** dialog box opens.
2. Expand the metamodels node, choose the desired metaclass, and click **Add**.
3. Use the **Add** and **Remove** buttons to make up a list of extended metaclasses.
4. Click OK when you are finished.

To use cut, copy, and paste operations

1. Cut, copy, or drag a metaclass in the Model Navigator
2. Paste and drop it to any diagram that is not a package diagram.

To use the drag-and-drop operation to create a shortcut on a package diagram, press and hold the CTRL and SHIFT keys while dragging an element from the Model Navigator to your package.

Related Procedures

Profile Definition Properties

Applying Profiles

Profile plugins that you create can be distributed among your team members.

If you have just created a profile plugin, you need to restart Together for the changes to take effect and for the plugin to become available in the program.

To enable a created profile

1. Select the **Navigator** view tab. If this view is not open, select **Window ▸ Show View ▸ Navigator** on the main menu.
2. In the **Navigator** view, right-click the root project folder, and select **Properties** from the context menu. The **Properties** dialog box displays.
3. From the list on the left, select **UML Profiles**.
4. Select the profile you created. More than one can be activated.
5. Click **OK**.

Note: There is no binary compatibility of compiled profiles across the various operating systems and versions of Together. You can copy a deployed profile to the plugins folder on another computer if the operating system and Together version are the same.

Related Procedures

UML Profiles Basics
Profile Definition Project
A Typical User Scenario of Working With Profiles

Converting Profile-Specific Properties

The converting profiles function helps you reuse projects from Together 2006 in which custom profiles were applied.

This feature is useful for the following scenario:

1. In Together 2006, a profile has been created and deployed. This results in creating a profile plugin.
2. This profile plugin is applied to a certain modeling project.
3. The same profile definition is reused and deployed in Together 2006 R2. This results in creating another profile plugin, which has different properties names.
4. The same modeling project is opened in Together 2006 R2. On an attempt to apply the new profile plugin to this project, the profile-specific properties will loose their values unless they are properly converted.

To convert profile-specific properties

1. On the main menu, choose **Model** ▸ **Profile** ▸ **Convert Properties**.
2. If there are no profile-specific properties in the project, no action is performed.

Related Concepts

UML Profiles

Creating Palette Contributions

In this section you will learn how to create a new tool, assign target diagrams and define the tool icon.

A Palette Contribution is defined by the two basic notions:

- its target diagram
- contributed stereotypes or pure metaclasses

To create a Palette Contribution

1. Using the **Profile Definition** tool, add a Palette Contribution node to the diagram background.
2. In the Properties View, choose **Profile Definition** node.
3. In the **diagrams** field, define the diagram types where you want the new creation tools to appear. Click the **Edit** button and in the **Select Diagrams** dialog box that opens, check the desired diagram types.
4. In the **icon** field, click the **Edit** button. In the **Select Icon** dialog box that opens, navigate to the `*.gif` file you want using the **Copy From File System** button.

   Tip: This dialog lets you arrange your icons in an orderly way. Use the **Create New Directory** button to create a special folder for storing icons, and populate it with the required images. This is useful for the large shared projects.

5. Using the Contribution link, define the contributed stereotype set:
   - Linking to a stereotype defines the contributed stereotype
   - Linking to a metaclass shortcut defines the contributed pure metaclass

Related Concepts

UML Profiles Basics

Related Procedures

Together Profiles

Creating Profile-Specific Constraints

When defining your profile, you can create a set of specific audits available only for projects with the applied profile. Such audits can be created as constraints linked to metaclasses in the profile definition project. Note that a constraint context can be represented only by a metaclass from the target metamodel.

For the following procedure, a stereotype `MyStereotype` has been defined for `uml20::classes::Class`, and you want to verify that the class with this stereotype only extends class with the same stereotype.

To provide the audit, do the following in your profile definition project:

1. Create a shortcut to the `uml20::classes::Class` metaclass.
2. Create a constraint element.
3. Link the created constraint with the metaclass shortcut (it gets the context `uml20::classes::Class`).
4. Type the following in the body of the constraint: `inv:stereotypes->includes('MyStereotype') implies generalizations->forAll( general.stereotypes->includes('MyStereotype'))`
5. Deploy the profile.

After the profile is applied to some project, it is possible to run profile-specific audits via the **Model ▸ Profile ▸ Run Profile Constrains** command.

Note: The **description** and **name** properties of the constraint element, specified in the Properties View, are used in the new audit. The value of the **description** property is used as the audit description, and the constraint name and invariant name are used as the audit name.

Related Concepts

UML Profiles Basics

Related Procedures

Together Profiles

Creating Stereotypes

To create a Stereotype

1. Using the **Profile Definition** palette, add a **Stereotype** node to the diagram background.
2. In the Properties View, choose **Profile Definition** node.
3. In the **Extended metaclass** field, click the **Edit** button.

   In the dialog box that opens, select the desired metaclasses from the **Model Elements** pane. Use the **Add** and **Remove** buttons to make up a list of extended metaclasses. Click **OK** when you are finished.
4. If necessary, specify the required stereotype property:

   Working with Required Stereotypes
5. Define view properties.

   Tip: View properties are not available for the stereotypes that extend across multiple metaclasses.

   Setting Viewmap Properties for Stereotypes
6. Add attributes (tagged values) to the stereotype:

   Adding Attributes to Stereotypes
7. Using **Contribution link**, connect the Stereotype to the desired Palette Contribution.

Related Procedures

Working with Required Stereotypes
Setting Viewmap Properties for Stereotypes
Adding Attributes to Stereotypes

Working with Required Stereotypes

In this section you will learn how to create a required stereotype and how to manage stereotypes in diagrams after applying or removing the parent profile of a stereotype.

To create a required stereotype

1 In a Profile Definition, select a stereotype that extends a metaclass.
2 Select an extension link. In the Model Navigator, expand the stereotype node and click the extension link.

> **Tip:** Alternatively, add a shortcut to the parent metaclass to the Profile Definition. The extension link to the extending stereotype is drawn automatically.

3 In the Properties View of the extension link, select the **Profile Definition** tab.
4 Set the **is required** property to `true`. The extension link in the diagram gets the `{isRequired}` label, and the **bind with profile** field appears in the Properties View.
5 Set the **bind with profile** field as required:

- ♦ If the field is set to `true`, after applying the profile to a project, the appropriate elements get the required stereotype. After the parent profile is turned off, this stereotype is removed from the elements.
- ♦ If the field is set to `false`, after applying the profile to a project, the appropriate elements get the required stereotype. After the parent profile is turned off, this stereotype is preserved.

Tip: This feature is useful for the large team projects and helps avoid confusion that might be caused by applying custom profiles.

To filter out required stereotypes in diagrams

1 On the main menu, choose **Window** ▶ **Preferences** ▶ **Modeling** ▶ **Profiles** ▶ **View Management**.
2 Click the tab that corresponds to the appropriate metamodel.
3 In the list of available profiles, check the stereotypes you would like to hide in diagrams.
4 Apply the changes and close the dialog.

Related Concepts

UML Profiles Basics

Related Procedures

Creating Stereotypes

Related Reference

Profile Definition Properties
UML Profiles Preferences View Management

Setting Viewmap Properties for Stereotypes

You can specify a visual representation of the elements in the profile you create. Values in the icon and viewmap properties affect the way the elements are displayed on your diagram.

Note: Viewmap property values are different for stereotypes that extend links.

To set viewmap and icon properties

1. Select a stereotype rectangle. The **Properties** view displays the properties of the selected stereotype.
2. In the **Profile Definition** node of the **Properties** view, choose the extended metaclass field and click the **Edit** button.
3. In the selection manager dialog, navigate to the a metaclass and click **Add**. You can select several extended metaclasses. Click **OK** when you are finished.

   Tip: The viewmap property is available for the stereotypes that extend one metaclass only.

4. After a value for the extended metaclass property is provided, viewmap and icon properties are displayed.
5. Select the viewmap field and click the **Edit** button. This opens the **Viewmap Editor** dialog box. Note that viewmap depends on the type of the extended metaclass. There are different sets of viewmap properties for the links and nodes.
6. Specify the viewmap properties. For a stereotype that extends a node, select **color** to specify color for the element, or select **svg** to browse for an `svg` file. If you choose **svg**, you can also specify the figure from those described in the selected `svg` file and specify whether the graphical node with the selected `.svg` figure will be resizable. For a stereotype that extends a link, specify values for Source decoration, Target decoration, Foreground, Background, and Line style options.
7. Click **OK** to save the changes and close the dialog box.

Adding Attributes to Stereotypes

In this section you will learn how to add attributes to stereotypes, and how to define attribute properties, groupings and descriptions. After applying a profile, stereotype attributes become visible in the Properties View of the elements with this stereotype.

You can add attributes to stereotypes in one of the following ways:

- Using the Properties View
- Using outgoing association links

To add attributes (tagged values) to a stereotype using the Properties View

1. Right-click on the selected stereotype in a profile definition diagram, and choose **New ▸ Attribute** on the context menu. Add as many attributes as required.
2. Select an attribute in the stereotype node. Its properties are displayed in the Properties View.
3. In the **Properties** tab:

    - Choose the **name** field and enter the attribute name.
    - Click the **type** field and specify the valid type using the combobox for primitive types or the selection manager dialog for the enumerations and metaclasses.

    Note: Invalid types are ignored during profile deployment.

4. In the **Profile Definition** tab, click the **inspector group** field, and select the inspector group from the list of existing groups, or create a new one. After applying the profile, the attribute is displayed in a separate tab (group) of the Properties View.
5. In the **Description** tab, choose the **Edit** tab and enter description text. After applying the profile, this description shows up as a tooltip in the tab (group) of the Properties View.

After the attribute is added to the stereotype, the tagged value name is equal to the attribute name. Multiplicity of the tagged value is equal to the attribute's multiplicity.

Note: If a profile defines attributes with a default value for a given type that is different from the default value in the underlying implementation, some side effects should be noted. For example, the default value for a Boolean property is `false` and is not persisted in the model instance. If a Boolean property is set to `true` as the default value in a profile definition and a user sets an instance value to `false`, its value is not persisted but interpreted as the default value (`true`) when read back in. Similarly, instance values changed to empty/null will not be persisted and will likewise be interpreted as the default value when read back in.

To add attributes (tagged values) to a stereotype using outgoing association links

1. On the profile definition diagram, create shortcuts to certain types (Primitive types, Enumerations or Metaclasses). Note that invalid types are ignored during profile deployment.
2. In the Class Diagram group of the Tool Palette, select the Association link and draw it from the stereotype to a shortcut to the type that you have chosen.
3. Select the created association link. Its properties are displayed in the Properties View.
4. In the **Supplier** tab, specify the following properties: `supplier multiplicity`, `supplier role`

5 If necessary, specify the inspector group and description, as described in the steps 4 and 5 of the previous procedure.

After the attribute is added to the stereotype, the supplier role (if specified) is used as the tagged value name. If the supplier role is not specified, the link name or default name is used as a tagged value name. Multiplicity of the tagged value is equal to the supplier multiplicity. If the upper value of the supplier multiplicity is greater than 1, the attribute is treated as multivalued.

Note: When the attribute of a metaclass type or the association to a metaclass shortcut is added to a stereotype, this attribute or association receives the viewmap and icon properties specified in the Profile Definition section of the Properties View. These properties allow you to select the viewmap and icon for the link that appears when this profile-specific property is set in the target project for the element with this stereotype.

Related Reference

Profile Definition Properties

Defining Profile Properties

When a Profile Definition project is created, you can specify or edit profile properties that are accessible via the default package diagram of the Profile Definition project. These properties are:

- Textual profile description
- Namespace, which identifies the profile

To specify profile definition properties

1. Open the default package diagram of the Profile Definition project.
2. On the context menu of the diagram, choose **Properties**. The Properties View opens.
3. In the Properties View, select the **Profile Definition** tab.
4. Click the **description** field and enter the description text. Optionally, click the **Edit** button.
5. Click the **namespace** field and enter a valid string.

Related Reference

Profile Definition Properties

Deploying Profiles

After you create one or more profiles, you can create profile plugins to share them with your team members.

To deploy a created profile

1. Select the Profile Definition project or any project element in the Diagram Editor or Navigator view.
2. Choose **Model** ▶ **Profile** ▶ **Deploy profile**. The **Deploy Profile** wizard opens.
3. In the **Profile Content Project Settings** page, update project and plugin settings as required and click **Next**.
4. In the **Behavior** page, define the way the new profile will be deployed. Follow the notes of the wizard. Click **Next**.
5. In the **Target Directory** page, select the directory where the plugin will be deployed. You can choose from the default location, linked folders or an external location outside of the Eclipse platform. Click **Finish**.
6. Any errors that occur during the profile validation are reported in the **Profile Validation Results** view. You can navigate from an error message to the respective profile definition element and correct the error. When you are ready, click the **Deploy** button in the view.
7. After the profile plugin is deployed, you will be prompted to restart the workbench and make the new profile available in the list of supported profiles. Click **Yes** to restart.

Note: Because the profiles are internationalized on creation, you can edit the `.properties` file inside your new profile plugin to provide any strings.

Related Procedures

Uninstalling Profiles

Enabling UML Profiles

There are several ways to enable UML profiles for Together projects.

To enable UML profiles support while creating a project

1. On the main menu, choose File ▸ New ▸ Project. The **New Project** wizard opens.
2. Expand the **Modeling** node in the tree view list, and select the UML project you want to create (UML 2.0 or UML 1.4). Click **Next**.
3. Follow the wizard to the **Profiles** screen. The **Profiles** screen of the wizard lists available profiles.
4. Select one or more profiles you want to enable and click **Next** to continue creating a new project with the **New Project** wizard.

To enable UML profiles support for existing projects

1. In the Model Navigator, right-click the root project folder, and select Properties on the context menu. The **Properties for <project>** dialog box displays.
2. From the list on the left, select **UML Profiles**.
3. Select any of the UML profiles that you want to enable. More than one can be activated.
4. Click **OK**.

Note: You can also access the **Properties for <project>** dialog box through the **Model Package Explorer** view and Navigator view.

To specify the default set of UML profiles enabled for all new workspace projects

1. Choose Window ▸ Preferences on the main menu.
2. In the left pane of the **Preferences** dialog box, expand the **Modeling** node.
3. Select the **UML Profiles** node.
4. Select the profiles you want to enable for UML 1.4 and UML 2.0 projects.

Note: The selected UML profiles are automatically enabled for projects created after you changed profile preferences. Profiles support of existing projects is not changed.

Exporting and Importing Profiles

In this section you will learn how to export and import profile plugins.

To export profiles

1. On the main menu, choose **File ▸ Export**.
2. In the **Export** dialog, under the **Modeling** node, choose **Profile Plug-ins** and click **Next** .
3. In the list of available profiles, check the ones to be exported.
4. Specify the target directory, entering its fully qualified name in the text field or clicking the **Browse** button.
5. Click **Finish**.

Note: When exporting, you do not have to copy default values for properties to the target directory. The data from the other model should provide similar functionality with default values if it has a similar active profile.

With XMI exports, default values are stored in the profile itself (the **defaultValue** property of the stereotype attribute). For example, when you export a Together model with a profile applied, two files are created: `model.uml` and `model.profile.uml`. The profile's default values are stored in the latter, and ownership of the profile default values are repeated.

To import profiles

1. On the main menu, choose **File ▸ Import**.
2. In the **Import** dialog, under the **Modeling** node, choose **Profile Plug-ins** and click **Next** .
3. In the **Search profile plug-ins in directory** field, specify the source directory where the plug-ins you want are stored. The list of available profiles is displayed.
4. In the list of profiles encountered in the specified folder, check the ones to be imported, and click **Next**.
5. Specify the target directory.
   - ♦ If you click the **Target plug-ins directory** radio button, the selected profile plug-ins will be imported to the default directory.
   - ♦ If you click the **Linked directory** radio button, the selected profile plug-ins will be imported to the linked directory of your choice. Optionally, you can link new directories using the **Link New Directory** button.
6. Click **Finish**.

Related Procedures

UML Profiles Basics

Opening Profile Definition

In this section you will learn how to view and modify definitions of the custom profiles and profiles that come bundled with Together. A profile definition opens as a UML 2.0 project.

To open a profile definition

1. On the main menu, choose **Model** ▸ **Profile** ▸ **Open Profile Definition**.

   The **Open Profile Definition** dialog displays the list of profiles that declare their definitions.

2. Check the profile you want and click **Finish**. The selected profile definition opens as a UML 2.0 project.

Related Concepts

Profile Definition Project

Related Procedures

UML Profiles Basics

Setting Viewmap Properties for Stereotypes

You can specify a visual representation of the elements in the profile you create. Values in the icon and viewmap properties affect the way the elements are displayed on your diagram.

Note: Viewmap property values are different for stereotypes that extend links.

To set viewmap and icon properties

1. Select a stereotype rectangle. The **Properties** view displays the properties of the selected stereotype.
2. In the **Profile Definition** node of the **Properties** view, choose the extended metaclass field and click the **Edit** button.
3. In the selection manager dialog, navigate to the a metaclass and click **Add**. You can select several extended metaclasses. Click **OK** when you are finished.

   Tip: The viewmap property is available for the stereotypes that extend one metaclass only.

4. After a value for the extended metaclass property is provided, viewmap and icon properties are displayed.
5. Select the viewmap field and click the **Edit** button. This opens the **Viewmap Editor** dialog box. Note that viewmap depends on the type of the extended metaclass. There are different sets of viewmap properties for the links and nodes.
6. Specify the viewmap properties. For a stereotype that extends a node, select **color** to specify color for the element, or select **svg** to browse for an `svg` file. If you choose **svg**, you can also specify the figure from those described in the selected `svg` file and specify whether the graphical node with the selected `.svg` figure will be resizable. For a stereotype that extends a link, specify values for Source decoration, Target decoration, Foreground, Background, and Line style options.
7. Click **OK** to save the changes and close the dialog box.

Uninstalling Profiles

The uninstalling profile feature enables you to correctly remove the unused profile plugins, which involves detaching them from all projects in your workspace and deleting them from the file system.

To uninstall a profile

1. On the main menu, choose **Model** ▸ **Profile** ▸ **Uninstall Profile**.
2. Select the profiles to be uninstalled.

Related Procedures

UML Profiles Basics

Verifying a Model Against Profile Constraints

Verification of a profile involves defining the necessary constraints within the target metamodel, and an actual verification of a model against the selected constraints.

To verify a model against an applied profile

1. On the main menu, choose **Window** ▸ **Preferences** ▸ **Profile** ▸ **Constraints**, and choose the desired constraints in the appropriate metamodel.
2. On the main menu, choose **Model** ▸ **Profile** ▸ **Run Constraints**.

Note: You can specify a scope for profile constraints after choosing **Model** ▸ **Profile** ▸ **Run Constraints**. The possible constraint restrictions are selected resource (single selection), package (shallow or deep), and project. The Profile Constraint history contains the project name. The project name is qualified with an actual item.

The results of verification display in the **Profile Constraints** view. You can navigate from any entry in the table to the respective element in diagram.

Related Procedures

UML Profiles Basics

Related Reference

UML Profiles Preferences Constraints

Working with Required Stereotypes

In this section you will learn how to create a required stereotype and how to manage stereotypes in diagrams after applying or removing the parent profile of a stereotype.

To create a required stereotype

1 In a Profile Definition, select a stereotype that extends a metaclass.

2 Select an extension link. In the Model Navigator, expand the stereotype node and click the extension link.

> **Tip:** Alternatively, add a shortcut to the parent metaclass to the Profile Definition. The extension link to the extending stereotype is drawn automatically.

3 In the Properties View of the extension link, select the **Profile Definition** tab.

4 Set the **is required** property to `true`. The extension link in the diagram gets the `{isRequired}` label, and the **bind with profile** field appears in the Properties View.

5 Set the **bind with profile** field as required:

- ♦ If the field is set to `true`, after applying the profile to a project, the appropriate elements get the required stereotype. After the parent profile is turned off, this stereotype is removed from the elements.
- ♦ If the field is set to `false`, after applying the profile to a project, the appropriate elements get the required stereotype. After the parent profile is turned off, this stereotype is preserved.

Tip: This feature is useful for the large team projects and helps avoid confusion that might be caused by applying custom profiles.

To filter out required stereotypes in diagrams

1 On the main menu, choose **Window** ▸ **Preferences** ▸ **Modeling** ▸ **Profiles** ▸ **View Management**.

2 Click the tab that corresponds to the appropriate metamodel.

3 In the list of available profiles, check the stereotypes you would like to hide in diagrams.

4 Apply the changes and close the dialog.

Related Concepts

UML Profiles Basics

Related Procedures

Creating Stereotypes

Related Reference

Profile Definition Properties
UML Profiles Preferences View Management

Configuring Implementation Projects

This part provides how-to information on setting Together preferences and options for the implementation projects.

In This Section

Configuring C++ Projects

In this section, you will learn how to define the project structure and processing options:

- Access C++ project properties
- Define source path
- Define entry points
- Include search paths
- Define C++ processing settings (for example, skip standard includes option, or suffixes for the C++ files)
- Define indexer
- Enable C++ formatting
- Set up formatting options

To configure a C++ project

1. Select the desired project in the Model Navigator.
2. On the main menu, choose **Project Properties**.

   Tip: Alternatively, choose **Properties** on the context menu of the project.

   The **Properties for <project>** dialog opens. Select the **Project Properties** page.
3. In the **Project source path** tab, click the **Link Additional Source to Project** button.
4. In the **Link Additional Source** dialog, specify the linked folder location and name, and click **OK**.

   Tip: Use the context menu of the linked folder to remove it, mark it read-only, or filter it.

5. Configure parsing entry points using the Configure Entry points dialog.
6. In the **Include paths** tab, click **Add**.
7. In the **Add Include Folder** dialog, enter the folder name or click **Browse** and navigate to the folder you want to add.
8. In the **C++ Processing Settings** tab, select your C++ project options.
   - To skip standard includes, check the **Skip standard includes** option.
   - If you want to use the preinclude file, specify its name in the **Preinclude file name** field.
9. Select the **C/C++ indexer** page, and select an indexer from the list. Among the available indexers, you can choose the Borland indexer.

To enable C++ formatter

1. On the main menu, choose **Window** ► **Preferences**
2. Under the C/C++ category, select the Code Formatter page.
3. From the list of available formatters, select Together C++ Code Formatter.

To set up formatting options

1. Under your Together installation, expand the `plugins` folder.
2. In the `com.borland.tg.cdtintegration` plugin, open the `formatter.properties` file.
3. Use the documentation provided with the file to edit as required.

Related Reference

New project Wizard C++ Language-Specific Options
C++ Projects

Configuring IDL Projects

In this section you will learn how to define the project structure and processing options:

- Access IDL project properties
- Define source path
- Include search paths
- Define IDL processing settings

To configure an IDL project

1. Select a project in the Model Navigator.
2. On the main menu, choose **Project Properties**.

    Tip: Alternatively, choose **Properties** on the context menu of the project.

3. In the **Project source path** tab, click the **Link Additional Source to Project** button.
4. In the **Link Additional Source** dialog, specify the linked folder location and name, and click **OK**.

    Tip: Use the context menu of the linked folder to remove it, mark it read-only, or filter it.

5. In the **Include paths** tab, click **Add**.
6. In the **Add Include Folder** dialog, enter the folder name or click **Browse** and navigate to the folder.
7. In the **IDL Processing Settings** tab, select your IDL project options. Refer to the IDL Language-Specific Options section for details.

Related Reference

New project Wizard IDL Language-Specific Options
IDL Language-Specific Information

Together UML 2.0 Diagrams

This section provides how-to information on using Together UML diagrams.

In This Section

UML 2.0 Class Diagrams Procedures

In This Section

Adding Owned Behavior to a Class

You can add behavior to a class. Behavior is defined by an activity, state machine or interaction.

Note: This feature is available in the design projects only.

To add a classifier behavior to a class

1. Select a class in a diagram.
2. In the Properties View, click the **classifier behavior** field.
3. In the **Select Behavior for Classifier Behavior Property** dialog, select the desired element in the Model Elements pane.
4. Use the **Add** and **Remove** buttons to make up a list of Selected Elements.
5. Click **OK** when you are finished.

The owned behaviors can be added to a classifier by pasting a behavior into the classifier (for example, cutting an activity and pasting it to a class) or via the Context menu. This way, the interaction can be added to the class, and activity and interaction can be added to the use case.

Related Concepts

UML 2.0 Class Diagram Definition

Related Procedures

UML 1.4 Class Diagrams Procedures

Changing the Appearance of Compartments

You can collapse or expand compartments for the different members of class, interface, and package elements. Use the **Preferences** dialog to set viewing preferences for compartment controls. Adding compartment controls is particularly useful when you have large container elements with content that does not need to be visible at all times.

To show compartment controls

1. On the main menu, choose **Window** ▸ **Preferences**.
2. Open the **Modeling** ▸ **View Management** page.
3. Check the **Always show Attributes and Operations compartments** option.

To collapse or expand compartments

1. Select the class (or interface) on the diagram.
2. Click the "**+**" or "**-**" in the left corner of the compartment.

Related Reference

UML 2.0 Class Diagrams
UML 1.4 Class Diagrams

Changing the Appearance of Interfaces

Note: This feature is available in the design projects only.

To show an interface as a circle using the context menu

1. Right-click the interface element in the **Diagram View** or **Model View**.
2. Choose **Show as circle**.

Tip: This menu item works as a toggle. Right-click again and choose **Show as circle** to show the interface element as a rectangle.

Note: Interfaces shown as small circles do not show their members in the **Diagram View**. Use the **Model View** to view the members.

To show an interface as a circle using the Properties View

1. Select the interface element in the **Diagram View** or **Model View**.
2. Press F4 to open the Properties View.
3. Set the **Circle view** property as True.

Tip: Choose False for the **Circle view** property to show the interface element as a rectangle.

Related Reference

UML 1.4 Class Diagrams
UML 2.0 Class Diagrams

Creating and Editing Properties

If recognizing Java Beans is enabled, creating a property results in creating a property attribute and its accessor methods in the source code. Properties display in a class icon in the Properties compartment. Optionally, you can show the participants of a property (its attribute and the getter and setter methods) in the Attributes and Operations compartments of the class icon, respectively. The participants are hidden by default.

When the Java Beans Properties Support is activated, using the cut, copy, paste, clone, or delete commands on a property member applies to the property attribute and to its getters and setters. If recognizing Java Beans is disabled, special care is required when editing or deleting properties. If you edit the property type using the in-place editor, the relevant types in the accessor methods will not be synchronized. The same result occurs when a property is deleted; that is, the accessor methods stay in place and should be deleted individually.

Note: This feature is available in the implementation projects only.

To activate or deactivate Java Beans

1. On the main menu, choose **Window** ▸ **Preferences**. The Preferences dialog opens.
2. Under the Modeling node, select Java.
3. Check the **Recognize Java Bean Properties** option. Refer to the Java Preferences description for details.

To add a property member to a class element

1. Select the target class in the diagram.
2. On the context menu, choose **New** ▸ **Property**.

To show property participants

1. On the main menu, choose **Window** ▸ **Preferences**. The Preferences dialog opens.
2. Under the Modeling node, select Java.
3. Uncheck the **Hide Java Bean Properties Participants** option. Refer to the Java Preferences description for details.

Related Procedures

Java Preferences

Creating Class By Template

Use the Class By Template button on the Palette diagram to implement source code constructions or solutions in your model.

Note: This feature is available in the implementation projects only.

To create a class by template

1. Select Class by Template in the Tools Palette.
2. Click on the diagram background. The **Apply template** dialog box opens.
3. Select the appropriate template from the **Templates** tree.
4. Set each value field within the **Parameters** area, or click **Finish** to apply default values.

Related Procedures

Apply Template Wizard

Creating Data Types

Data types are created as regular diagram elements, using the Tool Palette or **New Data Type** command on the diagram context menu. You can add attributes and operations to data types using the context menu.

Note: This feature is available in the design projects only.

To extend a data type

1. Select a data type in a diagram.
2. In the Properties View, select the **extends** field and click the chooser button.
3. In the **Select Data type for Extends Property** dialog, select the desired element in the Model Elements pane.
4. Use **Add** and **Remove** buttons to make up a list of Selected Elements.
5. Click **OK** when you are finished.

Related Concepts

UML 2.0 Class Diagram Definition

Related Procedures

Adding Owned Behavior to a Class

Creating Enumerations and Enumeration Literals

Enumerations are created as regular diagram elements, using the Tool Palette or the **New Enumeration** command on the diagram context menu.

To add an enumeration literal

1. Select an enumeration in the diagram.
2. On the context menu, choose **New Enumeration literal**. The new literal is added to the enumeration element.
3. In the Properties View, select the **name** field and enter the enumeration name. The name of the literal is displayed in diagram.
4. In the **specification** field enter the value that is displayed in the diagram next to the enumeration name, delimited by the equal sign.

To extend an enumeration

1. Select an enumeration in the diagram.
2. In the Properties View, select the **extends** field and click the chooser button.
3. In the **Select Enumeration for Extends Property** dialog, select the enumeration element you want in the Model Elements pane.
4. Use the **Add** and **Remove** buttons to make up a list of Selected Elements.
5. Click **OK** when you are finished.

Related Concepts

UML 2.0 Class Diagram Definition

Related Procedures

UML 1.4 Class Diagrams Procedures

Creating, Editing and Opening Header and Implementation Files in C++ Projects

C++ header files are automatically created when a class, interface or enumeration is added to a diagram. Creating an implementation file becomes possible when anything that is defined outside the header (for example, an operation or constructor) exists in the class.

To create an implementation file

1. Select a class in the diagram.
2. Right-click the selection and choose **New** on the context menu.
3. On the submenu, choose the member that you want (operation, constructor or destructor).
4. Right-click the member and choose **Create member definition** on the context menu.

An implementation file is created. If the implementation file already exists, the new member definition is added to this file. The implementation file opens in the separate tab of the editor.

To open a header file for editing

1. Select a class in the diagram.
2. Double-click the selected node.

To open an implementation file for editing

1. Select a class in the diagram.
2. Right-click the member and choose **Edit member definition** on the context menu.

Related Procedures

Special Considerations for C++ Projects

Working with a Constructor

You can create as many constructors in a class as needed using the **New ▸ Constructor** command of the context menu of a class.

In implementation projects, each new constructor is created with its unique set of parameters. In addition to creating parameters automatically, you can define the custom set of parameters using the Properties View.

In design projects, a constructor is created as an operation with the `<<create>>` stereotype.

Tip: You can move, copy and paste constructors and destructors between the container classes the same way as you would do the other members.

To define the constructor parameters

1. Select a constructor in a class.
2. In the Properties View, click the **Browse** button in the **parameters** field.
3. In the **Select Parameters for Operation** dialog that opens, click **Add**. A parameter is added with the default values. Edit the values as required, or use the defaults.

   Use the **Add** and **Remove** buttons to make up the list of parameters, and click **OK** when you are finished.

Tip: Alternatively, type the list of parameters in the text area. Use a comma as a delimiter.

Related Reference

UML 2.0 Class Diagrams
UML 1.4 Class Diagrams

Working with a Field

You can edit members using the Properties View, or the in-place editor of the Diagram Editor or Model Navigator. In the implementation projects, you can also use the source code editor to modify the members. In this section you will learn how to:

- rename a field
- define a visibility modifier
- define a stereotype
- define modifiers, initial values, and associated objects
- handle multi-declarations

To rename a field

1. Choose a field.
2. Enter the new name in the in-place editor of the Diagram Editor or Model Navigator, or use the **name** text field in the Properties View.

To define the visibility modifier

1. Choose a field.
2. Enter the visibility symbol in the in-place editor in the Diagram Editor , or select one from the **visibility** combobox in the Properties View.

To define the stereotype of a field

1. Choose a field.
2. Use the in-place editor in the Diagram Editor , or use the stereotype combobox of the Properties View.

To define modifiers, initial values, associated objects and so on

1. Choose a field.
2. Use the Properties View or the source code editor (for implementation projects).

When you do this, the model and the source code are kept in sync.

Note: You can type the *Value* property, an equal sign (`=`), and the *Name* property (for example, `EXCLUDE=2`) when adding an Enum literal with the inplace editor.

To use multi-declarations in the source code, consider the following:

1. In the source code of the Java and IDL projects, it is possible to declare several fields in one line. This notation is represented in the diagram as a number of separate entries in the Fields section in a class icon.
2. You can rename the fields, change modifiers, set initial values and so on, and all modifications will be applied to the respective field in the diagram icon.

3 You can copy and move such fields in a diagram (using the context menu commands or drag-and-drop), and the pasted field will appear in the target container separately.

In C++ projects, editing multi-declarations in the Diagram Editor or Properties View is not allowed.

Related Procedures

Navigating between the Tree View, Diagram, and Source Code
Adding a Single Model Element to a Diagram

Related Reference

UML 1.4 Class Diagrams
UML 2.0 Class Diagrams

Working with a Provided or Required Interface

To create a provided interface

1. Create class and interface node elements using the Palette buttons.
2. On the diagram Palette, click the **Provided Interface** button.
3. Click the client class and drag the mouse to the interface node.

To create a required interface

1. Create class and interface node elements using the Palette buttons.
2. On the diagram Palette, click the **Required Interface** button.
3. Click the client class and drag the mouse to the interface node.

Related Procedures

Changing the Appearance of Interfaces

Related Reference

UML 2.0 Class Diagrams
UML 2.0 Component Diagrams
UML 2.0 Composite Structure Diagrams

Working with a Relationship

Refer to the Getting Started Procedures to learn how to draw a link. This section describes how to change the link type and properties.

To change the type of an association link

1. Select an Association Link on the diagram.
2. In the Properties View, select the **Link** tab and click the **associates type** field.
3. Choose the link type (association, aggregation, or composition) from the drop-down list.

To set the directed property of an association link

1. Select the association link that you want on the diagram. The properties for the link appear in the Properties View.
2. In the **Link** tab of the Properties View, select the **directed** field.
3. Click the drop-down arrow and select the value for this Boolean property.

Related Procedures

Getting Started Procedures
Creating a Simple Link
Changing Type of an Association Link

Related Reference

UML 1.4 Class Diagrams
UML 2.0 Class Diagrams

Working with Association classes and n-ary associations

Association classes appear in diagrams as three related elements:

- Association class itself (represented by a class icon)
- N-ary association class link (represented by a diamond)
- Association connector (represented by a link between both)

To create an association class

1. On the diagram Palette, select the **Association Class** button.
2. Click the diagram background. This adds a regular class icon for the association class, connected with the diamond icon that represents the Association Class Link Aspect.
3. Create participant classes.
4. Using the **Association End** button, connect the diamond icon with the participant classes.

The source code of an association class now contains appropriate tags for the association class itself, and for each of the association end classes.

To delete an association class

1. Right-click an association class or its diamond icon.
2. Choose **Delete** on the context menu.

The whole association class construct is now deleted from the diagram.

Related Reference

Class Diagram Relationships
UML 2.0 Class Diagrams
UML 1.4 Class Diagrams

Working with Inner Classes

Both inner classes and inner interfaces in diagrams display within their own compartment field within the class. To create an inner class, do one of the following:

When the class already exists

1. Drag it over the target class.
2. Drop it.

Using the context menu

1. Right-click the parent class.
2. Select **New** ▸ **Inner Class** from the context menu.

Using Cut, Copy, and Paste

1. Use the clipboard operations to either cut or copy an existing inner class.
2. Select the parent class.
3. Use the clipboard operations to paste the selected class into the parent class.

Tip: Classes do not keep the same visibility after they are removed from the parent class.

Working with Instance Specifications

According to the UML 2.0 specification, an instance specification can instantiate one or more classifiers. You can instantiate a classifier using the **instantiates** property in the Properties View or the in-place editor.

To instantiate a classifier using the Properties View

1. Select an instance specification in your diagram.
2. In the **Properties** node of the Properties View, select the **instantiates** field.
3. Click the chooser button.
4. In the **Choose Classifier for 'instantiates' property**, select the classifiers from the available contents, using the Add/Remove buttons.
5. Click OK to save your changes.

To instantiate a classifier using the in-place editor

1. Select an instance specification in your diagram.
2. Press F2 to open the in-place editor. Alternatively, click twice on the instance specification name.
3. Type the name of an existing classifier, delimited by a colon, next to the instance specification name. For example, `InstanceSpecifcation1:Class1`.
4. Press Enter.

To define the features of an instance specification, you can insert slots into an instance specification element, associate the slots with the attributes of the instantiated classifiers, set the value, and define the slot stereotype.

To add a slot to an instance specification element

1. Add an instance specification element to your diagram.
2. Right-click the instance specification element and choose **New** **Slot** on the context menu.

To associate a slot with a structural feature

1. Select a slot in an instance specification element.
2. Click the **Properties** tab of the Properties View.
3. In the defining **feature field**, select the attribute you want from the list of attributes owned by the classifiers, which is instantiated by the instance specification (or their parents).

To set the slot value, do one of the following:

1. In the Properties View of the slot, select the **value** field, click the Editor button, and type the string in the **Edit property values** editor, OR
2. Invoke the in-place editor for the slot and type the value next to the slot name, delimited by an equal sign.

Related Procedures

UML 2.0 Component Diagrams Procedures
UML 2.0 Composite Structure Diagrams Procedures
UML 2.0 Class Diagrams Procedures

UML 2.0 Use Case Diagrams Procedures

This section outlines the procedures related to UML 2.0 Use Case diagrams.

In This Section

Creating an Extension Point

To create an extension point

1. Right-click the use case element.
2. Choose **Add ▸ Extension Point** on the context menu.
3. Type in a name.

Related Reference

UML 1.4 Use Case Diagrams
UML 2.0 Use Case Diagrams

Defining Includes and Extends Links

Includes and Extends links are created between Use Cases. The created links are marked with their stereotype.

To define condition properties of an Extends link

1. Select Extends link in the diagram.
2. In the Properties View, expand the **condition** node.
3. In the **language** field, choose the condition type (OCL or plain text).
4. In the **body** field, specify the condition text.

Tip: Alternatively, you can double-click on the condition element in the diagram and enter the condition text in the editor window.

Related Procedures

Setting Subject for a Use Case
Together Object Constraint Language (OCL)

Setting Subject for a Use Case

To set a subject for a Use Case

1. Select a Use Case in the diagram.
2. In the Properties View, expand the **Common properties** node and select the subject field.
3. Click on the chooser button.
4. In the **Select Classifier for Subject Property** dialog, select the desired classifiers from the metamodel. Use the **Add** and **Remove** buttons to create the list of selected elements.
5. Click **OK**.

UML 2.0 Interaction Diagrams Procedures

This section outlines the procedures related to UML 2.0 Sequence and Communication diagrams.

In This Section

A Typical Scenario of Designing a UML 2.0 Interaction Diagram

Use the following tips and techniques when you design a UML 2.0 Sequence or Communication Diagram. Usually you create interaction diagrams after class diagrams.

Whenever an interaction diagram is created, the corresponding interaction is added to the project. Interactions are represented as nodes in the Model Navigator.

You can view an interaction in two ways: as a Sequence Diagram or as a Communication Diagram. Any actions performed with either view are automatically reflected in the other views. Adding or deleting an element in an interaction results in the modification of the corresponding interaction diagram, and vice versa. An interaction diagram contains a reference to the underlying interaction.

Note: Unlike UML 1.4, it is not possible to switch a diagram that already exists from sequence to communication and vice versa. However, it is possible to create a sequence diagram and a communication diagram based on the same interaction.

To design a UML 2.0 Sequence Diagram, perform the following general actions:

1. Create an interaction with one or more lifelines, and open it in a sequence diagram. Associate the interaction with a class and operation.

   Working with Interactions
2. Create an interaction use.

   Creating an Interaction Use
3. Associate a lifeline with a classifier.

   Associating a Lifeline with a Classifier
4. Define the decomposition of a lifeline.

   Defining Decomposition of a Lifeline
5. Repeat the steps to create all required lifelines.
6. Link the created lifelines by using messages.

   Working with a UML 2.0 Message
7. Add combined fragments to the lifeline.

   Working with a Combined Fragment
8. Add state invariants.

   Creating a State Invariant

Related Reference

UML 2.0 Interaction Diagrams

Working with Interactions

You can start designing your sequence or communication diagram with creating an interaction. An interaction can be opened in a sequence or communication diagram.

In this section you will learn how to:

- Create an interaction
- Open an interaction in a sequence or communication diagram
- Define context and specification for an interaction

To create an interaction

1. In the Model Navigator, right-click a project or a package node.
2. On the context menu, choose **New** ▸ **Interaction diagram elements** ▸ **Interaction**.

To open an interaction in diagram

1. Select an interaction in the Model Navigator.
2. On the context menu, choose **Open full screen communication diagram** or **Open full screen sequence diagram**.

To define context and specification for an interaction

1. Select an interaction in the Model Navigator.

   Tip: Alternatively, click on the interaction diagram background.

2. In the Properties View, select the **Properties** tab.
3. In the **context** field, click the chooser button and in the **Choose referenced classifier** dialog, select a context.
4. In the **specification** field, click the chooser button and in the **Choose operation** dialog, select the operation for the specified context.

Related Reference

UML 2.0 Interaction Diagrams

Creating an Interaction Use

To create an interaction use

1. In the diagram Palette, choose the **Interaction Use** button.
2. Click on the target lifeline.
3. In the Properties View for the newly created interaction use, choose the **Properties** tab.
4. In the **interaction name** field, click the chooser button.

   Tip: Alternatively, just type the interaction name.

5. In the **Choose Referenced Interaction** dialog box, select the interaction and click **OK**.

An interaction use is initially created attached to a lifeline. You can further expand it over several lifelines, as well as detach it from and reattach it to lifelines.

Related Reference

UML 2.0 Interaction Diagrams

Associating a Lifeline with a Classifier

In this section you will learn how to:

- Associate a lifeline with an existing classifier using the lifeline context menu
- Associate a lifeline with a new classifier using the context menu
- Associate a lifeline with a classifier using the Properties View

To associate a lifeline with a classifier using the lifeline context menu

1. Select a lifeline on an Interaction diagram.
2. Right-click the lifeline and select **Choose Type** ▸ **<connectable element's type>** on the context menu.
3. If the desired type is not in the list, choose **More**. The **Choose represented connectable element's type** dialog box opens.

   Select a classifier to be associated with the lifeline from the tree of available model elements, and click **OK**.

To associate a lifeline with a new type using the lifeline context menu

1. Select a lifeline on an Interaction diagram.
2. Right-click the lifeline and select **New** ▸ **Type** on the context menu.
3. Select **Class** or **Interface** on the submenu. The new connectable element adds to the model.

To associate a lifeline with a classifier using the Properties View

1. Select a lifeline on an Interaction diagram.
2. In the Properties View of the lifeline, select **type** field.
3. Enter the classifier name in the text area, or click the file chooser button. The **Choose represented connectable element's type** dialog box opens.

   Select a classifier to be associated with the lifeline from the tree of available model elements, and click **OK**.

Related Procedures

Instantiating a Classifier

Related Reference

UML 2.0 Interaction Diagrams

Defining Decomposition of a Lifeline

To define decomposition for a lifeline

1. Select the desired lifeline in the Model Navigator or the Diagram Editor .
2. In the Properties View, select the **decomposition** field.
3. Click the chooser button.
4. In the **Choose Referenced Interaction** dialog box, select the desired interaction.
5. Click **OK**.

Tip: Decomposition, type, stereotype, and referenced element properties are also reflected in the corresponding Communication diagram.

Related Reference

UML 2.0 Interaction Diagrams

Working with a UML 2.0 Message

This section describes techniques for working with messages in sequence and communication diagrams. Although the two diagram types are equivalent, the techniques for dealing with messages differ.

In this section you will learn how to:

- Show or hide reply messages
- Create nested messages (Sequence diagram)
- Create a message from a lifeline back to itself
- Create a message link that corresponds to an operation call
- Create an asynchronous call, which enables you to extend or reduce the time of invocation specification and execution specification independently
- Create a found execution on a lifeline (that is, a message that comes from an object that is not shown on the diagram [Sequence diagram])

To show or hide reply messages

1. Select a call message on a diagram.
2. In the Properties View, set the **show reply message** value to `true` to show reply messages or to `false` to hide reply messages.

To create a nested message

1. Choose the **Message** icon on the diagram Palette.
2. Click an execution specification to originate the message and drag the link to the target lifeline.

Note: The nested message inherits the numbering of the parent message. For example, if the parent message has the number 1, its first nested message is 1.1.

To create a message from a lifeline back to itself

1. Choose the **Message** icon on the diagram Palette.
2. Double-click the target lifeline.

To create a message link that corresponds to an operation call

1. Create a message link between two lifelines in an interaction.
2. Make sure that the target lifeline has its type defined and the associated classifier contains at least one operation.
3. In the Properties tab of the Properties View, select the **signature** field and click the chooser button.
4. In the **Choose Operation** dialog box, select an operation.
5. Click **OK**.

The message link is named according to the name of the operation.

To de-synchronize invocation specification and execution specification

1. Select an invocation specification on a lifeline.
2. Click the **sort** property in the Properties View and select `asynchCall` in the list.

To create a found execution

1. In the Palette, click the **Found execution** button.
2. Click on a place of a lifeline. An execution specification bar is created in the target lifeline.

Related Procedures

Working with Instance Specifications

Related Reference

UML 2.0 Interaction Diagrams
UML 2.0 Message
Execution Specification and Invocation Specification

Working with a Combined Fragment

In this section you will learn how to:

- Create a combined fragment
- Create nested combined fragments
- Create nested operators
- Sever nested operators
- Create operands
- Expand combined fragments across several lifelines
- Detach a combined fragment from a lifeline

To create a combined fragment

1. Choose the **Combined Fragment** button in the diagram Palette, and click on the target lifeline.
2. In the **New Combined Fragment** dialog box that opens, choose an operator from the list of available operators and set the combined fragment options (operator name, arguments, or number of operands).
3. Click **OK**.

The combined fragment is added to the target lifeline or execution specification. Each new combined fragment has a different color to distinguish it from the other combined fragments within the same cluster of nested frames.

To create a nested combined fragment

1. Choose the **Combined Fragment** button in the diagram Palette.
2. Click on the target combined fragment that already exists in a lifeline.

Note: Each new node has a different color that is selected at random. You can work with the inner frames in the same way as with the outer frames: move along a lifeline, spread them over several lifelines, detach and tie frames. Note that drawing a message link from a frame automatically expands it, together with its outer frames, if any.

To create nested operators

1. Select a combined fragment.
2. In the **other operators** field of the Properties View, click the chooser button. The **Interaction Operators** dialog box opens, displaying the list of already defined operators in the current combined fragment.
3. Click the **Add** button. A new line is displayed below the existing entry in the list of operators.
4. If a certain operator enables arguments, enter them in the adjacent field in the Arguments column. Use a comma as a delimiter.
5. Use the **Add** and **Remove** buttons to compile your list of the nested operators. Use the **Up** and **Down** buttons to specify the proper order of nested operators.
6. Click **OK** to apply changes.

The nested operators are now listed in the descriptor of the combined fragment in the specified order.

To sever operators

1. Right-click a combined fragment that contains nested operators.
2. On the context menu, choose **Sever operators between**.
3. On the submenu, select the pair of operators between which the combined fragment will be divided.

A nested combined fragment is now created.

To combine with an outer fragment

1. Right-click an inner fragment.
2. On the context menu, choose **Combine with an outer fragment**.

To create an operand

1. Select a combined fragment or an operand in the Model Navigator or in the Diagram Editor .
2. On the context menu of the selection, choose **New ▸ Interaction Operand**.
3. In the **Interaction constraint** tab of the Properties View, select the language to be used for describing the constraint. To do this, click the Language drop-down list and choose OCL or plain text.
4. Type the constraint expression.
5. Add as many operands as required.
6. Apply changes.

A new operand is now created. If the operand was created from the context menu of a combined fragment, it will be added to the end of the combined fragment. If the operand was created from the context menu of an operand, it will be added just before this operand. Constraint text is displayed in the operand section of the combined fragment.

To expand a combined fragment across several lifelines

1. Select the combined fragment.

   Tip: You can expand both outer and inner combined fragments.

2. Click the anchor icon and drag it to the target lifeline.

The fragment now spans across lifelines, with the mounting links on each lifeline.

To detach a combined fragment from a lifeline

1. Select the mounting link of a combined fragment.
2. Choose **Delete** on the context menu.

Tip: You cannot delete the only mounting link of a combined fragment. A combined fragment must be attached to at least one lifeline.

Related Reference

Operator and Operand for a Combined Fragment

Creating a State Invariant

A state invariant is a constraint placed on a lifeline. This constraint is evaluated at runtime prior to execution of the next execution specification. State invariants are represented in the interaction diagrams in two ways: as OCL expressions or as references to the state diagrams. You can use the state invariants to provide comments to your interaction diagrams and to connect interactions with states.

Together provides validation of the state invariants represented as OCL expressions. An OCL editor with highlighting and validation is provided for typing your OCL expression.

The typed OCL expression is correct only if the context is specified for it. The context of the State Invariant connected to the LifeLine is the type of the part this LifeLine represents. If no type is set, the OCL expression is reported as invalid with OCL as the language of the expression. If no correct context can be specified, select text as the expression language.

To create a state invariant as an OCL expression:

1. On the diagram Tools Palette, click the state invariant button.
2. Click the target lifeline or execution specification.
3. In the **Properties** view of the state invariant, expand the **Common properties** node.
4. In the invariant kind field, choose OCL expression from the list. The shape of the state invariant diagram element changes to braces.
5. In the OCL invariant view that opens, select the language of the comment from the Language list. The possible options are OCL and plain text.
6. Type your expression text and apply changes.

To connect a state invariant to a state:

1. On the diagram Tools Palette, click the state invariant button.
2. Click the target lifeline or execution specification.
3. In the **Properties** view of the state invariant, expand the **Common properties** node.
4. In the invariant kind field, choose States/Regions from the list.
5. In the States/Regions field, click the Edit button.
6. In the **Choose States and/or Regions** dialog box, select the states and/or regions from the model. Use the **Add** button to add them to the **Selected** list.
7. Click OK to save your changes.
8. Alternatively, you can type the state or region name using the in-place editor. If the state or region belongs to a different package, specify its fully qualified name.

Related Concepts

About OCL Support in Together

Related Reference

UML 2.0 Interaction Diagrams

Associating a Lifeline with a Classifier

In this section you will learn how to:

- Associate a lifeline with an existing classifier using the lifeline context menu
- Associate a lifeline with a new classifier using the context menu
- Associate a lifeline with a classifier using the Properties View

To associate a lifeline with a classifier using the lifeline context menu

1. Select a lifeline on an Interaction diagram.
2. Right-click the lifeline and select **Choose Type** ▸ **<connectable element's type>** on the context menu.
3. If the desired type is not in the list, choose **More**. The **Choose represented connectable element's type** dialog box opens.

   Select a classifier to be associated with the lifeline from the tree of available model elements, and click **OK**.

To associate a lifeline with a new type using the lifeline context menu

1. Select a lifeline on an Interaction diagram.
2. Right-click the lifeline and select **New** ▸ **Type** on the context menu.
3. Select **Class** or **Interface** on the submenu. The new connectable element adds to the model.

To associate a lifeline with a classifier using the Properties View

1. Select a lifeline on an Interaction diagram.
2. In the Properties View of the lifeline, select **type** field.
3. Enter the classifier name in the text area, or click the file chooser button. The **Choose represented connectable element's type** dialog box opens.

   Select a classifier to be associated with the lifeline from the tree of available model elements, and click **OK**.

Related Procedures

Instantiating a Classifier

Related Reference

UML 2.0 Interaction Diagrams

Associating a Lifeline with a Referenced Element

To associate a lifeline with a referenced element

1. Make sure that your Interaction context or Interaction specification contains the referenced elements that should be represented by the lifelines.
2. Select the desired lifeline in the Model Navigator or the Diagram Editor .
3. In the Properties View, select the **represents** field.
4. Click the chooser button.
5. In the **Choose Represented Connectable Element** dialog box, select the desired part from the project or Favorites.
6. Click **OK**.

To navigate to a referenced interaction

1. Right-click on an interaction use that refers to another interaction.
2. On the context menu, choose **Select**.
3. Choose the desired destination on the submenu.

Related Reference

UML 2.0 Interaction Diagrams

Copying and Pasting an Execution or Invocation Specification

Clipboard operations are supported for the execution and invocation specifications.

To copy and paste an execution or invocation specification

1. **Cut**, **Copy**, and **Paste** commands are available on the context menu of an execution specification and invocation specification. It is possible to copy or move these elements within the same diagram or to another diagram.
2. When an execution or invocation specification is copied, it means that the entire branch of messages is copied also. Pasting the clipboard contents to a target lifeline results in changing the message numbers according to the numbering of messages in the target lifeline.
3. If you paste an invocation or execution specification to another diagram, the entire outgoing bunch of messages will be pasted also, with all the respective lifelines. If the target diagram does not contain lifelines for this execution specification, they will be created automatically.

Tip: It is also possible to move and copy message branches using the drag-and-drop technique. To move an execution or invocation specification, drag-and-drop it to the target location. To create a copy, drag-and-drop while holding the CTRL key down.

Related Procedures

Working with a UML 2.0 Message

Related Reference

UML 2.0 Interaction Diagrams

Creating a Full-Screen Sequence or Communication Diagram from an Interaction

To create a full-screen sequence or a communication diagram from an interaction

1. In the Diagram Editor or in the Model Navigator, choose an Interaction element.
2. Right-click the Interaction node and choose **Open Full-Screen Sequence diagram** or **Open Full-Screen Communication diagram**.

If such a diagram does not exist, it will be created and will open in the Diagram Editor .

Related Reference

UML 2.0 Interaction Diagrams

Creating a State Invariant

A state invariant is a constraint placed on a lifeline. This constraint is evaluated at runtime prior to execution of the next execution specification. State invariants are represented in the interaction diagrams in two ways: as OCL expressions or as references to the state diagrams. You can use the state invariants to provide comments to your interaction diagrams and to connect interactions with states.

Together provides validation of the state invariants represented as OCL expressions. An OCL editor with highlighting and validation is provided for typing your OCL expression.

The typed OCL expression is correct only if the context is specified for it. The context of the State Invariant connected to the LifeLine is the type of the part this LifeLine represents. If no type is set, the OCL expression is reported as invalid with OCL as the language of the expression. If no correct context can be specified, select text as the expression language.

To create a state invariant as an OCL expression:

1. On the diagram Tools Palette, click the state invariant button.
2. Click the target lifeline or execution specification.
3. In the **Properties** view of the state invariant, expand the **Common properties** node.
4. In the invariant kind field, choose OCL expression from the list. The shape of the state invariant diagram element changes to braces.
5. In the OCL invariant view that opens, select the language of the comment from the Language list. The possible options are OCL and plain text.
6. Type your expression text and apply changes.

To connect a state invariant to a state:

1. On the diagram Tools Palette, click the state invariant button.
2. Click the target lifeline or execution specification.
3. In the **Properties** view of the state invariant, expand the **Common properties** node.
4. In the invariant kind field, choose States/Regions from the list.
5. In the States/Regions field, click the Edit button.
6. In the **Choose States and/or Regions** dialog box, select the states and/or regions from the model. Use the **Add** button to add them to the **Selected** list.
7. Click OK to save your changes.
8. Alternatively, you can type the state or region name using the in-place editor. If the state or region belongs to a different package, specify its fully qualified name.

Related Concepts

About OCL Support in Together

Related Reference

UML 2.0 Interaction Diagrams

Creating an Interaction Use

To create an interaction use

1. In the diagram Palette, choose the **Interaction Use** button.
2. Click on the target lifeline.
3. In the Properties View for the newly created interaction use, choose the **Properties** tab.
4. In the **interaction name** field, click the chooser button.

   Tip: Alternatively, just type the interaction name.

5. In the **Choose Referenced Interaction** dialog box, select the interaction and click **OK**.

An interaction use is initially created attached to a lifeline. You can further expand it over several lifelines, as well as detach it from and reattach it to lifelines.

Related Reference

UML 2.0 Interaction Diagrams

Defining Decomposition of a Lifeline

To define decomposition for a lifeline

1. Select the desired lifeline in the Model Navigator or the Diagram Editor .
2. In the Properties View, select the **decomposition** field.
3. Click the chooser button.
4. In the **Choose Referenced Interaction** dialog box, select the desired interaction.
5. Click **OK**.

Tip: Decomposition, type, stereotype, and referenced element properties are also reflected in the corresponding Communication diagram.

Related Reference

UML 2.0 Interaction Diagrams

Roundtrip Engineering with UML 2.0 Sequence Diagrams

This section demonstrates a sample procedure if creating and editing a UML 2.0 sequence diagram that generates source code. To generate source code from a sequence diagram, you will:

- Create an implementation project and a class with the `main()` method.
- Generate a sequence diagram from the main method of the class.
- Create source-generating elements on the sequence diagram.
- Create messages. All the messages will have the same source and destination.

To create a project and class

1. Create a UML 2.0 source code project, and add a new class diagram.
2. Create a new class on the diagram.
3. Right-click on the class, and choose **New ▸ Operation** on the context menu. The in-place editor activates.
4. Create an operation. For example, add a main class method by entering the following code in the in-place editor: `main(args: String[]):void`
5. Press ENTER. The new `main()` method is created.

To generate a sequence diagram from the main method

1. Right-click on the main method, and choose **Generate Sequence Diagram** from the context menu. The **Generate Sequence Diagram** wizard displays.
2. Click **Next**, accepting the default settings for the first page of the wizard.
3. Click **Finish**, accepting the default settings for the second page of the wizard. The sequence diagram opens in a new diagram tab of the Diagram Editor . The lifeline gets the name `self`.

Tip: By default, the generated diagram gets the name `[Class_name].[method_name]`;

To create source-generating elements on the sequence diagram

1. Create a `for` statement in the `self` lifeline:

   — On the Tools Palette of the diagram, choose the **Combined Fragment** button and click on the execution specification of the message #1.

   — In the New Combined Fragment dialog, select `for` and click **OK**.

   — Select an operand in the combined fragment. In the Properties View of the operand, click the **Interaction Constraint** tab.

   — In the **language** field, select plain text. In the **body** field, enter the following code: `int i = 0; i < 4; i ++`

   The label displays in the operand of the combined fragment as: `[int i = 0; i < 4; i++]`
2. Add a new lifeline to the sequence diagram:

   — Click the **Lifeline** button on the Palette, and click on the diagram to create a new object with the name `Frame`.

3 Associate the new lifeline with a type:

— Right-click on the Frame lifeline, and choose **Choose type ▸ More** from the context menu. The **Choose represented connectable element's type** dialog box displays.

— Expand the **libraries** node and navigate to **javax > swing**. Select **JFrame** from the list, and click **OK**. The name of the selected class displays on the frame object.

To create messages on a sequence diagram

1 Draw a message link from the Execution Specification to the lifeline of the Frame lifeline. As you click the target lifeline, the **Choose operation** dialog opens.

2 Select JFrame constructor (`JFrame:void)` from the list. The message label becomes: `1.1: JFrame()`.

In the Properties View of the message, set **creation** property to `true`. The message changes visually to the creation type. Note that the message link now points to the frame object, which means that the object is being created.

3 Draw a new message. Its label is Message Link 1.2. In the **Choose operation** dialog, expand the JFrame node, scroll through the list, and select `setDefaultCloseOperation(int):void.` Click **OK**.

4 In the **arguments** field of the Properties View of the message 1.2, enter `JFrame.EXIT_ON_CLOSE`. The message label becomes: `1.2: setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)`

5 Draw Message Link 1.3. In the **Choose operation** dialog expand the Component node, scroll through the list, and select with the operation `setSize(int,int):void`. In the **arguments** field of the Properties View enter `600,400`. The message label becomes: `1.3: setSize(600,400)`

6 Draw Message Link 1.4. In the **Choose operation** dialog expand the Component node, scroll through the list, and select with the operation `setLocation(int,int):void` . In the **arguments** field of the Properties View, enter `(50*i,50*i)`. The message label becomes: `1.4: setLocation(50*i,50*i)`

7 Draw Message Link 1.5. In the **Choose operation** dialog, expand the Component node, scroll through the list, and select with the operation `show()`. The message becomes: `1.5:show():void`

To generate implementation code for a sequence diagram

1 Right-click on the background of the sequence diagram, and choose **Generate Implementation** from the context menu. The first page of the Sequence diagram refactoring wizard displays. Warning messages display, if applicable. Click **Next** to display the second page of the wizard.

2 The second page of the wizard displays changes that are necessary to perform refactoring. Notice the sections on the page highlighting the original source code to be replaced and the refactored source code to replace it. Click **Finish**.

Open implementation code of the Class1 in the Editor. Message labels, for which implementation code has been generated, display in bold on the diagram. If code generation fails for certain messages, those messages do not display in bold.

3 On the sequence diagram, double-click one of the message links displayed in bold, and observe that the Editor scrolls to the point of appropriate method invocation.

4 In the Editor, add an import statement for `javax.swing`.

Related Reference

UML 2.0 Interaction Diagrams

Working with a Combined Fragment

In this section you will learn how to:

- Create a combined fragment
- Create nested combined fragments
- Create nested operators
- Sever nested operators
- Create operands
- Expand combined fragments across several lifelines
- Detach a combined fragment from a lifeline

To create a combined fragment

1. Choose the **Combined Fragment** button in the diagram Palette, and click on the target lifeline.
2. In the **New Combined Fragment** dialog box that opens, choose an operator from the list of available operators and set the combined fragment options (operator name, arguments, or number of operands).
3. Click **OK**.

The combined fragment is added to the target lifeline or execution specification. Each new combined fragment has a different color to distinguish it from the other combined fragments within the same cluster of nested frames.

To create a nested combined fragment

1. Choose the **Combined Fragment** button in the diagram Palette.
2. Click on the target combined fragment that already exists in a lifeline.

Note: Each new node has a different color that is selected at random. You can work with the inner frames in the same way as with the outer frames: move along a lifeline, spread them over several lifelines, detach and tie frames. Note that drawing a message link from a frame automatically expands it, together with its outer frames, if any.

To create nested operators

1. Select a combined fragment.
2. In the **other operators** field of the Properties View, click the chooser button. The **Interaction Operators** dialog box opens, displaying the list of already defined operators in the current combined fragment.
3. Click the **Add** button. A new line is displayed below the existing entry in the list of operators.
4. If a certain operator enables arguments, enter them in the adjacent field in the Arguments column. Use a comma as a delimiter.
5. Use the **Add** and **Remove** buttons to compile your list of the nested operators. Use the **Up** and **Down** buttons to specify the proper order of nested operators.
6. Click **OK** to apply changes.

The nested operators are now listed in the descriptor of the combined fragment in the specified order.

To sever operators

1. Right-click a combined fragment that contains nested operators.
2. On the context menu, choose **Sever operators between**.
3. On the submenu, select the pair of operators between which the combined fragment will be divided.

A nested combined fragment is now created.

To combine with an outer fragment

1. Right-click an inner fragment.
2. On the context menu, choose **Combine with an outer fragment**.

To create an operand

1. Select a combined fragment or an operand in the Model Navigator or in the Diagram Editor .
2. On the context menu of the selection, choose **New** ▸ **Interaction Operand**.
3. In the **Interaction constraint** tab of the Properties View, select the language to be used for describing the constraint. To do this, click the Language drop-down list and choose OCL or plain text.
4. Type the constraint expression.
5. Add as many operands as required.
6. Apply changes.

A new operand is now created. If the operand was created from the context menu of a combined fragment, it will be added to the end of the combined fragment. If the operand was created from the context menu of an operand, it will be added just before this operand. Constraint text is displayed in the operand section of the combined fragment.

To expand a combined fragment across several lifelines

1. Select the combined fragment.

   Tip: You can expand both outer and inner combined fragments.

2. Click the anchor icon and drag it to the target lifeline.

The fragment now spans across lifelines, with the mounting links on each lifeline.

To detach a combined fragment from a lifeline

1. Select the mounting link of a combined fragment.
2. Choose **Delete** on the context menu.

Tip: You cannot delete the only mounting link of a combined fragment. A combined fragment must be attached to at least one lifeline.

Related Reference

Operator and Operand for a Combined Fragment

Working with a UML 2.0 Message

This section describes techniques for working with messages in sequence and communication diagrams. Although the two diagram types are equivalent, the techniques for dealing with messages differ.

In this section you will learn how to:

- Show or hide reply messages
- Create nested messages (Sequence diagram)
- Create a message from a lifeline back to itself
- Create a message link that corresponds to an operation call
- Create an asynchronous call, which enables you to extend or reduce the time of invocation specification and execution specification independently
- Create a found execution on a lifeline (that is, a message that comes from an object that is not shown on the diagram [Sequence diagram])

To show or hide reply messages

1. Select a call message on a diagram.
2. In the Properties View, set the **show reply message** value to `true` to show reply messages or to `false` to hide reply messages.

To create a nested message

1. Choose the **Message** icon on the diagram Palette.
2. Click an execution specification to originate the message and drag the link to the target lifeline.

Note: The nested message inherits the numbering of the parent message. For example, if the parent message has the number 1, its first nested message is 1.1.

To create a message from a lifeline back to itself

1. Choose the **Message** icon on the diagram Palette.
2. Double-click the target lifeline.

To create a message link that corresponds to an operation call

1. Create a message link between two lifelines in an interaction.
2. Make sure that the target lifeline has its type defined and the associated classifier contains at least one operation.
3. In the Properties tab of the Properties View, select the **signature** field and click the chooser button.
4. In the **Choose Operation** dialog box, select an operation.
5. Click **OK**.

The message link is named according to the name of the operation.

To de-synchronize invocation specification and execution specification

1 Select an invocation specification on a lifeline.
2 Click the **sort** property in the Properties View and select `asynchCall` in the list.

To create a found execution

1 In the Palette, click the **Found execution** button.
2 Click on a place of a lifeline. An execution specification bar is created in the target lifeline.

Related Procedures

Working with Instance Specifications

Related Reference

UML 2.0 Interaction Diagrams
UML 2.0 Message
Execution Specification and Invocation Specification

Working with Interactions

You can start designing your sequence or communication diagram with creating an interaction. An interaction can be opened in a sequence or communication diagram.

In this section you will learn how to:

- Create an interaction
- Open an interaction in a sequence or communication diagram
- Define context and specification for an interaction

To create an interaction

1. In the Model Navigator, right-click a project or a package node.
2. On the context menu, choose **New** ▸ **Interaction diagram elements** ▸ **Interaction**.

To open an interaction in diagram

1. Select an interaction in the Model Navigator.
2. On the context menu, choose **Open full screen communication diagram** or **Open full screen sequence diagram**.

To define context and specification for an interaction

1. Select an interaction in the Model Navigator.

   Tip: Alternatively, click on the interaction diagram background.

2. In the Properties View, select the **Properties** tab.
3. In the **context** field, click the chooser button and in the **Choose referenced classifier** dialog, select a context.
4. In the **specification** field, click the chooser button and in the **Choose operation** dialog, select the operation for the specified context.

Related Reference

UML 2.0 Interaction Diagrams

UML 2.0 State Machine Diagrams Procedures

In This Section

Associating a Transition or a State with a Behavior

You can associate an activity (created on a UML 2.0 Activity Diagram) with a state (upon entering the state, while doing the state activity, and upon exiting the state), or with a transition between states.

To associate a transition with an activity

1. Select a transition or a state on a UML 2.0 State Machine diagram.
2. In the Common properties tab of the **Properties** view, click the **Effect** (for a transition) or **Do Behavior**, **Entry Behavior** or **Exit Behavior** (for a state) field.
3. Click the chooser button to open the **Select Behavior for property** dialog box.
4. In the model tree view, locate your chosen activity and click **Add**.
5. Click **OK** to save the changes.

Related Procedures

Creating an OCL Guard Condition for a Transition

Related Reference

UML 2.0 State Machine Diagrams
State

Changing Regions Order in a State

If you have several regions in your State or State Machine, you can rotate them to place them either in horizontal or vertical order.

To change regions order

1. Right-click a State or a State Machine on your diagram
2. Select either **Order Regions Vertically** or **Order Regions Horizontally**.

Related Concepts

UML 2.0 State Machine Diagram Definition

Related Reference

UML 2.0 State Machine Diagrams

Creating an OCL Guard Condition for a Transition

An OCL expression can restrict a StateMachine transition by acting as a guard to that transition. In an OCL guard condition, the StateMachine must have a context that is a Classifier. The expression, which is evaluated when the guard's transition is attempted, is of type Boolean.

To create a guard condition for a transition

1. Select a transition on a diagram.
2. Select the **guard** tab in the **Properties** view.
3. Specify the language for your guard condition (OCL by default).
4. Select the **body** field and click the **Edit** button.
5. Type the condition expression and click OK to apply changes.

Related Concepts

About OCL Support in Together

Related Reference

UML 2.0 State Machine Diagrams

Creating and Editing States

Note: Because all State Diagram elements are created inside State Machines, create at least one State Machine before attempting to create other elements.

To create a state

1. On the Palette, click the **State** button.
2. Click the diagram background.

   Alternatively, right-click a region of the StateMachine element and select **New ▸ State** from the context menu.

   When a new state is placed on a diagram, you can use the Properties View to edit its properties.

   - Configure standard properties of the element.
   - In the **State Invariant** tab, select the language of the expression from the Language list box. The possible options are OCL and plain text.
   - In the **Properties** page, configure the behavior of the state by setting these additional properties:

Field	Description
Composite	Set to True if there is one or more regions in this state (not editable)
Orthogonal	Set to True if there are two or more regions in this state (not editable)
Simple	Set to True if there are no regions in this state (not editable)
Do Behavior	Specify the activity to be performed during execution of the current state by using the Properties View. This activity may be selected from any Activity diagram of the project.
Entry Behavior	Specify the activity to be performed when the current state starts executing by using the Properties View. This activity may be selected from any Activity diagram of the project.
Exit Behavior	Specify the activity to be performed when the current state finishes executing by using the Properties View. This activity may be selected from any Activity diagram of the project.

Note: You may place a state inside of the existing state. It is possible to hide individual states. For example, you can hide the content of composite states for better understanding of the whole diagram.

To make a state a submachine state

1. Select a state you want to make a submachine state.
2. In the Properties View of the state, click the **submachine** property and choose any model state machine.

For your convenience a StateMachine element can be opened in a separate Diagram Editor view.

To open a StateMachine element in a separate Diagram Editor view

1. Right-click a **StateMachine** element in the Diagram Editor .
2. Select **Open Full Screen in New Diagram**.

To define the do activity, entry or exit activities of a state

1. Select a state and click the appropriate field in the Properties view.
2. Click the **Edit** button. This opens the **Select Activity20 for property** dialog box.
3. Locate your chosen activity and use the **Add** and **Remove** buttons to add and remove activities.
4. Click **OK** to save your changes.

Related Concepts

About OCL Support in Together
UML 2.0 State Machine Diagram Definition

Related Procedures

Changing Regions Order in a State

Related Reference

UML 2.0 State Machine Diagrams

Creating History Elements

The Shallow History and Deep History elements are placed on regions of the states. Refer to the UML 2.0 Specification for more information about these elements.

You can create none or one Deep History, and none or one Shallow History elements in each region.

To add a history to a state, do one of the following:

1. Right-click a region in a state, point to **New**, and select one of the **History** elements on the shortcut menu, OR
2. Click one of the **History** elements on the Palette and then click the target state region.

Related Concepts

UML 2.0 State Machine Diagram Definition

Related Procedures

Changing Regions Order in a State

Related Reference

UML 2.0 State Machine Diagrams

Creating Members for State Machines, States, and Regions

To create a member for a state

1. Open the **Diagram View**.
2. Right-click an existing state and choose **New ▸ (member)** on the context menu.

   The following members are available:

   - Internal transition (also available on the context menu) – A shorthand for handling events without leaving a state and dispatching its exit/entry activities.
   - Region (also available on the context menu) – Use regions inside the states to group the substates. The regions may have different visibility settings and history elements. Each state has one region immediately after creation, although it can be deleted.
   - Reference to Entry point and Reference to Exit point – Use references to entry/exit points as sources/ targets of transitions, respectively.

   In the Regions, you can create all the elements that are available for the States except Internal transition.

   In addition to regions, you can create the following members for State Machines:

   - Entry point – Execution of the state starts at this point. It is possible to create several entry points for one state, which makes sense if there are substates.
   - Exit point – Execution of the state finishes at this point. It is possible to create several exit points for one state, which makes sense if there are substates.

Related Reference

UML 2.0 State Machine Diagrams

Designing a UML 2.0 State Machine Diagram

Following are tips and techniques that you can use when working with a UML 2.0 State Machine Diagram.

To design a UML 2.0 State Machine Diagram

1. Create initial and final nodes.
2. Create main states and substates.
3. Create regions.
4. Create entry and exit points.
5. Create pins.
6. Create transitions.
7. Create history nodes.
8. You can optionally create shortcuts to related elements of other diagrams.

Related Procedures

Creating a Shortcut

Related Reference

UML 2.0 State Machine Diagrams

Working with a Complex State

The techniques in this section apply to models of particularly complex composite states. These procedures are common for the State and Activity diagrams.

Create a composite state by nesting one or more levels of states within one state and draw transitions among the nested elements. You can place the following elements in a state:

- activity
- signal sending
- signal receipt
- start/end states
- history

Tip: You can nest multiple levels of states inside one state. For especially complex state modeling, however, you may find it more convenient to create different diagrams, model each of the state levels individually, and then hyperlink the diagrams sequentially.

Use the following techniques to create a composite (nested) state

1. Create a nested state using drag-and-drop.
2. Create a nested state using the context menu of the state element.

To create a nested state using drag-and-drop

1. Place a state element on the diagram background.
2. Drag a new state on top of an existing state.
3. Drop a new state.

To create a nested state using the context menu of the state element

1. Right-click the state (region) that will be the container.
2. Select **New ▸ State** on the context menu.

Tip: Using the **Shortcuts** command on the context menu of the diagram, you can reuse existing elements from the other state diagrams. Right-click the diagram and choose **New▸Shortcuts**, navigate within the pane containing the tree view of the available project contents to the existing diagram, and select its elements, states, histories, forks, and/or joins.

Related Concepts

Model Hyperlinking Overview

Related Procedures

Creating a Shortcut

Related Reference

UML 1.4 Activity Diagrams
UML 1.4 Statechart Diagrams
UML 2.0 State Machine Diagrams

Working with Activities and State Machines Full Screen Diagrams

It is possible to work with the individual activities and state machines in specific diagrams. You can open an activity or a state machine in a full-screen view, and the element is expanded to the whole diagram. The new diagram has the same name as the selected element. This kind of activity or state machine is transparent, which makes it possible to view the grid. You cannot move the container activity or state machine, but the nested elements are still selectable and movable.

To create a full-screen diagram for an activity or a state machine

1. Select the desired element in the diagram or in the Model Navigator.
2. On the context menu of the selection, choose **Open Full Screen in New Diagram**.

UML 2.0 Activity Diagrams Procedures

In This Section

Creating Activity Parameters

To add an activity parameter to an activity

1. On the Palette, click the **Activity Parameter** button.
2. Click the target activity.

or

1. Right-click an activity
2. Select **New ▸ Activity Parameter** on the context menu.

An Activity Parameter node is added to the activity as a rectangle. Note that the activity parameter node is attached to its activity. You can move the node only along the activity borders.

Creating Pins

Actions may require some input and produce some output. The input and output are defined by the input and output pins.

To add an input pin, output pin, or value pin, do one of the following:

1. Right-click an action
2. Select **New**, and choose either **Input Pin** or **Output Pin** or **Value Pin** on the context menu.

OR

1. In the Tools Palette, choose one of the pins
2. Click the target action.

The created pin is added to the target action as a square. Note that the pins are attached to their actions and can be dragged only along the action borders.

Designing a UML 2.0 Activity Diagram

Use the following tips and techniques when you design a UML 2.0 Activity Diagram. Usually you create Activity Diagrams after State Machine Diagrams.

To design a UML 2.0 Activity Diagram, follow this general procedure:

1. Create one or more activities. You can place several activities on a single diagram, or create a separate diagram for each.

   Warning: You cannot create nested activities.

2. Usually activities are linked to states or transitions on State Machine Diagrams. Switch to your State Machine Diagrams and associate the activities you just created with states and transitions.

   Tip: After you do this, you may find that some more activities must be created, or the same activity can be used in several places.

3. Switch back to the Activity Diagram. Think about flows in your activities. You can have an object flow (for transferring data), a control flow, both or even several flows in each activity.
4. Create starting and finishing points for every flow. Each flow can have the following starting points:

   - Initial node
   - Activity parameter (for object flow)
   - Accept event action
   - Accept time event action

   Each flow finishes with a **Activity Final** or **Flow Final** node.

   If your activity has several starting points, they can be used simultaneously.
5. Create object nodes. You do not link object nodes to classes on your Class Diagrams. However, you can use hyperlinks for better understanding of your diagrams.
6. Create action nodes for your flows. Flows can share actions.

   Warning: You cannot create nested actions.

7. For object flows, add pins to actions. Connect actions and pins by flow links.
8. Add pre- and post- conditions. You can create plain text or OCL conditions.
9. You can optionally create shortcuts to related elements of other diagrams.

Related Procedures

Creating a Shortcut

Related Reference

UML 2.0 Activity Diagrams

Rotating Activity Partitions

By default, activity partitions are created horizontally aligned. You can rotate those elements to fit your diagram.

To rotate activity partitions

1. Create one or more activity partitions within activities.
2. Right-click the Activity with activity partitions and select **Rotate Activity Partitions**.

Related Procedures

Creating a Shortcut

Related Reference

UML 2.0 Activity Diagrams

Using Control Flow Link

A transition link represents a control flow. It can be drawn between the following elements on the state or activity diagrams. It can also be drawn from an initial node, or to an activity final or final flow element from those elements listed below.

- state
- activity invocation
- decision
- fork/join
- history

To create a control link between two elements

1. Click the Control Flow link button on the tools palette.
2. On the Diagram, click the source element.
3. Drag the link to the destination element.
4. Drop when the second element is highlighted.

After the link has been drawn on the diagram, use the Properties view to update the link. Properties set for the link are shown on the diagram.

Related Reference

UML 2.0 Activity Diagram Elements

Working with Activities and State Machines Full Screen Diagrams

It is possible to work with the individual activities and state machines in specific diagrams. You can open an activity or a state machine in a full-screen view, and the element is expanded to the whole diagram. The new diagram has the same name as the selected element. This kind of activity or state machine is transparent, which makes it possible to view the grid. You cannot move the container activity or state machine, but the nested elements are still selectable and movable.

To create a full-screen diagram for an activity or a state machine

1. Select the desired element in the diagram or in the Model Navigator.
2. On the context menu of the selection, choose **Open Full Screen in New Diagram**.

Working with Activity Element

Because all activity diagram elements are enclosed into the Activity element, you should create at least one Activity to start modeling in the Activity diagram.

To create an activity element

1. On the Palette, click the **Activity** button.
2. Click the diagram background.

For your convenience, an Activity element can be opened in a separate Diagram Editor view.

To open an activity element in a separate Diagram Editor view

1. Right-click an activity element in the Diagram Editor
2. Select **Open Full Screen in New Diagram**.

Working with an Object Flow or a Control Flow

You can create control flow or object flow as an ordinary link between the two node elements. The valid nodes are highlighted when the link is established.

You can scroll to the target element if it is out of direct reach, or you can use the context menu command to avoid scrolling.

There are certain limitations stipulated by UML 2.0 specifications:

- An object flow link must have an object on at least one of its ends.
- It is impossible to connect two actions with an object flow except through an output pin on the source action.
- Control flow link may not connect objects and/or activity parameters.

Use the following techniques with an object flow or a control flow:

1. Create a flow. Flows are created as the regular links.
2. Create a fork or a join.
3. Create a decision or a merge.

To create a fork or a join

1. Identify the actions involved. If necessary, place all of the actions on the diagram first. Lay them out however you want.
2. Place either a fork or a join on the diagram. Resize as needed.
3. If depicting multiple sources, draw a control flow from each of the source actions to the join, and from the join to the target action. If depicting multiple targets, draw a control flow from the source action to the fork, and from the fork to each of the target actions.

To create a decision or a merge

1. Identify the actions involved. If necessary, place all of the actions on the diagram first. Lay them out however you want.
2. Place either a decision or a merge on the diagram. Resize as needed.
3. If merging multiple actions, draw a control flow from each of the source actions to the merge, and from the merge to the target action. If making a decision, draw a control flow from the source action to the decision, and from the decision to each of the target actions.

Related Procedures

Creating a Simple Link

Related Reference

UML 2.0 Activity Diagrams

UML 2.0 Component Diagrams Procedures

In This Section

Designing a UML 2.0 Component Diagram
How to design a UML 2.0 Component Diagram.

Working with a Provided or Required Interface
How to work with the provided and required interfaces. These procedures are common to UML 2.0 Class, Component and Composite Structure diagrams.

Working with Instance Specifications
Lists the steps for instantiating classifiers using the Properties View or the in-place editor.

Designing a UML 2.0 Component Diagram

You can use the following tips and techniques when working with UML 2.0 Component Diagrams. It might be convenient to begin creating a model with Component Diagrams if you are modeling a large system, such as a distributed, client-server software system, with numerous interconnected modules. You use Component Diagrams for modeling a logical structure of your system, and you use Deployment Diagrams for modeling a physical structure.

To design a UML 2.0 Component Diagram

1. Create a hierarchy of components. The largest component can be the whole system or its major part (for example, *server application*, *IDE*, *service*).

   Tip: You can create nested component nodes. There are two methods for creating a nested component node. You can select an existing component and add a child component inside. Alternatively, you can create two separate components and connect them with an Association-Composition link.

2. In the hierarchy of components, you can end up by adding concrete classes and instance specifications. You can create them on a Component Diagram directly or create them on a Class Diagram and put shortcuts on a Component Diagram.
3. Create interfaces. Each component can have a provided interface and a required interface.
4. Optionally, create artifacts. Usually, you describe physical artifacts of your system on Deployment Diagrams. But if a component is closely connected with its physical store, add and link an artifact to a Component Diagram.

   Tip: You can create nested artifacts.

5. Optionally, create ports for your components. You can attach a port to a component and link it with several classes or components inside. In this case, when a message arrives, this port decides which class must handle it.
6. Draw links between elements.
7. You can optionally create shortcuts to related elements of other diagrams.

Related Procedures

Working with a Provided or Required Interface
Creating a Shortcut

Related Reference

UML 2.0 Component Diagrams

Working with a Provided or Required Interface

To create a provided interface

1. Create class and interface node elements using the Palette buttons.
2. On the diagram Palette, click the **Provided Interface** button.
3. Click the client class and drag the mouse to the interface node.

To create a required interface

1. Create class and interface node elements using the Palette buttons.
2. On the diagram Palette, click the **Required Interface** button.
3. Click the client class and drag the mouse to the interface node.

Related Procedures

Changing the Appearance of Interfaces

Related Reference

UML 2.0 Class Diagrams
UML 2.0 Component Diagrams
UML 2.0 Composite Structure Diagrams

Working with Instance Specifications

According to the UML 2.0 specification, an instance specification can instantiate one or more classifiers. You can instantiate a classifier using the **instantiates** property in the Properties View or the in-place editor.

To instantiate a classifier using the Properties View

1. Select an instance specification in your diagram.
2. In the **Properties** node of the Properties View, select the **instantiates** field.
3. Click the chooser button.
4. In the **Choose Classifier for 'instantiates' property**, select the classifiers from the available contents, using the Add/Remove buttons.
5. Click OK to save your changes.

To instantiate a classifier using the in-place editor

1. Select an instance specification in your diagram.
2. Press F2 to open the in-place editor. Alternatively, click twice on the instance specification name.
3. Type the name of an existing classifier, delimited by a colon, next to the instance specification name. For example, `InstanceSpecifcation1:Class1`.
4. Press Enter.

To define the features of an instance specification, you can insert slots into an instance specification element, associate the slots with the attributes of the instantiated classifiers, set the value, and define the slot stereotype.

To add a slot to an instance specification element

1. Add an instance specification element to your diagram.
2. Right-click the instance specification element and choose New Slot on the context menu.

To associate a slot with a structural feature

1. Select a slot in an instance specification element.
2. Click the **Properties** tab of the Properties View.
3. In the defining **feature field**, select the attribute you want from the list of attributes owned by the classifiers, which is instantiated by the instance specification (or their parents).

To set the slot value, do one of the following:

1. In the Properties View of the slot, select the **value** field, click the Editor button, and type the string in the **Edit property values** editor, OR
2. Invoke the in-place editor for the slot and type the value next to the slot name, delimited by an equal sign.

Related Procedures

UML 2.0 Component Diagrams Procedures
UML 2.0 Composite Structure Diagrams Procedures
UML 2.0 Class Diagrams Procedures

UML 2.0 Deployment Diagrams Procedures

This section provides how-to information about designing UML 2.0 deployment diagrams.

In This Section

Designing a UML 2.0 Deployment Diagram

You can use the following tips and techniques when you design a UML 2.0 Deployment Diagram. It might be convenient to begin creating a model with Deployment Diagrams if you are modeling a large system that is comprised of multiple modules, especially if these modules reside on different computers. You use Deployment Diagrams for modeling a physical structure of your system, and you use Component Diagrams for modeling a logical structure.

To design a UML 2.0 Deployment Diagram, follow this general procedure

1. Create a hierarchy of execution environments, devices, and nodes. Execution environments usually represent a software environment that is used to execute your system, such as an operating system. Devices usually represent hardware equipment, such as a printer, a hard disk, or a computer. Nodes represent the remaining physical entities, such as a file.

   Tip: You can create nested execution environments, devices, and nodes. For example, you can add a node inside of an execution environment, or a node inside of a device.

2. Create artifacts.
3. Create deployment and instance specifications. By doing this, you arrange physical locations of objects and other entities of your system.
4. Add operations to artifacts.
5. After an operation is added, you can define its properties in the Properties View, which includes parameters, stereotype, multiplicity and more.
6. You can optionally create shortcuts to related elements of other diagrams.

To deploy an artifact to a target node

1. In the diagram Palette, choose the deployment button.
2. Drag-and-drop the deployment link from a node to an artifact.

To define parameters of an operation

1. Select the desired operation in an artifact.
2. In the Properties View, expand the **General** node and choose **Parameters** field.
3. Click the chooser button to open **Add/Remove Parameters** dialog box.
4. Click **Add**. This creates an entry in the parameters list.
5. Enter the parameter's name, type, multiplicity, default value, and direction. Note that parameter type can be selected from the list of predefined types, or from the model.
6. Using the **Add** and **Remove** buttons, create the list of parameters.
7. Click **OK** when ready.

Related Procedures

Creating a Shortcut

Related Reference

UML 2.0 Deployment Diagrams

Working with Artifacts

An artifact represents a physical entity and is depicted in a diagram as a rectangle with the `<<artifact>>` stereotype. An artifact may have properties that define its features, and operations that can be performed on its instances.

Physically, the artifacts can be model files, source files, scripts, binary executable files, a table in a database system, a development deliverable, a word-processing document, or a mail message.

A deployed artifact is one that has been deployed to a node used as a deployment target. Deployed artifacts are connected with the target node by deployment links.

Tip: You can create complex artifacts by nesting artifact icons.

To add operation to an artifact

1. Right-click an artifact icon in the diagram.
2. Choose **New Operation** on the context menu.

After an operation is added, you can define its properties in the Properties View.

To define parameters of an operation

1. Select the desired operation in an artifact.
2. In the Properties View, expand the Common properties node and click the parameters field.
3. Click the Edit button to open the **Select Parameters for Operation** dialog box.
4. Click **Add**. This creates an entry in the parameters list.
5. Enter the parameter's name, type, direction kind, multiplicity, default value. Note that the parameter type and direction kind can be selected from the list of predefined values.
6. Repeat steps 4–5 to create the list of parameters and click OK when you are finished.

To deploy an artifact to a target node

1. Click the deployment button in the Palette.
2. Click the element to be deployed. The valid source is highlighted.
3. Drag-and-drop the deployment link to a target node. The valid target is highlighted.

UML 2.0 Composite Structure Diagrams Procedures

In This Section

Creating a Port

To create a port:

1. Choose the port icon on the Palette.
2. Click the target class or part.
3. Create as many ports as required.

Related Reference

UML 2.0 Composite Structure Diagrams

Creating a Referenced Part

To create a referenced part:

1. Open the **Diagram View**.
2. Do one of the following:
   - Use the referenced part button on the diagram Palette.
   - Right-click a target container and choose **New ▸ Referenced part** on the context menu.
   - Select a part, open the Model Navigator, and check the option aggregated by reference.

Related Reference

UML 2.0 Composite Structure Diagrams

Creating an Internal Structure for a Node

To create an internal structure for a node

1. Choose the part icon on the diagram Palette.
2. Click the valid container (class or collaboration).
3. Repeat these steps to create as many participants as needed.

Tip: Choose the part icon on thePalette diagram while holding down the CTRL key. Each click on a valid container produces a new part.

4. Link the collaborating parts by connectors.
5. Use the Properties View to set up the properties of the part.

Related Reference

UML 2.0 Composite Structure Diagrams

Working with a Collaboration Use

To create a collaboration use

1. On the Tools Palette, choose the Collaboration Use button.
2. Click the target container.
3. Specify the name of the Collaboration Use.

To link to a collaboration type

1. Select a Collaboration Use element.
2. Specify the type of Collaboration Use using one of the following methods:
    - In the type field of the Collaboration Use in the Tools Palette, click the chooser button, and select the collaboration, which the Collaboration Use instantiates, from the model.
    - Next to the name of the Collaboration Use, insert a colon and the name of the collaboration, which the Collaboration Use instantiates.

The type of collaboration use is now indicated next to its name.

To bind the roles (parts) of the different classifiers via the collaboration use

1. Create a collaboration use and define its type.
2. Create one or more parts in the collaboration that represents the type.
3. Right-click the target collaboration use and choose Bind new role on its context menu.
4. In the Select Destination dialog box that opens, choose the role to be bound in the target classifier.

A role link is now created from the collaboration use to the role in the target classifier. The role link is now marked with the name of the role selected in the collaboration.

Note: Each role can be used for binding only once. With the next invocation of the Bind new role command, the list of available roles no longer displays the ones previously used.

To define an owner

1. Right-click a collaboration use and choose **Properties** on its context menu.
2. In the owning classifier field of the Properties View, click the chooser button.
3. In the **Select Owning Classifier** dialog box, navigate to the owner class or collaboration and click OK.

A link is now created between the owner as supplier and the collaboration use as the client. The link is marked with the `<<occurrence>>` label.

Related Reference

UML 2.0 Composite Structure Diagrams

Working with a Provided or Required Interface

To create a provided interface

1 Create class and interface node elements using the Palette buttons.
2 On the diagram Palette, click the **Provided Interface** button.
3 Click the client class and drag the mouse to the interface node.

To create a required interface

1 Create class and interface node elements using the Palette buttons.
2 On the diagram Palette, click the **Required Interface** button.
3 Click the client class and drag the mouse to the interface node.

Related Procedures

Changing the Appearance of Interfaces

Related Reference

UML 2.0 Class Diagrams
UML 2.0 Component Diagrams
UML 2.0 Composite Structure Diagrams

Working with Instance Specifications

According to the UML 2.0 specification, an instance specification can instantiate one or more classifiers. You can instantiate a classifier using the **instantiates** property in the Properties View or the in-place editor.

To instantiate a classifier using the Properties View

1. Select an instance specification in your diagram.
2. In the **Properties** node of the Properties View, select the **instantiates** field.
3. Click the chooser button.
4. In the **Choose Classifier for 'instantiates' property**, select the classifiers from the available contents, using the Add/Remove buttons.
5. Click OK to save your changes.

To instantiate a classifier using the in-place editor

1. Select an instance specification in your diagram.
2. Press F2 to open the in-place editor. Alternatively, click twice on the instance specification name.
3. Type the name of an existing classifier, delimited by a colon, next to the instance specification name. For example, `InstanceSpecifcation1:Class1`.
4. Press Enter.

To define the features of an instance specification, you can insert slots into an instance specification element, associate the slots with the attributes of the instantiated classifiers, set the value, and define the slot stereotype.

To add a slot to an instance specification element

1. Add an instance specification element to your diagram.
2. Right-click the instance specification element and choose **New** **Slot** on the context menu.

To associate a slot with a structural feature

1. Select a slot in an instance specification element.
2. Click the **Properties** tab of the Properties View.
3. In the defining **feature field**, select the attribute you want from the list of attributes owned by the classifiers, which is instantiated by the instance specification (or their parents).

To set the slot value, do one of the following:

1. In the Properties View of the slot, select the **value** field, click the Editor button, and type the string in the **Edit property values** editor, OR
2. Invoke the in-place editor for the slot and type the value next to the slot name, delimited by an equal sign.

Related Procedures

UML 2.0 Component Diagrams Procedures
UML 2.0 Composite Structure Diagrams Procedures
UML 2.0 Class Diagrams Procedures

Template Elements

This section describes how to create template elements in diagrams and define formal parameters.

In This Section

Creating Constraints

You can create constraints for all elements of the UML 2.0 diagrams. To describe a constraint, you can use plain text or OCL.

To create a constraint in a UML 2.0 diagram

1. Click the **Constraint Link** button on the diagram Palette and point to the model element that defines the context of your constraint (such as Class, Attribute or Operation), then hold down the left mouse button and draw the link to the place where you want to create the **Constraint** element.
2. Release the mouse button to insert the element.

   The element displays with the in-place editor open.
3. Type the constraint expression, save your changes, and close the **Constraint** editor.

   Tip: Alternatively, use one of the following methods:

   - On the context menu of an element, choose **New** ▸ **Linked Constraint** and enter the constraint expression.
   - Use the **Constraint** and **Constraint link** buttons on the Tools Palette to place a constraint node on the diagram and link it to the context element.

Related Concepts

About OCL Support in Together

Creating Generic Template Elements in LiveSource Projects

In this topic you will learn how to:

- Enable template specialization in a Java project
- Create a template specialization using the Properties View
- Create a template specialization using the **Class by Template** dialog.

To enable template specialization in a Java project

1. Select a Java project node in the Model Navigator.
2. On the main menu, choose **Project** ▸ **Properties**.
3. In the **UML Template Specialization** category of the **Project Properties** dialog, check the **Enable template specialization** option.

To create a template specialization element using the Properties View

1. Create a class or interface in a diagram.
2. In the Properties View of the class, select the **template parameters** field.
3. Enter one or more parameters in the text area. Use commas to separate multiple parameters.

To create a template specialization element using the Class by Template dialog

1. Right-click on the diagram background and choose **New** ▸ **Class by Template** on the context menu.
2. In the Templates list, locate the **Default Template <>** template and click **Finish**.

Related Concepts

Template Elements and Generics Overview

Creating Template Elements

In this topic you will learn how to:

- Add a template signature to a templateable element
- Bind a templateable element to a template.

Note: These tasks are common for all UML 2.0 diagrams.

To create a template element using the Tools Palette

1. Add a template signature to a templateable element using the **Template Signature** button on the Palette and clicking on the target model element in diagram. The template signature rectangle is added to the element.
2. Define formal parameters in the **Formal Template Parameters** dialog using the **Add** and **Remove** buttons. Specify each parameter's name, metaclass and constraint. Use the **Up** and **Down** buttons to define the order of parameters in the template signature.
3. Bind a templateable element to a template using the **Template Binding** button and drawing a link from a template to the template signature.

You can achieve the same goal using the Properties View.

To create a template element using the Properties View

1. In the Properties View of the selected model element, set the **isTemplate** field to `true`.
2. Click the **template binding** field to open the **Select Template Signature** dialog.
3. In the dialog, select the desired templates to be bound to the element. Use the **Add** and **Remove** buttons to make up the list of bound templates. Click **OK**.

Related Concepts

Template Elements and Generics Overview

Related Procedures

Defining Formal Parameters

Related Reference

Selection Manager

Defining Formal Parameters

Note: This task is common for all UML 2.0 diagrams.

To define formal parameters of a template using the Properties View

1. Select a template signature in the diagram.
2. In the Properties View, click the **formal parameters** field.
3. In the **Formal Template Parameters** dialog that opens, define formal parameters using the **Add** and **Remove** buttons. Specify each parameter's name, metaclass and constraint. Use the **Up** and **Down** buttons to define the order of parameters in the template signature.

This can also be accomplished using the Properties View.

Related Concepts

Template Elements and Generics Overview

Related Procedures

Creating Template Elements

Related Reference

Selection Manager

Editing Constraint Expressions

Constraint expressions are represented in plain text or in the OCL language. You can use the Editor view or the OCL tab of the Properties View to create or modify the constraint body.

To edit a constraint expression in the Editor view

1 Double-click a constraint element. The constraint test opens in its own tab of the Editor view.

2 In the **Language** drop-down list in the upper-right corner of the view, select the desired language of the expression.

   Note: If OCL is selected, the OCL editor provides syntax control and error highlighting. A red or green mark to the right indicates the validity of the OCL expression.

3 Apply changes.

To edit a constraint expression in the Properties View

1 Select a constraint element in diagram.

2 In the Properties View, select the **OCL** tab.

3 In the **language** field, select the desired language of the expression.

4 In the **body** field, enter the expression in the text area, or click the **Edit** button and enter text in the **Enter constraint** dialog box.

Tip: Alternatively, select a constraint element and press F2. Edit the constraint in the editor.

Related Concepts

About OCL Support in Together

Related Procedures

Creating Constraints

Related Reference

Diagram View

Together UML 1.4 Diagrams

This section provides how-to information on using Together UML diagrams.

In This Section

UML 1.4 Class Diagrams Procedures

In This Section

Changing the Appearance of Compartments

You can collapse or expand compartments for the different members of class, interface, and package elements. Use the **Preferences** dialog to set viewing preferences for compartment controls. Adding compartment controls is particularly useful when you have large container elements with content that does not need to be visible at all times.

To show compartment controls

1. On the main menu, choose **Window** ▶ **Preferences**.
2. Open the **Modeling** ▶ **View Management** page.
3. Check the **Always show Attributes and Operations compartments** option.

To collapse or expand compartments

1. Select the class (or interface) on the diagram.
2. Click the “**+**” or “**-**” in the left corner of the compartment.

Related Reference

UML 2.0 Class Diagrams
UML 1.4 Class Diagrams

Creating and Editing Constructors

To add a constructor to a class

1. Right-click the class.
2. Select **New** > **Constructor** from the context menu.

You can edit the constructor by using in-place editing or by using its Properties view.

Use drag and drop to move constructors between class elements on the diagram. The constructor name automatically updates to reflect the current class.

Creating Class By Template

Use the Class By Template button on the Palette diagram to implement source code constructions or solutions in your model.

Note: This feature is available in the implementation projects only.

To create a class by template

1. Select Class by Template in the Tools Palette.
2. Click on the diagram background. The **Apply template** dialog box opens.
3. Select the appropriate template from the **Templates** tree.
4. Set each value field within the **Parameters** area, or click **Finish** to apply default values.

Related Procedures

Apply Template Wizard

Expanding or Collapsing Compartments

To expand or collapse compartments for members

1. Select the class or interface.
2. Click the "+" or "-" in the section's upper left corner.

Related Procedures

Hiding and Showing Members

Extending and Implementing Classes and Interfaces

Use the Properties View or the **Generalization/Implementation Link** button to extend a class or implement an interface. A generalization/implementation link displays between the two elements for classes/interfaces located in the same package or diagram. However, for classes and interfaces that reside in the different packages, the base class name displays in the upper right corner of the corresponding class. You must use the Properties View to designate a base class if the base class (or interface) resides in a different package than the extending/implementing class.

A class may implement more than one interface. To show this on the diagram, draw Generalization/Implementation links from the class to each interface. You may also indicate this by updating the **implements** property for the class in the Properties View.

To choose an extending class or implement an interface using the Properties View

1. Right-click on the class.
2. In the Properties View, select the **extends** field or **implements** field.
3. In the **Model elements** tab, select the class or interface, and click Add.
4. Click OK.

Hiding and Showing Members

You can use the **Hide / Show** command for class or interface elements. This command gives you a more granular control over the class or interface element's hidden members.

It is also possible to use the **Show Hidden** command accessible via the diagram context menu to open the **Show Hidden** dialog. For more information on using the Show Hidden dialog and Show Hidden command via the diagram context menu, see Hiding and Showing Model Elements. Note that this topic is specific to using the Show Hidden command available on the class and interface element context menu.

To hide a member of a class or interface element

1. Select the member on the diagram. You can select more than one at a time by using `CTRL+Click`.
2. Open the context menu for your selection, and choose **Hide**. The selected members are hidden from the class or interface element.

To show the hidden members

1. Select the class or interface containing the hidden members and right-click to open the context menu.
2. Choose **Hide / Show** ▶ **Reveal hidden members** from the context menu. All of the previously hidden members are displayed on the class or interface element.

The **Reveal hidden** members command and the **Hide / Show** command provide the following options:

Option	Description
Attributes	Selecting Attributes from the submenu hides only the attributes of the class. The **Hide / Show** command for Attributes works as a toggle. To redisplay hidden attributes, right-click the class where the attributes are hidden, and select **Hide / Show** ▶ **Attributes** from the context menu.
Operations	Selecting Operations from the submenu hides only the operations of the class. The Hide / Show command for Operations works as a toggle. To redisplay hidden operations, right-click the class where the operations are hidden, and select **Hide / Show** ▶ **Operations** from the context menu.
Properties	Selecting Properties from the submenu hides only the properties of the class. The Hide / Show command for Properties works as a toggle. To redisplay hidden properties, right-click the class where the properties are hidden, and select **Hide / Show** ▶ **Properties** from the context menu.
Inner Classes	Selecting Inner Classes from the submenu hides only the inner classes of the class. The Hide / Show command for Inner Classes works as a toggle. To redisplay a hidden inner class, right-click the class where it was hidden, and select **Hide / Show** ▶ **Inner Classes** from the context menu.
Inner Interfaces	Selecting Inner Interfaces from the submenu hides only inner interfaces of the class. The Hide / Show command for Inner Interfaces works as a toggle. To redisplay a hidden inner interface, right-click the class where it was hidden, and select **Hide / Show** ▶ **Inner Interfaces** from the context menu.
All	Selecting All from the submenu hides attributes, operations, properties, inner classes, and inner interfaces of the class. The Hide / Show command for All works as a toggle. To redisplay hidden class elements, right-click the class where the elements are hidden, and select **Hide / Show** ▶ **All** from the context menu.

Related Procedures

Hiding and Showing Model Elements

Instantiating a Classifier

You can create an object that instantiates a class or interface from the same project or from a referenced project. You can create such links by using the Properties View or by using a Dependency link.

To instantiate a classifier

1. Select an object in a class diagram.
2. In the Properties View of the object, choose the **Instantiates** field.
3. Click the **Chooser** button. The **Choose Type to Instantiate** dialog box opens.
4. In this dialog box, choose a classifier (class or interface).

Tip: Alternatively, draw a **Dependency link** from this object to a classifier.

Related Procedures

Working with Instance Specifications

Setting Abstract or Final for a Class or Interface

To set Abstract or Final for a Class or Interface

1. From the Diagram editor, right-click the class or interface.
2. Select Modifiers from the context menu.
3. Choose either Final or Abstract.

Note: Setting a class modifier to Abstract italicizes the class name on the diagram.

Setting Visibility for a Class or Interface

To define the visibility modifier for a class or interface

1. In the Diagram Editor , right-click a class or interface.
2. Select **Properties** on the context menu.
3. In the Properties tab, check the **public** option to set the public modifier.

Setting Visibility for Members of a Class or Interface

You can set visibility modifiers for members of a class or interface by using the context menu for each element, or by using the Properties View.

Visibility modifiers include:

- public
- private
- protected
- package local

To set visibility for a member

1. Right-click a member in the Diagram editor.
2. Choose **Modifiers** on the context menu and select from Public, Protected, Private, or Package Local.

Tip: Values such as visibility can also be set by double-clicking the member on the diagram and entering Java format directly into the in-place editor: `private int attr1`, for instance, and `-attribute:int=10` would set an attribute's initial value to 10.

Alternatively, use the Properties view for the member

1. Select the member on the diagram. The Properties View displays the associated properties for the member selected.
2. Use the visibility field drop-down list in the Properties View.

Note: You can also use the Properties view to designate members as static.

Showing Different Modeling Views

By default, the Diagram Editor reflects the classic UML modeling view. Visibility modifiers are represented on the diagram in the following ways:

- **static:** members are underlined.
- **public:** members have a '+' symbol before the name.
- **private:** members have a '-' symbol before the name.

To show icons on class and package diagrams

1. From the main menu, select **Window** ▶ **Preferences**. The **Preferences** dialog opens.
2. From the options list on the left, expand **Modeling** node, and select **View Management.**
3. In the **Show Icons for** section, define the following options to better distinguish metaclasses of elements with similar looks:
    - Element shown as a label inside another element
    - Diagrams
    - Classifier shown as Class

    Refer to the View Management Preferences for detailed description of these options.

To show visibility modifiers

1. From the main menu, select **Window** ▶ **Preferences**. The **Preferences** dialog opens.
2. From the options list on the left, expand **Modeling** node, and select **View Management.**
3. In the **Detail level** section, click the **Implementation** radio-button.
4. Apply changes.

Related Reference

View Management Preferences

Showing Interfaces as Small Circles (lollipops)

Interfaces can be represented as rectangles or small circles ("lollipops") on your diagrams. You can change the representation of the interface element in your View Management preferences.

Note: Interfaces shown as small circles do not show their members in the Diagram editor. Instead, use the Model Package Explorer or Model Navigator to view the members.

To show an interface as a circle

1. Choose **Window** ▸ **Preferences** from the main menu. The Preferences window opens.
2. Expand Modeling and select View Management.
3. Check the Show Simple Java Interfaces option.
4. Click OK to close the dialog and apply the changes.

Working with a Constructor

You can create as many constructors in a class as needed using the **New ▸ Constructor** command of the context menu of a class.

In implementation projects, each new constructor is created with its unique set of parameters. In addition to creating parameters automatically, you can define the custom set of parameters using the Properties View.

In design projects, a constructor is created as an operation with the `<<create>>` stereotype.

Tip: You can move, copy and paste constructors and destructors between the container classes the same way as you would do the other members.

To define the constructor parameters

1. Select a constructor in a class.
2. In the Properties View, click the **Browse** button in the **parameters** field.
3. In the **Select Parameters for Operation** dialog that opens, click **Add**. A parameter is added with the default values. Edit the values as required, or use the defaults.

   Use the **Add** and **Remove** buttons to make up the list of parameters, and click **OK** when you are finished.

Tip: Alternatively, type the list of parameters in the text area. Use a comma as a delimiter.

Related Reference

UML 2.0 Class Diagrams
UML 1.4 Class Diagrams

Working with a Field

You can edit members using the Properties View, or the in-place editor of the Diagram Editor or Model Navigator. In the implementation projects, you can also use the source code editor to modify the members. In this section you will learn how to:

- rename a field
- define a visibility modifier
- define a stereotype
- define modifiers, initial values, and associated objects
- handle multi-declarations

To rename a field

1. Choose a field.
2. Enter the new name in the in-place editor of the Diagram Editor or Model Navigator, or use the **name** text field in the Properties View.

To define the visibility modifier

1. Choose a field.
2. Enter the visibility symbol in the in-place editor in the Diagram Editor , or select one from the **visibility** combobox in the Properties View.

To define the stereotype of a field

1. Choose a field.
2. Use the in-place editor in the Diagram Editor , or use the stereotype combobox of the Properties View.

To define modifiers, initial values, associated objects and so on

1. Choose a field.
2. Use the Properties View or the source code editor (for implementation projects).

When you do this, the model and the source code are kept in sync.

Note: You can type the *Value* property, an equal sign (`=`), and the *Name* property (for example, `EXCLUDE=2`) when adding an Enum literal with the inplace editor.

To use multi-declarations in the source code, consider the following:

1. In the source code of the Java and IDL projects, it is possible to declare several fields in one line. This notation is represented in the diagram as a number of separate entries in the Fields section in a class icon.
2. You can rename the fields, change modifiers, set initial values and so on, and all modifications will be applied to the respective field in the diagram icon.

3 You can copy and move such fields in a diagram (using the context menu commands or drag-and-drop), and the pasted field will appear in the target container separately.

In C++ projects, editing multi-declarations in the Diagram Editor or Properties View is not allowed.

Related Procedures

Navigating between the Tree View, Diagram, and Source Code
Adding a Single Model Element to a Diagram

Related Reference

UML 1.4 Class Diagrams
UML 2.0 Class Diagrams

Working with a Relationship

Refer to the Getting Started Procedures to learn how to draw a link. This section describes how to change the link type and properties.

To change the type of an association link

1. Select an Association Link on the diagram.
2. In the Properties View, select the **Link** tab and click the **associates type** field.
3. Choose the link type (association, aggregation, or composition) from the drop-down list.

To set the directed property of an association link

1. Select the association link that you want on the diagram. The properties for the link appear in the Properties View.
2. In the **Link** tab of the Properties View, select the **directed** field.
3. Click the drop-down arrow and select the value for this Boolean property.

Related Procedures

Getting Started Procedures
Creating a Simple Link
Changing Type of an Association Link

Related Reference

UML 1.4 Class Diagrams
UML 2.0 Class Diagrams

Working with Association classes and n-ary associations

Association classes appear in diagrams as three related elements:

- Association class itself (represented by a class icon)
- N-ary association class link (represented by a diamond)
- Association connector (represented by a link between both)

To create an association class

1. On the diagram Palette, select the **Association Class** button.
2. Click the diagram background. This adds a regular class icon for the association class, connected with the diamond icon that represents the Association Class Link Aspect.
3. Create participant classes.
4. Using the **Association End** button, connect the diamond icon with the participant classes.

The source code of an association class now contains appropriate tags for the association class itself, and for each of the association end classes.

To delete an association class

1. Right-click an association class or its diamond icon.
2. Choose **Delete** on the context menu.

The whole association class construct is now deleted from the diagram.

Related Reference

Class Diagram Relationships
UML 2.0 Class Diagrams
UML 1.4 Class Diagrams

Working with Inner Classes

Both inner classes and inner interfaces in diagrams display within their own compartment field within the class. To create an inner class, do one of the following:

When the class already exists

1. Drag it over the target class.
2. Drop it.

Using the context menu

1. Right-click the parent class.
2. Select **New** ▸ **Inner Class** from the context menu.

Using Cut, Copy, and Paste

1. Use the clipboard operations to either cut or copy an existing inner class.
2. Select the parent class.
3. Use the clipboard operations to paste the selected class into the parent class.

Tip: Classes do not keep the same visibility after they are removed from the parent class.

UML 1.4 Use Case Diagrams Procedures

In This Section

Creating an Extension Point

To create an extension point

1. Right-click the use case element.
2. Choose **Add ▸ Extension Point** on the context menu.
3. Type in a name.

Related Reference

UML 1.4 Use Case Diagrams
UML 2.0 Use Case Diagrams

UML 1.4 Interaction Diagrams Procedures

In This Section

Adding a Conditional Block

You can create one or more statement blocks on an activation bar of an object. The following statement blocks are available:

- If
- Else
- Else-If
- For
- While
- Do
- Try
- Catch
- Finally
- Switch

Note: When the statement blocks are created, you can update them using the Properties View or the in-place editor. The available properties fields depend on the type of the statement block. For example, the **type** field is not available for the statement blocks `switch, finally` and `catch`.

To add a statement block to the activation bar

1. Right-click an activation bar on a sequence diagram.
2. Choose **New** on the context menu.
3. On the submenu, select a conditional block.
4. In the in-place editor that opens, enter the body of the block statement, and press ENTER.

To edit a conditional block

1. Select a statement block in the diagram.
2. Open the Properties View.
3. Modify the editable fields as required. For example, select a new type of the statement block from the drop-down list in the **type** field.

Related Reference

UML 1.4 Interaction Diagrams

Branching Message Links

This section describes how to branch messages that start from the same location on the **lifeline**.

To branch a message link with the previous one

1. Select a message link on the sequence diagram.
2. Drag the message source to the source of the message you would like to branch with.
3. Drop the message source when a green circle mark appears.

To remove branching

1. Select the message link that you want to unbranch, and grab the bending point at the message number.
2. Drag and drop the message to the new source.

Related Procedures

Working with a UML 1.4 Message

Related Reference

UML 1.4 Interaction Diagrams

Converting Between UML 1.4 Sequence and Collaboration Diagrams

You can convert between sequence and collaboration diagrams. However, when you create a new diagram, you must specify that it is either a sequence diagram or a collaboration diagram.

To convert between sequence and collaboration diagrams

1. Right-click the diagram background.
2. If the diagram is a sequence diagram, choose **Show as Collaboration** on the context menu. If the diagram is a collaboration diagram, choose **Show as Sequence**.
3. Repeat this process to switch back and forth.

After you convert from a sequence diagram to a collaboration diagram for the first time, or if you have added new objects to the sequence diagram between conversions, it is recommended that you perform a full layout on the collaboration diagram.

Related Reference

UML 1.4 Interaction Diagrams

Creating Slots

To define the features of an object, you can insert slots into the object element, associate the slots with the attributes of the instantiated classifiers, and set values.

To add a slot to an object element

1. Add an object element to your diagram.
2. Right-click the object element on your diagram, and choose **New** ▸ **Slot** on the context menu.

To associate a slot with a structural feature and define the slot's value

1. Select a slot and open its Properties View.
2. In the **defining feature** field, type a value in a String format.
3. In the **value** field, enter the desired string.

Generating an Incremental Sequence Diagram

You can generate incremental sequence diagrams from a previously generated sequence diagram. In some cases, you can generate a sequence diagram with a low nesting value, such as 3 or 5. The nesting value limits how deep the parser traverses the source code calling sequence.

To generate an incremental sequence diagram from a previously generated sequence diagram

1. After you review the sequence diagram, you may decide that you want to see additional objects and messages that are currently not shown on the diagram because of the nesting value constraint.
2. In this case, select the **Generate Sequence Diagram** command from the context menu of an activation block. The nested messages and objects calling from that method are displayed on the diagram.

Related Procedures

Roundtrip Engineering with Sequence Diagrams

Related Reference

UML 1.4 Interaction Diagrams

Refining Collaboration Diagrams

This section provides techniques for refining collaboration diagrams. For information on sequence diagrams, see Refining Sequence Diagrams.

When working with the collaboration diagrams, consider the following:

- When you draw a message between objects, a generic link line displays between the objects, and a list of messages is created above it. The link line is present as long as there is at least one message between the objects.
- As you add messages, they display in time-ordered sequence from top to bottom of the messages list. You can select messages and edit their properties in the message Properties View just as you do in a sequence diagram.
- The collaboration diagram adds the capability of showing relationships between objects. In addition to the default link, you can add links to show association and aggregation relationships. These links are not available in the sequence diagram.
- Together allows you to quickly convert between sequence and collaboration diagrams. However, whenever you create a new diagram, you must specify that it is either a sequence or collaboration diagram, and Together tracks it as such. The diagram displays in the Model Package Explorer and Model Navigator as the type of origin, and opens in that view. For example, if you create a collaboration diagram, it will always display in the Model Package Explorer and Model Navigator and open in the Diagram editor as a collaboration diagram. However, you can view it as a sequence diagram.
- By default, message links are represented on a collaboration diagram as having an atomic delivery. This indicates the duration required to send the message is atomic, which means that nothing else can happen during the message transaction. If the message link requires some time to arrive, during which something else can occur, then designate the message link as non-atomic using the Non-atomic delivery command.

To convert between sequence and collaboration diagrams

1. Right-click on the diagram background.
2. From the context menu, select Show as Sequence.
3. Repeat this process to toggle between the two diagrams.

Note: You can also switch between diagrams using the context menu for the diagram in the Model Package Explorer or Model Navigator.

To create a message from an object back to itself

1. Click on the link button on the diagram's toolbar.
2. Click on the object where you want the message to appear.
3. Drag the link away from the object.
4. Drag the link back to the object and drop when the object is highlighted.

To associate a self-message link with an operation in its super class

1. Create a message link between two objects. The object that receives the message must be associated with a class.

2 Right-click the link and select **Choose Operation** on the context menu. The operations of the recipient object's class are listed in the drop-down list.
3 Select the operation and click OK. This renames the message link to the operation's name.

To create a message link that calls an operation

1 Create a message link between two objects. The object that receives the message must be associated with a class.
2 Right-click the link and select **Choose Operation** on the context menu. The operations of the recipient object's class are listed in the drop-down list.
3 Select the operation and click OK. This renames the message link to the operation's name.

To designate a message link as non-atomic

1 Right-click on the link.
2 Select Non-atomic Delivery from the context menu.

Related Procedures

Refining Sequence Diagrams

Refining Sequence Diagrams

This section describes techniques that enable you to present your interaction diagram in the most comprehensible way:

- **Switching between sequence and collaboration diagrams.** Together allows you to quickly convert between sequence and collaboration diagrams. Whenever you create a new diagram, you must specify that it is either a sequence or collaboration diagram, and Together tracks it as such. The diagram displays in the Model Navigator as the type of origin, and opens in that view. For example, if you create a sequence diagram, it will always display in the Model Navigator and open in the Diagram Editor as a sequence diagram. However, you can view it as a collaboration diagram.
- **Creating a Message-to-self**.
- **Specifying a Default Return Link.** To manually draw a return link, use the Palette of the sequence diagram. To avoid drawing a default return link, use the Properties View to create a default return link for you.
- **Setting Creation Messages.** Objects display with a default lifeline when placed on the diagram. Their tops align vertically. If you draw a message link to an object and then check the creation type of the message, the created object will move downward to show that it exists at a point later in time from its creator.
- **Specifying Non-Atomic Delivery.** By default, message links are drawn on the diagram horizontally. This indicates the duration required to send the message is atomic, which means that nothing else can happen during the message transaction. If the message link requires some time to arrive, during which something else can occur, then designate the message link as non-atomic using the Non-atomic delivery command.
- **Reordering and Moving Message Links.** To change the sequential order of messages, you can use the drag-and-drop technique, or the Properties View.
- **Adjusting the Size of Object Lifelines.** You can increase or decrease the length of an object's lifeline.
- **Changing the Order of a Sequence Diagram.** You can change the order of object lifelines while preserving the messages that exist between these lifelines.
- **Setting a Destruction Message**. Together indicates the destruction of a created object by rendering a bold **X** on a diagram.
- **Nesting messages.** You can nest messages by originating message links from an activation icon. The nested message inherits the numbering of the parent message. For example, if the parent message has the sequence number 1, its nested message has a sequence number 1.1.

To toggle between sequence and collaboration diagrams

1. Right-click on the diagram background.
2. From the context menu, select **Show as Collaboration**.
3. Repeat this process to toggle between the two diagrams.

Note: You can also switch between diagrams using the context menu for the diagram in the Model Package Explorer or Model Navigator.

To create a message from an object back to itself

1. Open the object.
2. Click the object's lifeline at the point where you want the message to appear.

To associate a self-message link with an operation in its super class

1. Right-click the link.
2. Choose **Select Overridden Operation** on the context menu.

To specify a default return link

1. Select the message link on the sequence diagram. The Properties View displays the associated link properties.
2. In the Properties View, select the return message field. By default, this field displays false.
3. To display a default return link, click the drop-down arrow for the return message field, and select true.

To set the creation type for a message link

1. Right-click the link.
2. Select **Type** ▸ **Creation** from the context menu.

To designate a message link as non-atomic

1. Right-click the link.
2. Select **Non-atomic Delivery** from the context menu.

To reorder message links

1. Select a link.
2. Drag the message link up or down along the lifeline. The sequence numbers of the message links are automatically updated. Alternatively, use the **sequence number** property of a link in the Properties View to reorder message links.

Note: If you select multiple message links (pressing the CTRL key), the links selected are moved with their increments intact.

Note: Moving a creation message below a second message pointing to the same object removes the creation type from the message. Moving a destruction message above another message pointing to the same object removes the destruction type from the message.

To change the size of the object lifeline

1. Select the bottommost message of the lifeline.
2. Drag the message link upward or downward.

To change the order of the object's lifelines

1. Select the object.
2. Drag the object horizontally to the position that you want.

Note: You cannot move Objects vertically along the Y-axis except as described in "Adjusting the Size of Object Lifelines" above.

To set the destruction type for a message link

1. Right-click the link.
2. Select **Type** ▸ **Destruction** from the context menu.

To create nested messages

1. Click the activation bar of the message link.
2. Create a new link originating from the activation bar.

Note: It is also possible to create message links back to the parent activation.

Related Procedures

Refining Collaboration Diagrams

Roundtrip Engineering with Sequence Diagrams

This section demonstrates how to create and edit a UML 1.4 sequence diagram that generates source code. To generate source code from a sequence diagram, you will:

1. Create a project and class that contains the operation that you want
2. Generate a sequence diagram from the main method
3. Create source-generating elements on the sequence diagram
4. Create a message
5. Generate source code for the sequence diagram

To create a project and class

1. Create a UML 1.4 Java modeling project, and add a new class diagram.
2. Create a new class on the diagram.
3. Right-click on the class, and choose **New ▸ Operation** on the context menu. The in-place editor activates.
4. Create an operation. For example, add a main class method by entering the following code in the in-place editor: `main(args: String[]):void`
5. Press ENTER. The new `main()` method is created.

To generate a sequence diagram from the main method

1. Right-click the main method, and choose **Generate Sequence Diagram** from the context menu. The **Generate Sequence Diagram** wizard is displayed.
2. Click **Next**, accepting the default settings for the first page of the wizard.
3. Click **Finish**, accepting the default settings for the second page of the wizard. The sequence diagram opens in a new diagram tab of the Diagram Editor .

Tip: By default, the generated diagram gets the name `[Class_name].[method_name]`

To create source-generating elements on the sequence diagram

1. Right-click the activation rectangle of message #1, and choose **New ▸ For Block** from the context menu.
2. The in-place editor is activated. Using the in-place editor, enter the following code: `int i = 0; i < 4; i++`
3. The label displays on the activation bar as: `for(int i = 0; i < 4; i++)`
4. Add an object to the sequence diagram. Click the **Object** button on the Palette, and click the diagram to create a new object with the name `Frame`.
5. Right-click the frame object, and choose **Select Class ▸ More** from the context menu. The **Select Class** dialog box is displayed.
6. In the **Model elements** list, expand the following nodes: **libraries > javax > swing**. Select **JFrame** from the list, and click **Add>>** .
7. Click **OK** to close the dialog. The name of the selected class is displayed on the frame object.

To create messages on a sequence diagram

1. Draw a message link from the `for(int i = 0; i < 4; i++)` statement block to the lifeline of the Frame object. Message Link 1.1 is created.

   Note: All of the messages created in the following steps should also have the same source and destination.

2. Right-click Message Link 1.1, and choose Select Operation from the context menu. Select the JFrame constructor (`JFrame:void)` from the list. The message label becomes: `1.1: <constructor>()// Message Link1`. The message changes visually to the creation type. Note that the message link now points to the frame object, which means that the object is being created.
3. Draw a new message. Its label is Message Link 1.2. Right-click the message link, and choose **Select Operation ▸ More** from the context menu. The **Select Operation** dialog is displayed. Expand the JFrame node, scroll through the list, and select `setDefaultCloseOperation(int):void.` Click **OK**. The message label becomes: `1.2:setDefaultCloseOperation(int):void //Message Link2`
4. In this step, we specify the parameters used on invoking an operation in the source code. These arguments can be entered in the **arguments** field in the Properties View of a message link.

   In the **arguments** field of the Properties View of the message 1.2, enter `JFrame.EXIT_ON_CLOSE`. The message label becomes: `1.2: setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE):void // Message Link2`
5. Draw Message Link 1.3. Similar to Step 3, use the **Select Operation** dialog (**Select Operation ▸ More**), expand the **Component** node, and select the `setSize(int,int):void` method. In the **arguments** field of the Properties View enter `600,400`. The message label becomes: `1.3: setSize(600,400):void // Message Link3`
6. Draw Message Link 1.4. Use the **Select Operation** dialog, expand the **Component** node, and select the `setLocation(int,int):void` method. Invoke the in-place editor for the message, and enter `(50*i, 50*i)` after the closing curly brace. The message label becomes: `1.4: setLocation(50*i, 50*i):void //Message Link4`
7. Draw Message Link 1.5. Using the **Select Operation** dialog, expand the **Window** node, and select the `show()` method from the list. The message becomes: `1.5:show():void //Message Link5`

To generate implementation code for sequence diagram

1. Right-click the background of the sequence diagram, and choose **Generate Implementation** from the context menu. The first page of the Sequence diagram refactoring wizard is displayed. Warning messages display, if applicable. Click **Next** to display the second page of the wizard.
2. The second page of the wizard displays changes that are necessary to perform refactoring. Notice the sections on the page highlighting the original source code to be replaced and the refactored source code to replace it. Click **Finish**.

   Open the implementation code of the Class1 in the Editor. Message labels, for which implementation code has been generated, are displayed in bold on the diagram. If code generation fails for certain messages, those messages are not displayed in bold.
3. On the sequence diagram, double-click one of the message links displayed in bold, and observe that the Editor scrolls to the point of appropriate method invocation.
4. In the Editor, add an import statement for `javax.swing`.

Related Reference

UML 1.4 Interaction Diagrams

Using AutoFix

Together automatically synchronizes changes between sequence/collaboration diagrams and class diagrams. Most problems on the sequence/collaboration diagrams will be highlighted on the diagram in red.

Red highlighting may not show every problem with a sequence/collaboration diagram. The AutoFix command runs a more comprehensive check to identify problems that may not be highlighted. Use the AutoFix command to resolve such issues.

The AutoFix command provides a dialog that describes any problems found. Select the problem to correct, and use the options listed in the "Available Solutions" list to fix the problem.

To use the AutoFix command

1. Right-click the background of a sequence or collaboration diagram, and select **AutoFix**. The **Solve Problems** dialog box is displayed.
2. A description of the problem is displayed in the list on the left. Select the problem from the list. The **Solution Description** field and **Available Solutions** field display appropriate descriptions and solutions.
3. Select an available solution from the list. For example, choose **Select Method** from the list.
4. Click **Select Method** to continue. The **Select Operation** dialog box is displayed.
5. Select an operation to link, and click **OK** to close the dialog.
6. Under **Solution Actions**, click **Accept**.
7. Click **Finish**.

Using AutoLink Labels

The AutoLink Labels command displays a dialog if there are operations on the diagram that were previously saved and unlinked using the Autofix dialog.

For example, if you have a message linked to an operation in a class, but you decide to delete the operation from the class, the sequence/collaboration diagram link becomes highlighted in red.

To correct this problem

1. Right-click the diagram background, and select **AutoFix**. The Solve Problems dialog is displayed.
2. In the dialog, select **Unlink and Save Text**.
3. Click **Accept Solutions**, and click **Finish**. Operations that are saved and unlinked are no longer highlighted in red on the sequence/collaboration diagram. They are enclosed in single quotes.

Later, you may decide to relink the message to a new operation.

To relink a message to a new operation

1. Right-click the diagram background, and select **AutoLink Labels**. The Convert Labels to Operations dialog is displayed.
2. Select the appropriate message to link from the list on the left.
3. Choose one of the actions from the Available Solutions list. For example, choose "Select Method" from the list.
4. Under Solution Actions, click **Select Method**. The Select Operation dialog is displayed.
5. Select an operation, and click **OK**.
6. Click **Finish**. The diagram is updated linking the message to the selected operation.

Related Procedures

Using AutoFix

Working with a UML 1.4 Message

This section describes techniques for working with messages in Sequence and Collaboration diagrams. Although the two diagram types are equivalent, the techniques for dealing with messages differ.

In a Collaboration diagram, all messages between the two objects are displayed as a generic link line, and a list of messages is created above it. The link line is present as long as there is at least one message between the objects. Messages display in time-ordered sequence from top to bottom of the messages list. In addition to the message links, you can add links that show association and aggregation relationships. These links do not display if you view the diagram as a sequence diagram.

When you draw messages between objects in a sequence diagram, each message is represented by its own link line. Messages in sequence diagrams have more editable properties than messages in collaboration diagrams.

In this section you will learn how to

1. Create a self message
2. Reorder a message link
3. Specify the creation of an object with a message
4. Specify the destruction of an object with a message
5. Specify a return link by using the Tool Palette (Toolbox)
6. Specify a return link by using the Object Inspector (Properties Window)

To create a self message

1. Click the **Self Message** button on the Palette.
2. For a Sequence diagram, click the lifeline of the object at the point where you want the message to appear. Clicking the object places the message-to-self first on the lifeline.

   For a Collaboration diagram, click the object.

To reorder a message link

1. Open a diagram.
2. To reorder messages, perform one of the following actions:

   - Drag message links up and down the object lifeline in the Diagram Editor . Reordering automatically updates the message link numbers.
   - Change the **Sequence Number** field in the Properties View.
   - In the Diagram Editor , use the in-place editor to change the sequence number.

To specify the creation of an object with a message

1. Select a message link in the Sequence diagram.
2. In the Properties View of the message link, click the **Creation** field.
3. Choose True from the list box.

The message link points to the recipient object icon rather than to its lifeline. The created object moves downward along the lifeline to show that it exists at a point later in time from its creator.

By default, the **Creation** property is set to False in the Properties Window.

To specify the destruction of an object with a message

1. Select a message link in the Sequence diagram.
2. In the Properties View of the message link, click the Destruction field.
3. Choose True from the list box.

The object is destroyed.

By default, the **Destruction** property is set to False in the Properties View.

To specify a return link by using the Palette

1. Click the **Return link** button in the Palette.
2. On the sequence diagram, click the object lifeline element at the supplier end of the message link to draw the return link.

To specify a return link by using the Properties View

1. Select the message link on the sequence diagram.
2. In the Properties View, click the drop-down arrow for the **Return Arrow** field and select True.

Related Procedures

Rerouting a Link

Related Reference

UML 1.4 Interaction Diagrams

Working with Classes in Sequence/Collaboration Diagrams

This section provides techniques for creating classes, linking and unlinking classes, and showing classes in sequence or collaboration diagrams.

To create a new class or interface

1. Select an actor or object on the sequence diagram.
2. Right-click the element and choose **New** ▸ **Class** or **New** ▸ **Interface**. The **New Object's Class** dialog is displayed.
3. Enter information required by the dialog to create a new class, and then click **Finish**.

To link an actor or object to a class

1. Select the actor or object on the sequence diagram.

   Tip: You can associate multiple objects with the same class. Use CTRL + CLICK to select the elements.

2. Choose **Select Class** on the context menu. The Select Class command expands to display a submenu that shows any classes that are "local" to the diagram and the **More** option to reveal inherited operations of the recipient class.
3. Select a class from the list, or **More**. This renames the actor or object to the chosen class name.
4. If you choose to associate a classifier to an object that is already instantiated with a different classifier, the **Solve Problems** dialog appears. Use this dialog to decide what to do with the linked operations that do not exist in the new classifier (options include unlink operation, save as text, and select another method).

To unlink a Class/Interface from an Actor or Object

1. Select the actor or object on the sequence diagram.
2. Choose **Unlink Class** from the context menu.

To show a class associated with an Actor or Object

1. Select the actor or object on the sequence diagram.
2. Choose **Show Class** from the context menu.
3. On the submenu, choose a view (for example, Model Navigator).

Note: To show the source code of a class in the editor, double-click the element in the sequence diagram. The source code is displayed highlighting the class name.

Working with Operations in Sequence/Collaboration Diagrams

This section provides techniques for creating, linking and showing operations while working with sequence or collaboration diagrams.

Use the context menu of the link to create a new operation or constructor. Double-clicking the message link displays the source code. This option is disabled if the object is associated with a read-only class.

After you create the operation, you can double-click the message link label on the diagram and enable the in-place editor to modify the operation; or as an alternative, you can modify the operation using the Properties view.

Tip: The maximum length of a message link label is 400 pixels. If your message label is longer than this, it displays on the diagram with an ellipse at the end to indicate that the message link holds more information. You can click the message link label to reveal the entire message.

To create a new operation or constructor

1. Create a message link between two objects. The supplier object must be associated with a class.
2. Right-click the link and choose **New ▸ Operation** or **New ▸ Constructor**. When creating a new operation, a dialog is displayed where you can designate the name, modifiers, return type, and parameters for the new operation. At a minimum, enter the name for the operation, and click Finish. Together uses the data entered in the dialog to create the operation in the class and update the link properties with the new operation.

Tip: Creating a new constructor on the message link sets the link type to `creation`.

To create a message link that calls an operation

1. Create a message link between two objects. The supplier object must be associated with a class.
2. Select the message link on a sequence diagram.
3. Right-click the link and choose **Select Operation** from the context menu. The Select Operation command expands to display a submenu that shows any operations that are "local" to the diagram and the More option for browsing operations that you can associate with the object.
4. Choose an operation from the list, or select More to associate an operation that is not local to the diagram. This renames the message link to the chosen operation's name.

After the operation has been associated with a message link, double-click the message link to display the source code of this operation in the editor.

Note: After a message link has been associated with an operation, you can rename the operation directly on the class diagram or in the source code, and the sequence/collaboration diagram displays the new name.

To unlink an operation from a message link

1. Select the link on the sequence diagram.
2. Right-click the link, and select **Unlink Operation** from the context menu.

To show an operation

1. Right-click the message link.

2 Select **Show Operation** ▸ **In Model Navigator**.

UML 1.4 Statechart Diagrams Procedures

This section outlines the procedures related to UML 1.4 Statechart diagrams.

In This Section

Choosing a Target Class for the State Diagram or Activity Diagram

Use the Properties View of a state or activity diagram to designate a target class.

To choose a target class for a diagram

1. Create a state or activity diagram within a package.
2. In the Properties View of the diagram, select the **context** field, and click the Browse button to open the **Select Class for 'context' Property** dialog.
3. In the **Select Class for 'context' Property** dialog, navigate to the target class for the diagram using the tree-view provided in the **Model Elements** tab.
4. Double-click the target class, or click **OK**.

Creating a Deferred Event

You can add a deferred event to a state or activity element.

To create a deferred event

1. Select the desired state or activity element in the diagram or in the Model Navigator.
2. Right-click the element, and select **New** ▶ **Deferred Event** on the context menu.

Related Reference

Deferred Event
UML 1.4 Activity Diagrams
UML 1.4 Statechart Diagrams

Creating a Self-Transition

To create a self-transition

1. Draw a transition from the state or activity element and drag the link away from the element.
2. Drag the link back to the element and drop it.

Alternatively:

1. Draw a transition between two activities (or states).
2. Drag the opposite end of the link line back to the desired activity (or state).

Related Procedures

Creating a Simple Link

Related Reference

UML 1.4 Activity Diagrams
UML 1.4 Statechart Diagrams
Tool Palette

Creating History

To create history for a state

1. Right-click on the state element.
2. From the context menu, choose **New > History**.
3. **Tip:** Alternatively, choose the history button in the Tool Palette and click on the target state.

Related Procedures

Setting Deep History

Creating internal transitions

An internal transition is a shorthand for handling events without leaving a state or activity and dispatching its exit/entry actions.

To create an internal transition

1. Select the desired state or activity on the diagram.
2. From the context menu of the selection, choose **New ▸ Internal Transition**.

Creating Multiple Transitions

A Transition can have multiple sources (it is a join from several concurrent states) or it can have multiple targets (it is a fork to several concurrent States).

You can show multiple transitions with either a vertical or horizontal orientation in your state and activity diagrams. Both the state and activity tools palette provide separate horizontal and vertical fork/join buttons for each orientation. The two orientations are semantically identical.

To create multiple transitions

1. Identify the nodes involved. If necessary, place all of the states on the diagram first, and lay them out as you want.
2. Place either a horizontal or vertical fork/join on the diagram. Resize as needed.
3. If depicting multiple sources, draw transitions from each of the source nodes to the fork/join.
4. If depicting multiple targets, draw a transition from the source node to the fork/join; next, draw transitions from the fork/join to each of the target nodes.

Setting Deep History

To designate deep history for a history element

1. Right-click the history element.
2. From the context menu, choose **Properties**.
3. Set the **deep** field to "true".

Specifying Entry and Exit Actions

You can create entry and exit actions for the states and activities as stereotyped **internal transitions**.

To specify entry and exit actions using the in-place editor

1. Create an internal transition in a state or activity.
2. Double-click the internal transition to enable in-place editing.
3. Rename the internal transition using the following syntax:

   ```
   stereotype/actionName(argument)
   ```

   For example:

   ```
   exit/setState(idle)
   ```

To specify entry and exit actions using the Properties View

1. Create the internal transition in a state or activity.
2. Make sure that a context is defined for the diagram, and the target class has at least one operation.
3. In the Properties View of the internal transition, specify the following properties:
   - **action expression:** Enter the expression in the text area, or click the **Browse** button and in the **Select Operation for the 'action expression' property** dialog, select an operation from the list operations in the target class of the diagram.
   - **name:** Specify the name of the event.
   - **event arguments:** Enter one or arguments. Use a comma as a delimiter.

Related Procedures

Choosing a Target Class for the State Diagram or Activity Diagram

Specifying entry/exit actions for a state

Entry and exit actions are executed upon entering or leaving a state, respectively. You can create entry and exit actions in Together state diagrams as stereotyped internal transitions. Use one of the following methods to specify entry/exit actions for a state.

Using the in-place editor:

1. Create an internal transition in the appropriate state.
2. Double-click the internal transition to enable in-place editing.
3. Rename the transition using the following syntax:

```
Name(event_arguments)[guard_condition]/action_expression^send_clause
```

Using the Properties view:

1. Right-click on the internal transition.
2. From the context menu, select **Properties**.
3. Set the event name, event arguments, and action expression properties in the Properties view.

Working with a Complex State

The techniques in this section apply to models of particularly complex composite states. These procedures are common for the State and Activity diagrams.

Create a composite state by nesting one or more levels of states within one state and draw transitions among the nested elements. You can place the following elements in a state:

- activity
- signal sending
- signal receipt
- start/end states
- history

Tip: You can nest multiple levels of states inside one state. For especially complex state modeling, however, you may find it more convenient to create different diagrams, model each of the state levels individually, and then hyperlink the diagrams sequentially.

Use the following techniques to create a composite (nested) state

1. Create a nested state using drag-and-drop.
2. Create a nested state using the context menu of the state element.

To create a nested state using drag-and-drop

1. Place a state element on the diagram background.
2. Drag a new state on top of an existing state.
3. Drop a new state.

To create a nested state using the context menu of the state element

1. Right-click the state (region) that will be the container.
2. Select **New** ▸ **State** on the context menu.

Tip: Using the **Shortcuts** command on the context menu of the diagram, you can reuse existing elements from the other state diagrams. Right-click the diagram and choose **New**▸**Shortcuts**, navigate within the pane containing the tree view of the available project contents to the existing diagram, and select its elements, states, histories, forks, and/or joins.

Related Concepts

Model Hyperlinking Overview

Related Procedures

Creating a Shortcut

Related Reference

UML 1.4 Activity Diagrams
UML 1.4 Statechart Diagrams
UML 2.0 State Machine Diagrams

UML 1.4 Activity Diagrams Procedures

In This Section

Choosing a Target Class for the State Diagram or Activity Diagram

Use the Properties View of a state or activity diagram to designate a target class.

To choose a target class for a diagram

1. Create a state or activity diagram within a package.
2. In the Properties View of the diagram, select the **context** field, and click the Browse button to open the **Select Class for 'context' Property** dialog.
3. In the **Select Class for 'context' Property** dialog, navigate to the target class for the diagram using the tree-view provided in the **Model Elements** tab.
4. Double-click the target class, or click **OK**.

Creating a Deferred Event

You can add a deferred event to a state or activity element.

To create a deferred event

1. Select the desired state or activity element in the diagram or in the Model Navigator.
2. Right-click the element, and select **New ▸ Deferred Event** on the context menu.

Related Reference

Deferred Event
UML 1.4 Activity Diagrams
UML 1.4 Statechart Diagrams

Creating a Self-Transition

To create a self-transition

1. Draw a transition from the state or activity element and drag the link away from the element.
2. Drag the link back to the element and drop it.

Alternatively:

1. Draw a transition between two activities (or states).
2. Drag the opposite end of the link line back to the desired activity (or state).

Related Procedures

Creating a Simple Link

Related Reference

UML 1.4 Activity Diagrams
UML 1.4 Statechart Diagrams
Tool Palette

Creating an Activity for a State

To create an activity for a state

1. Open the **Diagram View**.
2. Right-click a state and choose **Add ▸ Activity** on the context menu.

A new activity is created inside of a state.

Related Reference

UML 1.4 Activity Diagrams

Designing a UML 1.4 Activity Diagram

Use the following tips and techniques when you design a UML 1.4 Activity Diagram.

To design a UML 1.4 Activity Diagram

1. Create one or more swimlanes. You can place several swimlanes on a single diagram or create a separate diagram for each.

   Warning: You cannot create nested swimlanes.

2. Create one or more activities. You can place several activities on a single swimlane or create a separate swimlane for each.

   Warning: You cannot create nested activities.

3. For convenient browsing, first model the main flow. Next, cover branching, concurrent flows, and object flows.

   Tip: Use separate diagrams as needed and then hyperlink them.

4. Create **Start**, **End**, **Signal Receipt**, and **Signal Sending** elements for your swimlanes.
   If your activity has several **Start** points, they can be used simultaneously.
5. Create object nodes. Do not link object nodes to classes on your Class Diagrams. However, you can use hyperlinks for better understanding of your diagrams.
6. Create state nodes for your swimlanes.

   Tip: You can create nested states.

7. Optionally, create a **History** node.
8. Connect nodes by links.
9. You can optionally create shortcuts to related elements of other diagrams.

Related Procedures

Creating a Shortcut

Related Reference

UML 1.4 Activity Diagrams

Specifying Entry and Exit Actions

You can create entry and exit actions for the states and activities as stereotyped **internal transitions**.

To specify entry and exit actions using the in-place editor

1. Create an internal transition in a state or activity.
2. Double-click the internal transition to enable in-place editing.
3. Rename the internal transition using the following syntax:

   ```
   stereotype/actionName(argument)
   ```

   For example:

   ```
   exit/setState(idle)
   ```

To specify entry and exit actions using the Properties View

1. Create the internal transition in a state or activity.
2. Make sure that a context is defined for the diagram, and the target class has at least one operation.
3. In the Properties View of the internal transition, specify the following properties:

   - **action expression:** Enter the expression in the text area, or click the **Browse** button and in the **Select Operation for the 'action expression' property** dialog, select an operation from the list operations in the target class of the diagram.
   - **name:** Specify the name of the event.
   - **event arguments:** Enter one or arguments. Use a comma as a delimiter.

Related Procedures

Choosing a Target Class for the State Diagram or Activity Diagram

Using Object Flow Link

An object flow relationship can be drawn:

- from an Activity to an Object
- from a SignalSending element to an Object
- from an Object to a SignalReceipt element
- from/to an Object
- to/from a Fork/Join

To create an object flow link between two elements

1. On the Diagram, click the source element.
2. Drag the link to the destination element.
3. Drop when the second element is highlighted.

Related Procedures

Using Control Flow Link

Working with a Complex State

The techniques in this section apply to models of particularly complex composite states. These procedures are common for the State and Activity diagrams.

Create a composite state by nesting one or more levels of states within one state and draw transitions among the nested elements. You can place the following elements in a state:

- activity
- signal sending
- signal receipt
- start/end states
- history

Tip: You can nest multiple levels of states inside one state. For especially complex state modeling, however, you may find it more convenient to create different diagrams, model each of the state levels individually, and then hyperlink the diagrams sequentially.

Use the following techniques to create a composite (nested) state

1. Create a nested state using drag-and-drop.
2. Create a nested state using the context menu of the state element.

To create a nested state using drag-and-drop

1. Place a state element on the diagram background.
2. Drag a new state on top of an existing state.
3. Drop a new state.

To create a nested state using the context menu of the state element

1. Right-click the state (region) that will be the container.
2. Select **New** ▸ **State** on the context menu.

Tip: Using the **Shortcuts** command on the context menu of the diagram, you can reuse existing elements from the other state diagrams. Right-click the diagram and choose **New**▸**Shortcuts**, navigate within the pane containing the tree view of the available project contents to the existing diagram, and select its elements, states, histories, forks, and/or joins.

Related Concepts

Model Hyperlinking Overview

Related Procedures

Creating a Shortcut

Related Reference

UML 1.4 Activity Diagrams
UML 1.4 Statechart Diagrams
UML 2.0 State Machine Diagrams

UML 1.4 Component Diagrams Procedures

In This Section

Designing a UML 1.4 Component Diagram

The following tips and techniques can be used when working with UML 1.4 Component Diagrams. It can be convenient to start the creation of a model with Component Diagrams if you are modeling a large system (for example, a distributed, client-server software system, with numerous interconnected modules). Use Component Diagrams for modeling a logical structure of your system, and use Deployment Diagrams for modeling a physical structure.

To design a UML 1.4 Component Diagram

1. Create a hierarchy of Subsystems.
2. Create a hierarchy of Components. The largest component can be the whole system or its major part (for example, *server application*, *IDE*, *service*).
3. Create interfaces. Each component can have an interface.
4. Draw links between elements.
5. You can optionally create shortcuts to related elements of other diagrams.

Related Procedures

Creating a Shortcut

Related Reference

UML 1.4 Component Diagrams

Nesting Components

A component represents a modular and replaceable part of the system that complies to an interface. Examples of components include class libraries or binary programs. A component is used to package other logical elements, and represents things that participate in the execution of a system. Components also use the services of another component via one of its interfaces. Usually, components are used to visualize logical packages of source code (work product components), binary code (deployment components) or executable files (executions components).

To nest a component

1. Place a component element on the diagram background.
2. Drag the new component on top of an existing subsystem or existing component.
3. Drop the new component.

Related Concepts

UML 1.4 Component Diagram Definition

UML 1.4 Deployment Diagrams Procedures

In This Section

Designing a UML 1.4 Deployment Diagram

Use the following tips and techniques when you design a UML 1.4 Deployment Diagram. It can be convenient to start the creation of a model with Deployment Diagrams if you are modeling a large system that is comprised of multiple modules, especially if these modules reside on different computers. Use Deployment Diagrams for modeling a physical structure of your system, and use Component Diagrams for modeling a logical structure.

To design a UML 1.4 Deployment Diagram

1. Create a hierarchy of Nodes.

   Tip: You can create nested Nodes.

2. Create a hierarchy of Components. The largest component can be the whole system or its major part (for example, *server application*, *IDE*, *service*).

   Tip: You can create nested Components. There are two methods for creating a nested component:

   You can select an existing component and add a child component inside.

   Alternatively, you can create two separate components and connect them with an Association-Composition link.

3. Represent how Components reside on Nodes. You can represent this in two ways:
   - ♦ Use a supports link between the component and node. The supports link is a dependency link with the stereotype field set to support.
   - ♦ Graphically nest the Component within the Node.
4. Optionally, create Objects.
5. Create Interfaces. Each component can have an interface.
6. Indicate a temporary relationship between a Component and Node. Objects and components can migrate from one component instance to another component instance, and respectively from one node instance to another node instance. In such a case, the object (component) will be on its component (node) only temporarily. To indicate this, use the dependency relationship with a becomes stereotype.
7. You can optionally create shortcuts to related elements of other diagrams.

Related Procedures

Creating a Shortcut

Related Reference

UML 1.4 Deployment Diagrams

Together Business Process Modeling

This section provides how-to information on creating Business Process models with Together.

In This Section

Attaching External WSDL File

You can attach an external WSDL file to any participant on your business process diagram using the wsdl path property with chooser.

Note: Attaching an external WSDL file to the default pool or to all pools on a diagram will prevent the export process.

To attach a WSDL file to the participant of your process

1. Place your WSDL file inside the current project node (Eclipse will see external files after a Refresh is performed in the Navigator view).
2. Select the WSDL path property of the participant in the Properties View and click the **Edit** button.
3. Select the WSDL file in the **WSDL path** dialog box.
4. Click OK to save the changes.

Only valid files can be attached. If the program does not let you add the selected WSDL file, the file is not valid according to WSDL or BPMN specifications and cannot be used for correct export. After the file is added, it will be used for the BPEL mapping with the selected WSDL file for the process (no additional WSDL file will be generated in this case).

The `TargetNamespace` and `NamespacePrefix` properties are updated automatically (and the fields become read-only). Web Service Interfaces and Operations are available in the list boxes for the appropriate fields of the WebService element inside the Properties view.

When the WSDL file is removed from the WSDL path property of the process, previously entered values are restored and the native WSDL file can be generated for the BPEL file again.

Related Concepts

Business Process Modeling

Creating a BPMN Project

BPMN projects are created in Together with the help of the New Project wizard.

To create a BPMN project

1. Select **File ▶ New ▶ Project** on the main menu. The New Project wizard is displayed.
2. Expand the Modeling node in the tree view list, and select **Business Process Modeling Project**. Click **Next**.
3. Specify a name for a new BPMN project and the project location. Click **Next**.
4. Specify whether to create a BPMN diagram and specify the diagram's name. Click **Next**.
5. Select one or more profiles you want to enable for the created BPMN project and click **Next**.
6. Select referenced projects and click **Finish** to complete the wizard.

Note: To create a BPMN project with default parameters, click **Finish** after specifying the name of the project.

Related Concepts

Business Process Modeling

Related Reference

Business Process Diagram

Exporting to BPEL/WSDL Files

A Business Process diagram with an enabled BPEL profile can be exported to BPEL (Business Process Execution Language) for Web services for further deployment.

To export a Business Process diagram to BPEL/WSDL files

1. Open a Business Process Modeling project with the diagram you want to export.
2. Select **File** ▸ **Export** on the main menu.
3. Select **BPEL4WS File** under the **Modeling** node.
4. Select the diagram you want to export and the path to the export directory in the **Export to BPEL/WSDL** dialog box.
5. Check **Open file in Active BPEL Designer** if you want to open the generated BPEL file in the new view as the Active BPEL Designer file.

   Note: You can export BPEL/WSDL files to the current workspace project when Active BPEL Designer is already installed.

6. Click **Finish** to complete the procedure.

If the process is successful, your BPEL and WSDL files are created in the specified directory.

Note: Before a business process diagram is exported to BPEL/WSDL files, a diagram validation is performed.

Warning: If you want to open the export result in Active BPEL Designer, make sure your export result is located within one project opened in the workspace.

After you export your project to BPEL4WS, you can use Active BPEL Designer to work with BPEL files.

To install Active BPEL Designer in Together

1. Download and install Active BPEL Designer. See related links for the download location.
2. Copy plug-ins with names that start with **com.activee** from the `Designer installation\eclipse\plugins` folder to the `Together installation\plugins` folder.
3. Run Together with the `-cleanup` command line argument.

Related Concepts

Business Process Modeling

Related Procedures

Validating BPMN Diagrams

Importing BPEL File

BPEL files are imported to a Together BPMN project. A new BPMN diagram is created to represent the imported business process.

To import BPEL file

1. Select **File** ▸ **Import** on the main menu. The **Import** wizard is displayed.
2. Expand the **Modeling** node in the tree view list, and select **BPEL Import**. Click **Next**.
3. Select the BPEL file you want to import or click **Browse** to locate it.
4. Click **Add** or **Add folder** to add one or more WSDL files.
5. Specify a new diagram name in the **Diagram name** text box and select a BPMN project in which the new diagram will be created. Click **Finish** to import the selected BPEL file.

Related Concepts

Business Process Modeling

Importing BPMN Projects Created in Together 2006 for Eclipse

BPMN projects created in Together 2006 for Eclipse are not compatible with Together. You must perform a conversion before working with BPMN projects from Together 2006 for Eclipse.

To import a BPMN project created in Together 2006 for Eclipse

1. Switch to the workspace with old BPMN projects or import the old BPMN projects to the current workspace using the standard Eclipse tools.

   Note: After you can see your old BPMN projects in the Model Navigator, you can open the projects but you cannot modify the projects.

2. Select **File** ▸ **New** ▸ **Project** on the main menu. The **New Project** wizard is displayed.
3. Expand the Together node in the tree view list, and select **BPMN from Together 2006 Business Process Project**. Click **Next**.
4. Specify a name for a new BPMN project and the project location. Click **Next**.
5. Select the project you want to convert and click **Next**.

Related Concepts

Business Process Modeling

Performing Business Process Simulation

Together enables you to perform a simulated run of the designed business process. Simulation parameters are specified in the run configuration.

Note: Together automatically validates the diagram before performing a simulation.

To perform a simulated run of the business process

1. From the main menu, choose **Run** ▸ **Run**. The **Run** dialog box opens.
2. Click **Launch BPMN Simulation** and click to create a new launch configuration.
3. Specify run options and click **Run**.

   Alternatively, right-click the BPMN diagram background and select .

   Note: The **Simulate** command is available only for BPMN projects with enabled BPMN Simulation profile. When simulation is performed using the **Simulate** command on the context menu, simulation is run with default parameters.

To perform the step by step simulation

1. From the main menu, choose **Run** ▸ **Run**. The **Run** dialog box opens.
2. Click **Launch BPMN Simulation** and click to create a new launch configuration.
3. When specifying run options, select **Start with step by step execution**. Click **Run** to start the simulation process.

   The simulation stops after executing one step. To proceed to the next step, click **Next simulation step** in the **BPMN Simulation** view.

Related Procedures

Validating BPMN Diagrams
Creating a BPMN Project
Specifying BPMN Preferences

Related Reference

Launch BPMN Simulation
BPMN Validation View

Specifying BPMN Preferences

To set Business Process Modeling preferences

1 From the main menu, choose **Window** ▸ **Preferences**. The **Preferences** dialog box opens.
2 Expand the **Modeling** node and click **Business Process**.

To assign the default profile for BPMN diagram

1 From the main menu, choose **Window** ▸ **Preferences**. The **Preferences** dialog box opens.
2 Expand the **Modeling** node and click **Profiles**.
3 Click the **BPMN** tab and select the profiles you want to be enabled for a newly created BPMN project.

Related Reference

Business Process Preferences

Specifying Event and Trigger Type

To change the event type

1 Select an event.
2 In the Properties View, select the , **intermediate**, or **end** value for the **type** property.

> **Note:** By default, events are created with **start** type.

When the program detects an incorrect element type, it highlights the element.

To automatically correct the element type

1 Right-click the highlighted element.
2 Select **Fix Element Type** on the context menu.

To specify the trigger type

1 Select an event.
2 In the Properties View, select the **trigger type** property.
3 Select the trigger type from the list in the **Value** column.

> **Note:** There are ten triggers: None, Cancel, Compensation, Error, Link, Message, Multiple, Rule, Terminate, and Timer. There are also some constraints for event types and triggers (for example, a Compensation event cannot be a start event).

Related Concepts

Business Process Modeling

Related Procedures

Together Business Process Modeling

Using BPMN Layout Features

To align diagram elements using the diagram editor toolbar

1. Select one or more diagram elements, and click the drop-down arrow to the right of the Align Left button.
2. Choose one of the options to align the elements.

Note: The **Layout all** command performs the following:

- All pools are aligned and distributed with the constant distance between them.
- Sequence Flow links are aligned horizontally and directed from left to right.

Related Concepts

Business Process Modeling

Validating BPMN Diagrams

You can validate your BPMN diagram to check BPMN general rules, constraints to be met to make BPEL export possible, and prerequisites for simulation. Validation is performed for the entire diagram.

You can perform validation with or without export-specific errors and warnings.

Note: Validation is profile-sensitive. For example, there will be no validation for simulation action if simulation profile is turned off (default).

To validate a BPMN diagram for specification compliance

1. Open a Business Process Modeling project with the diagram you want to validate.
2. Right-click the diagram background and select **Validate BPMN diagram**.

To validate a BPMN diagram for BPEL4WS export

1. Open a Business Process Modeling diagram you want to validate.
2. Right-click the diagram background and select **Validate for BPEL4WS export**.

To navigate to an element that contains an error

1. Right-click an item in the **BPMN Validation** view.
2. Choose either **Select in Model Navigator** or choose **Select on Diagram**.

Note: Alternatively, double-click an item in the **BPMN Validation** view to select an element on the diagram.

To validate a BPMN diagram for simulation

1. Open a Business Process Modeling diagram you want to validate.
2. Right-click the diagram background and select **Validate for Simulation**.

Any errors that occurred during the validation for simulation are displayed in the same **BPMN Validation view**.

Note: Warnings in the **BPMN Validation view** denote some minor errors that should be corrected. Some warnings are provided for information only and do not stop the generation process (for example, add/remove links, type update, and properties update result in warnings). Errors in the BPMN Validation view imply that generation cannot be performed without correction. For example, errors can occur when unsupported elements and incorrect symbols are used in some names.

Related Reference

BPMN Validation View

Working with Groups

A Group on a BPMN diagram is a logical element that helps you to visualize the division of a BPMN diagram into logical parts.

To create a Group element on a BPMN diagram

1. Scroll to the **Artifacts** group on the Palette.
2. Click the **Group** element and then click the diagram. A new group is created inside the BusinessProcessDiagram element.

   Note: Alternatively, use the context menu to create a group. Right-click the diagram background and select **New** ▸ **Group**.

To add elements to or remove elements from the Group

1. Select the group that you want to edit.
2. Click the **Elements** tab in the **Properties** view.
3. Click the **grouped elements** property and click ⋯ in the **Value** column. The selection dialog box opens.

   Alternatively, right-click a group boundary on the diagram and select **Group Elements** ▸ **Edit**.
4. To add the elements, select the model elements in the left column and click **Add**.
5. To remove elements from the group, select the elements in the right column and click **Remove**. Alternatively, click **Remove All** to remove all elements from the group.
6. Click **OK** to save the changes and close the dialog box.

   Note: You also can add elements to the group by just dragging and dropping an element to a Group border.

To distinguish more clearly between different groups on a diagram, you can change the view of the group.

To change group colors

1. Select the group whose view you want to change.
2. Select the **View** tab in the **Properties** view.
3. To change the color of group elements, select the **background color** option, click ⋯ and select the color you want to use.
4. To change the color of the group boundary, select the **foreground color** option, click ⋯ and select the color you want to use.

To change the group title style

1. Select the group you want to edit the style for.
2. Select the **View** tab in the **Properties** view.
3. Select the **font** option, click ⋯ and select the font and size you want to use.

To select group members and navigate between a group and the group members

1. To select all members of a group, right-click the group and choose **Select Group Content**.
2. To navigate to an element in a group, right-click the group border, choose **Grouped Elements**, and then choose the element you want to select in the editor.
3. To navigate from an element to a group, right-click the element, choose **Groups**, and then choose the group you want to select.

Related Concepts

Business Process Modeling

Working with Projection Bars

Projection bars on the Business Process Modeling diagram provide placeholders for pools and lanes on the diagram and remain visible even if the pools are too long and you have to scroll through the diagram. To select a pool or lane on the diagram, click a pool or lane element in a projection bar.

To show or hide projection bars

1. From the main menu choose **Window** ▶ **Preferences**.
2. Expand the **Modeling** node and click **Diagram** in the left pane.
3. Check or clear the **Show projection bars** option in the **Other** group.

   Alternatively, right-click either the projection bars or a ruler and click the **Show projection bars** option.

Related Concepts

Business Process Modeling

Working With UML Links in a BPMN Project

Together provides UML links for a BPMN project with an enabled **UML Links** profile. The UML links feature helps you to use Class/Interface elements and their methods as values for the WSDL portType and operations. UML links are available for participant and service-related tasks (Send, Receive, User, Service).

To connect and remove a UML service

1. Locate a Participant of a process or a service-related task you want to work with in the Model Navigator.
2. Expand the node and right-click a WebService element. To create a Web Service, right-click an element and select **New** ▸ **Web Service**.

   Note: When the UML Links profile is enabled, you can see a **UML Services** menu with submenus.

3. To connect a UML service, click **UML Services** ▸ **Connect UML Service**.

   To remove a connection, click **UML Services** ▸ **Remove Link to Service**.

To locate a linked UML element

1. Locate a Participant of a process or a service-related task of the diagram you want to work with in the Model Navigator.
2. Expand the node and right-click a WebService element.

   Note: When the UML Links profile is enabled, you can see a **UML Services** menu with submenus.

3. To locate a UML element on a diagram, click **UML Services** ▸ **Select Service on Diagram**.

   To locate a UML element in the Model Navigator, click **UML Services** ▸ **Select Service in Model Tree**.

Note: When UML links exist, menu item names change and use UML element names instead of **Service**.

Related Concepts

Business Process Modeling

Data Modeling Procedures

This section describes how to work with ER diagrams and create logical and physical data models.

In This Section

Activating ER Logical Diagram Profile

This profile is available in UML 2.0 design projects. When this profile is activated, ER Logical Diagram elements are added to the Class Diagram Palette.

To activate ER Logical Diagram Profile

1. Select project in the Model Navigator or in the Navigator.
2. On the main menu, choose **Project** ▸ **Properties**.
3. On the **Profiles** page, check the ER Logical Diagram Profile check box.

Related Procedures

Creating a Project
Enabling UML Profiles
Data Modeling

Creating a Data Modeling Project

This section describes how to create a data modeling project for the development of a physical data model.

To create a data modeling project

1. On the main menu, choose **File ▸ New ▸ Project ▸ Modeling ▸ Data Modeling Project**.
2. On the first page of the **New Project** wizard, specify the project name and location. Click **Next**.
3. On the **Project Settings** page, select the target database server from the drop-down list. Check the **Default schema** option, if required, and specify the schema name. Click **Next**.
4. Follow the wizard to specify necessary options, and click **Finish**.
5. When you first create a data modeling project in a workspace, you will be prompted to associate the project with the Data Modeling perspective. You can confirm and memorize your decision.

Related Procedures

Creating a Project

Related Reference

New project Wizard Data Modeling Specific Options

Creating Connection Profile

Together supports a number of database servers. You can create connection profiles for each of the supported servers.

To create a new connection profile

1. On the main menu, choose **File Import**.
2. Expand the **Modeling** node, select **DB Schema from JDBC** and click **Next**.
3. In the **Import DB Schema from JDBC Connection** dialog, click **Connect**.
4. In the **Connect to Database** dialog that opens, select a database server, and click **New**.
5. Specify the connection parameters and click **Test**.
6. Click **Apply**.

Related Concepts

Data Modeling Overview

Related Procedures

Importing Data Model from Database

Related Reference

Connect to Database Dialog

Creating Foreign Key in a Physical Data Model

Warning: You cannot create a foreign key between tables from different schemata.

To create a foreign key

1. Create two tables in a schema diagram.
2. Draw a foreign key link between the child table and the parent table using the **Foreign Key** link button from the diagram Palette.
3. In the **foreign key** properties, locate the **parent key** field and select either **PK Constraint** or **Unique Constraint** from the parent table.

Note: Once the parent key property is specified, the **Propagate Attributes** context menu command becomes enabled. If not, make sure that at least one column is added to the Constraint selected as parent key. Use the **columns** field in Constraint Properties to add columns to the constraint. The foreign key columns propagated to the child tables display in red.

Related Concepts

Data Modeling Overview

Creating Logical Data Model

After your project is created and the ER Logical Diagram profile is activated, you can see the **ER Logical Diagram Elements** group on the diagram Palette.

Note that you can create only top-level ER logical elements (Entity, View, Subtype Cluster) and ER relationships using the Palette. For creating attributes, key groups, and so on, use the **New** context menu of their respective containers.

You can create ER Logical model elements the same way as any other diagram element.

To create a top-level element

1. Create a UML 2.0 project and activate the ER Logical Diagram Profile for it.
2. Create a new or open an existing Class20 diagram.
3. Click a button on the ER Logical Diagram Elements group in the Tools Palette and click the diagram background.

Note: The Properties View for ER elements contain an **ER** group where you can edit specific properties of the data modeling elements.

Related Concepts

Data Modeling Overview

Related Reference

ER Logical Diagram Elements
ER Physical Diagram Context Commands

Creating View Relationships in a Physical Data Model

To create a view relationship

1. In a schema diagram, create **Table** and **View** node elements.
2. Using the **View relationship** link button from the diagram Palette, draw a view relationship between a view and a table or between a view and another view.
3. Use the **Propagate Attributes** or **Propagate Attributes to All** context menu commands to propagate the view columns to a view or the entire view hierarchy.

Related Concepts

Data Modeling Overview

Generating Data Model from SQL (DDL) Script

After you have created a Data Modeling Project, you can reverse engineer a source SQL file from an existing file system to a database schema. The content of this file should contain a DDL statement (for example, `CREATE TABLE`). Every valid DDL statement will be translated into a corresponding physical data model object.

To import an SQL script

1. On the main menu, choose **File ▸ Import**.
2. In the **Import** wizard that opens, expand the **Modeling** node.
3. Select **DB schema from SQL script** and click **Next**.
4. In the **File** field, specify the path to the existing file. Select whether to open source SQL file in an SQL editor.
5. In the **Server** field, select the target database server. Its parser will be used to import the selected SQL file.
6. In the **Target project** field, specify the target Data Modeling project, where DB schema will be created.
7. Specify the target schema name.
8. Click **Finish** to start the import process. The new schema with the specified name is created in the selected project.

Related Concepts

Data Modeling Overview

Generating DDL Script from a Data Modeling Project

After you have designed a physical data model in your Data Modeling project, you can export it to a DDL statement and save in a `*.sql` or `*ddl` file. You can generate DDL only for one schema at a time. Each schema object is translated to a valid DDL statement with specific options depending on the current project RDBMS (database server).

To generate DDL script

1. In the Model Navigator view, select a schema that you want to export.
2. Select **File ▸ Export** on the main menu. The **Export** wizard is displayed.
3. Select the **DDL/SQL script** and click **Next**.
4. The Source Data Modeling project and schema that you selected in step 1 are selected by default. You can change the source schema by selecting one from the available schemata in all Data Modeling projects in your workspace.
5. The objects list contains tables and views in the selected schema. Select tables and views to be exported and click **Next**.
6. Select generation options that apply to the selected objects. Click **Next**. **Note**: You can preview the result by pressing the **Preview** button. This opens the **DDL Preview** dialog box with read-only contents.
7. Select a target file in which the generated script will be saved. You can make corrections in the result file after generation if necessary. Specify whether to open the resulting SQL file in the Eclipse SQL editor.
8. Click **Finish** to close the wizard and start the export process.

Tip: SQL keywords are highlighted in the editor. You can control the list of highlighted keywords by editing the `$TogetherArchitect_Home$/eclipse/plugins/com.borland.selena.dbmodeling_8.1.0/keywords.xml` file.

Related Concepts

Data Modeling Overview

Importing Data Model from Database

You can import a data model from a remote database using a JDBC connection. To reverse engineer a database, you need JDBC driver and JDBC connection parameters for your database.

To import a data model from a database

1. Choose **File** ▸ **Import** on the main menu. The **Import** wizard is displayed.
2. Expand the **Modeling** node and select **DB schema from JDBC** as the import source. Click **Next**.
3. Click **Connect** to establish a connection to the database. The **Connect to Database** dialog box displays.
4. In the Source tree view, select a connection profile. If necessary, create a new one.
5. Click **Connect**. If the connection is successful, you will see the source objects tree. Otherwise, an error message is displayed.
6. Select objects in the source objects tree to import them.
7. Specify the target Data Modeling project where the source object will be imported.
8. Click **Finish** to start the import process. New schemata are created in the selected project.

Related Concepts

Data Modeling Overview

Related Procedures

Creating Connection Profile

Transforming Logical Data Model to Physical Data Model

After you have created a Logical Data Model (with the help of ER Logical Diagram profile), you can convert it into a schema in an existing Data Modeling project.

To convert a Logical Data model to a schema

1. Select **File** ▸ **Import** on the main menu. The **Import** wizard is displayed.
2. Select **DB Schema from ER Logical Diagram Profile UML 2.0 project**. Click **Next**.
3. In the **Source** tree, select ER Logical Diagram elements. Select a valid UML project with ER Logical Diagram Profile.
4. Select the target Data Modeling project from the list where the schema will be created.
5. Select the target schema name (the name of the source file is used by default). Click **Next**.
6. Select transformation option.
7. Click **Finish** to start the import process. New schema with the specified name is created in the specified project.

Related Concepts

Data Modeling Overview

Model Driven Architecture

Topics in this section cover the most common tasks associated with developing model transformations.

In This Section

Opening MDA Views
How to open views related to **MDA** perspective.

Running a Composite Transformation script
How to run a Composite transformation script.

Running an Operational QVT
How to run an Operational QVT

Running Compiled Transformations
How to run compiled transformations.

Adding a New Ant Task to the Composite Transformation

To add a new Ant task to the Composite transformation

1. Open the composite transformation script in the **Ant Editor**, right-click in the editor area where you want to insert the task, and choose **Create MDA Ant Task** from the context menu.

   The **New MDA Ant Task** wizard is displayed.

   Note: If you create a new Composite transformation, click the **Add...** button on the **Composite Transformation Content** screen of the **New Composite Transformation** wizard.

2. Choose the **Enter data manually** option if you want to enter your task parameters manually.

   The **Select launch configuration type** wizard screen is displayed.

   Tip: Choose the **Select existing launch configuration** option to copy task parameters from an existing launch configuration to your task script.

3. Select the launch configuration type that matches your transformation and click the **Enter launch configuration data** button.

   The **Edit Launch Configuration** dialog box is displayed.

   Note: The **Edit Launch Configuration** dialog box is used here only to collect required task parameters, and no actual launch configuration is created.

4. Specify task parameters for the selected transformation type and click **OK** to return to the wizard. Click **Next**.

   The **Preview** wizard screen is displayed.

5. Check the result task script, and click **Finish** to insert it into your composite transformation script.

Related Concepts

Model Transformation Support

Related Procedures

Creating a Composite Transformation

Related Reference

QVT Ant Tasks
Model-To-Text Ant Tasks

Applying Model-To-Model Transformations

Use the **Apply Transformation** wizard to run the transformation on a particular model or model element.

To apply a Model-To-Model transformation to a model or a model element

1. Select a model or model element in the appropriate editor or navigator view and then choose **Model** ▸ **Apply Transformation** ▸ **QVT** ▸ **From Workspace...** from the main menu.

   Note: If you want to apply a compiled transformation, choose the **Compiled...** menu item instead of **From Workspace...**.

   The **Select Transformation** page of the **Apply Transformation Wizard** is displayed.
2. Select your transformation file and check the **Run in interpreted mode** check box if you want to use QVT Interpreter to run your the QVT code. Click **Next**.

   The **Select Destination** wizard page is displayed.

   Note: The **Run in interpreted mode** check box is disabled for Java-less projects.
3. Specify the target type, the URI of the target model, and the location of the trace file.

   For **Existing container** target type, specify the feature to which you want to place the transformation result, and whether you want to clear the existing feature contents before saving the result. Click **Next**.

   The **Configuration Properties** page displays. Note, that this page is displayed only if the applied QVT script accepts configuration properties.
4. Specify values for the configuration properties defined in your QVT script and click **Finish**.

   The generated target model and trace files appear at the specified location.

If you need to run the transformation repeatedly (for example, for testing purposes), create the Eclipse Launch Configuration for your transformation project, run it once and then press CTRL+F11 any time you need to reapply the transformation.

To create and run the Eclipse Launch Configuration for a Model-To-Model transformation

1. Select **Run Run...** from the menu. The **Run** dialog opens.
2. Select the **QVT Interpreter** configuration type and click **New**. The interpreter performs QVT. Select **QVT Transformation** if you want to execute Java generated by this QVT .
3. Specify the configuration name and URIs for input and output models and the trace file.
4. Click **Run**. The generated target model and trace files appear at the specified location.

Related Concepts

Model Transformation Support

Related Procedures

Creating a Model-To-Model Transformation
Debugging Model-To-Model Transformations
Deploying Transformations

Related Reference

Apply Transformation
Trace View

Applying Model-To-Text Transformations

Use the **Apply Transformation** wizard to run the transformation on a particular model or model element.

To apply a Model-To-Text transformation to a model or a model element

1. Select a model or model element in the **Model Navigator** or in the **Diagram Editor**, and then choose **Model ▸ Apply Transformation ▸ Model-To-Text ▸ From Workspace...** from the menu or use the context menu.

   Note: If you want to apply a compiled transformation, choose the **Compiled...** menu item instead of **From Workspace...**..

   The **Select Transformation to apply** page of the **Apply Transformation Wizard** is displayed.

2. Choose your Java transformation file (or your compiled transformation if you chose to apply the compiled transformation) and click **Next**.

   The **Specify output folder** wizard page is displayed.

3. Specify the folder where you want to save the transformation results and click **Finish**.

   The transformation output appears at the specified location.

If you need to run the transformation repeatedly (for example, for testing purposes), create the Eclipse Launch Configuration for your transformation project, run it once and then press CTRL+F11 any time you need to reapply the transformation.

To create and run the Eclipse Launch Configuration for a Model-To-Text transformation

1. Choose **Run ▸ Run...** from the menu.

   The **Run** dialog box is displayed.

2. Select the **Model-To-Text Transformation** configuration type and click **New**.
3. Specify the configuration name, Java transformation file, source model URI and the location for the transformation results.
4. Click **Run**.

   Transformation results are displayed at the specified location.

Related Concepts

Model Transformation Support

Related Procedures

Creating Model-To-Text Transformations
Debugging Model-To-Text Transformations
Deploying Transformations

Related Reference

Apply Transformation

Applying XSL Transformations

Use the **Apply Transformation** wizard to run the transformation on a particular model or model element.

To apply an XSL transformation to a model or a model element

1 Select a model or model element in the **Model Navigator** or in the **Diagram Editor**, and then choose **Model ▸ Apply Transformation ▸ XSL ▸ From Workspace...** from the menu or use the context menu.

Note: If you want to apply a transformation that is stored outside your workspace, choose **Model ▸ Apply Transformation ▸ XSL ▸ From File System...** instead.

The **Select Transformation** page of the **Apply Transformation Wizard** is displayed.

2 Choose your XSL transformation file and click **Next**.

The **Target file** wizard page is displayed.

3 In the **Target file** field, specify where you want to save the transformation results. Check the **Open result in editor** check box if you want to view the result file in the associated editor. Click **Next**.

The **Specify Parameters** wizard page is displayed.

4 Use the **Add...** button to define parameters and their values that you want to pass to your transformation. Click **Finish**.

The transformation output appears in the specified location.

If you need to run the transformation repeatedly (for example, for testing purposes), you can create the Eclipse Launch Configuration for your transformation project, run it once and then press CTRL+F11 any time you need to reapply the transformation.

To create and run the Eclipse Launch Configuration for an XSL transformation

1 Choose **Run ▸ Run...** from the menu.

The **Run** dialog box is displayed.

2 Select the **XSL Transformation** configuration type and click **New**.

3 Specify the configuration name, XSL transformation file, source model URI and the location for the transformation results. If your transformation accepts parameters, specify them in the Parameters section of the dialog box.

4 Click **Run**.

Transformation results appear in the specified location.

Related Concepts

Model Transformation Support

Related Procedures

Creating an XSL Transformation
Debugging XSL Transformations

Related Reference

Apply Transformation

Building MDA Projects from the Command Line

The MDA transformation framework uses the `com.borland.tg.mda.project.BuildApplication` Eclipse application for building or cleaning MDA Transformation projects from the command line or in the batch mode.

To build MDA Projects from the command line

1. Use the following command:

   ```
   Together -data <workspace> -application com.borland.tg.mda.project.BuildApplication
   <build | clean>
   ```

2. This command launches an instance of Together and then builds or cleans projects in the specified `workspace`. Errors generated by MDA, Java, or other builders are printed in the console window. If any errors occur during the build, the application returns nonzero exit code.

Related Concepts

Model Transformation Support

Related Procedures

Creating an MDA Transformation Project
Creating a Model-To-Model Transformation
Creating Model-To-Text Transformations

Related Reference

QVT Language
QVT Builder

Configuring Model-To-Model Transformation Builder

Before you can run a compiled transformation, you need to configure the corresponding transformation builder. Each type of transformation runs on a specific builder: **QVT Transformation Builder** or **Model-To-Text Transformation Builder**.

To configure QVT Transformation Builder

1. In the **Navigator** or **Model Navigator**, right-click your transformation project node and choose **Properties** from the context menu.
2. Choose **Builders** in the left pane and click **New**.

   The **Choose configuration type** dialog box is displayed.
3. Choose **Compiled Model-To-Model Transformation** and click **OK**.

   The **Properties for New_Builder** dialog box is displayed.
4. On the **Transformation** tab, specify the transformation ID, source model URI, and the location of the target folder.
5. On the **Build Options** tab, specify a working set of workspace resources. Any change to the specified resources will start the builder.
6. Save your changes and close the dialog box.

Related Concepts

Model Transformation Support

Related Procedures

Running Compiled Transformations

Related Reference

QVT Builder
Apply Transformation

Configuring Model-To-Text Transformation Builder

Before you can run a compiled Model-To-Text transformation, you need to configure the Model-To-Text transformation builder.

To configure Model-To-Text Transformation Builder

1. In the **Navigator** or **Model Navigator**, right-click your transformation project node and choose **Properties** from the context menu.
2. Choose **Builders** in the left pane and click **New**.

   The **Choose configuration type** dialog box is displayed.
3. Choose **Compiled Model-To-Text Transformation** and click **OK**.

   The **Properties for New_Builder** dialog box is displayed.
4. On the **Transformation** tab, specify the transformation ID, source model URI, and the location of the target folder.
5. On the **Build Options** tab, specify a working set of workspace resources. Any change to the specified resources will start the builder.
6. Save your changes and close the dialog box.

Related Concepts

Model Transformation Support

Related Procedures

Running Compiled Transformations

Related Reference

QVT Builder
Apply Transformation

Creating a Composite Transformation

To create a Composite transformation

1. Create an MDA Transformation project.
2. In **Navigator** or **Model Navigator**, right-click the project root and choose **New** ▸ **Other...** from the context menu.

   The **New** wizard is displayed.
3. Select **Modeling** ▸ **Composite Transformation** in the list of wizards tree and click **Next**.

   The **New Composite Transformation** wizard displays.
4. On the **MDA Composite Transformation** wizard screen, specify the folder where you want to store the composite transformation script and the name of the script file. Click **Next**.

   The **Composite Transformation Content** wizard screen is displayed.
5. In the **Project name** and **Default task name** fields, specify the ANT project name and the name of the default package. These parameters are inserted in the XML header of your ANT script and open in the **Ant** view.
6. Add one or more ANT tasks to your script and click **Finish**.

   Adding a New Ant Task to the Composite Transformation

Note: You can add or modify your ANT tasks at any time later using the **Ant Editor**.

Related Concepts

Model Transformation Support

Related Procedures

Creating an MDA Transformation Project

Related Reference

QVT Ant Tasks
Model-To-Text Ant Tasks

Adding a New Ant Task to the Composite Transformation

To add a new Ant task to the Composite transformation

1 Open the composite transformation script in the **Ant Editor**, right-click in the editor area where you want to insert the task, and choose **Create MDA Ant Task** from the context menu.

The **New MDA Ant Task** wizard is displayed.

> **Note:** If you create a new Composite transformation, click the **Add...** button on the **Composite Transformation Content** screen of the **New Composite Transformation** wizard.

2 Choose the **Enter data manually** option if you want to enter your task parameters manually.

The **Select launch configuration type** wizard screen is displayed.

> **Tip:** Choose the **Select existing launch configuration** option to copy task parameters from an existing launch configuration to your task script.

3 Select the launch configuration type that matches your transformation and click the **Enter launch configuration data** button.

The **Edit Launch Configuration** dialog box is displayed.

> **Note:** The **Edit Launch Configuration** dialog box is used here only to collect required task parameters, and no actual launch configuration is created.

4 Specify task parameters for the selected transformation type and click **OK** to return to the wizard. Click **Next**.

The **Preview** wizard screen is displayed.

5 Check the result task script, and click **Finish** to insert it into your composite transformation script.

Related Concepts

Model Transformation Support

Related Procedures

Creating a Composite Transformation

Related Reference

QVT Ant Tasks
Model-To-Text Ant Tasks

Creating a Model-To-Model Transformation

To create a Model-To-Model transformation

1. Create an MDA Transformation project.
2. In **Navigator** or **Model Navigator**, right-click the project root and choose **New** ▶ **QVT Transformation** from the context menu.

   The **New QVT Transformation** wizard is displayed.
3. On the **Transformation Input** wizard screen, choose a metamodel element that you want to use as the transformation input. You can expand **Together** project nodes to choose the input element from metamodels used in Together projects or expand the **metamodels** node to select an EMF class. Click **Next**.

   The **Transformation Output** wizard screen is displayed.

   Note: If you want to create an inplace transformation, check the Create inplace transformation check box. The **Transformation Output** wizard screen will not open.
4. Choose a metamodel element you want to use as your transformation output and click **Next**.

   The **New File** wizard screen is displayed.
5. Specify the transformation file name. Click **Next**.

   The **Import Metamodels** wizard screen is displayed.
6. Check the check boxes next to the auxiliary metamodels that you want to import into your transformation project and click **Next**.

   The **Import Libraries** screen is displayed.
7. Choose the libraries or compiled transformations that you want to use in your transformation and click **Next**. Together provides a number of pre-installed libraries that you can use. You can also create your own library.

   Note: You can find a list of methods defined in the pre-installed libraries on the **OCL library operations** tab of the **OCL Preferences** dialog box (**Window** ▶ **Preferences...** ▶ **Modeling** ▶ **OCL** ▶ **OCL library operations**).

   The **Transformation ID** screen is displayed.

   Note: The screen opens only when you create a QVT transformation in the MDA project that contains Java code.
8. Review the transformation ID assigned to your transformation and edit it, if necessary.
9. Click **Finish** to create the transformation.

   The transformation file opens in the **QVT Editor**.

Related Concepts

Model Transformation Support

Related Procedures

Creating an MDA Transformation Project
Applying Model-To-Model Transformations
Debugging Model-To-Model Transformations
Deploying Transformations

Related Reference

QVT Language
QVT Editor

Creating a QVT Library

To create a QVT library

1. Create an MDA Transformation project.
2. In **Navigator** or **Model Navigator**, right-click the project root and choose **New** ▸ **QVT Library** from the context menu.

   The **New QVT Library** wizard is displayed.
3. Enter or select a Java source container where you want to store your library and specify the library file name (*.qvt). Click **Next**.

   The **Import Metamodels** wizard screen is displayed.
4. Check the check boxes next to metamodel elements that you want to use as the library input. You can expand **Together** project nodes to choose input elements from metamodels used in Together projects or expand the **metamodels** node to select EMF classes. Click **Next**.

   The **Import Libraries** page is displayed.
5. Check the check boxes next to the libraries and transformations available in your workspace that you want to import into your QVT library and click **Next**.

   The **Library ID** page is displayed.
6. Specify the identifier that you want to use when referencing the compiled library and click **Finish**.

   The created library opens in the **QVT Editor**.

Related Concepts

Model Transformation Support

Related Procedures

Creating an MDA Transformation Project
Applying Model-To-Model Transformations
Deploying Transformations

Related Reference

QVT Language
QVT Editor

Creating an Example MDA Transformation Project

To create an MDA Sample Transformation Project

1. Choose **File** ▸ **New** ▸ **Example** from the menu.

   The **New Example** wizard is displayed.
2. Expand the **MDA** node in the tree view list and select the example project you want to create.
3. Click **Next**.

   The **Transformation Sample Wizard** is displayed.
4. Leave the project name as it is and click **Finish**.
5. The example project is created in your workspace.

Related Concepts

Model Transformation Support

Related Procedures

Creating an MDA Transformation Project

Related Reference

MDA Example Projects

Creating an MDA Transformation Project

To create a new MDA Transformation project

1. Choose **File** ▸ **New** ▸ **Project...** from the main menu.

   The **New Project** wizard is displayed.

2. Expand the **Modeling** node in the tree view list, select **MDA Transformation Project**, and then click **Next**.

   The **Transformation Project** wizard screen is displayed.

3. Specify the project name and location, and then click **Next**.

   Note: If you want to add a Model-To-Text transformation to your project, or compile your project transformations later, check the **Create a plug-in project** check box and then define your plug-in project settings on the **Transformation Project Content** wizard screen. Otherwise, Together will not generate Java code for your project transformations and you can run them in the interpreted mode only.

   The **Transformations** wizard screen is displayed.

4. If you want to add a transformation to your new project, check the **Create Transformation in the new Transformation Project** check box, select the required transformation type, and follow the wizard instructions for the selected transformation type. If you want to create an empty project, leave the page as it is and click **Finish**.

   The new MDA Transformation Project opens in the **Together Modeling** perspective.

Related Concepts

Model Transformation Support

Related Procedures

Creating a Model-To-Model Transformation
Creating Model-To-Text Transformations
Building MDA Projects from the Command Line

Creating an XSL Transformation

To create an XSL transformation

1. Create an MDA Transformation project.
2. In **Navigator** or **Model Navigator**, right-click the project root and choose **New** ▸ **XSL Transformation** from the context menu.

   The **Transformation Input** screen of the **New XSL Transformation** wizard is displayed.
3. Choose a metamodel element that you want to use for your transformation input and click **Next**.

   The **New File** screen is displayed.
4. Select the parent folder and the name for the XSL transformation file.
5. Click **Finish** to create the transformation.

   The XSL transformation file opens in the **XSL Editor**.

Related Concepts

Model Transformation Support

Creating Model-To-Text Transformations

To create a Model-To-Text transformation

1. Create a plug-in Transformation project or use an existing transformation project.
2. In **Navigator** or **Model Navigator**, right-click the project root and choose **New ▸ Model-To-Text Transformation** from the context menu.

   The **New Transformation Class** screen of the **New Model-To-Text Transformation** wizard is displayed.
3. In the **Source folder** field, specify the folder within your plug-in project that contains generated Java source files.

   In the **Class name** specify a fully qualified name of the Java class that you want to create.

   In the **Package name** filed, specify the name of the package that contains the class.

   Check the **Use JET** check box if you want to use JET.

   In the **Transformation ID** field, specify a unique ID assigned to the transformation. Click **Next**.

   The **Transformation Input** screen is displayed.
4. Choose a metamodel element that you want to use for your transformation input.

   You can expand the Together project nodes to select the metamodel element from the metamodels that are used in the projects or expand the metamodels node to select an EMF class.
5. Click **Finish** to create the transformation.

   The created class is opened in the Java Editor.

Related Concepts

Model Transformation Support

Related Procedures

Creating an MDA Transformation Project
Applying Model-To-Text Transformations
Debugging Model-To-Text Transformations
Deploying Transformations

Debugging Model-To-Model Transformations

To debug a Model-To-Model transformation

1. Open the transformation file in the **QVT Editor**.
2. Double-click the gray area on the left of the editor pane against the line where you want to set a breakpoint.
3. Choose **Run** ▸ **Debug...** from the main menu.

   The **Debug** dialog box is displayed.
4. Under **QVT Interpreter**, create the launch configuration for your transformation:

   On the **Transformation** tab, specify the configuration name, transformation module, source model URI, target model, and whether you want to generate a trace file.

   Note: If you check the **Clear contents** check box for **Inplace** or **Existing container** target types, only files created during the debugging process will be cleared.

   On the **Configuration** tab, set values for transformation properties, if needed.
5. Click the **Debug** button.

   The debugging session starts and then immediately stops at the first breakpoint that you set. Use the **Debug** and **Console** views to control the debugging process or switch to the **Debugging** perspective when prompted.

Related Concepts

Model Transformation Support

Related Procedures

Applying Model-To-Model Transformations
Creating an MDA Transformation Project

Related Reference

QVT Editor

Debugging Model-To-Text Transformations

Note: Together uses Eclipse Java Debugger for Model-To-Text transformations.

To debug a Model-To-Text transformation

1. Open the generated transformation file in the Java Editor.
2. Double-click the gray area on the left of the editor pane against the line where you want to set a breakpoint.
3. Choose **Run ▸ Debug...** from the main menu.

   The **Debug** dialog box is displayed.
4. Under **Model-To-Text Application**, create the launch configuration for your transformation:

   On the **Transformation** tab, specify the configuration name, transformation file, source model URI, target folder for transformation results, and whether you want to monitor changes in the target folder during the debugging process.

   On the **Workspace** tab, specify which additional projects you want to import to the temporary debugging workspace.
5. Click the **Debug** button.

   Together launches a “headless” instance of Eclipse that runs in the background. The projects required for the transformation are imported into the temporary debugging workspace.
6. The debugging session starts and then immediately stops at the first breakpoint that you set. Use the **Debug** and **Console** views to control the debugging process or switch to the **Debugging** perspective when prompted.

Related Concepts

Model Transformation Support

Related Procedures

Applying Model-To-Model Transformations
Creating an MDA Transformation Project

Debugging XSL Transformations

To debug an XSL transformation file

1. Open the transformation file in the **XSL Editor**.
2. Double-click the gray area on the left of the editor pane against the line where you want to set a breakpoint.
3. Select **Run > Debug...** from the menu. The **Debug** dialog is displayed.
4. Under **XSL Transformation**, choose the launch configuration you have created for your transformation and click **Debug**.

   The **Extensible Stylesheet Debug** perspective opens.
5. The debugging session will be started and immediately stopped at the first breakpoint specified.
6. As necessary, use standard debugger commands F5 = StepInto, F6 = StepOver, F7 = StepReturn, CTRL-R = run to line, F8 = Resume.

Related Concepts

Model Transformation Support

Related Procedures

Applying XSL Transformations

Related Reference

XSL Editor

Deploying Transformations

Note: You cannot deploy Java-less transformations.

To deploy compiled QVT and Model-To-Text transformations

1. Choose **File ▸ Export** from the menu.

   The **Select** page of the **Export** wizard is displayed.
2. Select the **Deployable Plug-ins and Fragments** item and click **Next**.

   The **Deployable Plug-ins and Fragments** page of the **Export** wizard is displayed.
3. In the **Available Plug-ins and Fragments** window, check the check boxes against the transformations that you want to export.
4. In the **Export Destination** area, specify the Eclipse installation directory.
5. Click **Finish**.

   Restart Together with `—clean` option to load the newly created transformation plug-in.

Related Procedures

Running Compiled Transformations

Related Reference

QVT Builder
Apply Transformation

Manually Registering a Metamodel for Use with QVTO

In addition to using deployed metamodels, you can also use metamodels from a workspace with Operational QVT. Metamodels should be defined in ecore models and then registered in the QVTO project properties.

To manually register a metamodel for use with QVTO

1. In the Model Navigator view, right-click the project node and select **Properties**.
2. In the **QVT Settings/Metamodel Mappings**, click **Add**, select the target ecore model URI, and specify the source model URI, which refers to the model in QVT. Click **OK** and close the project properties dialog.
3. After the new metamodel is available to QVT, reopen any open editors to use the newly registered metamodel.

Related Concepts

Model Transformation Support

Related Procedures

Running an Operational QVT

Opening MDA Views

To open MDA views:

1 Choose **Window** ▸ **Show View** ▸ **Other...**
2 In the **Show View** dialog box, expand **MDA** node.
3 Select **Metamodel Browser** and **OCL Expressions** views and click **OK**.
4 The views open in the Eclipse framework.

To open the Trace view

1 In the **Navigator** view, double-click a `.trace` file that you want to open in the **Trace** view.
2 The view displays the selected trace file.

Related Concepts

Model Transformation Support

Related Procedures

Creating an MDA Transformation Project

Related Reference

Metamodel Browser View

Running a Composite Transformation script

To run a Composite transformation script

1. Choose **Run** ▸ **External Tools** ▸ **External Tools...** from the menu.

   The **External Tools** dialog box is displayed.
2. Select the **Ant Build** node and click the **New** button to create a new launch configuration.

   The new configuration is added to the **Ant Build** node.
3. On the **Main** tab in the **Name** field, specify the name of the run configuration.
4. Under the **Buildfile**, click the **Browse Workspace...** button.

   The **Choose Location** dialog box is displayed.
5. Click the project that contains the composite transformation build file in the left pane and then select the required Ant build file (.xml) in the right pane. Click **OK** to close the dialog box.
6. On the **JRE** tab, select the **Run in the same JRE as the workspace** option.
7. Click the **Run** button to run the Ant build file.

To run an individual task in the Composite transformation script

1. Choose **Window** ▸ **Show view** ▸ **Other...**

   The **Show View** dialog box is displayed.
2. Expand the **Ant** node, select the **Ant** item, and click **OK**.

   The **Ant** view is displayed.
3. Drag and drop your composite transformation script file into the **Ant** view.

   The script file is displayed in the **Ant** view tree under the name specified in the project `name` property.
4. Expand the script file node, click the Ant task that you want to run and choose the **Run As** ▸ **Ant Build...** item from the context menu.

   The **Modify attributes and launch** dialog box is displayed.

   Note: If you have created Ant configurations for your script file, the **Ant Configuration Selection** dialog box is displayed instead. Select the configuration created for your script file and click **OK** to open the **Modify attributes and launch** dialog box.
5. On the **Targets** tab, check the check box next to the tasks that you want to run.
6. On the **JRE** tab, select the **Run in the same JRE as the workspace** option.
7. Click the **Run** button to run the specified Ant task(s).

Related Concepts

Model Transformation Support

Related Procedures

Creating a Composite Transformation

Related Reference

QVT Ant Tasks
QVT Ant Tasks

Running an Operational QVT

To run an operational QVT

1. Create a new run configuration by choosing **Run ▸ Run Configurations...** from the main menu.

   The Run manager dialog opens.

2. Right-click **Operation QVT Interpreter** and choose **New**.

   Type a descriptive title in **Name**.

3. Click **Browse** next to **Transformation module** and select the desired QVTO file. You can optionally specify a trace file that lists transformation mappings.

   Enable **Generate trace file** and click **Browse file**.

4. Click **Browse** next to **Model URI** for each input model.

   Select your model and click **OK**.

   Tip: For Together models, the process of entering URIs is automated. However, you can enter the model URI manually. This URI must follow the `together:/ + projectName + #model:project:: + packageName` convention. For example, if a UML20 project is named `u2`, the URI is `together:/u2#model:project::u2`, which selects the UML20 model's root package as the input scope.

5. Click **Browse** next to **Model URI** for each output model.

   Select your model and click **OK**. You can optionally select a model feature by clicking **Select**.

   Warning: Enabling **Clear contents** erases data in the target model.

   Note: If the QVT includes more input or output models, additional models might be selected.

6. The **Configuration** tab contains a list of any defined configuration properties in the QVT script. If properties are not defined in the QVT, the list is empty.

   Declarations for configuration properties resemble `configuration property modelName : String;`

   Enter appropriate data in **Value**.

   Note: Because raw string values are specified here, the launch configuration validates each passed value according to the property type and rejects invalid values. Because this check is not always sufficient, the configuration performs an additional validation at execution time. If the specified value is invalid (for example, if the configuration property type has been changed in the QVT script while the launch configuration retains the previous value for it), a QVT runtime exception occurs at the point of the configuration property initialization and the execution ends. If no value is set for a configuration property at execution time, the property defaults to a null value at initialization. Currently, only primitive types supported by QVT are supported for configuration properties.

7. Refer to the **Common** tab for general runtime configuration options. This content is inherited from the Eclipse platform.

8. Click **Apply** to save the configuration and **Run** to perform the transformation.

Related Concepts

Model Transformation Support

Related Procedures

Manually Registering a Metamodel for Use with QVTO

Running Compiled Transformations

You can run compiled transformations using:

- **QVT Transformation External Tool Builders**, which run transformations as a part of the project build.
- The **Apply Transformation** wizard for Model-To-Text or Model-To-Model transformation.

Note: You need to deploy your compiled transformation before you can run it.

To run a compiled transformation using transformation builders

1. Right-click your project root and choose **Properties** from the context menu.

   The **Properties** dialog box is displayed.
2. Click **Builders** and then click the **New...** button on the **Builders** page.

   The **Choose configuration type** dialog box is displayed.
3. Select the **Compiled Model-To-Text Transformation** or **Compiled QVT Transformation** item and click **OK**.

   The **Edit launch configuration properties** dialog box is displayed.
4. In the **Name** field, specify the name of the new builder.
5. Click the **Browse...** button next to the **Transformation id** field and select your transformation file in the **Select Transformation** dialog box.
6. Click the **Browse...** button next to the **Source model URI** field and choose the source model in the **Workspace Contents** dialog box.
7. For the Model-To-Model transformation, click the **Browse...** button next to the **Target folder** field and choose the folder where you want to store the transformation results.
8. For the Model-To-Text transformation, in the **Target model** area specify the target type, URI of the target model, and location of the trace file.
9. Click **OK** to close the dialog box.

   The compiled transformation runs as a part of the project build.

Related Concepts

Model Transformation Support

Related Procedures

Applying Model-To-Text Transformations
Applying Model-To-Model Transformations
Deploying Transformations

Comparing and Merging Models

Describes how to compare models and model elements with each other, and perform history comparison with the earlier versions of the model stored in VCS.

In This Section

Comparing and Merging Shared Models

Use the **Synchronize** view to compare shared models.

To compare and merge shared models

1. In the Team **Synchronize** view, select a model or model element version stored in the repository that you want to compare with the local version.
2. Choose **Model** ▸ **Compare With** ▸ **Local Version** from the main menu.

   Tip: Alternatively, right-click a model or model element and choose **Open In Model Compare Dialog** from the context menu.

   The comparison results display in the **Model Compare** dialog box.
3. Review the differences, apply your changes, and then commit the model to the repository using menu commands specific to your VCS.

Note: When merging a shared model, you can change your local version only. Together models consist of a large number of files, so you need to have all of these files locally.

Warning: Together performs merge on the model level, and existing file conflicts may still remain after the model merge. To commit these changes, use the "forced commit" mechanism provided by your version control system, (for example, the "Override and Commit" option in CVS).

Related Concepts

Model Compare and Merge

Related Procedures

Merging Models

Related Reference

Model Compare/Merge
EMF Model Compare Preferences

Comparing Models

You can compare two or three models against each other and review differences.

To compare models

1. In the **Model Navigator** or **Navigator** view, select two or three models or model elements.
2. Choose **Compare With ▸ Each Other (as Models)** from the context menu.

   The comparison results display in the **Compare** editor.

Related Concepts

Model Compare and Merge

Related Procedures

Merging Models

Related Reference

Model Compare/Merge
EMF Model Compare Preferences

Merging Models

How to merge models using the **Compare** editor.

Warning: Source code elements merged using the **Compare** editor become design elements after merging.

To merge models

1. Compare models or model elements.

   Comparing Models
2. Double-click the first difference displayed in the **Structure Compare** section of the **Compare** editor.

   The difference details are displayed in the **Substructure/Properties Merge** section of the **Compare** editor.

   Tip: Use the **Show Containment References** button on the editor toolbar to toggle views of the comparison results.
3. In the **Substructure Merge** tab, select an element of the input model and click the **Copy to the Left** or **Copy to the Right** button to copy it to the target model.
4. In the **Properties Merge** tab, select a property of the input model and click the **Copy to the Left** or **Copy to the Right** button to copy it to the target model.

Tip: You can undo and redo operations, using the **Undo** and **Redo** toolbar buttons, or keyboard shortcuts CTRL+Z and CTRL+SHIFT+Z.

Related Concepts

Model Compare and Merge

Related Procedures

Comparing Models

Related Reference

Model Compare/Merge
EMF Model Compare Preferences

Comparing Models

You can compare two or three models against each other and review differences.

To compare models

1. In the **Model Navigator** or **Navigator** view, select two or three models or model elements.
2. Choose **Compare With ▸ Each Other (as Models)** from the context menu.

   The comparison results display in the **Compare** editor.

Related Concepts

Model Compare and Merge

Related Procedures

Merging Models

Related Reference

Model Compare/Merge
EMF Model Compare Preferences

Together Object Constraint Language (OCL)

This section provides how-to information on using Together OCL facilities.

In This Section

Creating an OCL Guard Condition for a Transition

An OCL expression can restrict a StateMachine transition by acting as a guard to that transition. In an OCL guard condition, the StateMachine must have a context that is a Classifier. The expression, which is evaluated when the guard's transition is attempted, is of type Boolean.

To create a guard condition for a transition

1. Select a transition on a diagram.
2. Select the **guard** tab in the **Properties** view.
3. Specify the language for your guard condition (OCL by default).
4. Select the **body** field and click the **Edit** button.
5. Type the condition expression and click OK to apply changes.

Related Concepts

About OCL Support in Together

Related Reference

UML 2.0 State Machine Diagrams

Creating Constraints

You can create constraints for all elements of the UML 2.0 diagrams. To describe a constraint, you can use plain text or OCL.

To create a constraint in a UML 2.0 diagram

1. Click the **Constraint Link** button on the diagram Palette and point to the model element that defines the context of your constraint (such as Class, Attribute or Operation), then hold down the left mouse button and draw the link to the place where you want to create the **Constraint** element.
2. Release the mouse button to insert the element.

   The element displays with the in-place editor open.
3. Type the constraint expression, save your changes, and close the **Constraint** editor.

   Tip: Alternatively, use one of the following methods:

   - On the context menu of an element, choose **New ▸ Linked Constraint** and enter the constraint expression.
   - Use the **Constraint** and **Constraint link** buttons on the Tools Palette to place a constraint node on the diagram and link it to the context element.

Related Concepts

About OCL Support in Together

Editing Constraint Expressions

Constraint expressions are represented in plain text or in the OCL language. You can use the Editor view or the OCL tab of the Properties View to create or modify the constraint body.

To edit a constraint expression in the Editor view

1. Double-click a constraint element. The constraint test opens in its own tab of the Editor view.
2. In the **Language** drop-down list in the upper-right corner of the view, select the desired language of the expression.

   Note: If OCL is selected, the OCL editor provides syntax control and error highlighting. A red or green mark to the right indicates the validity of the OCL expression.

3. Apply changes.

To edit a constraint expression in the Properties View

1. Select a constraint element in diagram.
2. In the Properties View, select the **OCL** tab.
3. In the **language** field, select the desired language of the expression.
4. In the **body** field, enter the expression in the text area, or click the **Edit** button and enter text in the **Enter constraint** dialog box.

Tip: Alternatively, select a constraint element and press `F2`. Edit the constraint in the editor.

Related Concepts

About OCL Support in Together

Related Procedures

Creating Constraints

Related Reference

Diagram View

Enabling Source Code Generation from OCL Constraint

After your design project is finished, you can generate Java or C++ code from it. When generating Java, you can use your OCL constraints.

To enable or disable OCL constraints processing

1. Click **Window** ▸ **Preferences** on the main menu.
2. Expand the **Modeling** node and then the **Source Generation** node and select **Java**.
3. Select 5.0 in the **Source compatibility** list.

    Note: Because java asserts are generated for some expressions, it is recommended to select `java5` as the target java.

4. Switch to the **OCL** tab and select whether to generate invariants and pre/post conditions.
5. Click **OK** to save the changes and exit the dialog box.

Note: All the listed settings can be made in the Export wizard (**File** ▸ **Export** ▸ **Modeling** ▸ **GenerateJavaProject**) if you select the **Enable project-specific settings** option in the **Generation Options** tab.

Related Concepts

About OCL Support in Together

OCL in Documentation Templates

Together allows you to compose model queries and define enable conditions using OCL syntax, and then use them in a template for generating documentation. OCL or Legacy type expressions can be entered in the template element's properties dialog box using the provided Expression Editor. Where applicable, the editor is either opened in the tab or you can use the Edit Expression button to open the editor.

In addition to the standard OCL operations, the special native OCL extensions are provided for the functions that are specific for Documentation Generation. Native OCL extensions tend to have the same signature and meaning as the legacy Documentation Generation functions have. Code sense suggests these operations along with the standard OCL ones.

To add an expression to your template

1. Open a template where you want to add an OCL expression.
2. In the **Properties** dialog box, open the tab where you want to type the expression. If the Expression Editor is not opened in the tab, click the **Edit Expression** button.
3. Specify the context for your expression in the **Context** field.
4. In the **Body** area, you can type the expression text. The code sense, syntax highlighting and validating are available.
5. Click **OK** to save the expression in the template.

Related Concepts

Documentation Template Controls
About OCL Support in Together
Enable Conditions

Searching Model with OCL queries

Together lets you search for models using OCL queries.

To find model elements that match the specified OCL query

1. On the main menu, choose **Search** ▸ **Model**.

   The **Search** dialog box is displayed.
2. Click the **OCL Model Search** tab.
3. Specify the context for your expression in the **Context** field.

   Tip: Use the drop-down list or the Content Assistant. To open the Content Assistant, click on the **Context** field and press CTRL +SPACE. Choose your element from the list.

   For example, to search for all UML 2.0 classes that have the stereotype `MyStereotype`, enter:

   ```
   uml20::classes::Class
   ```
4. In the **Invariant** field, type the query expression.

   For example, to complete your search for all UML 2.0 classes that have the stereotype `MyStereotype`, enter:

   ```
   self.stereotypes->includes('MyStereotype')
   ```
5. In the **Scope** section, click the appropriate radio button to select the search area. The possible options are workspace, selected resources, the current project or a predefined working set.

   To select a working set, click the **Choose** button. In the **Select Working Set** dialog, choose your working set and click **OK**. If there are no available working sets, use the **New** button to create one.
6. Click **Search**.

A tree with the list of matching elements opens. You can navigate to the corresponding diagram from this view by double-clicking the selected element.

Related Concepts

OCL Support

Related Procedures

Searching Model Elements

Using OCL in Model Audits and Metrics

You can run audits and metrics in your design model using OCL expressions.

You can create custom OCL audits and metrics that operate with metamodel types and run them against the model that is an instance of the same metamodel. Together also contains a set of sample audits (see the following procedure to access them). The ideas of most of them are taken from Ambler and Fowler books. These audits can be used as examples for custom rules creation. For a description of the predefined model audits and metrics provided in Together, refer to "Model Audits and Metrics Descriptions."

To define audits and metrics

1 Choose **Window** ▸ **Preferences...** from the menu.

The **Preferences** dialog box is displayed.

2 Expand the **Modeling** node and select **QA Model**.

3 Select either the **Audits** or **Metrics** tab.

4 Click **New** to add an audit or metric. The **Edit Audit** or **Edit Metric** dialog box is displayed, respectively.

5 Specify your audit or metric name, description, severity, and select the context of the OCL expression. The code for your new audit or metric is displayed in the standard OCL editor in the **Body** text area.

The audit expression should be a valid invariant that returns `Boolean`. Each metric expression should return an `Integer` value.

To run defined audits and metrics

1 In the diagram, select model elements against which you want to run audits or metrics.

2 Select **Model** ▸ **Run Model Audits** or **Model** ▸ **Run Model Metrics** from the main menu.

The results are displayed in the **Model Audit** or **Model Metrics** view, respectively.

Note: The scope depends on the current selection made on the diagram. If the project default diagram is selected, the entire project will be checked. If a single element is selected, only this element will be checked.

Note: If the first operand of an audit returns false, the second operand is not computed and errors are logged. This ensures precise execution semantics and is necessary because the second operand computation may have side effects and may call mapping operations.

Related Concepts

OCL Support
Model Metrics

Related Procedures

Running Model Audits and Metrics

Related Reference

Model Audits and Metrics Descriptions
OCL

Working with a Combined Fragment

In this section you will learn how to:

- ♦ Create a combined fragment
- ♦ Create nested combined fragments
- ♦ Create nested operators
- ♦ Sever nested operators
- ♦ Create operands
- ♦ Expand combined fragments across several lifelines
- ♦ Detach a combined fragment from a lifeline

To create a combined fragment

1 Choose the **Combined Fragment** button in the diagram Palette, and click on the target lifeline.
2 In the **New Combined Fragment** dialog box that opens, choose an operator from the list of available operators and set the combined fragment options (operator name, arguments, or number of operands).
3 Click **OK**.

The combined fragment is added to the target lifeline or execution specification. Each new combined fragment has a different color to distinguish it from the other combined fragments within the same cluster of nested frames.

To create a nested combined fragment

1 Choose the **Combined Fragment** button in the diagram Palette.
2 Click on the target combined fragment that already exists in a lifeline.

Note: Each new node has a different color that is selected at random. You can work with the inner frames in the same way as with the outer frames: move along a lifeline, spread them over several lifelines, detach and tie frames. Note that drawing a message link from a frame automatically expands it, together with its outer frames, if any.

To create nested operators

1 Select a combined fragment.
2 In the **other operators** field of the Properties View, click the chooser button. The **Interaction Operators** dialog box opens, displaying the list of already defined operators in the current combined fragment.
3 Click the **Add** button. A new line is displayed below the existing entry in the list of operators.
4 If a certain operator enables arguments, enter them in the adjacent field in the Arguments column. Use a comma as a delimiter.
5 Use the **Add** and **Remove** buttons to compile your list of the nested operators. Use the **Up** and **Down** buttons to specify the proper order of nested operators.
6 Click **OK** to apply changes.

The nested operators are now listed in the descriptor of the combined fragment in the specified order.

To sever operators

1. Right-click a combined fragment that contains nested operators.
2. On the context menu, choose **Sever operators between**.
3. On the submenu, select the pair of operators between which the combined fragment will be divided.

A nested combined fragment is now created.

To combine with an outer fragment

1. Right-click an inner fragment.
2. On the context menu, choose **Combine with an outer fragment**.

To create an operand

1. Select a combined fragment or an operand in the Model Navigator or in the Diagram Editor .
2. On the context menu of the selection, choose **New** ▸ **Interaction Operand**.
3. In the **Interaction constraint** tab of the Properties View, select the language to be used for describing the constraint. To do this, click the Language drop-down list and choose OCL or plain text.
4. Type the constraint expression.
5. Add as many operands as required.
6. Apply changes.

A new operand is now created. If the operand was created from the context menu of a combined fragment, it will be added to the end of the combined fragment. If the operand was created from the context menu of an operand, it will be added just before this operand. Constraint text is displayed in the operand section of the combined fragment.

To expand a combined fragment across several lifelines

1. Select the combined fragment.

   Tip: You can expand both outer and inner combined fragments.

2. Click the anchor icon and drag it to the target lifeline.

The fragment now spans across lifelines, with the mounting links on each lifeline.

To detach a combined fragment from a lifeline

1. Select the mounting link of a combined fragment.
2. Choose **Delete** on the context menu.

Tip: You cannot delete the only mounting link of a combined fragment. A combined fragment must be attached to at least one lifeline.

Related Reference

Operator and Operand for a Combined Fragment

Working with Custom OCL Operations

Custom OCL operations can be used in OCL queries throughout Together—including in audits, metrics, search expressions, and gendoc templates. In this section you will learn how to perform the following actions with the custom OCL operations:

- Create
- Delete
- Edit
- Export
- Import
- Clone

To create a custom OCL operation

1. On the main menu, choose **Window** ▸ **Preferences**.
2. In the **Preferences** dialog, choose **Modeling OCL** and open the **OCL Operations** tab.
3. On the toolbar of the tab, click the **New** button.
4. In the Edit Operation dialog that opens, choose a context and enter the valid OCL expression as the body of the operation.
5. Click **OK**.

To delete an OCL operation

1. On the main menu, choose **Window** ▸ **Preferences**.
2. In the **Preferences** dialog, choose **Modeling OCL** and open the **OCL Operations** tab.
3. Select the operation to be deleted.
4. On the toolbar of the tab, click the **Remove** button.

To edit an OCL operation

1. On the main menu, choose **Window** ▸ **Preferences**.
2. In the **Preferences** dialog, choose **Modeling OCL** and open the **OCL Operations** tab.
3. Select the operation to be modified.
4. On the toolbar of the tab, click the **Edit** button.
5. In the Edit Operation dialog that opens, update the context and the body of the operation.
6. Click **OK**.

Tip: Alternatively, you can update the body of the operation in the OCL text area.

To export an OCL operation

1. On the main menu, choose **Window** ▸ **Preferences**.

2 In the **Preferences** dialog, choose **Modeling OCL** and open the **OCL Operations** tab.
3 Select the operation to be exported.
4 On the toolbar of the tab, click the **Export** button.
5 In the export dialog that opens, navigate to your target location and save the operation as type `*.oclOperations`.

To import an OCL operation

1 On the main menu, choose **Window** ▶ **Preferences**.
2 In the **Preferences** dialog, choose **Modeling OCL** and open the **OCL Operations** tab.
3 On the toolbar of the tab, click the **Import** button.
4 In the import dialog that opens, find the file of type `*.oclOperations` that you want to import, and click **OK**.

To clone an OCL operation

1 On the main menu, choose **Window** ▶ **Preferences**.
2 In the **Preferences** dialog, choose **Modeling OCL** and open the **OCL Operations** tab.
3 Select the operation to be copied.
4 On the toolbar of the tab, click the **Clone** button.

Related Concepts

OCL Support

Related Procedures

Together Object Constraint Language (OCL)

Related Reference

OCL

Patterns and Templates

This section provides how-to information on using patterns with Together.

In This Section

Adding a Pattern Part

To add a part to a pattern

1. Right-click an oval that represents the pattern instance in the diagram.
2. On the context menu of the pattern instance, choose **Create pattern part**.
3. On the submenu of this menu node, select a part to be created. The **Create Pattern Part** wizard opens.
4. On the first page of the wizard, specify the target package where the part will be created. Click **Next**.
5. On the second page of the wizard, specify properties for the new pattern part, using the **Set values** section.
6. Click **Finish**.

Related Concepts

Patterns and Templates

Related Reference

Pattern Explorer

Building Pattern

In this section, you will learn how to build a pattern from a pattern definition project.

To build a pattern from a pattern definition project

1. Right-click your pattern definition project in the Model Navigator, and choose **Patterns Build pattern** on the context menu.
2. Specify the name of the pattern and select a folder in the pattern registry (in the local workspace-specific part) where the shortcut to the new pattern will be created.
3. Click **Finish** to start the compilation process.

Note: During the compilation process, the project validation is performed. If an error preventing the project from compiling is found, the compilation is aborted and errors are displayed in the Pattern definition validation results view.

Creating Model Element by Pattern

To create model elements by pattern

1. On the main menu, choose **File** ▶ **New** ▶ **Other**.
2. Expand the **Modeling** node and select **Model element by pattern**.
3. Select patterns to apply and click **Next**. You can optionally define the values for user-editable properties of the pattern and select whether to add a pattern instance into the model. If you choose to add a pattern instance, the new entity is displayed in the Patterns root on your project and in the Pattern Explorer, and the oval is displayed in the diagram.
4. Click **Finish** to create elements based on the selected pattern definition.

Tip: If your diagram looks entangled after adding pattern instances, use the **Layout** ▶ **Layout All** command on the diagram context menu.

Related Concepts

Patterns and Templates

Related Reference

Pattern Explorer

Creating Pattern Definition

New pattern definition projects are created the same way as any other project in Together. While creating a new project, select **Pattern definition** under the **Together** node. The project is created with an empty model for you to design your pattern from scratch.

Together suggests a simplified way to create a pattern definition project by exporting a selection of model elements to a pattern definition.

Note: Patterns can be created only for UML 2.0 projects.

To create a pattern definition project from selected model elements

1. Select the elements you want to transform to a pattern definition.
2. Right-click the selection and choose **Export** ▸ **Pattern definition** from the context menu.
3. In the **Create pattern from elements** dialog, specify the name of the pattern definition and the target category. You can choose to display the existing patterns and select the transformation profile as necessary. Click **Next**.
4. In the **Set role names** page, edit the role names of the elements involved in the pattern definition, or accept defaults. Click **Next**.
5. In the **Set default values** page, specify the default values for each role. Click **Finish**.

The newly created model will be filled with images (represented as **InstanceSpecifications**) of selected elements of the source model. Properties of the source model elements are represented by **Slots**.

The new pattern definition is displayed in the Pattern Registry view.

Related Procedures

Creating a Project

Related Reference

Pattern Registry
Create Pattern from Elements

Deleting Patterns Instances

You can delete elements of the patterns using the Diagram Editor , the Model Navigator, or the **Pattern Explorer**.

To delete a pattern instance

1. In the Diagram Editor or Model Navigator, select the pattern instance to be deleted.
2. On the context menu, choose **Delete** to delete the selected pattern instance from the diagram and model.

Tip: Alternatively, select a pattern instance in the **Pattern Explorer** and choose **Delete instances** on the context menu. The selected instance is deleted from the model.

To delete all instances of a pattern

1. In the **Pattern Explorer**, select a pattern node.
2. On the context menu of the selection, choose **Delete instances**.

Related Concepts

Patterns and Templates

Related Reference

Pattern Explorer
Pattern Registry

Editing Templates

You can edit templates through the **Templates** view. To open the **Templates** view:

From the main menu, choose **Window** ▸ **Show View** ▸ **Templates**. If **Templates** is not a menu option, choose **Window** ▸ **Show View** ▸ **Other** ▸ **Patterns and Templates** ▸ **Templates**.

To rename a template

1. On the **Templates** view, right-click a template and select **Rename**. This opens the **Rename Template** dialog box.
2. Enter the new name and click **OK**.

To modify a template

1. In the **Templates** view, right-click a Template.
2. Select **Open**. This opens the **Template** editor.
3. Make your edits.

There are three separate editors: one for link templates, one for class templates, and one for package templates. Some edits are entered in a text box, others through a dialog that opens when you click **Edit**.

To restore a template that you have changed to its original state

1. In the **Templates** view, right-click a template.
2. Select **Restore**.

Note: This option is available only after changes to a template have been saved.

To delete a template

1. In the **Templates** view, right-click a template.
2. Select **Delete**.
3. Confirm the deletion in the **Delete** dialog box.

You can restore templates that you have deleted. You can do this only at the category level, which means all templates deleted from the category are restored simultaneously. You cannot restore deleted templates individually.

To restore deleted templates

1. In the **Templates** view, right-click the category for the deleted template.
2. Choose **Restore deleted** from the context menu.

Related Concepts

Patterns and Templates

Managing Pattern Definitions in the Pattern Registry

In this section you will learn how to:

- Open the Pattern Registry
- Create a new pattern from the registry
- Edit pattern definitions from the registry
- Export pattern definitions as a plugin

To open a Pattern Registry

1. Select **Window Show view Other** from the main menu.
2. Expand the **Patterns and Templates** node and select **Pattern Registry**.

To create a new pattern from the registry

1. Right-click the Workspace folder in the **Pattern Registry** and choose **New Pattern** on the context menu.
2. Specify a name for the new pattern and select a pattern definition project that will be compiled to a pattern.

To edit a definition from the registry

1. Right-click a pattern in the Pattern Registry.
2. Choose **Edit Definition** on the context menu.

To export a definition as a plugin

1. Click the **Export...** button on the toolbar of the **Pattern Registry**.
2. Specify necessary parameters for a new plugin and select patterns you want to include in the plugin.
3. Click **Finish** to create a new Eclipse extension plugin.

Recognizing Patterns

You can examine your project for patterns.

To recognize patterns in a project

1. Select a project in the Model Navigator.
2. On the context menu of the project, choose **Patterns** ▸ **Recognize Patterns**.

   The **Recognize Patterns** wizard opens.
3. In the **Choose Pattern Definitions** page, choose definitions of the patterns you want to find in the project. To do this, select patterns in the tree of available patterns, and click **Next**.
4. In the **Recognition Scope** page, select model elements. Use the **Add** and **Remove** buttons to make up the list of model elements to be examined. Click **Next**.
5. Observe the list of recognized patterns, or repeat the process with the other patterns and scope.

Warning: For the large projects, recognizing patterns increases memory consumption, which can result in the “out of memory” exception. Change the JVM parameters to increase memory limit. To do that, restart the application with the following Eclipse command line argument: `-vmagrs -Xmx600M`. This adds 600 MB of RAM. Depending on your hardware configuration, you can try the different additional memory values.

Recognition results are presented in a dialog box allowing you to choose instances that should be created and stored in the project model. After some of them are selected, new model entities are created in a special model node called "Patterns" and grouped by pattern definition instances in the Pattern Explorer view. References to pattern instances from a model can be created on any diagram by using the Add shortcut command.

Related Concepts

Pattern recognition

Using Conditions in Templates

When using the template editor to write the code snippets and syntax for fields, methods and classes, you can write conditional logic to control what is created when the template is applied.

The following example shows how conditional logic applies to the Applet template.

To apply conditional logic to the Applet template

1 From the main menu, choose **Window** ▶ **Show View** ▶ **Other** ▶ **Patterns and Templates** ▶ **Templates** to display the **Templates** view.

2 Expand the **Local Templates** node, then **Java Package**, and then **Standard**.

3 Double-click the Applet template to open the template editor.

4 Click the **Variables** tab. Notice the Boolean variable named `init`, which controls the code generation by the Applet template.

5 Click the **Units** tab. Notice that `init`, the Boolean variable declared on the Variables tab, controls the creation of the methods `init()` and `destroy()`.

Note: Variable names declared on the Variables tab must be preceded by a `$` symbol. Otherwise, they are treated as normal text.

Related Reference

Syntax and Conditions in Templates

Using the Class Template Editor

To edit class templates in the Templates view

1 Open the **Templates** view: From the main menu, choose **Window** ▸ **Show View** ▸ **Other** ▸ **Patterns and Templates** ▸ **Templates**.

2 Right-click a template in the Java Class templates section, and select **Open**.

By opening and exploring the default templates that come with Together, you can get a better understanding of how easy it is to create your own class templates.

Related Procedures

Using Conditions in Templates

Related Reference

Template Variable Types

Using the Link Template Editor

You can edit link templates though the Templates view.

To edit a link template

1. Open the **Templates** view: From the main menu, choose **Window** ▶ **Show View** ▶ **Other** ▶ **Patterns and Templates** ▶ **Templates**.
2. Expand the **Local Templates** node and the **Java Link** node.
3. Right-click the template in this section and select **Open**. The link template editor opens.

Related Procedures

Using Conditions in Templates

Using the Package Template Editor

You can edit package templates though the Templates view.

To edit a package template

1 Open the **Templates** view: From the main menu, choose **Window** ▶ **Show View** ▶ **Other** ▶ **Patterns and Templates** ▶ **Templates**.

2 Right-click a template in the Java Package templates section and select **Open**.

Related Procedures

Using Conditions in Templates
Editing Templates

Related Reference

Template Variable Types

Validating Pattern Definition Projects

To avoid errors and inconsistencies, you need to check if the pattern you designed is valid and can be compiled before creating a pattern definition from your Pattern Definition project.

Pattern validation checks for the following:

- Every instance specification in the project has a non-null classifier assigned via `InstantiatesLink`. The classifier is a class from the “metamodels” root of the model.
- Every slot of every instance in the project has a defining feature that can be found as an attribute of the class representing the metaclassifier of the slot’s owning instance specification.
- Slots are assigned only those values that are acceptable for corresponding features.
- Every instance specification representing a link must be tied by an aggregation-shaped link (`PatternDefinitionAssociationLink` with property `instanceAggregated` set to `true`) to an instance of a non-link metaclass.
- Every link instance must have two participants attached to it by means of `PatternDefinitionAssociationLink` with property `instanceAggregated` set to `false`. The link stereotype must be set to `client` or `supplier`.
- The link client and supplier attached to a link instance by association links must comply to the metamodel description (classifiers of the link participants must inherit metaclasses mentioned in the descriptions of the corresponding link participant role).
- No cyclical aggregations are allowed in the pattern definition model.
- Each constraint in the pattern definition project must have the `constraintType` property set to the name of one of the available constraints.
- Each constraint must be linked by `ConstraintParameterLinks` to all parameters that this constraint is checked against. Each such link must have its parameter role set in the `parameterRole` property. Roles of the parameter links must not duplicate one another, and all parameters of the constraint should be defined.
- The union of all participant sets selected for pattern parts should not be equal to the set of all participants in the definition.

To validate a pattern definition

1. In the Model Navigator view, select your Pattern definition project
2. Right-click the selection and choose **Patterns** ▸ **Check pattern definition project validity** from the context menu.

The validation results are displayed in the **Pattern definition validation results** view.

Related Reference

Last Validation Results View

Verifying Pattern Instances

If modification of a pattern instance violates its validity, the pattern instance is displayed red in the diagram. Verification of a pattern instance helps update relationships between the pattern and participants, and makes the pattern valid, if possible. In case such an update is not possible, Together displays a warning message.

To verify a pattern instance

1. Select a pattern oval in the diagram and right-click it.
2. On the context menu, choose **Patterns** ▶ **Verify pattern**.

Tip: Alternatively, choose **Verify pattern** on the context menu of the pattern in the **Pattern Explorer**.

You might want to get rid of invalid pattern instances. Use the Pattern Explorer context menu command.

To delete invalid pattern instances

1. In the **Pattern Explorer**, right-click a pattern node.
2. On the context menu of the node, choose **Clear Invalid Instances**.

Related Concepts

Patterns and Templates

Related Reference

Pattern Explorer

Working with the Pattern Instances

Managing pattern instances involves the following procedures

1 Create a pattern instance in diagram:

   Creating Model Element by Pattern

2 Create additional parts in a pattern:

   Adding a Pattern Part

3 Check validity of a pattern instance:

   Verifying Pattern Instances

4 Examine a diagram for patterns:

   Recognizing Patterns

5 Delete pattern instances:

   Deleting Patterns Instances

Related Concepts

Patterns and Templates

Related Reference

Pattern Explorer

Creating Model Element by Pattern

To create model elements by pattern

1. On the main menu, choose **File** ▶ **New** ▶ **Other**.
2. Expand the **Modeling** node and select **Model element by pattern**.
3. Select patterns to apply and click **Next**. You can optionally define the values for user-editable properties of the pattern and select whether to add a pattern instance into the model. If you choose to add a pattern instance, the new entity is displayed in the Patterns root on your project and in the Pattern Explorer, and the oval is displayed in the diagram.
4. Click **Finish** to create elements based on the selected pattern definition.

Tip: If your diagram looks entangled after adding pattern instances, use the **Layout** ▶ **Layout All** command on the diagram context menu.

Related Concepts

Patterns and Templates

Related Reference

Pattern Explorer

Adding a Pattern Part

To add a part to a pattern

1 Right-click an oval that represents the pattern instance in the diagram.

2 On the context menu of the pattern instance, choose **Create pattern part**.

3 On the submenu of this menu node, select a part to be created. The **Create Pattern Part** wizard opens.

4 On the first page of the wizard, specify the target package where the part will be created. Click **Next**.

5 On the second page of the wizard, specify properties for the new pattern part, using the **Set values** section.

6 Click **Finish**.

Related Concepts

Patterns and Templates

Related Reference

Pattern Explorer

Verifying Pattern Instances

If modification of a pattern instance violates its validity, the pattern instance is displayed red in the diagram. Verification of a pattern instance helps update relationships between the pattern and participants, and makes the pattern valid, if possible. In case such an update is not possible, Together displays a warning message.

To verify a pattern instance

1. Select a pattern oval in the diagram and right-click it.
2. On the context menu, choose **Patterns** ▸ **Verify pattern**.

Tip: Alternatively, choose **Verify pattern** on the context menu of the pattern in the **Pattern Explorer**.

You might want to get rid of invalid pattern instances. Use the Pattern Explorer context menu command.

To delete invalid pattern instances

1. In the **Pattern Explorer**, right-click a pattern node.
2. On the context menu of the node, choose **Clear Invalid Instances**.

Related Concepts

Patterns and Templates

Related Reference

Pattern Explorer

Recognizing Patterns

You can examine your project for patterns.

To recognize patterns in a project

1. Select a project in the Model Navigator.
2. On the context menu of the project, choose **Patterns ▸ Recognize Patterns**.

   The **Recognize Patterns** wizard opens.
3. In the **Choose Pattern Definitions** page, choose definitions of the patterns you want to find in the project. To do this, select patterns in the tree of available patterns, and click **Next**.
4. In the **Recognition Scope** page, select model elements. Use the **Add** and **Remove** buttons to make up the list of model elements to be examined. Click **Next**.
5. Observe the list of recognized patterns, or repeat the process with the other patterns and scope.

Warning: For the large projects, recognizing patterns increases memory consumption, which can result in the "out of memory" exception. Change the JVM parameters to increase memory limit. To do that, restart the application with the following Eclipse command line argument: `-vmagrs -Xmx600M`. This adds 600 MB of RAM. Depending on your hardware configuration, you can try the different additional memory values.

Recognition results are presented in a dialog box allowing you to choose instances that should be created and stored in the project model. After some of them are selected, new model entities are created in a special model node called "Patterns" and grouped by pattern definition instances in the Pattern Explorer view. References to pattern instances from a model can be created on any diagram by using the Add shortcut command.

Related Concepts

Pattern recognition

Deleting Patterns Instances

You can delete elements of the patterns using the Diagram Editor , the Model Navigator, or the **Pattern Explorer**.

To delete a pattern instance

1. In the Diagram Editor or Model Navigator, select the pattern instance to be deleted.
2. On the context menu, choose **Delete** to delete the selected pattern instance from the diagram and model.

Tip: Alternatively, select a pattern instance in the **Pattern Explorer** and choose **Delete instances** on the context menu. The selected instance is deleted from the model.

To delete all instances of a pattern

1. In the **Pattern Explorer**, select a pattern node.
2. On the context menu of the selection, choose **Delete instances**.

Related Concepts

Patterns and Templates

Related Reference

Pattern Explorer
Pattern Registry

Working with the Templates

Working with templates involves the following procedures

1. Edit, rename, delete and restore templates:

   Editing Templates

2. Use the class template editor:

   Using the Class Template Editor

3. Use the link template editor:

   Using the Link Template Editor

4. Use the package template editor:

   Using the Package Template Editor

5. Create conditional code templates:

   Using Conditions in Templates

Related Concepts

Patterns and Templates

Related Reference

Templates View

Editing Templates

You can edit templates through the **Templates** view. To open the **Templates** view:

From the main menu, choose **Window** ▶ **Show View** ▶ **Templates**. If **Templates** is not a menu option, choose **Window** ▶ **Show View** ▶ **Other** ▶ **Patterns and Templates** ▶ **Templates**.

To rename a template

1. On the **Templates** view, right-click a template and select **Rename**. This opens the **Rename Template** dialog box.
2. Enter the new name and click **OK**.

To modify a template

1. In the **Templates** view, right-click a Template.
2. Select **Open**. This opens the **Template** editor.
3. Make your edits.

There are three separate editors: one for link templates, one for class templates, and one for package templates. Some edits are entered in a text box, others through a dialog that opens when you click **Edit**.

To restore a template that you have changed to its original state

1. In the **Templates** view, right-click a template.
2. Select **Restore**.

Note: This option is available only after changes to a template have been saved.

To delete a template

1. In the **Templates** view, right-click a template.
2. Select **Delete**.
3. Confirm the deletion in the **Delete** dialog box.

You can restore templates that you have deleted. You can do this only at the category level, which means all templates deleted from the category are restored simultaneously. You cannot restore deleted templates individually.

To restore deleted templates

1. In the **Templates** view, right-click the category for the deleted template.
2. Choose **Restore deleted** from the context menu.

Related Concepts

Patterns and Templates

Using the Class Template Editor

To edit class templates in the Templates view

1. Open the **Templates** view: From the main menu, choose **Window** ▸ **Show View** ▸ **Other** ▸ **Patterns and Templates** ▸ **Templates**.
2. Right-click a template in the Java Class templates section, and select **Open**.

By opening and exploring the default templates that come with Together, you can get a better understanding of how easy it is to create your own class templates.

Related Procedures

Using Conditions in Templates

Related Reference

Template Variable Types

Using the Link Template Editor

You can edit link templates though the Templates view.

To edit a link template

1 Open the **Templates** view: From the main menu, choose **Window** ▸ **Show View** ▸ **Other** ▸ **Patterns and Templates** ▸ **Templates**.
2 Expand the **Local Templates** node and the **Java Link** node.
3 Right-click the template in this section and select **Open**. The link template editor opens.

Related Procedures

Using Conditions in Templates

Using the Package Template Editor

You can edit package templates though the Templates view.

To edit a package template

1 Open the **Templates** view: From the main menu, choose **Window** ▶ **Show View** ▶ **Other** ▶ **Patterns and Templates** ▶ **Templates**.

2 Right-click a template in the Java Package templates section and select **Open**.

Related Procedures

Using Conditions in Templates
Editing Templates

Related Reference

Template Variable Types

Using Conditions in Templates

When using the template editor to write the code snippets and syntax for fields, methods and classes, you can write conditional logic to control what is created when the template is applied.

The following example shows how conditional logic applies to the Applet template.

To apply conditional logic to the Applet template

1. From the main menu, choose **Window ▸ Show View ▸ Other ▸ Patterns and Templates ▸ Templates** to display the **Templates** view.
2. Expand the **Local Templates** node, then **Java Package**, and then **Standard**.
3. Double-click the Applet template to open the template editor.
4. Click the **Variables** tab. Notice the Boolean variable named `init`, which controls the code generation by the Applet template.
5. Click the **Units** tab. Notice that `init`, the Boolean variable declared on the Variables tab, controls the creation of the methods `init()` and `destroy()`.

Note: Variable names declared on the Variables tab must be preceded by a `$` symbol. Otherwise, they are treated as normal text.

Related Reference

Syntax and Conditions in Templates

Together Quality Assurance

This section provides how-to information on using Together Audits and Metrics.

In This Section

Searching QA Results
How to search in the source code QA results.

Specifying Quality Assurance Preferences
How to perform quality assurance tasks.

Using OCL in Model Audits and Metrics
How to use OCL expressions in Audits and Metrics.

Using QA History
How to use QA results history.

Viewing and Finding QA Descriptions
How to view and search source code QA results descriptions.

Viewing Audit Results
How to view audit results.

Viewing Metric Results
How to view metric results.

Viewing Metrics as Graphs
How to view metrics as graphs.

Viewing Problem Detection Audits (Detection Metrics)
How to view Problem Detection Audits.

Copying QA Results to Clipboard

To quickly copy the currently displayed QA results and paste them into any document

1. Select one or more QA results by using CTRL+CLICK.
2. Right-click any of the highlighted results and choose **Copy**.
3. Open any document and use the clipboard **Paste** option (or CTRL+V) to paste the copied text.

Your results will be pasted in formats similar to the following:

Audits:

```
Statement is unreachable
\33151\Chill.hpp:22
```

Metrics:

```
Resource SaleDM Metric:AC: Metric:AHF: Metric:AID: 0 Metric:AIF: Metric:AIUR:
 Metric:ALD: 4 Metric:AOFD: Metric:AUF: Metric:CBO: Metric:CC: 1 Metric:CF:
 Metric:CIW: Metric:CL: Metric:CM: Metric:COC: Metric:CR: Metric:ChC:
 Metric:DAC: Metric:DD: Metric:DOIH: Metric:FO: Metric:HDiff: Metric:HEff:
 Metric:HPLen: Metric:IUR: Metric:LCOM3: Metric:LOC: 10 Metric:MHF:
 Metric:MIC: Metric:MIF: Metric:MNOB: 0 Metric:MNOL: Metric:MPC: 6
 Metric:MSOO: Metric:NAM: Metric:NCC: Metric:NIC: 0 Metric:NOA: Metric:NOAM:
 Metric:NOC: Metric:NOCC: Metric:NOCP: Metric:NOED: Metric:NOIS: Metric:NOLV:
 0 Metric:NOM: Metric:NOO: Metric:NOOM: Metric:NOP: 1 Metric:NOPA:
 Metric:NORM: Metric:PC: Metric:PF: Metric:PIS: Metric:PS: Metric:PUR:
 Metric:RFC: Metric:TCC: Metric:WCM: Metric:WMPC1: Metric:WOC:
```

Related Procedures

Running Source Code Metrics
Running Source Code Audits

Creating a Metrics Chart

You can create a chart in the **Metric Results Pane**.

Metrics charts are created in temporary files, which are deleted when the charts are closed. However, you can save graphical information in text files, export it to any graphical format, and include graphics in the project that way.

To create a bar chart

1. Select a column that contains the result for a particular metric.
2. Right-click the column and choose **Bar Graph**.

To create a Kiviat chart

1. Select the row that contains the results for a particular element.
2. Right-click the row and choose **Kiviat Graph**.

To auto-update a Kiviat chart while browsing metric results

1. After the Kiviat chart is created, switch to the Metrics view.
2. Right-click any row and choose **Link Kiviat Graph**. The Kiviat chart will update every time you change the highlighted row in the Metrics view.
3. To disable the auto-update, right-click any row in the Metrics view and choose **Unlink Kiviat Graph**.

Note: The Link Kiviat Graph menu command is available when the Kiviat chart is created based on the source code modeling project.

To save a chart as an image

1. In the chart view pull-down menu, choose **Save Image As BMP Picture...** or **Save Image As SVG Picture...**.
2. In the **Save Chart As Image** dialog box, navigate to the target location and click **Save**.

To print a chart

1. Browse to the chart you want to print.
2. In the chart view pull-down menu, choose **Print...**.

Related Concepts

Quality Assurance

Related Procedures

Viewing Metric Results

Creating and Using Code QA Sets

For code audits and metrics, Together uses a default set of values.

To create and use your own set of values

1. Select **Window** ▸ **Preferences** on the main menu.
2. Expand the **Modeling** node, expand the **QA Source** node and select either **C++** or **Java**.
   Alternatively, right-click the QA results table (after you have run audits or metrics) and select **Preferences**.

To create a set of audits or metrics

1. Select either the **Audits** or **Metrics** tab.
2. Use the check boxes to select categories to include in your set. Use **Select All** or **Clear All** to select or deselect all boxes at once.
3. Use the editor in the lower section of each tab to edit the parameters.
4. Click **Save** and choose a location for the set. To share this QA set, you can save it with your project and share it using version control. The file is saved with a `.qa` extension.

Note: This new set now becomes the default set that is used when you run a QA task.

Note: To load an existing set of audits or metrics, click **Load the set of options from a file** in either **C++** or **Java** under the **QA Source** page of the **Preferences** dialog box.

Related Procedures

Using Version Control and Teams in Together
Running Source Code Audits
Running Source Code Metrics
Using OCL in Model Audits and Metrics

Related Reference

Quality Assurance

Exporting and Importing Model Audits/Metrics

To export model audits or metrics to a file

1. Choose **Window ▸ Preferences** on the main menu.
2. Expand the **Modeling** node and select the **QA Model** node.

   Alternatively, right-click the QA results table (after you have run model audits or metrics) and select **Preferences**.
3. Select either the **Audits** or **Metrics** tab.
4. Use the check boxes to select categories that you want to export.

   Tip: Use **Select All** or **Clear All** to select or deselect all the boxes at once.
5. Use the OCL editor in the lower section of each tab to edit the parameters.

   Tip: You can edit the context and values of each audit or metric. The editor provides code highlighting and validation features.
6. Click **Export** and choose the location of the exported file. To share your customized audits and metrics, you can save it in your project and share it using the version control.

   The exported file has an `.modelMetrics` or `.modelAudits` extension and contains the full information about your audits or metrics, including names, query bodies, and severity.

To import model audits or metrics

1. Choose **Window ▸ Preferences** on the main menu.
2. Expand the **Modeling** node and select the **QA Model** node.

   Alternatively, right-click the QA results table (after you have run model audits or metrics) and select **Preferences**.
3. Select either the **Audits** or **Metrics** tab.
4. Click **Import** and choose a previously exported file with an `.modelMetrics` or `.modelAudits` extension.

   Warning: The audits or metrics in the imported file *completely* overwrite your current audits or metrics settings, including names, OCL query bodies, and severity.

Related Procedures

Using Version Control and Teams in Together
Running Source Code Audits
Running Source Code Metrics
Using OCL in Model Audits and Metrics

Related Reference

Quality Assurance

Exporting QA Results

In this topic, you will learn how to export audit and metric results to XML or HTML files so that you can share them with team members or review them later.

To export the audit or metric results to a separate file

1. Select the rows of the table that you want to save. Do not select anything if you want to print the entire list.
2. Right-click the results view and choose **Export**. The **Export QA results to file** dialog box displays.
3. Type the path and file name you want to use or click **Browse** to specify the path. By default, the QA results are saved in the current project directory under the `/out/qa/` folder.

   Note: You do not need to add an extension to the file name.

4. Select the type of the file you want to export the results to in the **Type** list and specify auxiliary export options.

   Note: The available formats include **Text file (comma separated)**, **Text file (tab separated)**, **HTML file**, **Summary HTML report** (only for Audit export), and **Save in loadable format** for further importing the results to Together.

   Tip: For large scale projects, use the HTML format, which lets you export results to multiple connected HTML files.

5. Click **OK**.

Related Procedures

Running Source Code Audits

Flagging Audits in Code

For online audits, the affected source code can be flagged in the Editor view.

To flag audit results

1. In the **Model Package Explorer** view, right-click a project.
2. Select **Properties** from the context menu.
3. Select **QA Builder** in the pane on the left.
4. Check **Run incremental QA Builder** in the right pane. (This option can also be set after you run audits to flag the results.)
5. Click **OK**.
6. Run audits.
7. Double-click any result in the **Audit results** table to open the code in the **Editor** view. Results are indicated by the icons in the editor view's marker bar on the left. Possible Quick Fix solutions are offered for certain items (indicated by the light bulb icon).

Note: For more information on using Quick Fix, refer to the documentation set provided with your IDE.

You can also activate the QA Builder automatically for each new project created using the Quality Assurance Preference pages.

To add the QA Builder to new projects automatically

1. Choose **Window** ▸ **Preferences** from the main menu.

   The **Preferences** dialog box is displayed.
2. Choose **Modeling** ▸ **QA Source**.
3. Check the **Add QA Builder to new projects automatically** check box and click **OK**.

Related Procedures

Running Source Code Audits

Generating QA Report

After you run audits or metrics, you can generate reports based on the findings.

To generate QA reports

1. Right-click the QA results table and select **Export Results**.
2. In the **Save as** field, type the path and the name for the new report.

   Note: Directories and files that do not exist will be created during the process.

3. In the **Type** field, select the output format. For a list of all possible file formats, see “Audit View” and “Metric View” in the Reference section.
4. Click **OK**.

Related Procedures

Running Source Code Metrics
Running Source Code Audits
Exporting QA Results
Running Audits and Metrics from the Command Line

Related Reference

Audit View
Metric View

Grouping and Ungrouping

You can group the results of generated audits using one of four different categories:

- Severity
- Description
- Resource
- In Folder

To group audit results

1. Right-click the audit results table.
2. Select **Group by** and a category.
3. Together groups the results based on your choice.

To ungroup audit results

1. Right-click the audit results table.
2. Select **UnGroup**.

Related Procedures

Running Source Code Audits

Hiding and Showing Audit Results

You can hide or show specific audit results using the following criteria:

- Row(s)
- Description
- Resource
- Folder

To hide results

1. Right-click the audit results table.
2. Select **Hide Selected**.
3. Select a category from the submenu.
4. Together hides that category based on the audit row selected.

Tip: Use SHIFT+CLICK or CTRL+CLICK to select more than one audit row.

To show hidden results

1. Right-click the audit results table.
2. Select **Show All Hidden**.

Related Procedures

Running Source Code Audits

Navigating to Problems

You can jump directly from problems identified in the QA results table to the corresponding section of your code.

To navigate to the corresponding section of your code from the QA results table

1. Right-click the result.
2. Select **Go To**.

   Alternatively, double-click the result in the table.

   This will open the source file in the editor and highlight the problem line. If the source file is already open in the editor, it brings that tab to the front.

Related Procedures

Running Source Code Metrics
Running Source Code Audits

Printing Audit Results

You can print the entire table of audit violations using the Print command on the **Audit** view context menu.

Warning: This feature is available for implementation projects only.

To print the list of audit violations

1. Navigate to the audit results you want to print.
2. Right-click the **Audit** view field. The system **Print** dialog box opens.
3. If necessary, adjust the page and printer settings.
4. Click **Print** to send the file to the printer.

Related Concepts

Quality Assurance

Related Procedures

Viewing Audit Results

Related Reference

Print Audit dialog box

Refreshing QA Results

To rerun the same QA check on selected resources from the QA results table, perform one of the following

1. Right-click the results table and select **Refresh**.
2. Click **Refresh** on the **Audit** view toolbar.

Related Procedures

Running Source Code Metrics
Running Source Code Audits

Running Audits and Metrics from the Command Line

Use the `audit.cmd`, `metric.cmd`, `model-audit.cmd`, and `model-metric.cmd` command files to run source code and model audits and metrics from the command line and store the results in a file of the desired format. The command files are located in the Together installation folder.

Before you run audits or metrics from the command line, make sure that the path to the JDK/JRE installation folder is added to the PATH environment variable, and the target project is opened. If the target project is located in workspace other than `%ECLIPSEHOME%/workspace`, make sure to pass `-data workspacePath` to the launcher, or change `%TG_WORKSPACE%` parameter in cmd file.

To run source code and model audits and metrics on platforms other than Windows, use the following shell script files: `audit.sh`, `metric.sh`, `metric-report.sh`, `model-audit.sh`, and `model-metric.sh`.

Note: Search Linux help for instructions on how to run the files on those platforms.

To run audits or metrics from the command line (for Windows):

1. Click the **Start** button and then choose **Run...**.

   The Windows command prompt is displayed.
2. Change the current directory to the Together installation folder (for example, `cd C:\Borland\Together`).
3. To display the list of available command line options for the required command file, enter `audit --help` or `metric --help` for source code audits or metrics, or `model-audit --help` or `model-metric --help` for model audits or metrics respectively.
4. Compose the command line using the options listed. Specify the audit or metric target, the location of the result file, and the file format.
5. Click ENTER to run the specified audit or metric.

Related Concepts

Exporting and Importing Audits and Metrics

Related Procedures

Running Source Code Metrics
Running Source Code Audits

Running Model Audits and Metrics

To run model audits or metrics

1. Select one or more model elements you want to run model audits or metrics against.
2. Choose **Model** ▸ **Run Model Audits** or **Model** ▸ **Run Model Metrics** from the main menu.

Note: Model audits and metrics are performed recursively; that is, the program analyzes the selected element and all elements that are children of the selected one.

After you run model audits or metrics, the results are displayed in the **Model Audits** or **Model Metrics** view, respectively.

Related Concepts

Quality Assurance
Model Audits

Related Procedures

Running Source Code Metrics
Running Source Code Audits
Using OCL in Model Audits and Metrics
Running Audits and Metrics from the Command Line
Running Model Audits and Metrics as Ant Tasks

Related Reference

Model Audits View
Model Metrics View

Running Model Audits and Metrics as Ant Tasks

You can run model audits and metrics as Ant tasks. These tasks can be run either from the Eclipse user interface or the command line.

To create an Ant build script

1 Create an Ant build file with content similar to the following script:

```
<project default="build.modelQA">
        <target name="build.modelQA">
                <tg-audits file="c:/testout/audits.xmi">
                        <project name="OCL Sample"/>
                        <project name="AreaService Sample"/>
               </tg-audits>
               <tg-metrics file="c:/testout/metrics.xmi">
                        <project name="OCL Sample"/>
                        <project name="AreaService Sample"/>
               </tg-metrics>
        </target>
</project>
```

Ant tasks used to run audits and metrics are named `tg-audits` and `tg-metrics`, respectively. In this example, `c:/testout/audits.xmi` and `c:/testout/metrics.xmi` are output files and `OCL Sample` and `AreaService Sample` are the names of the Together modeling projects you run audits for.

2 Run the build file from either the Eclipse user interface or the command line.

To run audits and metrics as an Ant task from the Eclipse user interface

1 You must make sure that the script will be run in the same JRE as your workspace. To do this, right-click your build file in the Navigator View and choose **Run As** ▸ **Ant Build...**. In the **Edit Configuration** dialog, open the **JRE** tab and select the **Run in the same JRE as the workspace** option. This step must be completed before you run the build script for the first time. You do not have to perform this action every time.

2 Run the build file by right-clicking it in the Navigator View and choosing **Run As** ▸ **Ant Build**.

To run audits and metrics as an Ant task from the command line

1 In a command shell, run a command similar to the following one:

```
java -Xms128m -Xmx1024m -XX:MaxPermSize=256m -jar
./plugins/org.eclipse.equinox.launcher_1.0.101.R34x_v20080819.jar -data
<WORKSPACE_PATH> -nosplash -consolelog -application
org.eclipse.ant.core.antRunner -f <BUILDFILE>
```

where `<WORKSPACE_PATH>` is the path to your workspace and `<BUILDFILE>` is the path to your build file.

2 For more guidance in creating a shell command, refer to one of the root command line launchers provided at product installation for an example.

To work with the output

1. After the script outputs the results to an `audits.xmi` file (for audits) and a `metrics.xmi` file (for metrics), you can import the results into the corresponding view (Model Audits view for audits, Model Metrics view for metrics). From these views, you can also export the results into other formats, such as HTML.
2. If you want to transform the results into a proprietary HTML format or any other format, write a transformation in Java that uses EMF. Refer to the `com.borland.selena.ocl.gdm` plug-in for the Ecore and Java classes of the audits metamodel (registered by the `http:///com/borland/selena/audits.ecore` URI). The same metamodel is used for metrics.

   Note: The audits and metrics results do not contain model element information, such as the name of an element. Rather, the results refer to model elements by their unique identifier. Because of this, such a transformation would probably require the corresponding Together models on which you ran the audits and metrics. These models can be accessed by using the EMF API for Together models.

Note: Ant scripts for the audits and metrics of legacy models cannot be migrated to new DSL Toolkit audits and metrics.

Related Procedures

Running Model Audits and Metrics
Saving and Loading Audit Results
Saving and Loading Metric Results
Running Source Code Audits
Running Source Code Metrics
Exporting QA Results

Related Reference

Quality Assurance

Running Source Code Audits

Together adds a full range of QA audits to run against your code. The results can be compiled in a separate QA report or in a document generation you perform for your project. In Together, audits can be run on several levels of your development:

- Project: Run audits on all packages and classes/interfaces in a project.
- Package: Run audits on all the classes/interfaces within a package.
- Class: Run audits on only a particular class/interface.

To run audits

1. In the **Model Package Explorer**, **Model Navigator**, or **Packages** view, right-click any of the resources listed above.

   Note: You can choose multiple elements in multiple projects or multiple projects.

2. Choose **QA Source** ▸ **Audits...**.

   The **Run QA** dialog is displayed.

3. Select which resources to process from the list. Click **Preferences** to choose which audit categories/properties you want to run against the selection, or accept the current QA set.
4. Click **OK** to run the audits.

   The results are displayed in the Audit view.

Related Procedures

Generating QA Report
Running Audits and Metrics from the Command Line

Related Reference

QA Source
Audit View

Running Source Code Metrics

Together adds a full range of QA metrics to run against your code. The results can be compiled in a separate QA report or in a document generation you perform for your project. In Together, metrics can be run on several levels of your development:

- Project: Run metrics on all packages and classes/interfaces in a project.
- Package: Run metrics on all the classes/interfaces within a package.
- Class: Run metrics on only a particular class/interface.

To run code metrics

1. In the **Model Package Explorer**, **Model Navigator**, or **Packages** views, select any of the resources mentioned above.

   Note: You can choose multiple elements in multiple projects or multiple projects.

2. Choose **QA Source** ▸ **Metrics...**.

   The **Run QA** dialog box is displayed.

3. Select which resources to process from the list.
4. Click **Preferences** to choose which audit categories/properties you want to run against the selection, or accept the current QA set.
5. Click **OK** to run the selected metrics.

   The results are displayed in the Metric view.

Related Procedures

Generating QA Report
Running Audits and Metrics from the Command Line
QA Model

Related Reference

Metric View
Model Metrics View

Saving and Loading Audit Results

After you have run audits on a project or part of a project, you can save those results and view them whenever you like.

To save a set of audit results

1. Right-click on the **Audit results** table.
2. Select **Export**. The **Export QA results to file** dialog box opens.
3. Enter the path and file name you want to use. You can also click **Browse** to specify the path. By default, the audit results file is saved in the corresponding project directory under the `out/qa` directory.

   Note: You do not need to add an extension to the file name. Together will still open the file when you next load the results.

4. For the file Type, select **Save in loadable format** from the drop down list.

   Note: Other formats include Text file (comma separated), Text file (tab separated), HTML file, and Summary HTML report.

5. Click OK.

To load a set of audit results

1. Right-click a project, package, class, or interface in the Model Package Explorer or Model Navigator. (You can also select this command from the Audit View context menu.)
2. Select your saved audit results file.
3. Click **Open**.
4. View the results in the **Audit results** table.

Alternatively

1. Right-click on the Audit results table.
2. Select your saved audit results file.
3. Click **Open**.
4. View the results in the **Audit results** table.

Related Procedures

Running Source Code Audits

Saving and Loading Metric Results

After you have run metrics on a project or part of a project, you can save those results and view them whenever you like.

To save a set of metric results

1 Right-click the Metric results table and select **Export Metric Results**. The **Export QA results to file** dialog box opens.

2 Enter the path and file name you want to use. You can also click **Browse** to specify the path. By default, the metric results file is saved in the corresponding project directory under the `out/qa` directory.

> **Note:** You do not need to add an extension to the file name. Together will still open the file when you next load the results.

3 For the file Type, select **Save in loadable XMI format** from the drop down list.

> **Note:** Other formats include Text file (comma separated), Text file (tab separated), and HTML file.

4 Click OK.

To load a set of metric results

1 Right-click the Metric results table and select **Load Metric Results**.

2 Select your saved metric results file.

3 Click **Open**.

4 View the results in the Metric results table.

Related Procedures

Running Source Code Metrics

Searching QA Results

You can search through source code QA results for specified text.

To search the generated QA results for specified text

1. Right-click within the results table.
2. Select **Search in...**.
3. In the **Search in** field of the **Search** dialog box, specify which column to search. (In a metrics search, this is limited to Resource.)
4. Select which direction to move in the results table while searching.
5. Enter the text you are looking for.
6. Click **Search**.
7. The first row that matches the criteria is highlighted. Click **Search** again to move to the next matching row.

Related Procedures

Running Source Code Metrics
Running Source Code Audits

Related Reference

QA Search

Specifying Quality Assurance Preferences

Together provides a full range of code and model audits and metrics to run against your project. You can perform the following:

- Select which audits and metrics values to run against your project.
- Save customized "QA sets" to run against future builds.
- Create reports based on QA results.
- Include QA results in project documentation.
- View QA results as Kiviat or bar graphs.
- Sort, search, and copy your QA results.
- Alter the values used for quality assurance.

Note: Model audits and metrics results display in the Model Audits View and Model Metrics View. Code audits and metrics results are displayed in the Audits View and Metrics View. The actions available in Audits View and Metrics View, or in Model Audits View and Model Metrics View, are similar.

To open Quality Assurance preferences

1. Choose **Window** ▶ **Preferences...** from the main menu.
2. Choose **Window** ▶ **Preferences...** ▶ **Modeling** ▶ **QA Model** for model audits and metrics or **Window** ▶ **Preferences...** ▶ **Modeling** ▶ **QA Source** ▶ **C++ (or Java)** for source code audits and metrics.

The Quality Assurance Preferences contain lists of audits and metrics that you can select as well as tools for saving and loading (exporting and importing for model) QA sets.

Related Procedures

Activating Together Capabilities
Running Source Code Audits
Running Source Code Metrics
Running Model Audits and Metrics

Related Reference

QA Model
QA Source

Using OCL in Model Audits and Metrics

You can run audits and metrics in your design model using OCL expressions.

You can create custom OCL audits and metrics that operate with metamodel types and run them against the model that is an instance of the same metamodel. Together also contains a set of sample audits (see the following procedure to access them). The ideas of most of them are taken from Ambler and Fowler books. These audits can be used as examples for custom rules creation. For a description of the predefined model audits and metrics provided in Together, refer to "Model Audits and Metrics Descriptions."

To define audits and metrics

1. Choose **Window** ▸ **Preferences...** from the menu.

   The **Preferences** dialog box is displayed.
2. Expand the **Modeling** node and select **QA Model**.
3. Select either the **Audits** or **Metrics** tab.
4. Click **New** to add an audit or metric. The **Edit Audit** or **Edit Metric** dialog box is displayed, respectively.
5. Specify your audit or metric name, description, severity, and select the context of the OCL expression. The code for your new audit or metric is displayed in the standard OCL editor in the **Body** text area.

The audit expression should be a valid invariant that returns `Boolean`. Each metric expression should return an `Integer` value.

To run defined audits and metrics

1. In the diagram, select model elements against which you want to run audits or metrics.
2. Select **Model** ▸ **Run Model Audits** or **Model** ▸ **Run Model Metrics** from the main menu.

   The results are displayed in the **Model Audit** or **Model Metrics** view, respectively.

Note: The scope depends on the current selection made on the diagram. If the project default diagram is selected, the entire project will be checked. If a single element is selected, only this element will be checked.

Note: If the first operand of an audit returns false, the second operand is not computed and errors are logged. This ensures precise execution semantics and is necessary because the second operand computation may have side effects and may call mapping operations.

Related Concepts

OCL Support
Model Metrics

Related Procedures

Running Model Audits and Metrics

Related Reference

Model Audits and Metrics Descriptions
OCL

Using QA History

Each time you run QA, Together keeps a history of the results.

To use the QA results history

1. On the **Audit** or **Metric** view toolbar, locate the icon that looks like a clock.
2. Click the down arrow next to the icon to see a list of previous results.

   Note: This list only applies to your current session of Together. Once you close the application, the list is deleted. You can also clear the history by selecting the Clear History command beneath the list.

3. Select an item in the list to display those results in the table.

Related Procedures

Running Source Code Metrics
Running Source Code Audits

Viewing and Finding QA Descriptions

After running source code audits or metrics, you can view the related description for each result.

To view the related description for each result

1. Right-click the result row.
2. Select **Description**. A description of the related audit or metric opens in the help browser.

To find a particular audit or metric

1. Choose **Window** ▸ **Preferences** from the main menu.
   The **Preferences** dialog is displayed.
2. Choose **Modeling** ▸ **QA Source** ▸ **C++** or **Modeling** ▸ **QA Source** ▸ **Java**, depending on your project type.
3. Click **Find an analyzer**.
   The **Find Analyzer** dialog box is displayed.
4. Enter a string to search for in the **Choose an analyzer** field at the top of the dialog.
   The results are displayed in the lower portion of the dialog.
5. Double-click any of the results to display the appropriate audit or metric in the **Quality Assurance** list.
6. Double-click a result to view its description in the help browser.

Related Procedures

Running Source Code Metrics
Running Source Code Audits

Viewing Audit Results

When viewing audit results, you can compare and organize items in the results report.

The results report is tightly connected with the diagram elements and the source code. Using the report, you can navigate to the specific location of the violation.

Use the following techniques when viewing audit results

1. Sort all the items according to the values for a specific column
2. Group items according to the current column
3. Navigate to the specific location of the violation

To sort all the items according to the values for a specific column

1. Switch to the audit results table.
2. Click the column heading. The arrow in the heading displays whether sorting is ascending or descending.

To group items according to the current column

1. Right-click the Audit results table and choose Group By. This enables you to organize the results by changing the relationship of rows and columns.
2. To ungroup the results, right-click the table, and choose Ungroup.

To navigate to the specific location of the violation

1. Select any element in the results report.
2. Choose **Open** on the context menu or simply double-click the row to navigate directly to the location of the violation (a line of the source code for source code projects and a model element for design projects).

Related Concepts

Quality Assurance

Related Procedures

Running Source Code Audits

Viewing Metric Results

Use the following techniques when viewing metric results

1. Sort results by column
2. Filter results
3. Update results
4. Navigate to the source code
5. View the metric description

To sort results by column

1. Select the desired column in the metrics result table.
2. Click the column header to change the sorting order.

To filter results

1. You can filter the displayed results to improve the meaningfulness of the results report.
2. Use the following toolbar buttons to show and hide elements:

Button
Namespaces
Classes
Methods
Child elements

Note: Filtering is available only for source code metric results.

To update results

1. You can update or refresh the results table.
2. Use the following **Tool Palette Toolbox** buttons:

Button	Description
Refresh	Recalculate the results that are currently displayed
Restart	Open the Metrics dialog window, define new settings and start new metrics analysis.

To navigate to the location of the violation

1. Select the row in the results table that is of interest to you.

2 Right-click and choose **Open** on the context menu to navigate directly to it (a line of the source code for source code projects and a model element for design projects).

To view the metric description

1 Select the column in the results table that corresponds to the metrics of interest to you.

2 Right-click and choose **Show description** on the context menu.

Note: Description is available only for source code metric results.

Related Concepts

Quality Assurance
Running Source Code Metrics

Viewing Metrics as Graphs

Together can create visual representations of your metrics results.

- As Kiviat graphs are created for each resource, select the resource row you want to chart.
- When you choose Refresh from the Metric results table context menu, the graph clears. Follow the steps below to regenerate the graph based on the new results.

To view the metrics results as graphs for any row in the results table

1. For the Kiviat graph, right-click the row and select **Kiviat Graph**. The graph opens in a new view.
   - If the information is crowded or overlaps, expand the window.
   - To see the value of an individual data point on the Kiviat graph, hover over it to display a popup.
   - Red dots represent values above the maximum for the metric.
   - Blue dots represent values below the minimum for the metric.
   - Green dots represent values that fall within the required range of the metric.

   The graph view pull-down menu allows you to save your Kiviat graph as an image file or print the graph.
2. For the bar graph, right-click on the metric and select **Bar Graph**. The graph opens in a new view.
   - If the information is crowded or overlaps, expand the window.
   - The bar graph displays only those branches in the results that are expanded, so if you have the project node collapsed, you will see only one bar representing the project.

   The graph view pull-down menu allows you to save your bar graph as an image file or to print the graph.

Related Procedures

Refreshing QA Results
Running Source Code Metrics

Viewing Problem Detection Audits (Detection Metrics)

Some source code audits, known as problem detection audits, are based on a group of metrics. These indicate possible design problems that are broader in scope than individual audits can reveal. For more information on problem detection audits, see “Code Audits” in the Concepts section. You can recognize these audits because they have no data in the Location column.

To view the detection metrics

1. Right-click on the selected result.
2. Select **Detection Metrics**.

   The metrics table for the selected result opens. You can view the list and results of the metrics used for that audit.

Note: Problem detection audits and metrics are available only for source code projects.

Related Concepts

Code Audits

Related Reference

Metric View

Using Version Control and Teams in Together

This section describes the use of Version Control Systems (VCS) with Together. You can use VCS to keep track of changes and store versions of the work you do.

In This Section

Comparing and Merging Shared Models

Use the **Synchronize** view to compare shared models.

To compare and merge shared models

1. In the Team **Synchronize** view, select a model or model element version stored in the repository that you want to compare with the local version.
2. Choose **Model** ▸ **Compare With** ▸ **Local Version** from the main menu.

   Tip: Alternatively, right-click a model or model element and choose **Open In Model Compare Dialog** from the context menu.

   The comparison results display in the **Model Compare** dialog box.
3. Review the differences, apply your changes, and then commit the model to the repository using menu commands specific to your VCS.

Note: When merging a shared model, you can change your local version only. Together models consist of a large number of files, so you need to have all of these files locally.

Warning: Together performs merge on the model level, and existing file conflicts may still remain after the model merge. To commit these changes, use the "forced commit" mechanism provided by your version control system, (for example, the "Override and Commit" option in CVS).

Related Concepts

Model Compare and Merge

Related Procedures

Merging Models

Related Reference

Model Compare/Merge
EMF Model Compare Preferences

Setting Up ClearCase Support

The following steps describe how to set up Together to work with ClearCase.

To open a custom toolbar for working with ClearCase

1 From the main menu, select **Window** ▸ **Customize Perspective**. The **Select Perspective** dialog box opens.
2 Click the **Command** tab and check the **ClearCase for Modeling** option.
3 Restart Together for the changes to take effect.

To turn on ClearCase icon decorators

1 From the main menu, select **Window** ▸ **Preferences**.
2 Expand the **General** node.
3 Expand the **Appearance** node and select the **Label Decorations** node.
4 Check the **ClearCase SCM Adapter for Modeling Views** option.

Related Concepts

Version Control in Together

Related Procedures

Sharing Projects
Sharing Templates

Setting Up Repositories

The following steps describe how to set up CVS repositories for use in Together. This is a prerequisite to sharing projects.

To open the CVS Repository Exploring Perspective

1 From the main menu, select **Window** ▸ **Open Perspective** ▸ **Other**. The **Select Perspective** dialog box opens.

2 Choose **CVS Repository Exploring** from the list, and click OK.

Using this perspective, you can add repositories for various VCS systems.

Before you share a project, you need to define the repositories that Together will use. Prior to this, the CVS Repositories view is blank. Use the **Add CVS Repository** wizard to define a repository.

To define a repository

1 Right-click in an empty area of the **CVS Repositories** view, and select **New** ▸ **Repository Location**. The **Add CVS Repository** wizard opens.

2 Provide the information required by the wizard as listed below. When finished, click **Next**.

This list provides option descriptions.

- **Host** – Type the host name for your CVS server. For example, if your login command begins with: `CVS -d :pserver:/jane.doe@CVS-host`, enter: `CVS-host`
- **Repository path** – Enter the CVS repository as you would in the pserver section of the CVS login command. For example, if your login command begins with: `CVS -d :pserver:/jane.doe@CVS-host/repository_alias`, enter: `/repository_alias`
- **User** – Enter the user name. For example, if your login command begins with: `CVS -d :pserver:/jane.doe@CVS-host/repository_alias`, enter: `jane.doe`
- **Password** – Enter your valid password.
- **Connection type** – Choose from the list: pserver, ext, extssh
- **Use Default Port** and **Use Port** – Select **Use Port** to define a custom port for the connection. **Use Default Port** is enabled by default.
- **Validate Connection on Finish** – This option is checked by default. Leaving the option checked allows you to attempt to connect with the host server to ensure that all information was entered correctly.
- **Save Password** – Check to save your password locally on your computer. Note the warning message about the password file at the bottom of the wizard.

3 Click **Finish**. The **CVS Repositories** view is updated with the new CVS repository location.

For instructions on sharing a project, refer to Sharing Projects topic. Consult the documentation set provided with your IDE for complete details on using version control. From the main menu, choose **Help** ▸ **Help Contents**. You can find more information on CVS at http://ximbiot.com/cvs/wiki/index.php?title=Main_Page .

Related Concepts

Version Control in Together

Related Procedures

Sharing Projects
Sharing Templates

Sharing Projects

Before you can share a project, you need to define a repository location. If you have not already, you may do so by following the steps in “Setting up Repositories” topic. Or you can proceed with step one below and set up your repository in step two.

Note: When viewed in the Model Package Explorer (with filters off), a Java or Java Modeling Project contains at least two resource divisions:

- The first is your normal source code project structure: the directories/packages you have created and your source code.
- The second section is the Together Model directory. This contains the actual diagram files generated and used by Together and includes the .project and .classpath files.

To share a project using a project's context menu in the Model Package Explorer, Model Navigator or Navigator view

1. Right-click the project, and select **Team** ▸ **Share Project**.
2. In the first **Share Project** dialog box, choose a version control system. If using StarTeam, consult the StarTeam User Guide (located in the Contents section of this help viewer). Otherwise, choose CVS, and click **Next** to continue.
3. Choose whether you want to use the existing repository location or create a new repository location. If you choose to create a new location, the **Share Project** wizard opens as described in “Setting Up Repositories.” Click **Next**.
4. In the new dialog, choose either to use the project name as the module name, specify a different module name, or to use an existing module. Click **Next**.
5. In the final screen, the wizard explains your status in the process. Click **Finish**, and the wizard imports your project to the repository.

   From now on, when using the **Team** command from this project's context menu, the **Update** and **Commit** commands are enabled. Respectively, these options commit your changes to the repository and refresh your view with changes made by other team members.

   Recommendations and Tips

   - Use the **AutoCheckout modeling resource on edit** option on the **Modeling resources** page of the **Team** preferences to check out model files when Together attempts to change the corresponding model entities. This mode requires a VCS provider to mark files that are not checked out as read-only. For example, you can configure StarTeam for Eclipse (**Window** ▸ **Preferences** ▸ **Team** ▸ **StarTeam** ▸ **File**) by checking the **Mark unlocked working files read-only** and **Exclusively lock files on checkout**. In this case every unlocked file is marked as read-only and upon any attempt to modify this file in the **Java Editor**, **Diagram Editor** or **Model Navigator**, Eclipse asks StarTeam to check out the file and StarTeam displays a dialog box asking you to lock the file. You can also check the **Auto lock read-only files** option to avoid this dialog. If the file cannot be locked (set in the manual mode) or a server refuses to do it, all changes will be reverted to the previous state. If **AutoCheckout modeling resource on edit** is deselected but a VCS provider is configured to mark working files as read-only, all element context modification actions in the **Diagram Editor** and **Model Navigator** are disabled until the read-only flag is cleaned from the working files.
   - For sharing Together model projects with ClearCase, use the **ClearCase Adapter for Modeling** wizard.
   - Any source files/design diagrams that you created before you shared the project should be added to the repository.

- The project's project file can be shared, but only if the team will use the exact same project name.
- You should not version control the default package diagrams (those files with the .txvpck extension) because these change frequently and are easily regenerated automatically by Together if they are missing. To exclude the default package diagrams from sharing, choose **Preferences** ▸ **Team** ▸ **Modeling resources** and check **Ignore default package diagrams**.
- You can include the .classpath file if you are using variables to point to libraries, or if you are sure that each team member will have the libraries in exactly the same place (for Windows, this includes the drive letter).
- You can specify directories and files that you do not want to share: from the main menu, select **Window** ▸ **Preferences** ▸ **Team** ▸ **Ignored Resources**. Add "*.txvpck" and the Commit and Update actions will ignore these default package diagrams.

Note: Respectively, each team member would also have to set this up locally.

Refer to the documentation set provided with your program for additional details on using version control.

Related Concepts

Version Control in Together

Related Procedures

Setting Up Repositories
Sharing Templates

Related Reference

Sharing Design Elements: Special Considerations

Sharing Templates

By default, Together stores its ready-to-use templates in the `Local Templates` directory. If you change these templates, your changes will be available only locally.

Use the **Templates** view to store templates that you want to share with your team.

To store and share your templates using the Templates view

1. Open the **Templates** view (**Window ▸ Show View ▸ Other**, expand **Patterns and Templates** node and choose **Templates**).

   The **Templates** view displays.
2. Right-click the **Local Templates** node.
3. Choose **New**. From the submenu, choose the type of template you want to create (for example, class template, which is used here as an example).

   Tip: Alternatively, you can right-click a design element for which you want to create a template in the **Model Navigator** or **Diagram** view and choose the **Save As Template...** context menu option.

   The **Save Template** wizard is displayed.
4. Select the shared project where you want to store the template in the **Location** list.
5. Specify the name of the new template in the **Name** field.

   This creates a template directory within your project. When you return to the **Templates** view, you should be able to see your project. The new template is displayed under your project node, in the `Java Class` node within the `Misc` category.
6. Right-click the `Java Class` node and select **Create Category...** to add subsections to the node.

   The created categories are displayed in the location list under the project. Use categories to organize, manage, and share your templates. After you have shared the project, use the **Team** context menu command in the **Templates** view to add, commit, and update templates using VCS.

Related Concepts

Version Control in Together

Related Procedures

Sharing Projects

Related Reference

Sharing Design Elements: Special Considerations

Managing Requirements with Together

This section provides how-to information on using Together for creating requirements, managing traces, generating requirements documentation and more.

In This Section

Creating Requirements Based on Use Case

This topic describes how to create CaliberRM or RequisitePro requirements from use case diagrams.

You can create traced requirements directly from Use Case model elements. Before creating requirements, make sure that you have set up **Create traces between Requirement and Use Case** as enabled in **Requirements Preferences**. Set up other related requirements preferences to suit your needs.

To create a requirement based on use case

1. Open a Use Case Diagram from which you want to create a new requirement.
2. In the diagram, right-click a use case element and choose **Requirements** ▸ **Create Requirement(s)...**.
3. The **Create Requirement** dialog box is displayed. In the dialog box, select an appropriate requirement under which a new requirement will be created.
4. Click **OK**. The new requirement receives the name of the traced use case element. The name of the traced element will be displayed in green.

Tip: You can select multiple use case elements in the Diagram Editor or **Model Navigator** (using SHIFT and CTRL keys) and create requirements for them. When you select the traced use case element on the diagram, you can see a trace to the requirement in the **Element Traces** view.

Related Concepts

Requirements Management

Related Reference

Element Traces View

Creating Traces from Requirements to Model Elements

There are several ways to create a trace between a Together model element and a requirement.

To create a trace from an element to a requirement using the Together Diagram editor

1. Open the diagram from which you want to create a trace to a requirement.
2. Right-click the appropriate element and choose **Requirements** ▶ **Manage Traces** from the context menu.
3. In the **Manage Traces** dialog box, select the tab with requirements of a particular vendor and expand the project nodes to locate the appropriate requirement.
4. Select the requirement and click **Add**. The selection moves to the **Selected** pane.
5. Click **OK**.

To create a trace from an element to a requirement using the Model Navigator

1. In the Model Navigator view, right-click an element and choose **Requirements** ▶ **Manage Traces** from the context menu.
2. In the **Manage Traces** dialog box, select the tab with requirements of a particular vendor and expand the project nodes to locate the appropriate requirement.
3. Select the requirement and click **Add**. The selection moves to the **Selected** pane.
4. Click **OK**.

Related Concepts

Requirements Management

Related Procedures

Deleting Traces

Related Reference

Manage Traces Dialog

Deleting Traces

To delete a trace using the Diagram editor

1. Open the diagram in which you want to remove a trace.
2. In the diagram, right-click the appropriate element and select **Requirements** ▸ **Manage Traces** from the context menu.
3. In the **Selected** pane of the **Manage Traces** dialog box, select the traces you want to remove and click **Remove**. To remove all traces, click **Remove All**.
4. Click **OK**.

To delete a trace using the Element Traces view

1. Open the diagram in which you want to remove a trace.
2. Open the **Element Traces** view.
3. In the diagram, select the appropriate element. Traced requirements and/or model elements are displayed in the **Element Traces** view.
4. Right-click a trace you want to remove and select **Remove** from the context menu.
5. Click **OK**.

Warning: Traces are deleted without confirmation. You might be prompted to log on to the repository that stores the traced requirements.

Related Concepts

Requirements Management

Related Procedures

Creating Traces from Requirements to Model Elements

Related Reference

Manage Traces Dialog

Generating Documentation for Requirements

You can generate documentation from the requirement point of view using one of the default Together templates.

To generate documentation for requirements

1. Choose **File** ▸ **Export** and then select **Documentation Using Template** in the **Export** dialog box.

   Alternatively, choose **Project** ▸ **Documentation** ▸ **Documentation Using Template**.

   Click **Next**.
2. In the opened step of the wizard select the **Requirements Online Report** template in the **Default** list and specify other options.
3. Click **Finish** to generate documentation.

Related Concepts

Requirements Management

Related Reference

Trace Synchronizer View
Element Traces View

Modifying Requirement Preferences

The Requirements preferences control how Together creates requirements from Use Case diagram elements (and vice versa) and how it updates traces on a traced element move.

To modify the Requirements preferences

1. Select **Window** ▸ **Preferences** from the main menu. The **Preferences** dialog box is displayed.
2. In the dialog box, expand **Modeling** and select **Requirements**. The **Requirements Preferences** are displayed.
3. Modify the preferences and close the dialog saving your settings.

The Requirements preferences also include a section specific to CaliberRM. When converting legacy traces imported from TCC to the current format, use these preferences to specify which CaliberRM server stores the imported data.

To modify the CaliberRM Requirements preferences:

1. Select **Window** ▸ **Preferences** from the main menu. The **Preferences** dialog box is displayed.
2. In the dialog box, expand **Modeling**, then **Requirements** and then select **CaliberRM**. The **CaliberRM Requirements Preferences** are displayed.
3. Modify the preferences and close the dialog saving your settings.

Related Reference

CaliberRM

Navigating from Model Elements to Requirements

To navigate from a model element to a requirement

1. Right-click a traced model element in either the **Diagram** editor or **Model Navigator** view. In the **Model Navigator**, traces are displayed with a decorator; on the diagram, the names of traces are displayed in green.
2. Point to **Requirements** and select the traced requirement you want to view.
3. The requirement is selected in the **CaliberRM Navigator** or **RequisitePro** view, depending on the requirement type.

Related Concepts

Requirements Management

Related Procedures

Viewing Element Traces

Opening Requirements Views

You can manage traces between requirements and model elements using dedicated Requirements views.

To open the Requirements views

1. On the main menu, click **Window** ▸ **Show View** ▸ **Other**.
2. In the **Show View** dialog box, expand **Requirements**.
3. Using the SHIFT or CTRL key, select **Trace Synchronizer** and **Element Traces** views under the **Requirements** folder.
4. Click **OK**.
5. The new views open in the Eclipse framework.

Tip: For ease of use you can redock the views by clicking and dragging the title bars.

Related Concepts

Requirements Management

Related Reference

Trace Synchronizer View
Element Traces View

Searching for Traced Elements

You can search for specific elements that have traces to requirements or find all traced elements in the defined search scope using the **Requirement Traces Search** dialog.

To find specific traced elements

1. Choose Search ▸ Requirement Traces Search... from the menu.

   The **Search** dialog is displayed.
2. Select the **Requirement Traces Search** tab.
3. In the **Containing Text** field, define a search string.
4. Check the **Case Sensitive** check box if you want your search to be case-sensitive.
5. If your search string contains regular expressions, check the **Regular expression** check box.
6. In the **Available Attributes** field, select in which trace attributes you want to search (server name, project name or requirement name).
7. In the **Available Integrations**, select the Integration plug-ins (CaliberRM and/or RequisitePro) requirements you want to search.
8. In the **Scope** area, define the search scope (workspace, enclosing projects or a working set).
9. Click **Search**.

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

To find all traced elements

1. Choose Search ▸ Requirement Traces Search... from the menu.

   The **Search** dialog is displayed.
2. Select the **Requirement Traces Search** tab.
3. Click **OK**. The list of found desynchronized traces is displayed in the **Trace Synchronizer** view.
4. Check the **Search All Traced Elements** check box.
5. In the **Scope** area, define the search scope (workspace, enclosing projects or a working set).
6. Click **Search**.

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

Related Concepts

Requirements Management

Related Reference

Requirement Traces Search Dialog Box

Synchronizing Traces

To synchronize traces for a diagram element using Diagram Editor

1. In the Diagram editor, right-click a traced element for which you want to synchronize traces and choose **Requirements ▸ Synchronize Traces**.
2. The **Trace Synchronizer** view opens displaying the list of the traces associated with the selected element. During the operation, you might be asked to provide your login credentials to access the requirements project.
3. In the view, sort the list by **Status** to find problematic traces.
4. Review the list and individually decide how to handle each problematic trace. The details on every synchronization problem are given in the **Status summary** field. You can update a local trace from the repository, restore a local copy of the trace, or delete an obsolete trace.
5. Right-click a trace or select multiple traces and choose an appropriate option from the context menu.
6. To monitor changes in the view, use the **Refresh** button.

To synchronize traces for a requirement or a model element using Trace Synchronizer view

1. In the **Trace Synchronizer** view, click the **Synchronize** button. The **Trace Synchronizer** dialog box is displayed.
2. In the dialog box, define where to search for desynchronized traces. You can search either within requirements or model elements. To search in requirements, click the **Requirements** tab to define the search scope. You can select a CaliberRM requirement, requirement type, baseline or a server connection. During the operation, you might be asked to provide your login credentials to access the requirements project. To search in model elements, click the **Model Elements** tab to define the search scope. You can select a single model element within any modeling project available in your current Eclipse workspace.
3. Click **OK**. The list of found desynchronized traces is displayed in the **Trace Synchronizer** view.
4. Review the list and individually decide how to handle each problematic trace. The details on every synchronization problem are given in the **Status summary** field. You can update a local trace from the repository, restore a local copy of the trace, or delete an obsolete trace.
5. Right-click a trace or select multiple traces and choose an appropriate option from the context menu.
6. To monitor the changes in the defined search scope, use the **Refresh** button.

Related Concepts

Requirements Management

Related Procedures

Opening Requirements Views

Related Reference

Trace Synchronizer View
Trace Synchronizer Dialog Box

Viewing Element Traces

To view element traces using Element Traces view

1. Select a traced model element in either the **Diagram** editor or **Model Navigator**. In the **Model Navigator**, traces are displayed with a decorator; on the diagram, the names of traces are displayed in green.
2. Open the **Element Traces** view. The list of traces to and from the selected element is displayed in the view.
3. Using the context menu, you can open a traced requirement in the CaliberRM Navigator or RequisitePro view (depending on the requirement type) or delete a trace.

Related Concepts

Requirements Management

Related Procedures

Deleting Traces
Navigating from Model Elements to Requirements

Related Reference

Element Traces View

Generating Project Documentation

This section provides how-to information on using Together Documentation Generation facilities.

In This Section

Configuring the Documentation Generation Facility

To configure the documentation generation facility

1. On the main menu, choose **Window** ▶ **Preferences** ▶ **Generate Documentation**.
2. Under the Generate HTML category, enter the options for HTML documentation:

   classes and members to be included in the documentation

   tags to be included in the documentation

   documentation title, window title, header, and footer.
3. Under the HTML Output category, choose the option of processing line breaks.
4. Under the RTF Output category, set up text formatting options

Related Concepts

Documentation Generation Overview

Related Reference

Generate Documentation Preferences

Generating HTML Documentation

In this section you will learn how to generate project documentation in HTML format using the default template supplied with the product.

To generate HTML project documentation

1. Select **Project** ▶ **Documentation** ▶ **Generate HTML**on the main menu.
2. In the **Generate HTML Documentation** dialog box that opens, specify the output folder, and select your preferred **Scope** and **Options** settings.
3. Click **Finish** to generate documentation.

Related Concepts

Documentation Generation Overview

Related Procedures

Configuring the Documentation Generation Facility

Related Reference

Documentation Generation
Generate HTML Documentation dialog box

Generating Project Documentation as Ant Task

To generate project documentation as an Ant task

1. Create a new Ant buildfile
2. Specify the parameters for the `gendoc` utility. For example:

```
<gendoc workspace="C:\Work\Together\runtime-workspace" project="uml" package="package1"
outdir="c:\out" template="Project Report" format="RTF" nodiagrams="true"
hyperlinks="true" audits="true"/>
```

Example:

```
<project name="gendoc" default="exec">
<taskdef name="gendoc"
    classname="com.borland.gendoc.launchers.ant.GenDocTask"
    classpath="C:\Together\plugins\com.borland.gendoc.core\gendoc.jar"/>
<target name="exec">
    <gendoc
       workspace="C:\Together\workspace"
       project="uml20_test"
       package="package2"
       format="RTF"
       outdir="C:\tmp"
       audits="true"
       />
</target>
</project>
```

Related Concepts

Documentation Generation Overview

Related Procedures

Generating Project Documentation from Command Line

Related Reference

Gendoc Utility Syntax
Genhtml Utility Syntax

Generating Project Documentation from Command Line

You can update project documentation as part of a periodic automated build process by having the process script call the documentation generator in Together via the command line interface.

Together provides the following utilities that enable you to generate project documentation without launching the product:

- `genhtml` for generating HTML documentation, with the launchers `genhtml.cmd` and `genhtml.sh`
- `gendoc` for generating documentation by template, in one of the supported output formats, with the launchers `gendoc.cmd` and `gendoc.sh`

To generate project documentation using the documentation generation utility

1. Use the following command: `<utility> [project filename] [options] [packagenames]`
2. Specify parameters and options as described in the utility references.

Note: If you run the utility without parameters, documentation is generated for the entire workspace and stored in the `out` subfolder of the workspace.

Related Concepts

Documentation Generation Overview

Related Reference

Gendoc Utility Syntax
Genhtml Utility Syntax

Generating Project Documentation Using Template

In this section you will learn how to generate project documentation in the desired format, using one of the default or custom templates of your choice.

To generate project documentation using a template

1. Select **Project ▸ Documentation ▸ Generate Using Template** on the main menu.
2. In the **Generate Documentation Using Template** dialog box that opens, specify the following settings:
   - In the **Output Path** field, enter the fully qualified path to the target folder. Alternatively, use the browse button.
   - In the **Format** field, select an output format from the drop-down list.
   - In the **Templates** section, select a Default or Custom template.
   - In the **Scope** section, click the radio button for a scope. Note that you can generate documentation for all open projects in the workspace, for a single project, or for the current package or diagram.
   - In the **Include** section, select artifacts to be included in the generated output. Note that you can include audit results.
   - If you want to open the generated report immediately, check the **Open in Viewer** option.
3. Click **Finish** to generate documentation.

Related Concepts

Documentation Generation Overview

Related Procedures

Configuring the Documentation Generation Facility
Creating Custom Documentation Template

Related Reference

Generate Documentation Using Template dialog box

Together Documentation Templates Procedures

This section provides how-to information on creating and editing custom documentation templates using the Together Documentation Template Designer.

In This Section

[Setting Frame and Frameset Properties]
How to define properties of a frameset or a frame within a Frameset Template.

[Setting Section Properties]
After a section is created, define its properties.

[Setting Template Properties]
After a documentation template is created, you can define its properties.

[Using Word Documents in Documentation Templates]
It is possible to use styles, headers and footers of the Word documents in the custom documentation templates. In this section you will learn how to attach and detach a Word document.

A Typical Scenario of Creating a Custom Documentation Template

Creating a custom documentation template and populating it with sections and controls involves the following general steps:

To create and customize a documentation template

1. Create a stub template:

   Creating Custom Documentation Template
2. Create template structure:

   Creating Sections
3. Provide reusable elements:

   Creating Stock Sections
4. Define sections properties:

   Setting Section Properties
5. Define area properties:

   Setting Area Properties
6. Define template properties:

   Setting Template Properties
7. Add controls to the sections:

   Creating Controls
8. Customize controls:

   Moving, Resizing and Aligning Controls

Find useful information in the following related sections:

Related Procedures

Using Word Documents in Documentation Templates
Reusing documentation templates from TCC/TA 1.x

Creating Custom Documentation Template

To create a documentation template

1. On the main menu, choose **File ▸ New ▸ Other**
2. In the **New** dialog that opens, expand the **Modeling** node.
3. Choose **Documentation Template** and click **Next**.
4. In the **GenDoc Template Designer File** dialog, specify the following:
    - In the **File Name** field, enter the fully qualified name of the template file.
    - In the **Template type** field, choose a template type from the drop-down list (documentation template or frameset template).
5. Use the Template Designer toolbox or context menus to create the template structure.

Note: After a documentation template is created, you can view and modify its properties using the **Template Properties** dialog.

Related Procedures

Setting Template Properties

Related Reference

Documentation Template Properties

Creating Sections

A nested section can be created for an existing folder or iterator; a sibling section can be created for any existing section. For these sections, the **Insert Nested Section** or **Insert Sibling Section** toolbar buttons and menu commands are enabled, unlike the report header and footer, which lie outside the template body.

To create a new section in a documentation template

1. Select an existing section in the template.
2. Right-click an existing section in the scope pane, point to the **Insert Sibling Section** or **Insert Nested Section** on the context menu and select the type of new section.

   Tip: Alternatively, use the toolbar buttons of the Documentation Template Designer.

3. When necessary, determine the essential template information for the new section.
   - **Static section**: None required at creation.
   - **Element iterator**: Select the element metatype from the list of available types.
   - **Element property iterator**: Select the scope.
   - **Folder section**: None required at creation.
   - **Call to stock section**: Select the stock section from the list of existing sections. Refer to the section "Creating Stock Sections."
   - **Call to template section**: None required at creation.

4. Click **OK** to complete the insertion.
5. Set section properties as required.

Iterators and folder sections can also contain headers and footers. If such a section does not contain a header or a footer, its context menu provides **Add Header** or **Add Footer** commands. If a header or a footer exists, the context menu provides **Delete Header** or **Delete Footer** commands.

To add a header or a footer to an iterator

1. Select an existing element iterator or folder section.
2. On the context menu, choose **Add Header** or **Add Footer**.

Related Concepts

Documentation Template
Documentation Template Sections

Related Procedures

Creating Stock Sections
Setting Section Properties

Related Reference

Documentation Template Designer

Creating Stock Sections

Stock sections are reusable folders or iterators that reside in the template's collection of stock sections. Each stock section displays in a separate named tab in a documentation template.

In this section you will learn how to:

- Create stock sections from scratch
- Create stock sections from an existing section
- Delete stock sections
- Show stock section

To create a stock section

1. On the toolbar of the **Template Designer**, click the **New Stock Section (Element Iterator)** button or **New stock section (Folder)** button. The **New Stock Section** dialog opens.
2. In the **Name** field, enter the name of the new stock section.
3. For the element iterators, select a metatype from the list of available metatypes.
4. Check the **Intrinsic to Property Iterator** option, if necessary. If this flag is checked, the scope of the stock section root iterator or folder section depends on the Properties Iterator this section is called from: the only available iteration scopes for an element iterator are *customized* and *programmed*. For folder sections, it means that the metatype chooser tab is absent.
5. Click **OK**.

To create a stock section from an existing section

1. In the scope pane of a template, select an element iterator or folder section from which you want to create a stock section.
2. Right-click the section and choose **Copy into Stock** on the context menu.
3. In the **New stock section (Folder)** dialog, enter the stock section name.
4. Click **OK**.

To delete a stock section

1. Right-click the stock section tab be deleted.
2. On the context menu of the tab, choose **Remove Stock Section**.

To show a stock section for a call to stock section

1. Right-click the call stock section.
2. On the context menu of the section, choose **Show Stock Section**.

Related Concepts

Documentation Template
Documentation Template Sections

Related Reference

Documentation Template Designer

Setting Section Properties

After a section is created, define its properties. Section properties are defined in the **Properties** dialogs, which are specific for each kind of sections.

You can invoke the Properties dialog for the iterators and folder sections from the scope pane or from the details pane of a section. For the static sections, the Properties dialog is invoked from the scope pane only.

To set properties of a section

1. In the scope section of the **Template designer**, select a template section.
2. On the context menu, choose **Properties**.
3. Define properties as required and click **OK**.

Note: Properties dialogs are specific to each section type. Refer to the dialog descriptions for details.

Related Concepts

Documentation Template Sections
Enable Conditions

Related Reference

Element Iterator Properties
Property Iterator Properties
Static Section Properties
Call to Stock Section Properties
Call to Template Properties

Setting Area Properties

Area properties apply to static sections, headers and footers. They are defined in the **Area Properties** dialog, which is common for static sections, headers and footers, and is invoked from the details pane. Refer to the dialog descriptions for details.

To set area properties

1. Select a static section, header or footer of a template.
2. On the context menu of the details pane, choose **Area Properties**.
3. In the **Area Properties** dialog, specify settings and click **OK**.

Related Concepts

Documentation Template Sections

Related Reference

Area Properties

Setting Template Properties

To set properties of a documentation template

1. On the toolbox of the **Template Designer**, click the **Show Template properties** button.
2. In the **General** tab, you can change the following:
   - Enter template description.
   - Define the report title expression, clicking the editor button to open the **Edit Expression** dialog.
   - Select Root Object Metatype from the list of available metatypes.
   - Attach a Word document as a formatting template. Refer to “Using Word Documents in Documentation Templates” for details.
   - Check or clear options to generate headers and footers as required.
3. In the **Page Settings** tab, you can specify page size, margins, and landscape or portrait orientation.
4. In the **Formatting Styles** tab, you can change formatting styles. Refer to “Creating Formatting Styles” for details.
5. In the **Template Parameters** tab, specify the formal parameters that will be used for calling this template from another template.

Related Procedures

Using Word Documents in Documentation Templates
Creating Formatting Styles for Documentation Templates

Related Reference

Documentation Template Properties

Creating Controls

To create a new control in a static section, header, or footer

1. Select a section in the details pane.
2. Click the type of control you want to insert on the template designer toolbar.

   Tip: Alternatively, you can right-click the section in the details pane, point to **Insert Control** and select the type of control from the menu.

3. On the context menu of a control, choose **Properties** and fill in the control properties in the tabbed **<Control> Properties** dialog box.
4. Click **OK**.

Note: The **Insert Control** command is also available on the context menu of a panel control.

Related Concepts

Documentation Template Controls

Related Procedures

Hyperlinking Documentation

Related Reference

Control Properties

Moving, Resizing and Aligning Controls

The Documentation Template Designer displays a newly created control as a rectangle positioned at the insertion point in the details pane. You can move the control to change where its output is displayed in the generated documentation. You can also modify the size of the rectangle to determine the approximate size of the region for the output. Increasing the rectangle size is especially important for a label with a default size that is not large enough to allow its entire text to be displayed in the report.

If you select two or more controls in a section, you can use the context menu of the selection to uniformly align controls within the section. Be aware that when you apply multiple alignments, it is possible for controls to overlap.

Tip: Precise positioning and sizing is not possible.

Warning: There is no simple undo for changes in alignment or size. When you change alignment or resize controls, you must manually readjust the controls to return them to the former status.

To move a control within a section

1. Select a control. Notice that the mouse pointer changes to a cross with double-ended arrows.
2. With the control selected, drag and drop the control to a position within the section.

To copy or move a control to another section

1. Select a control.
2. On the context menu of the selected control, choose **Copy** or **Cut**.
3. Right-click the target section and choose **Paste** on the context menu.

To resize a control

1. Place the mouse pointer on either the right or left edge of the rectangle. Notice that the mouse pointer changes to a double-ended arrow.
2. Drag the edge of the rectangle to the required size.

To align controls

1. Select two or more controls within a section.
2. Right-click the selection and choose **Alignment** on the context menu.
3. On the submenu, choose one of the following options:
   - Left Side
   - Right Side
   - Top Side
   - Bottom Side
   - Make same width
   - Make same height

- Make same size

Related Concepts

Documentation Template
Documentation Template Controls

Related Reference

Documentation Template Designer

A Typical Scenario of Creating a Template for Multi-Frame Documentation

This part discusses how to use the Documentation Template Designer to create the templates for multi-frame HTML documentation.

To create a template for multi-frame documentation

1. Create a stub multi-frame template:

   Creating Custom Documentation Template
2. Create the frameset structure to describe how the frames are organized within the browser window:

   Defining Frameset Structure
3. Design the body of a frameset template, keeping in mind that for the multi-frame templates, static sections and headers and footers for the folder sections and iterators are prohibited. Create controls. Set section, area and template properties as described in the section

   A Typical Scenario of Creating a Custom Documentation Template
4. Set call to template section properties:

   Setting Call to Template Section Properties
5. Create hypertext links, including image maps and Javadoc link references:

   Hyperlinking Documentation

Related Concepts

Multi-frame Documentation Templates

Related Reference

Call to Template Properties

Creating Custom Documentation Template

To create a documentation template

1. On the main menu, choose **File** ▸ **New** ▸ **Other**
2. In the **New** dialog that opens, expand the **Modeling** node.
3. Choose **Documentation Template** and click **Next**.
4. In the **GenDoc Template Designer File** dialog, specify the following:
   - In the **File Name** field, enter the fully qualified name of the template file.
   - In the **Template type** field, choose a template type from the drop-down list (documentation template or frameset template).
5. Use the Template Designer toolbox or context menus to create the template structure.

Note: After a documentation template is created, you can view and modify its properties using the **Template Properties** dialog.

Related Procedures

Setting Template Properties

Related Reference

Documentation Template Properties

Defining Frameset Structure

The frameset structure of a multi-frame HTML document describes how the frames are organized within the browser window. After a frameset template is created, you can define its structure through the template properties.

To define the structure of a frameset template

1. On the toolbar of the **Template Designer**, click the **Show Properties** button. The **Template Properties** dialog opens with the root frameset highlighted.
2. Define template properties in the **General** and **Template Parameter** tabs, as described in

   Setting Template Properties
3. In the **Frameset Structure** tab, choose the layout of the root frameset template, clicking either the **Columns** or **Rows** radio-buttons.
4. Add a frame or a frameset to the root frameset. To add a frame, click the **Add Frame** button. To add a frameset, click the **Add Frameset** button.

   Repeat this step to create a structure.
5. For each frame or frameset node, define its properties, as described in

   Setting Frame and Frameset Properties
6. Click **OK** when ready.

Related Procedures

Setting Frame and Frameset Properties

Related Reference

Frameset Template Properties

A Typical Scenario of Creating a Custom Documentation Template

Creating a custom documentation template and populating it with sections and controls involves the following general steps:

To create and customize a documentation template

1. Create a stub template:

   Creating Custom Documentation Template
2. Create template structure:

   Creating Sections
3. Provide reusable elements:

   Creating Stock Sections
4. Define sections properties:

   Setting Section Properties
5. Define area properties:

   Setting Area Properties
6. Define template properties:

   Setting Template Properties
7. Add controls to the sections:

   Creating Controls
8. Customize controls:

   Moving, Resizing and Aligning Controls

Find useful information in the following related sections:

Related Procedures

Using Word Documents in Documentation Templates
Reusing documentation templates from TCC/TA 1.x

Setting Call to Template Section Properties

The section properties of a call to template determine how the output for a template call can be used. With multi-frame HTML documentation, call to template sections typically generate separate files that can be loaded into a frame of the resulting HTML project documentation.

To access the properties of a call to template section, select **Properties** from the section's right-click menu. Refer to the **Call to Template Properties** dialog description for details.

To define properties of a call to template section

1. In the **Template** field of the **General** tab, click the **Browse** button, and select a template.
2. Select the type of generated output. If the output generated from the template is to be loaded into a frame, select Separate File from the radio buttons.
3. Define the name of the generated document. Click the **Edit Expression** button to create the expression.

   Note: If a particular call of a template is to be iterated many times to produce multiple documents, derive the output document name from the properties of the current model element. You can use the `getProperty("$name")` expression to get the name of the current model element.

4. Define the name of the output directory. Click the **Edit Expression** button to create the expression.
5. Define the output image subdirectory for the images files.

Related Reference

Call to Template Properties

Hyperlinking Documentation

HTML documentation requires hypertext links. A hypertext link connects a link reference (source) and a link destination (target). The link reference is a piece of text or an image. The link destination is a file or an anchor in a file.

To create hypertext links, refer to the following sections

1. Creating hyperlinks in the ordinary documentation:

   Hyperlinking Controls to Element Documentation
2. Creating hyperlinks in the multi-frame documentation:

   Creating Hypertext Links (Advanced)
3. Creating image mapping for the model elements in diagrams:

   Image Mapping (Advanced)
4. Converting Javadoc link references to hyperlinks:

   Creating Javadoc Link References (Advanced)

Creating Controls

To create a new control in a static section, header, or footer

1. Select a section in the details pane.
2. Click the type of control you want to insert on the template designer toolbar.

   Tip: Alternatively, you can right-click the section in the details pane, point to **Insert Control** and select the type of control from the menu.

3. On the context menu of a control, choose **Properties** and fill in the control properties in the tabbed **<Control> Properties** dialog box.
4. Click **OK**.

Note: The **Insert Control** command is also available on the context menu of a panel control.

Related Concepts

Documentation Template Controls

Related Procedures

Hyperlinking Documentation

Related Reference

Control Properties

Creating Custom Documentation Template

To create a documentation template

1. On the main menu, choose **File** ▸ **New** ▸ **Other**
2. In the **New** dialog that opens, expand the **Modeling** node.
3. Choose **Documentation Template** and click **Next**.
4. In the **GenDoc Template Designer File** dialog, specify the following:
   - In the **File Name** field, enter the fully qualified name of the template file.
   - In the **Template type** field, choose a template type from the drop-down list (documentation template or frameset template).
5. Use the Template Designer toolbox or context menus to create the template structure.

Note: After a documentation template is created, you can view and modify its properties using the **Template Properties** dialog.

Related Procedures

Setting Template Properties

Related Reference

Documentation Template Properties

Creating Formatting Styles for Documentation Templates

In this section you will learn how to create and edit formatting styles.

Tip: Create as many Formatting Style types as you would like, then assign a Formatting Style to a Control. After the Formatting Style has been assigned to controls, to change any style properties for these controls, change the property in the appropriate Formatting Style. After the Formatting Style is updated, the change will show immediately in the controls.

To create a new formatting style

1. On the tool palette of the **Template Designer**, click the **Show Template Properties** button. The **Template Properties** dialog opens.
2. In the **Formatting Styles** tab, click the **New** button. The **Style** dialog opens.
3. Select **Main** tab of the dialog, and specify the following parameters:
   - In the **Name** field, specify the style name. As you enter the name, it is displayed in the title bar of the **Style** dialog.
   - In the **Type** field, choose whether the style applies to a paragraph or to a character.
   - In the **Level** field, choose the nesting level of the style.
4. In the Font, Color and Border tabs, define the respective parameters of the style.
5. Click **OK**.

To edit formatting style

1. On the tool palette of the **Template Designer**, click the **Show Template Properties** button. The **Template Properties** dialog opens.
2. In the **Formatting Styles** tab, select a style and click the **Edit** button. The **Style** dialog opens.
3. Repeat steps 3 to 5 of the previous task.

Related Concepts

Documentation Generation Overview

Related Procedures

Creating Custom Documentation Template

Related Reference

Documentation Template Designer

Creating Hypertext Links (Advanced)

Multi-frame HTML documentation requires hypertext links. A hypertext link connects a link reference (source) and a link destination (target). The link reference is a piece of text or an image; it is the property of a control. The link destination is a file or an anchor in a file; it is the area property of a static section, header or footer.

By default, if no target frame for a hyper-reference is specified, the referenced document is loaded into the same frame window as the page that contains the link reference. The target frame parameter alters this behavior to load the target file into a named frame, so the source document is not replaced.

To assign a target frame to a link reference

1. On the context menu of the control, choose **Properties**.
2. Select the **Hyperlink to Elements** tab.
3. Click the **Edit Expression** button next to the **Target Frame Name Expression** field.
4. Select a notation and enter an expression for the name of a frame window defined in the frameset structure. The expression should return the name of one of the frame windows defined in the FrameSet Structure.

To create a link reference for a control

1. Create a target from a section as described in “Hyperlinking Controls to Element Documentation.”
2. Right-click the desired control and choose **Properties** on the context menu.
3. In the **Hyperlinks to Elements** tab, click the **Link to Element’s Specific Doc** radio button.
4. Click the **Edit Expression** button next to the **Expression for Model Element** field to determine which elements' documentation is the link target.
5. If the target area is marked as a Documentation Subject Selector, click the **Edit Expression** button next to the **Expression for Documentation Subject Selector** field to match the expression for the template area described above.

Related Concepts

Documentation Template Controls

Related Procedures

Hyperlinking Controls to Element Documentation

Related Reference

Control Properties

Creating Javadoc Link References (Advanced)

The Documentation Template Designer provides conversions for Javadoc References represented in the following forms:

- inside `{@link}` tags
- as the value of some Javadoc element's properties

Note: You need to specify the conversion in the properties of the control.

To convert a {@link} tag

1 On the context menu of the desired image control, choose **Properties**.
2 Select the **Other** tab.
3 Check the **Render Embedded Javadoc Tags** option.

Tip: Only a text control (label control, data control, or formula control) can generate documentation text.

Converting a value of an element's property to a hyperlink is more complicated than converting an `{@link}` tag. Such conversions require using one of these documentation generation functions:

```
getJDRefDisplayName()
getJDRefElement()
```

The following procedure gives a general outline of actions required to perform conversion. Most often this kind of conversion is used for the `see` property.

To convert the value of an element property

1 Create a property iterator that will go through the instances of the property.
2 Create one or more static sections that correspond to the Javadoc references of the desired type.
3 Provide an enable condition for each section, which activates it for the appropriate JDRef.

```
getJDRefType(getDGVariable("curPropertyInstance")) == "<JDRef type>"
```

where `"<JDRef type>"` is `"element", "URL"` or `"text"`.

4 Create a formula control in each section.
5 On the context menu of a formula control, choose **Properties**.
6 In the **Formula** tab of the **Formula Control** dialog, enter the following expression:

```
getJDRefDisplayName(getDGVariable("curPropertyInstance"))
```

Note: This expression is common for all types of Javadoc references. The value of the expression is the text that will be displayed in the documentation.

7 In the **Hyperlinks to Elements** tab of the **Formula Control** dialog, select the hyperlink type:

- For the JDRefs of the “element” type, click the **Link to Element's Specific Docs** radio button and enter the following expression in the **Expression for model element** field: `getJDRefDisplayName (getDGVariable("curPropertyInstance"))`
- For the JDRefs of the “URL” type, click the **URL Link** radio button and enter the following expression in the **Expression for URL** field: `getJDRefURL(getDGVariable("curPropertyInstance"))`
- For the JDRefs of the “text” type, the hyperlink definition is not defined. Because such a JDRef does not refer to any element, the function `getJDRefElement()` always returns null, producing no hyperlinks.

Related Concepts

Javadoc Link References

Related Reference

Control Properties

Creating Sections

A nested section can be created for an existing folder or iterator; a sibling section can be created for any existing section. For these sections, the **Insert Nested Section** or **Insert Sibling Section** toolbar buttons and menu commands are enabled, unlike the report header and footer, which lie outside the template body.

To create a new section in a documentation template

1. Select an existing section in the template.
2. Right-click an existing section in the scope pane, point to the **Insert Sibling Section** or **Insert Nested Section** on the context menu and select the type of new section.

   Tip: Alternatively, use the toolbar buttons of the Documentation Template Designer.

3. When necessary, determine the essential template information for the new section.
   - **Static section**: None required at creation.
   - **Element iterator**: Select the element metatype from the list of available types.
   - **Element property iterator**: Select the scope.
   - **Folder section**: None required at creation.
   - **Call to stock section**: Select the stock section from the list of existing sections. Refer to the section "Creating Stock Sections."
   - **Call to template section**: None required at creation.
4. Click **OK** to complete the insertion.
5. Set section properties as required.

Iterators and folder sections can also contain headers and footers. If such a section does not contain a header or a footer, its context menu provides **Add Header** or **Add Footer** commands. If a header or a footer exists, the context menu provides **Delete Header** or **Delete Footer** commands.

To add a header or a footer to an iterator

1. Select an existing element iterator or folder section.
2. On the context menu, choose **Add Header** or **Add Footer**.

Related Concepts

Documentation Template
Documentation Template Sections

Related Procedures

Creating Stock Sections
Setting Section Properties

Related Reference

Documentation Template Designer

Creating Stock Sections

Stock sections are reusable folders or iterators that reside in the template's collection of stock sections. Each stock section displays in a separate named tab in a documentation template.

In this section you will learn how to:

- Create stock sections from scratch
- Create stock sections from an existing section
- Delete stock sections
- Show stock section

To create a stock section

1. On the toolbar of the **Template Designer**, click the **New Stock Section (Element Iterator)** button or **New stock section (Folder)** button. The **New Stock Section** dialog opens.
2. In the **Name** field, enter the name of the new stock section.
3. For the element iterators, select a metatype from the list of available metatypes.
4. Check the **Intrinsic to Property Iterator** option, if necessary. If this flag is checked, the scope of the stock section root iterator or folder section depends on the Properties Iterator this section is called from: the only available iteration scopes for an element iterator are *customized* and *programmed*. For folder sections, it means that the metatype chooser tab is absent.
5. Click **OK**.

To create a stock section from an existing section

1. In the scope pane of a template, select an element iterator or folder section from which you want to create a stock section.
2. Right-click the section and choose **Copy into Stock** on the context menu.
3. In the **New stock section (Folder)** dialog, enter the stock section name.
4. Click **OK**.

To delete a stock section

1. Right-click the stock section tab be deleted.
2. On the context menu of the tab, choose **Remove Stock Section**.

To show a stock section for a call to stock section

1. Right-click the call stock section.
2. On the context menu of the section, choose **Show Stock Section**.

Related Concepts

Documentation Template
Documentation Template Sections

Related Reference

Documentation Template Designer

Defining Frameset Structure

The frameset structure of a multi-frame HTML document describes how the frames are organized within the browser window. After a frameset template is created, you can define its structure through the template properties.

To define the structure of a frameset template

1. On the toolbar of the **Template Designer**, click the **Show Properties** button. The **Template Properties** dialog opens with the root frameset highlighted.
2. Define template properties in the **General** and **Template Parameter** tabs, as described in

   Setting Template Properties
3. In the **Frameset Structure** tab, choose the layout of the root frameset template, clicking either the **Columns** or **Rows** radio-buttons.
4. Add a frame or a frameset to the root frameset. To add a frame, click the **Add Frame** button. To add a frameset, click the **Add Frameset** button.

   Repeat this step to create a structure.
5. For each frame or frameset node, define its properties, as described in

   Setting Frame and Frameset Properties
6. Click **OK** when ready.

Related Procedures

Setting Frame and Frameset Properties

Related Reference

Frameset Template Properties

Setting Template Properties

To set properties of a documentation template

1. On the toolbox of the **Template Designer**, click the **Show Template properties** button.
2. In the **General** tab, you can change the following:
   - Enter template description.
   - Define the report title expression, clicking the editor button to open the **Edit Expression** dialog.
   - Select Root Object Metatype from the list of available metatypes.
   - Attach a Word document as a formatting template. Refer to “Using Word Documents in Documentation Templates” for details.
   - Check or clear options to generate headers and footers as required.
3. In the **Page Settings** tab, you can specify page size, margins, and landscape or portrait orientation.
4. In the **Formatting Styles** tab, you can change formatting styles. Refer to “Creating Formatting Styles” for details.
5. In the **Template Parameters** tab, specify the formal parameters that will be used for calling this template from another template.

Related Procedures

Using Word Documents in Documentation Templates
Creating Formatting Styles for Documentation Templates

Related Reference

Documentation Template Properties

Setting Frame and Frameset Properties

In this section you will learn how to define properties of each frame and frameset that comprise a multi-frame template.

To define properties of a frame

1. In the **Frameset Structure** tab of the **Template Properties** dialog, select frame.
2. Specify the frame name, percent size and scrolling mode.
3. Click the **Edit Expression** button in the **Source File Name Expression** field. In the **Edit Expression** dialog, select an expression type and enter the expression body.
4. Click the **Edit Expression** button in the **Enable Condition** field. In the **Edit Expression** dialog, select an expression type and enter the expression body.

To define properties of a frameset

1. In the **Frameset Structure** tab of the **Template Properties** dialog, select a frameset.
2. Choose the layout of a frameset.
3. Click the **Edit Expression** button in the **Enable Condition** field. In the **Edit Expression** dialog, select an expression type and enter the expression body.

Related Reference

Frameset Template Properties

Hyperlinking Controls to Element Documentation

Any generated output that contains an anchor or a bookmark can be a link target. Documentation templates have facilities for inserting anchors at the “main documentation” of model elements. You can insert anchors for static sections, headers, and footers.

When you define the location of a model element main documentation, you can specify hyperlink references to it for any control that is not a panel. The control can be in the same template as the main documentation or it can be in a different template.

In this section you will learn how to:

- Make a target from a static section, footer, or header
- Link a control to a target

To make a target from a static section, footer, or header

1. Right-click the details pane of a section and choose **Area Properties**.
2. Click the **Hypertext Target** tab.
3. Specify **Expression for the Target Bookmark Selector**. When specified, this option inserts a bookmark into a file used as the File Link target. Click the **Edit Expression** button to create the expression in OCL or legacy notation.
4. Define **Start of the Current Element’s Specific Documentation**. If this option is checked, the output of this section is identified as the “main documentation” for the current element. This option is used for hyperlinks of Link to Element’s specific Doc type.
5. Specify **Expression for the Documentation Subject Selector**. This option is only enabled if the **Start of the Current Element’s Specific Documentation** option is checked. It marks the location of the elements' specific documentation with the appropriate Documentation Subject Selector, allowing users to create hyper-references to different documentation locations generated by the same model element. Click the Edit Expression button to create the expression in OCL or legacy notation.

To link a control to a target

1. In the details pane of a section, select a label, image or formula control.
2. On the context menu of the control, choose Properties.
3. Click the **Hyperlinks to Element** tab.
4. Define the ink type:
    - **Link to Element’s specific Doc:** You must identify the element whose documentation is to be the target in the text field for Expression for RWI-Element.
    - **File Link:** You must fill in the path to the file. If the hyperlink target is a bookmark in the file, you must supply that as well.
    - **URL Link:** You must supply the URL.
5. Specify the link settings, depending on the selected link type.
6. Optionally, provide compound hyperlinks, clicking the **Add Hyperlink <n>** button. This adds a new **Hyperlinks to Element** tab to the dialog.

Related Concepts

Documentation Template Controls

Related Procedures

Setting Area Properties

Related Reference

Area Properties

Hyperlinking Documentation

HTML documentation requires hypertext links. A hypertext link connects a link reference (source) and a link destination (target). The link reference is a piece of text or an image. The link destination is a file or an anchor in a file.

To create hypertext links, refer to the following sections

1. Creating hyperlinks in the ordinary documentation:

   Hyperlinking Controls to Element Documentation
2. Creating hyperlinks in the multi-frame documentation:

   Creating Hypertext Links (Advanced)
3. Creating image mapping for the model elements in diagrams:

   Image Mapping (Advanced)
4. Converting Javadoc link references to hyperlinks:

   Creating Javadoc Link References (Advanced)

Hyperlinking Controls to Element Documentation

Any generated output that contains an anchor or a bookmark can be a link target. Documentation templates have facilities for inserting anchors at the “main documentation” of model elements. You can insert anchors for static sections, headers, and footers.

When you define the location of a model element main documentation, you can specify hyperlink references to it for any control that is not a panel. The control can be in the same template as the main documentation or it can be in a different template.

In this section you will learn how to:

- Make a target from a static section, footer, or header
- Link a control to a target

To make a target from a static section, footer, or header

1. Right-click the details pane of a section and choose **Area Properties**.
2. Click the **Hypertext Target** tab.
3. Specify **Expression for the Target Bookmark Selector**. When specified, this option inserts a bookmark into a file used as the File Link target. Click the **Edit Expression** button to create the expression in OCL or legacy notation.
4. Define **Start of the Current Element’s Specific Documentation**. If this option is checked, the output of this section is identified as the “main documentation” for the current element. This option is used for hyperlinks of Link to Element’s specific Doc type.
5. Specify **Expression for the Documentation Subject Selector**. This option is only enabled if the **Start of the Current Element’s Specific Documentation** option is checked. It marks the location of the elements' specific documentation with the appropriate Documentation Subject Selector, allowing users to create hyper-references to different documentation locations generated by the same model element. Click the Edit Expression button to create the expression in OCL or legacy notation.

To link a control to a target

1. In the details pane of a section, select a label, image or formula control.
2. On the context menu of the control, choose Properties.
3. Click the **Hyperlinks to Element** tab.
4. Define the ink type:

   - **Link to Element’s specific Doc:** You must identify the element whose documentation is to be the target in the text field for Expression for RWI-Element.
   - **File Link:** You must fill in the path to the file. If the hyperlink target is a bookmark in the file, you must supply that as well.
   - **URL Link:** You must supply the URL.

5. Specify the link settings, depending on the selected link type.
6. Optionally, provide compound hyperlinks, clicking the **Add Hyperlink <n>** button. This adds a new **Hyperlinks to Element** tab to the dialog.

Related Concepts

Documentation Template Controls

Related Procedures

Setting Area Properties

Related Reference

Area Properties

Creating Hypertext Links (Advanced)

Multi-frame HTML documentation requires hypertext links. A hypertext link connects a link reference (source) and a link destination (target). The link reference is a piece of text or an image; it is the property of a control. The link destination is a file or an anchor in a file; it is the area property of a static section, header or footer.

By default, if no target frame for a hyper-reference is specified, the referenced document is loaded into the same frame window as the page that contains the link reference. The target frame parameter alters this behavior to load the target file into a named frame, so the source document is not replaced.

To assign a target frame to a link reference

1. On the context menu of the control, choose **Properties**.
2. Select the **Hyperlink to Elements** tab.
3. Click the **Edit Expression** button next to the **Target Frame Name Expression** field.
4. Select a notation and enter an expression for the name of a frame window defined in the frameset structure. The expression should return the name of one of the frame windows defined in the FrameSet Structure.

To create a link reference for a control

1. Create a target from a section as described in “Hyperlinking Controls to Element Documentation.”
2. Right-click the desired control and choose **Properties** on the context menu.
3. In the **Hyperlinks to Elements** tab, click the **Link to Element’s Specific Doc** radio button.
4. Click the **Edit Expression** button next to the **Expression for Model Element** field to determine which elements' documentation is the link target.
5. If the target area is marked as a Documentation Subject Selector, click the **Edit Expression** button next to the **Expression for Documentation Subject Selector** field to match the expression for the template area described above.

Related Concepts

Documentation Template Controls

Related Procedures

Hyperlinking Controls to Element Documentation

Related Reference

Control Properties

Image Mapping (Advanced)

When an image control is for the whole diagram in the model, the reference definition creates link references for all model elements depicted on the diagram. To create the image map, you must enter all expressions in the hyper-reference definition relative to the element returned by the `context OclAny getDGRwiElement ('diagramMapElement')` call.

When Documentation Generator generates the image of a diagram, it creates the image map, which includes all model elements depicted on the diagram. While doing this, it iterates through diagram elements, substituting the `diagramMapElement` variable with every diagram element, calculating a hyper-reference for it, and then inserting it into the image map. For example: `context uml::kernel::Elementif getDGRwiElement ('diagramMapElement').isDiagram() then getDGRwiElement('diagramMapElement') else getDGRwiElement('') endif`

To create an image map

1 On the context menu of the image control, choose **Properties**.
2 In the **Image** tab, select Diagram as an image type.
3 Select the **Hyperlink to Elements** tab.
4 Click the **Link to Element's Specific Doc** radio button.
5 Click the **Edit Expression** button next to the **Expression for Model Element** field and enter all expressions in the hyper-reference definition relative to the element returned by the call:

```
context OclAny

getDGRwiElement('diagramMapElement')
```

Related Concepts

Documentation Template Controls

Related Reference

Control Properties

Creating Javadoc Link References (Advanced)

The Documentation Template Designer provides conversions for Javadoc References represented in the following forms:

- inside `{@link}` tags
- as the value of some Javadoc element's properties

Note: You need to specify the conversion in the properties of the control.

To convert a {@link} tag

1. On the context menu of the desired image control, choose **Properties**.
2. Select the **Other** tab.
3. Check the **Render Embedded Javadoc Tags** option.

Tip: Only a text control (label control, data control, or formula control) can generate documentation text.

Converting a value of an element's property to a hyperlink is more complicated than converting an `{@link}` tag. Such conversions require using one of these documentation generation functions:

```
getJDRefDisplayName()
getJDRefElement()
```

The following procedure gives a general outline of actions required to perform conversion. Most often this kind of conversion is used for the `see` property.

To convert the value of an element property

1. Create a property iterator that will go through the instances of the property.
2. Create one or more static sections that correspond to the Javadoc references of the desired type.
3. Provide an enable condition for each section, which activates it for the appropriate JDRef.

   ```
   getJDRefType(getDGVariable("curPropertyInstance")) == "<JDRef type>"
   ```

   where `"<JDRef type>"` is `"element", "URL"` or `"text"`.
4. Create a formula control in each section.
5. On the context menu of a formula control, choose **Properties**.
6. In the **Formula** tab of the **Formula Control** dialog, enter the following expression:

   ```
   getJDRefDisplayName(getDGVariable("curPropertyInstance"))
   ```

   Note: This expression is common for all types of Javadoc references. The value of the expression is the text that will be displayed in the documentation.
7. In the **Hyperlinks to Elements** tab of the **Formula Control** dialog, select the hyperlink type:

- For the JDRefs of the “element” type, click the **Link to Element's Specific Docs** radio button and enter the following expression in the **Expression for model element** field: `getJDRefDisplayName (getDGVariable("curPropertyInstance"))`
- For the JDRefs of the “URL” type, click the **URL Link** radio button and enter the following expression in the **Expression for URL** field: `getJDRefURL(getDGVariable("curPropertyInstance"))`
- For the JDRefs of the “text” type, the hyperlink definition is not defined. Because such a JDRef does not refer to any element, the function `getJDRefElement()` always returns null, producing no hyperlinks.

Related Concepts

Javadoc Link References

Related Reference

Control Properties

Image Mapping (Advanced)

When an image control is for the whole diagram in the model, the reference definition creates link references for all model elements depicted on the diagram. To create the image map, you must enter all expressions in the hyper-reference definition relative to the element returned by the `context OclAny getDGRwiElement ('diagramMapElement')` call.

When Documentation Generator generates the image of a diagram, it creates the image map, which includes all model elements depicted on the diagram. While doing this, it iterates through diagram elements, substituting the `diagramMapElement` variable with every diagram element, calculating a hyper-reference for it, and then inserting it into the image map. For example: `context uml::kernel::Elementif getDGRwiElement ('diagramMapElement').isDiagram() then getDGRwiElement('diagramMapElement') else getDGRwiElement('') endif`

To create an image map

1 On the context menu of the image control, choose **Properties**.

2 In the **Image** tab, select Diagram as an image type.

3 Select the **Hyperlink to Elements** tab.

4 Click the **Link to Element's Specific Doc** radio button.

5 Click the **Edit Expression** button next to the **Expression for Model Element** field and enter all expressions in the hyper-reference definition relative to the element returned by the call:

```
context OclAny

getDGRwiElement('diagramMapElement')
```

Related Concepts

Documentation Template Controls

Related Reference

Control Properties

Moving, Resizing and Aligning Controls

The Documentation Template Designer displays a newly created control as a rectangle positioned at the insertion point in the details pane. You can move the control to change where its output is displayed in the generated documentation. You can also modify the size of the rectangle to determine the approximate size of the region for the output. Increasing the rectangle size is especially important for a label with a default size that is not large enough to allow its entire text to be displayed in the report.

If you select two or more controls in a section, you can use the context menu of the selection to uniformly align controls within the section. Be aware that when you apply multiple alignments, it is possible for controls to overlap.

Tip: Precise positioning and sizing is not possible.

Warning: There is no simple undo for changes in alignment or size. When you change alignment or resize controls, you must manually readjust the controls to return them to the former status.

To move a control within a section

1. Select a control. Notice that the mouse pointer changes to a cross with double-ended arrows.
2. With the control selected, drag and drop the control to a position within the section.

To copy or move a control to another section

1. Select a control.
2. On the context menu of the selected control, choose **Copy** or **Cut**.
3. Right-click the target section and choose **Paste** on the context menu.

To resize a control

1. Place the mouse pointer on either the right or left edge of the rectangle. Notice that the mouse pointer changes to a double-ended arrow.
2. Drag the edge of the rectangle to the required size.

To align controls

1. Select two or more controls within a section.
2. Right-click the selection and choose **Alignment** on the context menu.
3. On the submenu, choose one of the following options:
    - Left Side
    - Right Side
    - Top Side
    - Bottom Side
    - Make same width
    - Make same height

- Make same size

Related Concepts

Documentation Template
Documentation Template Controls

Related Reference

Documentation Template Designer

OCL in Documentation Templates

Together allows you to compose model queries and define enable conditions using OCL syntax, and then use them in a template for generating documentation. OCL or Legacy type expressions can be entered in the template element's properties dialog box using the provided Expression Editor. Where applicable, the editor is either opened in the tab or you can use the Edit Expression button to open the editor.

In addition to the standard OCL operations, the special native OCL extensions are provided for the functions that are specific for Documentation Generation. Native OCL extensions tend to have the same signature and meaning as the legacy Documentation Generation functions have. Code sense suggests these operations along with the standard OCL ones.

To add an expression to your template

1. Open a template where you want to add an OCL expression.
2. In the **Properties** dialog box, open the tab where you want to type the expression. If the Expression Editor is not opened in the tab, click the **Edit Expression** button.
3. Specify the context for your expression in the **Context** field.
4. In the **Body** area, you can type the expression text. The code sense, syntax highlighting and validating are available.
5. Click **OK** to save the expression in the template.

Related Concepts

Documentation Template Controls
About OCL Support in Together
Enable Conditions

Reusing documentation templates from TCC/TA 1.x

Normally the legacy documentation templates are compatible with Together, except for the two major differences: adapter classes (special extensions written in Java) and changes in the metamodel.

'For example, consider the legacy `Class.tpl` template from predefined TCC templates. Errors occur when this template executes because it calls methods of the `SpecialScopeProvider` adapter class, which is missing in Together. Users need to modify this template manually to make sure that calls to these methods are replaced with OCL expressions.

Namely, the users should change expressions for `Class.tpl` for Programmed iteration scope in two Element iterators. The sample procedure is described below.

To modify a legacy documentation template using OCL

1 Remove class and method names.

2 Open Iterator properties.

3 Change the value of the **programmed iteration scope** property from **specify class and method** to **specify expression**.

4 Set the expression type to 'OCL', with `uml::kernel::Element` as context.

5 Specify the following bodies:

— for Element iterator in the `Direct Known Subinterfaces` folder: `getKnownSubclasses()`

— for Element iterator in the `All Known Implementing Classes` folder: `getImplementingClasses()`

Note: Adapter classes written for TCC do not work for Together. Instead of the TCC GenDoc extensions, Together makes use of the native OCL extensions (special Eclipse plugins that use the `com.borland.selena.ocl.gdm.nativeExtension` extension point and contain Java implementations of the corresponding methods). `getKnownSubclasses()` and `getImplementingClasses()` are the examples of such extensions. Users should create their own OCL extensions to be used in their templates.

To make sure that metamodels are compatible

1 Open your template in the **Template Designer**.

2 Make sure there are no iterations by **<Any>** or folders without metatype specified. If such iterations exist, it means that metatypes for these iterators/folders are not recognized.

3 Change metatypes as required.

Related Concepts

Documentation Template

Related Procedures

Together Documentation Templates Procedures

Related Reference

Documentation Template Designer

Setting Area Properties

Area properties apply to static sections, headers and footers. They are defined in the **Area Properties** dialog, which is common for static sections, headers and footers, and is invoked from the details pane. Refer to the dialog descriptions for details.

To set area properties

1. Select a static section, header or footer of a template.
2. On the context menu of the details pane, choose **Area Properties**.
3. In the **Area Properties** dialog, specify settings and click **OK**.

Related Concepts

Documentation Template Sections

Related Reference

Area Properties

Setting Call to Template Section Properties

The section properties of a call to template determine how the output for a template call can be used. With multi-frame HTML documentation, call to template sections typically generate separate files that can be loaded into a frame of the resulting HTML project documentation.

To access the properties of a call to template section, select **Properties** from the section's right-click menu. Refer to the **Call to Template Properties** dialog description for details.

To define properties of a call to template section

1. In the **Template** field of the **General** tab, click the **Browse** button, and select a template.
2. Select the type of generated output. If the output generated from the template is to be loaded into a frame, select Separate File from the radio buttons.
3. Define the name of the generated document. Click the **Edit Expression** button to create the expression.

   Note: If a particular call of a template is to be iterated many times to produce multiple documents, derive the output document name from the properties of the current model element. You can use the `getProperty("$name")` expression to get the name of the current model element.

4. Define the name of the output directory. Click the **Edit Expression** button to create the expression.
5. Define the output image subdirectory for the images files.

Related Reference

Call to Template Properties

Setting Frame and Frameset Properties

In this section you will learn how to define properties of each frame and frameset that comprise a multi-frame template.

To define properties of a frame

1. In the **Frameset Structure** tab of the **Template Properties** dialog, select frame.
2. Specify the frame name, percent size and scrolling mode.
3. Click the **Edit Expression** button in the **Source File Name Expression** field. In the **Edit Expression** dialog, select an expression type and enter the expression body.
4. Click the **Edit Expression** button in the **Enable Condition** field. In the **Edit Expression** dialog, select an expression type and enter the expression body.

To define properties of a frameset

1. In the **Frameset Structure** tab of the **Template Properties** dialog, select a frameset.
2. Choose the layout of a frameset.
3. Click the **Edit Expression** button in the **Enable Condition** field. In the **Edit Expression** dialog, select an expression type and enter the expression body.

Related Reference

Frameset Template Properties

Setting Section Properties

After a section is created, define its properties. Section properties are defined in the **Properties** dialogs, which are specific for each kind of sections.

You can invoke the Properties dialog for the iterators and folder sections from the scope pane or from the details pane of a section. For the static sections, the Properties dialog is invoked from the scope pane only.

To set properties of a section

1. In the scope section of the **Template designer**, select a template section.
2. On the context menu, choose **Properties**.
3. Define properties as required and click **OK**.

Note: Properties dialogs are specific to each section type. Refer to the dialog descriptions for details.

Related Concepts

Documentation Template Sections
Enable Conditions

Related Reference

Element Iterator Properties
Property Iterator Properties
Static Section Properties
Call to Stock Section Properties
Call to Template Properties

Setting Template Properties

To set properties of a documentation template

1. On the toolbox of the **Template Designer**, click the **Show Template properties** button.
2. In the **General** tab, you can change the following:
   - ♦ Enter template description.
   - ♦ Define the report title expression, clicking the editor button to open the **Edit Expression** dialog.
   - ♦ Select Root Object Metatype from the list of available metatypes.
   - ♦ Attach a Word document as a formatting template. Refer to “Using Word Documents in Documentation Templates” for details.
   - ♦ Check or clear options to generate headers and footers as required.
3. In the **Page Settings** tab, you can specify page size, margins, and landscape or portrait orientation.
4. In the **Formatting Styles** tab, you can change formatting styles. Refer to “Creating Formatting Styles” for details.
5. In the **Template Parameters** tab, specify the formal parameters that will be used for calling this template from another template.

Related Procedures

Using Word Documents in Documentation Templates
Creating Formatting Styles for Documentation Templates

Related Reference

Documentation Template Properties

Using Word Documents in Documentation Templates

The **Template Designer** provides a way to use styles, headers and footers of the `*.rtf`, `*.dot` and `*.doc` Word files in your custom documentation templates. When the path to the Word file is specified, the styles of the referenced Word document are displayed in the list of formatting styles; the header and footer of the referenced Word document are displayed on each page of the generated report.

In this section you will learn how to:

- attach a Word document to a documentation template
- detach a Word document from a documentation template

To attach a Word document to a documentation template

1. Open a documentation template in the **Template Designer**, or create a new one.
2. On the tool palette of the **Template Designer**, click the **Show Template Properties** button. The **Template Properties** dialog opens.
3. In the **Formatting Template** field, specify the path to the Word file.

   Tip: Alternatively, click the **Browse** button and navigate to the Word file.

   The styles of the referenced document display in the **Formatting Styles** tab of the **Template Properties** dialog.

   Tip: The **Delete** and **Edit** buttons do not work for the styles of the attached document.

4. Click **OK**.

Warning: Together supports simple text Word `*.rtf`, `*.dot` and `*.doc` templates headers/footers only. The headers and footers with embedded images, objects, complex formatted text and fields are not processed.

To detach a Word document from a documentation template

1. Open a documentation template in the **Template Designer**.
2. On the tool palette of the **Template Designer**, click the **Show Template Properties** button. The **Template Properties** dialog opens.
3. Remove the path from the field **Formatting Template**.
4. Click **OK**.

Related Concepts

Documentation Generation Overview

Related Procedures

Creating Custom Documentation Template
Creating Formatting Styles for Documentation Templates

Related Reference

Documentation Template Designer

Interoperability and Migration

This section provides how-to information on exchanging model information between the various products of the Together product line.

In This Section

Converting Profile-Specific Properties

The converting profiles function helps you reuse projects from Together 2006 in which custom profiles were applied.

This feature is useful for the following scenario:

1. In Together 2006, a profile has been created and deployed. This results in creating a profile plugin.
2. This profile plugin is applied to a certain modeling project.
3. The same profile definition is reused and deployed in Together 2006 R2. This results in creating another profile plugin, which has different properties names.
4. The same modeling project is opened in Together 2006 R2. On an attempt to apply the new profile plugin to this project, the profile-specific properties will loose their values unless they are properly converted.

To convert profile-specific properties

1. On the main menu, choose **Model** ▸ **Profile** ▸ **Convert Properties**.
2. If there are no profile-specific properties in the project, no action is performed.

Related Concepts

UML Profiles

Importing a Project in an IBM Rational Rose MDX Model

Together enables you to create projects around an IBM® Rational® XDE .mdx file.

Note: Together design projects that are created on the basis of the imported MDX models always comply with the UML 2.0 specification.

To create a project from an MDX model

1. On the main menu, choose **File ▸ Import**. The **New Project** wizard opens.
2. Select **Project from MDX file** and click **Next**.
3. Specify the path to the MDX file you want to import or click **Browse** to locate the file. You can also specify the following:
   - Use the **Scale factor** field to specify the element dimensions coefficient. By default, the scale factor is 0.03.
   - **Preserve diagram nodes and bounds**: If this option is selected, user-defined bounds are preserved in the resulting diagrams. Otherwise the default values are applied.
4. Click **Next**.
5. Specify new project name. Click **Next**.
6. Specify the diagram to start with. Click **Next**.
7. Select one or more profiles you want to enable for this project. Click **Next**.
8. Select any referenced projects.
9. Click **Finish** to complete the wizard. A new project will be created with elements from the MDX file.

Note: If a profile was applied to the Rational XDE model while importing the MDX model to Together, the properties from this profile are imported as custom properties.

Related Concepts

Model Import and Export Overview

Related Reference

MDX Import Wizard
MDX Projects Import Options

Importing a Project in IBM Rational Rose (MDL) Format

Together enables you to create projects around IBM® Rational® Rose model files (.mdl, .ptl, .cat, .sub).

Note: You can import a set of petal and subunit files.

Warning: Together projects created on the basis of the imported MDL models always comply with the UML 1.4 specification.

To create a design project from an IBM Rational Rose (MDL) project

1. On the main menu, choose File ▸ Import. The **New Project** wizard opens.
2. Select **Project from MDL file** and click **Next**.
3. Click either **Add** or **Add Folder** to designate the MDL project path. This step specifies the name (or names) of the Rational Rose project file (or files) to be imported (several model files can be imported at once). Click **Remove** to delete the selected file or files from the Paths list. Click **Remove all** to delete all files from the Paths list.

   Note: Avoid adding a model file along with its subunit to the import list because this results in invalid project.

4. Use the **Scale factor** field to specify the element dimensions coefficient. By default, the scale factor is 0.3.
5. Specify the following options for the project:
   - **Convert Rose default colors**: If this option is selected, the default Rational Rose colors will be replaced with the default Together colors.
   - **Preserve diagram nodes and bounds**: If this option is selected, user-defined bounds are preserved in the resulting diagrams. Otherwise the default values are applied.
   - **Convert Rose actors**: This option enables you to choose mapping for the Rose actors. If the option is selected, the Rose actors are mapped to Together actors. If the option is not selected, the Rose actors are mapped to the classes with the Actor stereotype, such as Actor, Business Actor, Business Worker, or Physical Worker.
   - **Generate source code**: If this option is selected, a new Java Modeling project is created; otherwise, a Modeling project is created from imported MDL.
6. Click **Finish**.
7. When prompted, supply a name for your project and click **Finish**.
8. Follow the remaining steps in the wizard to specify options for your new project, and click **Finish** to complete the wizard.

After the import process is complete, you can view the project structure in the Model Navigator view. The `mdlimport.log` file is generated by default and lists any errors encountered during the import process.

Note: After entering a project name, you can click **Finish** without completing the remaining steps of the wizard. The project is created using the remainder of default settings.

Related Concepts

Model Import and Export Overview

Related Procedures

Generating Source Code from Design Project

Related Reference

Together Projects
MDL Projects Import Options
MDL Import Wizard

Importing a Project in XMI Format

You can import projects or sections of projects that were created in other modeling tools and saved in XMI format.

Note: For UML 1.4 and Java Modeling projects only, XMI 1.1/1.2 imports are supported. Attempting to import an XMI 1.0 file results in an empty project.

To import a project from an XMI file

1 Select **File** ▸ **Import** on the main menu. The **Import** dialog box opens.

2 Select **XMI File** and click **Next**.

3 In the **Import Project from XMI File** dialog box, specify the following:

- ♦ The Together project to which your XMI data will be imported in the **Select destination project** field.
- ♦ The full path to the .xml, .xmi, or .uml2 file you want to import in the **Select source .xmi file** field.

4 Click **Finish**.

Note: A .xml or .xmi file can be imported to UML 1.4 and Java Modeling projects; a .uml2 file can be imported to UML 2.0 projects.

After you are notified that the import process is complete, you can view the results in the **Model Navigator**.

Note: When importing UML 2.0 models with profile files related to the model, for the models originally exported from Together for Eclipse, select model .uml2 file as a source and make sure that all the profile files are located in the same folder with the model file.

If there are any warnings produced during XMI import, the XMI Import dialog notifies you to refer to the **Task** view. To open the Task view, select **Window** ▸ **Show View** ▸ **Other** ▸ **Basic** ▸ **Tasks** from the main menu.

Related Concepts

Model Import and Export Overview

Related Procedures

XMI Export and Import of the Models with Cross-Project References

Importing Java Modeling Projects Created in Together Edition for Eclipse 7.0

You can import projects created in Together Edition for Eclipse 7.0.

The general procedure for importing a project created in Together Edition for Eclipse 7.0 consists of the following steps:

1. Importing your existing project into a workspace
2. Creating a Java modeling project from a Java project

To import an existing project from TEC 7.0

1. Select **File** ▸ **Import** on the main menu.
2. Select **Existing Projects into Workspace** and click **Next**.
3. In the **Import Projects** dialog box, specify the path to your project's root directory and select one or more projects you want to import.
4. Click **Finish** when you specified all necessary options.

   The new Java project is created and opened in your workspace.

Note: The name of the imported project cannot be changed during the import process. Therefore, the projects are created with the same name as the imported projects.

To create a Java modeling project from a Java project

1. Select **File** ▸ **New** ▸ **Project** on the main menu. The **New Project** wizard opens.
2. Expand the **Together** node in the tree view list and select **Java Modeling projects from Java projects**. Click **Next**.
3. Select the Java project you created from the project created in Together Edition for Eclipse. Click **Next**.
4. Specify other project-related options.
5. Click **Finish** when you specified all necessary options.

Related Concepts

Together Interoperability and Migration

Importing Legacy Projects

Together allows you to import projects from some of the previously released Together products. Considering the differences between the products, Together suggests two ways to accomplish this import. You can merge all roots of a legacy multi-rooted project into a single root, or you can create a separate project for each root of the source project.

- The **Merge option** is recommended for typical cases of when the input project has one design root and several source code roots.
- The **Separate projects option** is recommended when your input project has nonstandard configuration with several design roots, which you would like to preserve as separate projects.

To import a legacy project merging all source roots into a single project

1. Select **File** ▸ **Import** on the main menu
2. In the **Import Wizard**, select **Modeling Together Project** and click **Next**. The second page of the wizard opens.
3. Click **Browse** to specify the fully qualified name of the project you want to import.
4. In the **Design elements storage policy** section, choose whether the design elements of the resulting project will be stored as standalone design elements or as filemates.
5. In the **Migration type** section, select the **Merge all roots contents into the new project** option.
6. Click **Next**. The third page of the wizard opens.
7. Specify the name of the target project. The default project name is constructed from the names of the last two folders of the source project file location.
8. Click **Finish** to import the selected project.

Warning: TVS projects and projects created in Together Editions for Eclipse prior to version 7.0 cannot be imported to Together.

To create separate projects for each selected root

1. Select **File** ▸ **Import** on the main menu
2. In the **Import Wizard**, select **Modeling Together Project** and click **Next**. The second page of the wizard opens.
3. Click **Browse** to specify the fully qualified name of the project you want to import.
4. In the **Design elements storage policy** section, choose whether the design elements of the resulting project will be stored as standalone design elements or as filemates.
5. In the **Migration type** section, select the **Create a separate project for each root** option.
6. On the third page of the wizard, the **Root location** table displays the list of folders of the source project. Select each root from the list and define the way you want to handle the root and its contents:
   - In the **Together project name** field, specify the name of the target project for the selected root. The default name is constructed from the package prefix, if any. If there is no package prefix, the project name is created from the names of the last two folders of the root location.
   - The read-only **Content type** and **Diagram format** fields display the corresponding information for the selected root.
   - In the **Decision** field, choose the way to handle information of the selected root. If the root contains design files, you can either copy them to the target location or skip the root. If the root contains source code files,

you have the choice to copy it as is, copy and convert it to design language, or skip the root. The option **Copy and convert to design language** is the default choice for the roots that contain Java files.

- In the **Dependencies to be preserved while importing** field, you can specify whether the import handles links and references between projects created for the currently selected root and projects created for other roots. All dependencies are processed by default. However, if you are aware of any one-way dependencies between the original roots, and the selected root does not refer to any elements from other roots, uncheck those corresponding projects listed in the field to save CPU resources and complete the import faster.

7. Click **Next**. The fourth page of the wizard opens.
8. Specify the name of the master project that contains references to all projects created in the course of the migration. The default name of the master project is based on the source project name.

   Note: The master project is created to demonstrate the contents and structure of the source project. It is read-only and not intended for editing. Use the real projects to create or edit contents and establish dependencies.

9. Click **Finish** to import the selected project.

All resulting projects belong to the same type, which is defined by the properties of the source project and your choice in the **Decision** field of the **Import Wizard**. Java modeling projects are created if there is at least one Java source root for which the **Copy** option is selected. UML 1.4 modeling projects are created if there are no Java source roots, or if such roots exist but the **Decision** field is set to **Skip** or **Convert to design language**.

Related Concepts

Together Interoperability and Migration

Related Reference

Import Together Project Wizard

Reusing documentation templates from TCC/TA 1.x

Normally the legacy documentation templates are compatible with Together, except for the two major differences: adapter classes (special extensions written in Java) and changes in the metamodel.

'For example, consider the legacy `Class.tpl` template from predefined TCC templates. Errors occur when this template executes because it calls methods of the `SpecialScopeProvider` adapter class, which is missing in Together. Users need to modify this template manually to make sure that calls to these methods are replaced with OCL expressions.

Namely, the users should change expressions for `Class.tpl` for Programmed iteration scope in two Element iterators. The sample procedure is described below.

To modify a legacy documentation template using OCL

1. Remove class and method names.
2. Open Iterator properties.
3. Change the value of the **programmed iteration scope** property from **specify class and method** to **specify expression**.
4. Set the expression type to 'OCL', with `uml::kernel::Element` as context.
5. Specify the following bodies:

   — for Element iterator in the `Direct Known Subinterfaces` folder: `getKnownSubclasses()`

   — for Element iterator in the `All Known Implementing Classes` folder: `getImplementingClasses()`

Note: Adapter classes written for TCC do not work for Together. Instead of the TCC GenDoc extensions, Together makes use of the native OCL extensions (special Eclipse plugins that use the `com.borland.selena.ocl.gdm.nativeExtension` extension point and contain Java implementations of the corresponding methods). `getKnownSubclasses()` and `getImplementingClasses()` are the examples of such extensions. Users should create their own OCL extensions to be used in their templates.

To make sure that metamodels are compatible

1. Open your template in the **Template Designer**.
2. Make sure there are no iterations by **<Any>** or folders without metatype specified. If such iterations exist, it means that metatypes for these iterators/folders are not recognized.
3. Change metatypes as required.

Related Concepts

Documentation Template

Related Procedures

Together Documentation Templates Procedures

Related Reference

Documentation Template Designer

XMI Export and Import of the Models with Cross-Project References

You can import and export multi-root projects using XMI. Note that XMI import and export is implemented differently for UML 1.4 and Java modeling projects, and for UML 2.0 projects.

- **UML 1.4 and Java modeling projects:** When a project that contains cross-project references is exported to an XMI file, the main project root and referenced roots are exported to the same XMI file. The **Use prefix of imported root** option of the **Export Wizard** enables you to reproduce the package structure of each root in top-level packages named as the root prefixes. If the option is unchecked, all same-named packages from the different roots are merged. When an XMI file is imported, the resulting project contains all packages and elements from the main model and referenced roots.
- **UML 2.0 projects:** When a project that contains cross-project references is exported to an XMI file, `*.imports.uml2` special files are created for each referenced root. The exported XMI file contains references to these files. When an XMI file is imported, the resulting project contains the main model only. If the referenced roots still exist in the workspace, the resulting UML 2.0 model recognizes them. References to the elements from these roots can be resolved only if the unique identifiers (UINs) of the elements have not been changed since export. Note that when an element is moved, its container is changed, and this can change the UIN.

To export a UML 1.4 and Java modeling project with cross-project references

1. On the main menu, choose **File Export**.
2. On the first page of the **Export Wizard**, select XMI file under Modeling and click **Next**.
3. On the second page of the wizard:
   - Select the project to be exported;
   - Select the XMI type and encoding;
   - Specify the export destination;
   - Check the **Use prefix of imported root** option if you want to reproduce the package structure of each root in top-level packages named as the root prefixes. By default, this option is unchecked.
4. Click **Finish**.

Tip: Package prefixes of the referenced roots are never used if you perform an export via the `XMIExport.cmd` command line utility.

To export a UML 2.0 project with cross-project references

1. On the main menu, choose **File Export**.
2. On the first page of the **Export Wizard**, select XMI file under Modeling and click **Next**.
3. On the second page of the wizard:
   - Select XMI for UML 2.0 as the project to be exported
   - Specify export destination
4. Click **Finish**.

Related Concepts

Together Interoperability and Migration
Model Import and Export Overview

Related Procedures

Importing a Project in XMI Format
Exporting a Project to XMI Format

Reference

Reference

This part contains reference information.

In This Section

Together Glossary

This topic contains a dictionary of specific terms used in the Together user interface and documentation. This dictionary is sorted alphabetically.

Term	Description
Behavior	In Together, a group of the following UML 2.0 **model elements**: activity, state machine, and interaction.
Cardinality	The number of elements in a set. See also **multiplicity**.
Classifier	In general, a classifier is a classification of instances. It describes a set of instances that have features in common. In Together, a classifier is a group of the following model elements: class, interface, association class, structure, delegate, enumeration, module, interaction. In UML 2.0 projects this group includes the **data type** element as well. Some of the elements can include members or other classifiers. A classifier inserted into another classifier is called an **inner classifier**.
Compartment	Some of Together model elements (basically, classes) are represented by rectangles with several **compartments** inside. You can change the appearance of the compartments. See **Related Reference** below for details.
Container	A container is a **classifier** that can include one or more model elements, or **members**.
Design project	One of the two basic project types supported by Together: **design** and **implementation**. A design project is language-neutral. It does not contain source code.
Diagram	A graphical presentation of a collection of shortcuts to **model elements** from one or more **packages** or **namespaces**. Most often a diagram is rendered as a connected graph of arcs (relationship links) and vertices (nodes). The set of available diagrams for a project depends on the project type.
Domain-specific language (DSL)	A language designed to accomplish a set of tasks within a particular domain. Compared to a general-purpose language, DSLs are typically smaller, more declarative and less expressive.
Domain model	A domain model is the part of a **DSL** that describes the entities and their relationships within a domain.
Implementation project	One of the two basic project types supported by Together: **design** and **implementation**. An implementation project is language-specific. It includes diagrams and source code.
Inner classifier	An inner classifier is a **classifier** inserted into another classifier.
Invocation specification	An area on a UML 2.0 Sequence Diagram. Most often an **invocation specification** is located within an **execution specification**. This element is not defined in the UML 2.0 specification, but is introduced in Together. It is a useful tool for modeling synchronous invocations with the reply messages. A message in a UML 2.0 Sequence Diagrams has its origin in an invocation specification.
Member	A member is a **model element** inserted into a classifier, or a **container**. If a member is a classifier, it is called an **inner classifier**.
Model element	Any component of your model that you can put into a package or a namespace. Model elements include **nodes** and **links** between them.

Multiplicity	A specification of the range of allowable cardinalities that a set may assume, for example: `0..*`. Multiplicity specifications can be given for association ends, parts within composites, and other purposes. A multiplicity is a subset of the non-negative integers. See also **cardinality**.
N-ary association	An association among three or more nodes. In Together, an **association class** implements this functionality.
Object Constraint language (OCL)	A declarative language used to describe expressions on models. Typically, OCL describes constraints (or rules) about models.
Package	An element for storing diagrams, model elements, and other packages. For implementation projects, the same elements are known as a **namespace** and are connected to namespaces in the source code. Every project in Together consists of one or more packages or namespaces. You cannot delete the `default` package (namespace).
Pattern instance	An oval model element that represents a pattern with a special predefined behavior.
Practitioner	The role in a software development team that uses the **DSL** created by the **Toolsmith**.
Toolsmith	The role in a software development team that amplifies and extends software tools. Toolsmiths often create **DSLs** but also provide tool configurations, customizations and extensions.
Shortcut	A presentation of a model element, diagram, namespace, package, or some external artifact placed on a diagram.
View filter	A mechanism to show or hide a specific kind of model element. With large projects, the amount of information shown on a diagram can become overwhelming. In Together, you can selectively show or hide information. See **Related Reference** below for details.

Related Concepts

Help on Help
Together Overview

Related Procedures

Working with Inner Classes
Changing the Appearance of Compartments

Together Keyboard Shortcuts

Together enables you to perform many diagram actions without using the mouse. You can navigate between diagrams and diagram elements, create diagram elements, use drag-and-drop operation, and more, using the keyboard only.

Navigational shortcut keys

Keyboard shortcuts for navigation and browsing:

Action	Shortcut	Notes
Navigate between open diagrams in the Diagram Editor	CTRL+TAB	The title of the diagram that has focus is in bold text.
Navigate between elements on a diagram	Arrow keys	
Select elements	SHIFT + arrow key	
Expand node in Model Navigator	RIGHT ARROW	
Collapse node in Model Navigator	LEFT ARROW	
Open the Properties View	F4, or ALT + ENTER	
Close current diagram	CTRL+F4	
Toggle between a selected container node and its members	PGDOW /PGUP	
Navigate between nodes or node members	Arrow keys, SHIFT + arrow keys	
In the Diagram Editor , toggle focus between selected element and diagram.	CTRL+SPACE	
Select on Diagram	CTRL+F3	

Shortcut keys for editing

Keyboard shortcuts for editing:

Action	Shortcut
Cut, Copy, or Paste model elements or members	CTRL+X, CTRL+C, CTRL+V
Activate the in-place editor for a diagram element to edit or rename a member	F2
Undo	CTRL+Z
Redo	CTRL+Y, CTRL+SHIFT+Z
Select all elements on the diagram	CTRL+A
Close the Overview window	ESC
Add a new package to a diagram	CTRL+E
Add a new class to a diagram	CTRL+L
Add a new method (operation) to a class or interface	CTRL+M
Add a new field (attribute) to a class	CTRL+W
Add a new interface to diagram	CTRL+SHIFT+L
Add shortcuts	CTRL+SHIFT+N

Add a new diagram from the Model Navigator	CTRL+SHIFT+D
Invoke **Content Assist** in the OCL editor, or check spelling in Description tab of Properties View	CTRL+SPACE

Zoom shortcut keys

Keyboard shortcuts for zooming the diagram image:

Action	Shortcut	Notes
Zoom in	+	Use the numeric keypad
Zoom out	-	Use the numeric keypad
Fit the entire diagram in the Diagram Editor	*	Use the numeric keypad
Display the actual size	/	Use the numeric keypad

Cycling between the Diagram Editor and the Palette

Keyboard shortcuts for cycling between the diagram editor and the palette:

Action	Shortcut
Navigate from the diagram name in focus to the Palette	TAB
Navigate from the Palette in focus to Palette **Minimize** button	TAB
Navigate from the Palette **Minimize** button in focus to Palette items	TAB
Navigate from the Palette item in focus to the Diagram Editor and place focus on any selected item	TAB
Navigate from the Diagram Editor in focus to the Palette items. The last Palette item used is selected (otherwise defaults to the **Select** option)	SHIFT+TAB
Navigate from the Palette item in focus to Palette **Minimize** button	SHIFT+TAB
Navigate from Palette **Minimize** button in focus to the Palette	SHIFT+TAB
Navigate from the Palette in focus to the Diagram Editor	SHIFT+TAB

Palette item navigation

Keyboard shortcuts for navigating between Palette items:

Action	Shortcut
Expand or collapse the selected drawer in the Palette	SPACE
Select the current Palette item in focus	SPACE
Move the focus between Palette items	UPARROW or DOWNARROW
Create a new shape item on the diagram if the **Shape** Palette item is in focus	ENTER
Create a new connection between the two selected diagram elements if the **Connection** Palette item is in focus	ENTER
Navigate to the diagram if the Palette item is in focus	TAB
Deselect the Palette item that is in focus	ESC

Display stack popup list if **Stack** Palette item is in focus	ALT+DOWNARROW
Navigate between the available Palette tools on the stack if the **Stack** Palette item is in focus	UPARROW or DOWNARROW
Select an item from the stack popup list if the stack popup list is in focus	ENTER

Diagram navigation

Keyboard shortcuts for navigating between diagrams:

Action	Shortcut
Select a shape in the selected diagram	ALT+DOWNARROW
Cycle through the shapes that exist in the selected diagram. A shape is selected when the eight side and corner size handles are displayed.	UPARROW, DOWNARROW, LEFTARROW, or RIGHTARROW
Multi-select shapes on the selected diagram	SHIFT+UPARROW, SHIFT+DOWNARROW, SHIFT +LEFTARROW, or SHIFT+RIGHTARROW
Invoke the context menu for the shape for the selected diagram	SHIFT+F10

Shape navigation

Keyboard shortcuts for navigating between shapes:

Action	Shortcut
Invoke the context menu for the selected shape	SHIFT+F10
Cycle through the Position Handle, 8 Side and Corner Size Handles, and Position Handle for the selected shape (navigates in clockwise rotation)	. (period)
Cycle through the Position Handle, 8 Side and Corner Size Handles, and Position Handle for the selected shape (navigates in counter-clockwise rotation)	SHIFT+. (period)
Select any available connection (navigates clockwise among the existing connections)	/
Select any available connection (navigates counter-clockwise among the existing connections)	\
Select the shape compartment for the selected shape	ALT+DOWNARROW
Deselect the selected shape by displaying the shape in an outline	CTRL+SPACE
Change the shape size or position for the selected shape handle. A shaded shape is displayed showing the new size or position.	UPARROW, DOWNARROW, LEFTARROW, or RIGHTARROW
Change the shape size or position for the selected shape handle in respect to the aspect ratio. A shaded shape is displayed showing the new size or position.	CTRL+UPARROW, CTRL+DOWNARROW, CTRL +LEFTARROW, or CTRL+RIGHTARROW
Change the shape size or position for the selected shape with respect to the shape's center. A shaded shape is displayed showing the new size or position.	CTRL+SHIFT+UPARROW, CTRL+SHIFT+DOWNARROW, CTRL+SHIFT+LEFTARROW, or CTRL+SHIFT+RIGHTARROW

Deselect the selected shape handle	ESC
Accept the current shaded shape	ENTER
Revert to the original shape size or position for the selected shape	ESC
Select a compartment within the selected shape	ALT+DOWNARROW
Navigate between the available compartments if a compartment is selected	UPARROW or DOWNARROW
Select the shape compartment items. The first compartment item is selected.	ALT+DOWNARROW
Deselect the compartment and select the shape if a compartment is selected	ALT+UPARROW
Navigate between the available compartment items	UPARROW or DOWNARROW

Connection navigation

Keyboard shortcuts for navigating between connections

Action	Shortcut
Invoke the context menu for the connection	SHIFT+F10
Cycle through all of the connection labels if a connection is selected. A connection label is selected when the four corner size handles are displayed. Connection labels can be navigated in the same manner as shapes.	ALT+DOWNARROW
Deselect the connection label and select the connection if a connection label is selected	ALT+UPARROW
Deselect the connection and select the shape if a connection is selected	UPARROW, DOWNARROW, LEFTARROW, or RIGHTARROW
Deselect the selected connection by displaying the connection in an outline	CTRL+SPACE
Cycle through the endpoints, bendpoints, and midpoints of the selected connection	. (period) or SHIFT+. (period)
Allow the selected connection endpoint to be moved to a new shape	UPARROW, DOWNARROW, LEFTARROW, or RIGHTARROW
Move the connection bendpoint if the cursor is over a bendpoint	UPARROW, DOWNARROW, LEFTARROW, or RIGHTARROW
Accept the current location if the cursor is over a bendpoint	ENTER
Revert to the original location if the cursor is over a bendpoint	ESC
Move the new bendpoint	UPARROW, DOWNARROW, LEFTARROW, or RIGHTARROW
Accept the new bendpoint	ENTER
Remove the bendpoint	ESC

Properties view navigation

Keyboard shortcuts for navigating between Properties views:

Action	Shortcut
Cycle through the fields in the property section if a **Properties** field is selected and eventually highlight the currently active **Properties** tab (cycles through the Properties View UI)	SHIFT+TAB
Cycle through the fields in the property sections of the active **Properties** tab (cycles through the Properties View UI in the opposite direction of SHIFT+TAB)	TAB
Move focus among tabs in the **Properties** view if a **Properties** tab is highlighted	UPARROW or DOWNARROW

Other shortcut keys

Other keyboard shortcuts:

Action	Shortcut	Notes
Open the Print Diagram dialog box	CTRL+P	
Diagram update	F5	
Drag-and-drop operation	>	While the focus is on the necessary element, press this key until the move handle is displayed. Move the element using the arrow keys and press ENTER to drop the element.

Related Concepts

Help on Help
Together Overview

Additional Resources

The following supplemental resources provide further insights into modeling, architecture, and design.

- *High-Assurance Design: Architecting Secure and Reliable Enterprise Applications* by Clifford Berg
- *Beyond Software Architecture: Creating and Sustaining Winning Solutions* by Luke Hohmann
- *Design Patterns: Elements of Reusable Object-Oriented Software* by Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides
- *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions* by Gregor Hohpe and Bobby Woolf
- *Enterprise Service Bus* by David Chappell
- *Enterprise SOA: Service-Oriented Architecture Best Practices (The Coad Series)* by Dirk Krafzig, Karl Banke, and Dirk Slama
- *Object Design: Roles, Responsibilities, and Collaborations* by Rebecca Wirfs-Brock and Alan McKean
- *Patterns of Enterprise Application Architecture* by Martin Fowler
- *Refactoring to Patterns* by Joshua Kerievsky
- *Streamlined Object Modeling: Patterns, Rules, and Implementation* by Jill Nicola, Mark Mayfield, Mike Abney, and Michael Abney
- *UML Distilled: A Brief Guide to the Standard Object Modeling Language, Third Edition* by Martin Fowler
- *Workflow Modeling: Tools for Process Improvement and Application Development* by Alec Sharp and Patrick McDermott
- *The Object Constraint Language: Getting Your Models Ready for MDA, Second Edition* by Jos Warmer and Anneke Kleppe
- *Eclipse Modeling Framework 2.0* by Frank Budinsky, Ed Merks, and David Steinberg

Components of the Together User Interface

This section describes GUI components of the Together user interface you use for modeling, quality assurance, requirements management and more.

In This Section

Quality Assurance GUI Components
Describes GUI components of the Together interface that you use for Together Quality Assurance features.

Menus

In This Section

Menus

If you have all Modeling capabilities enabled, which is the default configuration, and the Modeling perspective opened, the following menu items are visible. The menu items that appear also depend on which view you have opened.

Note: Because it shares the same user interface environment as the Eclipse platform, Together is able to extend or replace existing Eclipse menus or add its own menus. For more information on the standard Eclipse menus, refer to the “Workbench menus” topic in the *Workbench User Guide*.

Item	Description
File menu	The File menu extends the capabilities of submenus by providing the ability to create, import, and export Together artifacts.
Edit menu	Use the Edit menu to cut, copy, paste, and delete diagrams and diagram elements, select all items on a diagram, and undo/redo actions.
Source menu (Model Navigator View)	The Source menu contains commands for working with logical data model elements.
Refactor menu (Model Navigator View)	The Refactor menu contains refactoring commands for the implementation projects.
Navigate menu	Use the Navigate menu to locate and navigate through items displayed on your workbench.
Search menu	The Search menu lets you explore all facilities to locate specific content or elements.
Project menu	Use the Project menu to open and close projects, build projects and working sets, discard build problems and built states, and generate documentation.
Diagram menu	Use the Diagram menu to create a new diagram or to configure how an existing diagram is displayed.
Model menu	Use the Model menu to run audits and metrics (for the implementation projects), compare models, manage profiles, and apply transformations.
Run menu	Use the Run menu to run and debug applications, and to configure your run and debug options.
Window menu	Use the Window menu to select perspectives, views, and editors, and to set your Together preferences.
Help menu	Use the Help menu to access product and platform documentation, tips and cheat sheets, a search engine, and software updates.

Related Reference

Diagram View

Model Navigator

The Model Navigator provides the logical representation of the model of your project: namespaces (packages) and diagram nodes. Using this view, you can add new elements to the model; cut, copy, paste and delete elements, and more. Context menu commands of the Model Navigator are specific to each node. Explore these commands as you encounter them.

In the Model Navigator, only the nodes and their respective subnodes shown in the Diagram Editor are listed under the corresponding diagram node. For example, if you have a package containing a class, both the package and class are shown under the diagram node in the Model Navigator. However, members of the class are not shown under the diagram node because they are displayed under the namespace (package) node only.

The Model Navigator is a dockable window. The docking areas are any of the four borders of the Together window. You can position the Model Navigator according to your preferences.

The Model Navigator has several different context menus, each specific to the resource selected. This section discusses the various context menus for the following resource levels in the Model Navigator:

- Project Level
- Package Level
- Diagram Level
- Element Level, Class
- Element Level, Operation
- Element Level, Link

Note: Depending on your development platform, some of the menu commands described may not be applicable. For more information, refer to the documentation set provided with your IDE: select Help Contents on the Help menu.

Project Level

Model Navigator projects offer the following context menu commands. Some of the commands are the same across all the context menus. For commands that are not mentioned here, refer to the Common Diagram Context Commands section.

Option	Description	
New	Displays a submenu with all basic elements that can be added to the project.	
New Diagram	Creates a new diagram: Activity, Collaboration, Class, Component, Deployment, State, Use Case, Sequence.	
Open in New Tab	Opens the project level diagram in the Diagram editor in a new tab.	
Open	Opens the project level diagram in the Diagram editor. It will replace the currently shown diagram with this one.	
Open Type Hierarchy	Highlights the node selected in the Hierarchy view. The Hierarchy view will expand and highlight that element in the tree-view. If closed, the Hierarchy view will open. For more information, refer to the documentation set provided with your IDE. Select Help Contents on the Help menu.	
Delete	Deletes the selection. Together prompts for confirmation.	
Source	Format	Uses the code formatter to format the current selection. For more information, refer to the documentation set provided with your IDE.
	Organize Imports	Generates a list of import statements based on the import order preference, and replaces old statements. For more

		information, refer to the documentation set provided with your IDE. Select Help Contents on the Help menu.
	Find Strings to Externalize	Launches the Externalize Strings Wizard. For more information, refer to the documentation set provided with your IDE. Select Help Contents on the Help menu.
Refactor		Begins the refactoring process to restructure your code without changing its observable behavior. For more information, refer to the documentation set provided with your IDE. Select Help Contents on the Help menu.
	Rename	Starts the Rename refactoring dialog. Renames the selection and (if enabled) corrects all references to the elements (also in other files).
	Move	Starts the Move refactoring wizard. Moves the selection and (if enabled) corrects all references to the elements (also in other files).
Import		Opens the Import wizard for a number of import options. Follow this link for more information about the XMI File option. TEC also offers the ability to import Together Control Center projects. For details on other import options, refer to the documentation set provided with your IDE. Select Help Contents on the Help menu.
Export		Opens the Export wizard for a number of export options, some of which are Together-specific: UML Documentation and XMI File, for instance. For details on other export options, refer to the documentation set provided with your IDE. Select Help Contents on the Help menu.
Quality Assurance	Audit	Runs audits against the selection. See "Running Model Audits and Metrics" for details.
	Metrics	Runs metrics against the selection. See "Running Model Audits and Metrics" for details.
	Load Metric Results	Loads a previously saved set of metric results. See "Saving and Loading Metric Results" for details.
Refresh		Refreshes the current view.
Properties		Displays the properties for the selection in the Properties view. While the Properties command accessed through the Model Navigator opens the Properties view for the selected resource, you should be aware that this differs from the Properties command accessed through the Navigator and Packages views. The Navigator view Properties command, for instance, opens a project-specific Properties dialog. Among other things, it allows you to turn templates and pattern-recognition on or off per project. These selections override those made in the global Preferences dialogs.
Update		Using the update command will update the diagram.

Package Level

Within the Model Navigator at the package level of a project, the context menu displays many of the same commands as at the project level; however, there are some additional options available described below.

Option	Description
Select on Diagram	Using the Select on Diagram command, you can open the corresponding diagram that the element belongs to in the Diagram editor and highlight the element on the diagram.
Select in Project	Displays project names that contain other occurrences of the selected design element. The icons next to the project names let you distinguish whether the referenced occurrence is in the home project of the element or in the read-only root of another project referencing the home project. Selecting a project name from this menu changes (or navigates) the element selection from the current project to the element's other occurrence in the referenced project. This navigation takes place whether the selection was made in the Model Navigator

	context or in a diagram context. Refer to the topics on navigation and cross-project references for further information.
Show in Packages View	The Show in Packages View command highlights the node selected in the Packages tree-view. The Packages view will expand and highlight that element in the tree-view. If closed, the Packages view will open.
Show in Model Package Explorer	The Show in Model Package Explorer command highlights the node selected in the Model Package Explorer tree-view. The Model Package Explorer view will expand and highlight that element in the tree-view. If closed, the Model Package Explorer view will open.

Diagram Level

The context menu of the diagram level of a project displays the same commands as at the package level.

Element Level, Class

The context menu for class and interface elements has the following specific commands:

Select on Diagram	The Select on Diagram command highlights the node selected in the Diagram editor. If the diagram is closed, it opens to display the node.
Select in Project	Displays project names that contain other occurrences of the selected design element. The icons next to the project names let you distinguish whether the referenced occurrence is in the home project of the element or in the read-only root of another project referencing the home project. Selecting a project name from this menu changes (or navigates) the element selection from the current project to the element's other occurrence in the referenced project. This navigation takes place whether the selection was made in the Model Navigator context or in a diagram context. Refer to the topics on navigation and cross-project references for further information.

Element Level, Operation

The context menu for an operation has the following specific commands:

Select on Diagram	The Select on Diagram command highlights the operation selected in the Diagram editor. If the diagram is closed, it opens to display the operation.
Select in Project	Displays project names that contain other occurrences of the selected design element. The icons next to the project names let you distinguish whether the referenced occurrence is in the home project of the element or in the read-only root of another project referencing the home project. Selecting a project name from this menu changes (or navigates) the element selection from the current project to the element's other occurrence in the referenced project. This navigation takes place whether the selection was made in the Model Navigator context or in a diagram context. Refer to the topics on navigation and cross-project references for further information.

Element Level, Link

For information on the link level context menu, see Common Link Context Commands.

Related Procedures

Running Model Audits and Metrics
Saving and Loading Metric Results
Establishing cross-project references
Navigating between the Tree View, Diagram, and Source Code

Related Reference

Common Link Context Commands

Model Package Explorer Context Menus

The Model Package Explorer has many different context menus, each specific to the resource selected. This section discusses the following Model Package Explorer context menus.

Note: Together can leverage all Java development functionality from Eclipse's JDT, and it therefore inherits all the corresponding Eclipse interface's views and menus while adding some modeling-specific items to those menus. Depending on your development platform, some of the following menu commands described might not be applicable. For more information, refer to the documentation set provided with your IDE. On the main menu, click **Help ▸ Help Contents**.

Project Menu

New		Displays a submenu with all elements that can be created.
Go Into		Displays the contents of the selected item in the Model Package Explorer. All others are hidden. To return to views of parent resources, click the Up button.
Open in New Window		Displays the contents of the selected item in a new instance of Together.
Open Type Hierarchy		Opens the type Hierarchy view. For more information, refer to the documentation set provided with your IDE. From the menubar, choose **Help ▸ Help Contents**.
Copy		Copies the selection.
Paste		Pastes a copied selection.
Delete		Deletes selection. A confirmation dialog opens before deletion.
Source	Format	Uses the code formatter to format the current selection. For more information, refer to the documentation set provided with your IDE.
	Organize Imports	Generates list of import statements based on the import order preference and replaces old statements. For more information, refer to the documentation set provided with your IDE.
	Find Strings to Externalize	Launches the **Externalize Strings Wizard**. For more information, refer to the documentation set provided with your IDE.
Refactor		Begins the refactoring process to restructure your code without changing its observable behavior. For more information, refer to the documentation set provided with your IDE.
	Rename	Starts the Rename refactoring dialog. Renames the selection and (if enabled) corrects all references to the elements (also in other files).
	Move	Starts the Move refactoring wizard. Moves the selection and (if enabled) corrects all references to the elements (also in other files).
Import		Opens the Import wizard for a number of import options. Follow this link for more information about the XMI File option. TEC also offers the ability to import Together Control Center projects. For details on other import options, refer to the documentation set provided with your IDE.
Export		Opens the Export wizard for a number of export options, some of which are Together-specific: UML Documentation and XMI File, for instance. For details on other export options, refer to the documentation set provided with your IDE.
Refresh		Refreshes the selection.
Close Project		Closes selected project.
Run		This menu contains commands similar to the Run command on the main menu. See Workbench User Guide for details.
Debug		This menu contains commands similar to the Debug command on the main menu. See Workbench User Guide for details.

Team	Use the Team command to add a project to the repository. After a project is added, you can create patches, commit, synchronize with repository, and so on. For more information, see “Using Version Control and Teams in Together.”	
Compare With	Use Compare With to compare the resources in the workspace with the resources held within the repository. Offers a submenu with the following options:	
	Patch	Allows you to share work with other team members without storing it in a repository. Use the Patch command to access this type of resource.
	Each Other	Compares two files. Select two diagrams in the Model Package Explorer view by using CTRL+CLICK, then invoke **Compare With ▸ Each Other**.
	Restore From Local History	Allows you to restore a resource with a saved version from your local history. For more information, refer to the documentation set provided with your IDE.
Quality Assurance	Audits	Run audits on the selection. Together allows you to run a default QA set or create your own. See “Running Model Audits and Metrics” for more information.
	Metrics	Run metrics on the selection. Together allows you to run a default QA set or create your own. See “Running Model Audits and Metrics” for more information.
	Load Metric Results	Loads the results of a previous metrics run. See “Saving and Loading Metric Results” for more information.
	Load Audit Results	Loads the results of a previous audit run. See “Saving and Loading Metric Results” for more information.
CaliberRM connections	Displays the Open Connection dialog box with available connections. Connections to CaliberRM servers can be specified on the CaliberRM page of the Preferences dialog box.	
Properties	Opens the selected **Project Properties** dialog box.	

Folder Menu

The folder menu commands are documented in the Project Menu section.

Package Menu

The package menu commands are documented in the Project Menu section except for the following:

Select on Diagram	Opens the corresponding diagram that the element belongs to in the Diagram editor and highlights the element on the diagram. This command is also available for compilation units in the **Package Explorer** view.	
Select in Project	Displays project names that contain other occurrences of the selected design element. The icons next to the project names let you distinguish whether the referenced occurrence is in the home project of the element or in the read-only root of another project referencing the home project. Selecting a project name from this menu changes (or navigates) the element selection from the current project to the element's other occurrence in the referenced project. This navigation takes place whether the selection was made in the Model Navigator context or in a diagram context. Refer to the topics on navigation and cross-project references for further information.	
Open Diagram	Opens the package diagram. If necessary, the Diagram Editor opens first.	
References	Workspace	Searches the entire workbench for references to the selection.

	Working Set	Opens the **Select Working Sets** dialog, allowing you to specify which working set to search.
Declarations	Workspace	Searches the entire workbench for references to the selection.
	Working Set	Opens the **Select Working Sets** dialog, allowing you to specify which working set to search.

Package Declaration Menu

The commands for this menu are documented in the Package Menu section.

Package Diagram Menu

The menu shown below is for a default package diagram. Menus for specific diagram types may not contain all of the commands listed.

New	Displays a submenu with all basic elements that can be added to a diagram.	
New Diagram	Creates a new diagram.	
Generate Class Diagram	Creates a new class diagram from the selection.	
Select in Model Tree	Highlights the selection in the Model Navigator. If closed, the Model Navigator will open.	
Select in Project	Displays project names that contain other occurrences of the selected design element. The icons next to the project names let you distinguish whether the referenced occurrence is in the home project of the element or in the read-only root of another project referencing the home project. Selecting a project name from this menu changes (or navigates) the element selection from the current project to the element's other occurrence in the referenced project. This navigation takes place whether the selection was made in the Model Navigator context or in a diagram context. Refer to the topics on navigation and cross-project references for further information.	
Show in Packages View	Highlights the selection in the **Package Explorer**. If closed, the **Package Explorer** will open.	
Open	Opens the selected diagram in the Diagram Editor .	
Open in New Tab	Opens the selected diagram on a new tab in the Diagram Editor .	
Cut	Removes the selection to the clipboard. You can later choose to Paste the element.	
Copy	Copies the selection to the clipboard. You can later choose to Paste the element.	
Paste	Pastes a cut or copied selection in a new location.	
Rename	Opens the **Rename** dialog to rename the selection.	
Delete	Opens a confirmation dialog before deleting the selection.	
Import	Opens the **Import** wizard.	
Export	Opens the **Export** wizard.	
Quality Assurance	Audit	Runs audits against the selection. See “Running Model Audits and Metrics” for details.
	Metrics	Runs metrics against the selection. See “Running Model Audits and Metrics” for details.
	Load Metric Results	Loads a previously saved set of metric results. See “Saving and Loading Metric Results” for details.
Add Bookmark	Allows you to bookmark the selection. You can view bookmarks in the Bookmarks view. You can also bookmark individual lines of code in the editor. For more information on using bookmarks, refer to the documentation set provided with your IDE.	
Hyperlinks	The Hyperlinks command offers a submenu with the following options:	
	Edit	Using Edit, you can view, add, and remove hyperlinks to a project. For more information on hyperlinks, see “Hyperlinking Diagrams.”

	<Hyperlink>	<Hyperlink> represents an actual hyperlinked element. If there are no hyperlinks for the diagram, then only the Edit option is available.
Requirements	Allows you to edit requirements traces. For more information, refer to the CaliberRM plugin help.	
References	Workspace	Searches the entire workbench for references to the selection.
	Hierarchy	Searches the Hierarchy view for references to the selection.
	Working Set	Opens the **Select Working Sets** dialog, allowing you to specify which working set to search.
Declarations	Workspace	Searches the entire workbench for references to the selection.
	Hierarchy	Searches the Hierarchy view for declarations of the selection.
	Working Set	Opens the **Select Working Sets** dialog, allowing you to specify which working set to search.
Export to Image	Opens the Export Diagram to Image dialog box, which lets you create an image of the selection. You can export to Bitmap, JPEG, GIF, and SVG formats.	
Hide/Show	Hides individual elements on a diagram. If you do not see an element in a diagram, choose Show hidden from the diagram context menu and check the hidden elements list. For more information, see “Hiding and Showing Model Elements.”	
Team	Use the Team command to add a project to the repository. After a project is added, you can create patches, commit, synchronize with repository, and so on. For more information, see “Using Version Control and Teams in Together.”	
Compare With	Use Compare With to compare the resources in the Workbench with the resources held within the repository.	
Replace With	Use Replace With to replace Workbench resources with versions in the repository.	
Properties	Opens the Properties view for the selection.	
Update	Refreshes the selection.	

Compilation Unit Menu

The compilation unit menu commands are explained in the above sections except for the following:

Open With	Allows you to choose, from a submenu, an editor with which to open the selection. You can determine which editors are available in the File Associations Preferences dialog. For more information on using system, default, and external editors, refer to the documentation set provided with your IDE.
Apply Template	Opens the Apply Template wizard, which lets you apply templates. For more information, see “Working with the Templates.”
Save As Template	Selecting the Save As Template command displays the Create Template dialog, which lets you save the class or interface as a template.

Java Scrapbook Page Menu

The commands on this menu are documented in the above sections.

No-Icon File Menu

The commands on this menu are documented in the above sections.

XML File Menu

The commands on this menu are documented in the above sections.

Plugin Menu

The commands on this menu are documented in the above sections except for the following:

Update Classpath	Opens the **Java Classpath** wizard. The dialog displays a list of plug-ins and fragments in your workspace. Select the ones for which you want to recompute classpaths.
Create Plugin JARs	Creates an Ant buildscript and opens the Run Ant wizard with the build.jars option selected by default. This may be done to deploy the plugins/fragments in your workspace. For more on creating and deploying plugins, see the Tool Developer guide.

Image File Menu

The commands on this menu are documented in the above sections.

Cascading Style Sheet Menu

The commands on this menu are documented in the above sections.

HTML File Menu

The commands on this menu are documented in the above sections.

Type/Interface Menu

The commands on this menu are documented in the above sections.

Method Menu

The commands on this menu are documented in the above sections.

Field Menu

The commands on this menu are documented in the Compilation Unit Menu section except for the following:

Read Access	Finds all read accesses to the selection. Search the Workspace/Hierarchy or select a Working Set to search.
Write Access	Finds all write accesses to the selection. Search the Workspace/Hierarchy or select a Working Set to search.

Import Container Menu

The commands on this menu are documented in the above sections.

Import Menu

The commands on this menu are documented in the above sections.

JAR Menus

The commands on this menu are documented in the above sections, but may also include the following:

Open With	Allows you to choose, from a submenu, an editor with which to open the selection. You can determine which editors are available in the File Associations Preferences dialog. For more information on using system, default, and external editors, refer to the documentation set provided with your IDE.

Related Procedures

Using Version Control and Teams in Together
Running Model Audits and Metrics
Saving and Loading Metric Results
Hyperlinking Diagrams
Hiding and Showing Model Elements
Using Version Control and Teams in Together
Working with the Templates
Establishing cross-project references
Navigating between the Tree View, Diagram, and Source Code

Related Reference

Common Link Context Commands

Common Diagram Context Commands

The context menus of the various diagrams provide functions specific to each diagram. Explore the context menus of the different diagrams as you encounter them to see the commands available for each one.

However, for the most part, all of the UML diagrams do share common context menu commands. To use the context menu for a diagram, simply right-click the background of the diagram in the Diagram editor.

From the Diagram editor, each diagram has the following common context menu commands. Most of the commands are similar in all the Together projects.

New

Each diagram has the New command. Each diagram has a submenu specific to the New command containing each diagram's specific elements.

Select in Model Tree

The Select in Model Tree command highlights the node selected in the Model Navigator tree-view. The Model Navigator will expand and highlight that element in the tree-view. If closed, the Model Navigator will open. For more information, see “Navigating between the Tree View, Diagram, and Source Code.”

Cut

This action removes the diagram. You can then choose to Paste the element into a new location.

Copy

This action copies the selected diagram. After copying an element, you can Paste it into a new location.

Clone

The Clone command lets you quickly create a new diagram or element with the same content as the existing one. An element that can be cut/copied and pasted can also be cloned by using the Clone command. Cloning is basically a one-step copy-and-paste. For more information, see “Working with Custom OCL Operations.”

Paste

Using the Paste command, you can paste a diagram that has been cut or copied.

Paste element

Use this command to paste a copied diagram or element as a shortcut to another diagram.

Rename

The Rename dialog renames the element and refactors the change throughout the project.

Delete

The Delete command will delete the element from the project. A Confirmation dialog opens to confirm the deletion.

Export

Opens the **Select** dialog box with available export destinations.

Import

Opens the **Select** dialog box with available import sources.

Add Linked

Provides search options for references, implementations, and inheritance according to the specified types and scopes.

Refactor

Moves/Renames and Refactors a project or element. Other options are displayed in the submenu based on the selection. Dialogs let you enter the needed information or locations.

Model Bookmark

Bookmarks allow you to navigate to resources that are frequently used. You can set, remove, and view bookmarks using the Bookmarks view.

Hyperlinks

The Hyperlinks command offers a submenu with the following options:

Option	Description
Edit	Using Edit, you can view, add, and remove hyperlinks to a project. For more information on hyperlinks, see “Hyperlinking Diagrams.”
<Hyperlink>	<Hyperlink> represents an actual hyperlinked element. If there are no hyperlinks for the diagram, then only the Edit option displays.

Requirements

The Requirements Management command offers a submenu with the following commands:

Option	Description
Manage Traces	Opens the Manage Traces dialog, which lets you define traces from a model element selected in the Diagram editor or Model Navigator to CaliberRM or RequisitePro requirements. Using the dialog, you can view, add, and remove requirements associated with the element. For more information on linking requirements to diagram elements, see “Creating Traces from Requirements to Model Elements.”
Synchronize Traces	Opens the Trace Synchronizer dialog, which lets you search for traced CaliberRM requirements or model elements with local and server copies that become desynchronized

for some reason. The found desynchronized traces are displayed in the Trace Synchronizer view. For more information on synchronizing traces, see “Trace Synchronizer Dialog Box.”

If the diagram is associated with requirements, the submenu lists such requirements. Click on a requirement to open it in the CaliberRM or RequisitePro Navigator.

Layout

The Layout command offers a submenu with options for laying out your diagram elements.

Hide / Show

Note that individual elements on a diagram can be hidden using the element context menu. If you do not see an element in a diagram, choose Hide/Show from the diagram context menu and check the hidden elements list in the Show Hidden dialog. For more information, see “Hiding and Showing Model Elements.”

Team

Use the Team command to add a project to the repository. Once the project is added, you can create patches, commit, synchronize with repository, and so on. Refer to the documentation set provided with your IDE for complete information. From the menubar, choose **Help** ▸ **Help Contents**.

Additional commands for CVS are provided when the Capability **Team** ▸ **CVS support for Modeling** is enabled.

Using these commands assumes that option **Team** ▸ **Modeling Resources** ▸ **Auto Checkout modeling resources on edit** is off and that the project is checked out from CVS with option **Team** ▸ **CVS** ▸ **Watch/Edit** ▸ **Configure project to use Watch/Edit on checkout** turned on.

The **Team** command offers a submenu with the following commands:

Command	Description
Edit View	Performs the edit command for the diagram file (*.txv*) only. Model or diagram files (*.txv*/*.txa*) containing elements shown on the diagram are not affected.
Edit View and Model	Performs the edit command for the diagram file and also for all files containing elements shown on the diagrams regardless of their containing package.
Edit View and Package Locally	Performs the edit command for the diagram file and also files containing elements shown on the diagrams from the same package.
Edit View and Package Recursively	Performs the edit command for the diagram file and also files containing elements shown on the diagrams from the same package and all its subpackages recursively.
Unedit View	Performs the unedit command for the diagram file (*.txv*) only. Model or diagram files (*.txv*/*.txa*) containing elements shown on the diagram are not affected.
Unedit View and Model	Performs the unedit command for the diagram file and also for all files containing elements shown on the diagrams regardless of their containing package.
Unedit View and Package Locally	Performs the unedit command for the diagram file and also files containing elements shown on the diagrams from the same package.
Unedit View and Package Recursively	Performs the unedit command for the diagram file and also files containing elements shown on the diagrams from the same package and all its subpackages recursively.

Properties

The Properties command opens the Properties view for the current element or diagram. For more information, see “Properties View.”

Related Concepts

Together Capabilities Activation
Version Control in Together

Related Procedures

Navigating between the Tree View, Diagram, and Source Code
Working with Custom OCL Operations
Hyperlinking Diagrams
Creating Traces from Requirements to Model Elements
Hiding and Showing Model Elements
Common Diagrams Procedures

Related Reference

Trace Synchronizer Dialog Box
Properties View
Package Context Menu

Package Context Menu

All of the UML diagram types share common context menu commands. To use the context menu for a diagram, simply right-click in the Diagram Editor .

To view the common context menu commands, see "Common Diagram Context Commands."

The context menu for a package element residing on a diagram differs slightly from the package diagram context menu. It includes the Open, Open in New Tab, and New Diagram commands. The package context menu shares the common context menu commands as well as commands specific to it:

- New
- **New Diagram** (for package elements)
- Generate Class Diagram
- Show in Packages View
- Show in Model Package Explorer
- Open (for package elements)
- Open in New Tab (for package elements)
- Quality Assurance

New

The New command for the package element offers a submenu with the following options:

- Selecting Class from the submenu adds a class element to the diagram.
- Selecting Interface from the submenu adds an interface element to the diagram.
- Selecting Package from the submenu adds a package element to the diagram.
- Selecting Object from the submenu adds an object element to the diagram.
- Selecting Class by Template launches the Apply Template dialog displaying the available templates. Make a selection from the list to apply a template.
- Selecting Note from the submenu adds a note element to the diagram.
- To refer to an element located outside of the current diagram, or to another diagram, you can use shortcuts. Invoking the Shortcut command displays a selection dialog, where you can choose the desired element (or diagram) from the appropriate location.

New Diagram

The New Diagram command for the package element offers a submenu allowing you to create new diagrams. The new diagrams are created in the current package.

Generate Class Diagram

The Generate Class Diagram command creates an exact copy of the package diagram as a new class diagram.

Show in Packages View

The Show in Packages View command highlights the node selected in the Packages tree-view. The Packages view will expand and highlight that element in the tree-view. If closed, the Packages view will open.

Show in Model Package Explorer View

The Show in Model Package Explorer View command highlights the node selected in the UML Explorer tree-view. The Model Package Explorer view will expand and highlight that element in the tree-view. If closed, this view will open.

Open

When selecting a package element on a diagram, you can open the package diagram in the Diagram editor by using the Open command. Using the Open command will keep the current diagram open, while the newly opened diagram opens in its own tabbed page in the Diagram editor. This command resides on the context menu for package elements on diagrams.

Open in Active Editor

When selecting a package element on a diagram, you can open the package diagram in the Diagram editor by using the Open in Active Editor command. Using the Open in Active Editor command will replace the currently opened diagram with the newly opened diagram. This command resides on the context menu for package elements on diagrams.

Quality Assurance

The Quality Assurance command for the class diagram offers a submenu with the following options:

- After you have run metrics on a project or part of a project, you can save those metric results and view them whenever you like. Use this command to load a set of metrics results. For more information, see “Saving and Loading Metric Results.”
- Selecting Metrics from the submenu processes metrics for only the specific diagram. For more information, see “Running Source Code Metrics.”
- Selecting Audit from the submenu processes audits for only the specific diagram. For more information, see “Running Source Code Audits.”

Team

Use the Team command to add a project to the repository. Once the project is added, you can create patches, commit, synchronize with repository, and so on. Refer to the documentation set provided with your IDE for complete information. From the menubar, choose **Help** ▸ **Help Contents**.

Additional commands for CVS are provided when the Capability **Team** ▸ **CVS support for Modeling** is enabled.

Using these commands assumes that option **Team** ▸ **Modeling Resources** ▸ **Auto Checkout modeling resources on edit** is off and that the project is checked out from CVS with option **Team** ▸ **CVS** ▸ **Watch/Edit** ▸ **Configure project to use Watch/Edit on checkout** turned on.

The Team command offers a submenu with the following commands:

Command	Description
Edit	Performs edit command for the diagram and model files (*.txv*/*.txa*) in the selected package.
Edit Recursively	Performs edit command for the diagram and model files in the selected package and all its subpackages recursively.
Unedit	Performs unedit command for the diagram and model files (*.txv*/*.txa*) in the selected package.
Unedit Recursively	Performs unedit command for the diagram and model files in the selected package and all its subpackages recursively.

Related Concepts

Together Capabilities Activation
Version Control in Together

Related Procedures

Saving and Loading Metric Results
Running Source Code Metrics
Running Source Code Audits

Related Reference

Common Diagram Context Commands

Common Element Context Commands

Select element ▸ Right-click

The context menus of the various elements provide functions specific to each element. For example, you can add or delete members from a class, cut-copy-paste, hide and show elements, and more. Explore the context menus of the different elements as you encounter them to see the commands available for each one.

However, for the most part, all of the UML elements do share common context menu commands. To use the context menu for an element, simply right-click on the element in the Diagram Editor .

Each element has the following common context menu commands:

Select in Model Tree

The Select in Model Tree command highlights the node selected in the Model Navigator tree-view. The Model Navigator will expand and highlight that element in the tree-view. If closed, the Model Navigator will open.

Select in Project

The Select in Project command displays project names that contain other occurrences of the selected design element. The icons next to the project names let you distinguish whether the referenced occurrence is in the home project of the element or in the read-only root of another project referencing the home project. Selecting a project name from this menu changes (or navigates) the element selection from the current project to the element's other occurrence in the referenced project. This navigation takes place whether the selection was made in the Model Navigator context or in a diagram context. Refer to the topics on navigation and cross-project references for further information.

Cut

One of the usual edit operations. This action deletes the source element from the current Diagram Editor . You can then Paste the element into a new location.

Copy

One of the usual edit operations. This action copies the selected element. After copying an element, you can Paste it into a new location.

Clone

The Clone command lets you quickly create a new element with the same content as the existing one. An element that can be cut/copied and pasted can also be cloned by using the Clone command. Cloning is basically a one-step copy-and-paste. For more information, see “Working with Custom OCL Operations.”

Paste

One of the usual edit operations. Using the Paste command, you can paste an element that has been cut or copied.

Note: Pasting elements from one package to another also maps relationships of those elements to the target package. By default, a prompt appears warning users of this before the paste is complete. To disable this warning, select **Window ▸ Preferences** from the main menu, choose the **Modeling** node, and uncheck the **Show warning about relationships when elements copied** option.

Paste shortcut

Use this command to paste a copied diagram element as a short cut on another diagram.

Rename

The Rename dialog renames the element and refactors the change throughout the project.

Delete

The Delete command will delete the element from the project. A Confirmation dialog opens to confirm the deletion.

Add Linked

Provides search options for references, implementations, and inheritance according to the specified types and scopes.

Model Bookmark

Bookmarks allow you to navigate to resources that are frequently used. You can set, remove, and view bookmarks using the Bookmarks view.

Hyperlinks

The Hyperlinks command offers a submenu for managing and viewing hyperlinks:

Option	Description
Edit	Using Edit, you can view, add, and remove hyperlinks to a project. For more information on hyperlinks, see “Hyperlinking Diagrams.”
<Hyperlink>	<Hyperlink> represents an actual hyperlinked element. If there are no hyperlinks for the diagram, then only the Edit option displays.

Requirements

The Requirements Management command offers a submenu with the following commands:

Option	Description
Manage Traces	Opens the Manage Traces dialog, which lets you define traces from a model element selected in the Diagram Editor or Model Navigator to CaliberRM or RequisitePro requirements. Using the dialog, you can view, add, and remove requirements associated with the element. For more information on linking requirements to diagram elements, see “Creating Traces from Requirements to Model Elements.”
Synchronize Traces	Opens the Trace Synchronizer dialog, which lets you search for traced CaliberRM requirements or model elements with local and server copies that become desynchronized

for some reason. The found desynchronized traces are displayed in the Trace Synchronizer view. For more information on synchronizing traces, see “Trace Synchronizer Dialog Box.”

If the selected element is associated with requirements, the submenu lists such requirements. Click on a requirement to open it in the CaliberRM or RequisitePro Navigator.

Print

Use the Print command to print a single diagram element. Use CTRL + CLICK to select multiple diagram elements for printing.

Optimize Size

Use this command to resize an element to its default size. For elements that contain subelements, the default size will respect the position of the subelements and remain large enough to show them.

Hide

Individual elements can be hidden in diagrams using the Hide command. If you do not see an element in a diagram, choose **Hide/Show** from the Diagram context menu and check the hidden elements list.

Properties

Using the Properties command opens the Properties View for the current element. For more information, see “Properties View.”

Related Procedures

Working with Custom OCL Operations
Hyperlinking Diagrams
Creating Traces from Requirements to Model Elements
Common Diagrams Procedures
Establishing cross-project references
Navigating between the Tree View, Diagram, and Source Code

Related Reference

Trace Synchronizer Dialog Box
Properties View

Common Link Context Commands

The context menus of the various link elements provide functions specific to each link. Explore the context menus of the different link elements as you encounter them to see the commands available for each one.

Shared Commands

For the most part, all of the link elements share common context menu commands. To use the context menu for a link, simply right-click the link in the Diagram editor.

Each link has the following common context menu commands:

Option	Description
Scroll to Source	If the required end of the link is out of reach, choose Scroll to Source to scroll to the client end of the link.
Scroll to Destination	If the required end of the link is out of reach, choose Scroll to Destination to scroll to the supplier end of the link.
Select in Model Tree	The Select in Model Tree command highlights the node selected in the Model Navigator tree-view. The Model Navigator will expand and highlight that element in the tree-view. If closed, the Model Navigator will open. For more information, see "Navigating between the Tree View, Diagram, and Source Code."
Delete	The Delete command will delete the element from the project. A Confirmation dialog opens to confirm the deletion.
Add Linked	Provides search options for references, implementations, and inheritance according to the specified types and scopes.
Hyperlinks	The Hyperlinks command offers a submenu with the following commands: **Edit**: Using Edit, you can view, add, and remove hyperlinks to a project. For more information on hyperlinks, see "Hyperlinking Diagrams." **<Hyperlink>**: <Hyperlink> represents an actual hyperlinked element. In the example shown above, Activity is a hyperlinked element. If there are no hyperlinks for the diagram, then only the Edit option displays.
Hide	Individual elements can be hidden in diagrams using the Hide command. If you do not see an element in a diagram, choose Hide / Show from the Diagram context menu and check the hidden elements list.
Properties	The Properties command opens the Properties view for the current element. For more information, see "Properties View."

Widely Encountered Commands

Perhaps not available on context menus for all diagram elements, the following commands are often encountered.

Option	Description
Add Linked	Provides search options for references, implementations, and inheritance according to the specified types and scopes.
Generate Class Diagram	The Generate Class Diagram command creates an exact copy of the diagram as a new class diagram. Active only where applicable. For more information, see "Creating a Diagram."
Link Type	Use the submenu list of the link types to specify an association, aggregation, or composition link.
Client Cardinality	Choose the appropriate cardinality from the drop-down list. Available cardinality choices are:

	0..1 1 0..* 1..*
Supplier Cardinality	Choose the appropriate cardinality from the list. Available cardinality choices are: 0..1 1 0..* 1..*
Add client qualifier	Use this command to designate a qualified association to reduce multiplicity for associations.
Add supplier qualifier	Use this command to designate a qualified association to reduce multiplicity for associations.

Related Procedures

Navigating between the Tree View, Diagram, and Source Code
Hyperlinking Diagrams
Creating a Diagram

Related Reference

Properties View

Model Bookmarks View

The Model Bookmarks view lists book-marked model elements. Using the context menu, you can navigate to the book-marked element on the diagram or remove a bookmark from the list.

Context Menu Commands

Show in Model Navigator	Highlights the element in the Model Navigator.
Select All	Selects all bookmarks.
Remove Bookmark	Removes one or more selected bookmarks.
Select on Diagram	Highlights the selected bookmark on the diagram. The diagram opens in the Diagram Editor, if necessary.

Compare Editor

Model ▸ Compare With ▸ Each Other (As Models or Model Elements)

Use the **Compare Editor** to review and merge differences in the structure and properties of the models which you have compared.

Structure Compare

Use the **Structure Compare** area to view the differences found during model comparison. Double-click the difference to open details in the **Changed Properties** and **Substructure/Properties Merge** areas.

Show Containment References	Toggles plain and structured view of the **Structure Compare** area. The structured view displays references to containment features (if any) for a particular difference.
Expand All	Expands all nodes in the **Structure Compare** area.
Collapse All	Collapses all nodes in the **Structure Compare** area.
Export Result	Opens the **Export Result** wizard that allows you to export the comparison results to a text file.

Substructure/Properties Merge

Use the **Substructure/Properties Merge** area to review the differences found in the structure and between properties of the compared models.

Undo	Steps one step back in the history of the applied commands.
Redo	Steps one step forth in the history of the applied commands.
Copy to the left	Moves the selected change from the right window (remote model) to the left window (local model).
Copy to the right	Moves the selected change from the left window (local model) to the right window (remote model).
Navigate	Shows the selected element in the **Model Navigator** view.

Problems

Use the **Problems** area to view problems found during model comparison.

Description	Displays the problem description.
Element	Displays the name of the problematic model element.
Source	Specifies in which model the problem occurs: **Left** (local model) or **Right** (remote model).

Related Concepts

Model Compare and Merge

Related Procedures

Comparing and Merging Models

Tool Palette

Window ▸ Show View

Together extends Palette of the platform of by adding model elements to it.

The Together diagram Palette displays special buttons for the supported UML diagrams. When a diagram opens in the Diagram Editor , the appropriate buttons are also displayed in the Palette.

In the diagram Palette you see the top level model elements that can be placed on the current diagram.

Note: The set of available model elements depends on the type of the current diagram and active profiles.

Related Procedures

Adding a Single Model Element to a Diagram
Creating a Simple Link

Diagram View

Context menu (in the Model View) ▸ Open Diagram

The **Diagram View** displays model diagrams. Each diagram is presented in its own tab.

To open the **Diagram View**, choose a diagram, namespace or a package in the **Model View**, right-click it and choose **Open Diagram** on the context menu.

Most manipulations with diagram elements and links involve drag-and-drop operations or executing right-click (or context) menu commands on the selected elements.

Some of the actions provided by the context menus are:

- Add or delete diagram elements and links
- Add or delete members in the elements
- Create elements by pattern
- Cut, copy, and paste the selected items
- Navigate to the source code
- Hyperlink diagrams
- Zoom in and out

Item	Description
Working area	The main part of the **Diagram View** shows the current diagram.
Context menu	The context menus of the **Diagram View** are context-sensitive. Right-clicking model elements, including class members, provides access to element-specific operations on the respective context menu. Right-clicking the diagram background opens the context menu for the diagram.
Overview button	Opens the Overview pane (see below).

Overview pane

The overview feature of the **Diagram View** provides a thumbnail view of the current diagram. The Overview button is located in the bottom right corner of every diagram.

OCL Editor

The OCL Editor is used to enter and edit OCL expressions. Any changes to the names of your model components (classes, operations, attributes, and so on) used in these expressions are automatically updated by Together. This guarantees that your OCL constraints always stay up-to-date.

Related Concepts

Together Diagram Overview
About OCL Support in Together

Related Procedures

Creating a Diagram
Navigating between the Tree View, Diagram, and Source Code

Metamodel Browser View

Window ▸ Show View ▸ MDA ▸ Metamodel Browser

Use the **Metamodel Browser** view to look up a metamodel that can be selected as a source or a target of a QVT transformation.

Toolbar

Item	Description
Go into	Expands the selected node and displays the nested contents.
Go back	Returns to the parent node.
Go home	Returns to the metamodel tree.
Collapse all	Collapses all open nodes.
Show inherited features	If this button is pressed, the inherited features are displayed in each node.
Show fully-qualified names	If this button is pressed, the fully-qualified names of the classifiers display.
Find classifier	Opens the **Find Classifier** dialog, where you can locate a metaclass by any substring of its name. Wildcards are supported.

Related Concepts

Model Transformation Support

Related Procedures

Applying Model-To-Model Transformations
Applying Model-To-Text Transformations

Related Reference

QVT Editor
QVT Builder

Model Package Explorer View

The Model Package Explorer displays the UML Content for all open Projects. This view combines useful visual and functional advantages from the Together Model Navigator with the Package Explorer. You can see all of your diagram elements at a glance while also keeping an eye on .java, XML, and all other project files. Context menus in this view also access commands from both the Model Navigator and Package Explorer.

Buttons

Button	Description
Back	Returns to previous tree view display.
Go Into	Expands the selected node and hides others from view.
Up	Returns view to parent structure level. This button becomes active after you use the **Go Into** context command.
Collapse	Collapses nodes in view.

Filters

In the upper-right corner of the Model Package Explorer view, the down-arrow menu offers the following commands:

Option	Description
Name filter patterns	If selected, opens the edit field in which you can type your own filter patterns to hide from view. Off by default.
Select the filters	Specifies resources to hide from view (.*files, empty parent packages, referenced libraries, Together diagram files, Together Model folder).
Filter description	Displays a description of the selected filter. Empty by default.

Select Working Set

Opens the Select Working Set dialog. Working sets allow you to customize your view, determining which resources are displayed. After a working set is selected, it becomes an additional option in the Model Package Explorer filters menu.

For more information on working sets, consult the documentation set provided with your IDE. From the menu bar, choose **Help** ▸ **Help Contents**.

Warning: Exercise caution when using the Remove button. Deletion is performed without prompt.

Deselect Working Set

Returns the Model Package Explorer to the default view.

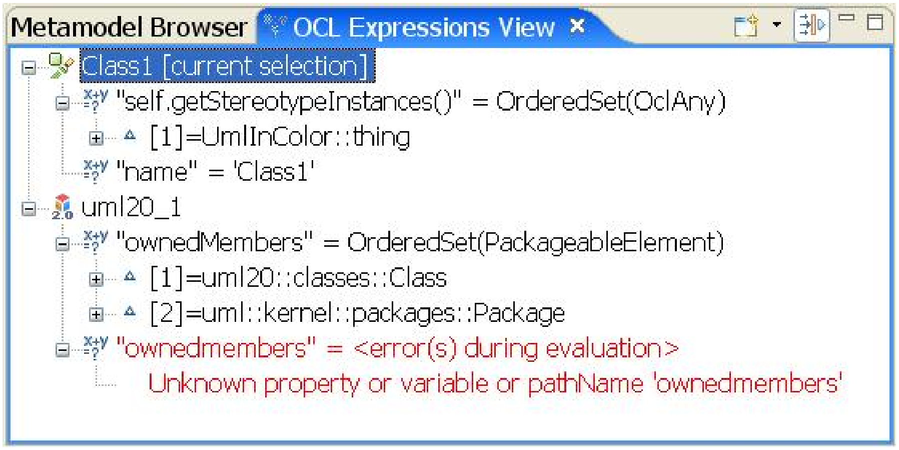
Edit Active Working Set

Makes changes to the current working set.

OCL Expressions View

OCLExpressionsView editor_context

Use the **OCL Expressions** view to quickly evaluate OCL expressions in the explicitly specified context (a Together or EMF model element), or in the context of the current selection. The screen shot below shows `Class1` from a UML 2.0 model in the current selection, and the model itself explicitly added as the second context.



```
Class1 [current selection]
  "self.getStereotypeInstances()" = OrderedSet(OclAny)
    [1]=UmlInColor::thing
  "name" = 'Class1'
uml20_1
  "ownedMembers" = OrderedSet(PackageableElement)
    [1]=uml20::classes::Class
    [2]=uml::kernel::packages::Package
  "ownedmembers" = <error(s) during evaluation>
    Unknown property or variable or pathName 'ownedmembers'
```

The children of the `Class1` element are expressions in the `uml20::classes::Class` OCL context. The children of the second element are expressions in the `uml::together::Model` OCL context . The **OCL Expressions** view displays a tree structure with several levels. The first level contains contexts, with the predefined **Current selection** context. The second level contains OCL expressions.

A context element specifies the model element against which Together evaluates child expressions. The OCL context of child expressions is defined by the type of the context element. For example, children of a UML 2.0 class context element will be evaluated in the `uml20::classes::Class` OCL context.

The **OCL Expressions** view preserves manually added contexts and OCL expressions after the Workbench restart. The **OCL Expressions** view evaluates your OCL expressions in real time as you work on your model. Evaluation results are displayed either in expression labels (for simple results), or as expression children (for model- or collection-like results). Evaluation errors display in red with the `<errors during evaluation>` label, and error messages are displayed as children of the expression node.

Note: To use OCL in EMF models, enable the corresponding metamodels in the **OCL Preferences** dialog box.

Toolbar

Item	Description
Add Together Model Element...	Opens the **Model Elements** dialog box, which lets you choose a context model or model element from your workspace and add it to the current list of contexts.
Add EMF Model Element...	Opens the **Workspace Contents** dialog box, which lets you choose a context EMF model or model element from your workspace and add it to the current list of contexts.
Add URI...	Opens the **Add model** dialog box, which lets you specify the URI of a context model or model element that you want to add to the current list of contexts.
Hide/Unhide Empty Nodes	Hides/Displays empty expression nodes.
Refresh	Reevaluates the selected OCL expressions and reloads the selected contexts. Note, that the **OCL Expressions** view does not automatically refresh contexts and expressions when the referenced models change.

Context Menu

Item	Description
Add Together Model Element...	Opens the **Model Elements** dialog box, which lets you choose a context model or model element from your workspace and add it to the current list of contexts.
Add EMF Model Element...	Opens the **Workspace Contents** dialog box, which lets you choose a context EMF model or model element from your workspace and add it to the current list of contexts.
Add URI...	Opens the **Add model** dialog box, which lets you specify the URI of a context model or model element that you want to add to the current list of contexts.
Add Expression...	Opens the **Edit...** dialog box, which lets you compose a new OCL expression and add it as a direct child of the selected context model or model element. The dialog box supports OCL error reporting and code completion options.
Edit...	Opens the **Edit...** dialog box, which lets you edit the selected OCL expression.
Enable Expression	Enables real-time evaluation of the selected OCL expression node and all its children.
Disable Expression	Disables real-time evaluation of the selected OCL expression node and all its children.
Delete	Removes the selected expression or context model element from the view.
Refresh	Reevaluates OCL expressions currently in the **OCL Expressions** view.

Related Concepts

About OCL Support in Together

Related Procedures

Creating Constraints
Opening MDA Views

Properties View

Context menu ▸ Properties

Every diagram and element has a general Properties View. The composition of the Properties View changes depending on the element or diagram selected in the Diagram Editor or Model Navigator. Use the Properties View to set properties for all diagrams and elements.

The Properties View displays properties in two columns: **Property** and **Value**. The **Property** column displays the names of the properties of a selected resource. The **Value** value column displays the values of the properties of a selected resource. Double-click on a value to edit.

Groups of properties display on the left of the Properties View.

The following keyboard shortcuts are available:

Ctrl+A	Select all
Ctrl+ ->/<-	Next or previous word
Del/Backspace	Remove current or previous character
Ctrl+Z/Y	Undo or redo
Ctrl+C/X/V, Shift+Del/Ins	Copy, cut, or paster
Ctrl+B/I/U	Bold, italic, or underline
Ctrl+M/Ctrl+Shift+M	Increase/Decrease indent (in a lists context)
Ctrl+Shift+Space	Insert non-breaking whitespace
Ctrl+-	Insert acronym
Ctrl+Space	Clear all formatting for the selection
Ctrl+Shift+8	Show whitespace characters (linebreaks)
F11	Show WYSIWYG only (without Source view; that is, in full-screen mode in the Inspector context)
Ctrl+Alt+I	Show Real Objects' About dialog with license information

Properties

This section displays the common properties of the selected object. The number of fields in this section varies depending on the selected diagram or element. See each element description for details.

Custom

This section displays the custom properties and their values specified for the selected object.

Add	Creates a new entry in the list of properties.
Remove	Deletes the selected entry from the list of properties.

Description

Use this field to add description text for a diagram or element. There are 'WYSIWYG and Source View tabs. Text in the Source tab is displayed with HTML formatting.

Words that are displayed with a red underline are not recognized by the editor's dictionary. A list of spelling checkersuggestions is available in the context menu. Select the word with a red underline and open the context menu to see spelling suggestions. You can disable automatic spell checking.

Note: You can configure the spelling settings from the **WYSIWYG** tab. The available Spelling properties are American English, British English, French, German, and Spanish.

Edit tab item	Description
Undo	Undoes an action
Redo	Redoes an action
B	Applies bold style
I	Applies italic
U	Applies underlining
S	Applies strikethrough
Align left	Applies align left
Align center	Applies align center
Align right	Applies align right
Align justify	Applies align justify
Decrease indent	Applies decrease indent
Increase indent	Applies increase indent
Add ordered list	Adds an ordered list
Add unordered list	Adds an unordered list
Insert table	Inserts a table
Insert row below	Inserts a row below the current row
Insert column left	Inserts a column to the left of the current column
Insert image	Inserts an image
Insert hyperlink	Inserts a hyperlink
Spelling settings	Apply spelling checker settings
Auto spelling checker	Enable or disable automatic spell checking

Hyperlinks

Use this field to add hyperlinks to the element.

Item	Description
Add hyperlink	Opens the Hyperlinks dialog, where you can select elements to be linked with the current element.
Remove	Deletes the selected hyperlink.
Remove all	Removes all hyperlinks from the element.

Requirements

You can track various requirements properties including type, priority, and difficulty for diagrams and individual elements. You can specify a requirements document for the diagram or elements.

traces	Displays the number of traces associated with the selected element.
author	Use this field to add author properties. Click the ellipse button to add values.
difficulty	Use the drop-down list to set difficulty to High or Low. Medium represents the default value.
document	Use the document field to link to a specific document.

number	Use the number field to assign a requirement number.
priority	Use the drop-down list to set priority to High or Low. Medium represents the default value.
req. description	Use this field to add a requirements description. Click the ellipse button to add text.
testcase	Use the text field to designate a testcase requirement.
type	Use the drop-down list to set the appropriate type. You can choose from Business Rule, Feature, Performance, Product Requirement, or User Need.

View

Diagrams do not have view properties. Each element (class, interface, object, and so on) will have the view properties listed below:

2D look	The property value can be set as Yes or No. Yes represents the default value.
background color	This field sets the RGB background color for the element. {255, 255, 255} represents the default color value. Use the drop-down list to choose a color.
font	This fields sets the font for the element.
foreground color	This field sets the RGB foreground color for the element. {0, 0, 0} represents the default color value. Use the drop-down list to choose a color.

Buttons for the Properties View

Show/Hide Categories:	This button groups lines under their appropriate categories.
Filter Properties	This button determines whether advanced properties are displayed in this view. Basic properties are always shown.
Restore Default Value	If you make changes to a value, this button restores the selected property to its default value.
Menu	Displays the Show/Hide Categories and Filter Properties commands.
Minimize	Minimizes current properties view to the view title. To show the entire view, click the Restore button.
Maximize	Maximizes the current view to the entire window. To restore the view, click the Restore button.
Information button	Available for certain properties. May display a small text editor for larger text entries, a selection wizard, or a file chooser dialog.
Chooser button	Available for certain properties. Displays the Selection dialog, enabling you to select an element from the Model.
drop-down button	Available for certain properties. Displays a list of available options to choose from.

Rich Text Editor Shortcuts

The following keyboard shortcuts are available for the Rich Text Editor:

Ctrl+A	Select all
Del/Backspace	Remove current/previous character
Ctrl+Z/Y	Undo/Red
Ctrl+C/X/V,Shift+Del/Ins	Copy, Cut, Past
Ctrl+B/I/U	Bold, Italic, Underlin
Ctrl+M/Ctrl+Shift+M	Increase/Decrease indent (this is reasonable in a lists context
Ctrl+Space	Clear all formatting in the selection

Ctrl+Shift+8	Show whitespace characters (linebreaks). Works with Windows and Linux, but has not been integrated on Mac OSX
Ctrl+->/<-	Next/Previous wor
Ctrl+Shift+Space	Insert non-breaking whitespace
F11	Shows WYSIWYG view only (without a Source view; that is, full screen in the context of the Inspector)
Ctrl+Alt+I	Show Real Objects' About dialog with license informatio

QVT Builder

Use **QVT Builder** to generate Java code from your QVT source files.

The generated code is placed in the Java source folder specified in the **QVT Settings** page of your transformation project **Properties** dialog box. You can then compile Java files using the Eclipse Java builder.

Related Concepts

Model Transformation Support

Related Procedures

Applying Model-To-Model Transformations
Applying Model-To-Text Transformations

Related Reference

QVT Editor

QVT Editor

Use the **QVT Editor** to write your QVT transformation.

The **QVT Editor** provides QVT code sense and auto-complete (CTRL+SPACE) options.

The QVT-specific **Outline** view displays an outline of the structure of the currently active QVT file in the editor area.

Related Concepts

Model Transformation Support

Related Procedures

Creating a Model-To-Model Transformation
Creating Model-To-Text Transformations

Related Reference

QVT Language
QVT Builder

XSL Editor

Use the **XSL Editor** to write your XSL transformation scripts.

The **XSL Editor** provides XSL code sense and auto-complete (CTRL+SPACE) options.

The XSL-specific **Outline** view displays an outline of the structure of the currently active XSL file in the editor area.

Related Concepts

Model Transformation Support

Related Procedures

Creating an XSL Transformation

Trace View

Use the **Trace** view to inspect the results of a Model-To-Model QVT transformation.

The view displays the trace file, which contains detailed information about every transformation step performed.

Item	Description
#	Displays the number of the executed transformation step.
From	Displays the object before the transformation.
To	Displays the object after the transformation.
Method	Displays the method in the source code that executes the transformation step.

Context Menu

Item	Description
Show Source	Navigates to the element that is the source of the selected transformation step.
Show Target	Navigates to the element that is the result of the selected transformation step.

Related Concepts

Model Transformation Support

Related Procedures

Opening MDA Views

Trace Synchronizer View

Window ▸ Show View ▸ Other... ▸ Requirements ▸ Trace Synchronizer

This topic provides information about the **Trace Synchronizer** view. You can use this view to find and fix desynchronized traces to CaliberRM or RequisitePro requirements.

Toolbar buttons and context menu items

Synchronize Traces	Opens the **Trace Synchronizer** dialog box.
Refresh trace synchronization information	Refreshes the trace information displayed in the **Trace Synchronizer** view.
Save as HTML	Opens the **Save As** dialog box, where you can export the current content of the **Trace Synchronizer** view to an HTML file.
Update Trace	Discards local changes and updates the selected traces from the repository.
Restore Trace	Discards changes in the repository and restores the requirement information stored in the model.
Delete Trace	Deletes the trace.
Navigate to Trace Source	Opens the trace source (requirement) in the CaliberRM or RequisitePro Navigator depending on the requirement type.
Navigate to Trace Target	Opens the trace target (model element) in the appropriate editor.

Columns

Status	Displays the status of the trace source.
Trace from	Displays the name of the trace source.
Trace from project	Displays the name of the source CaliberRM or Together project.
Status	Displays the status of the trace target.
Trace to	Displays the name of the trace source.
Trace to project	Displays the name of the target CaliberRM or Together project.
Status summary	Displays the summary information about the current trace status.

Status items

Not Found	Information about the object is not found.
Current	Information about the object is up to date.
Missed	Information about the object is missing.
New	The object is new.
Modified	The object has been modified.
Outdated	The object becomes outdated.

Related Concepts

Requirements Management

Templates View

Window ▸ Show view ▸ Other ▸ Patterns and Templates ▸ Templates

The **Templates** view displays currently available templates. These templates can be applied to open projects that have been template-enabled.

Together is shipped with a number of templates for the supported languages (Java, OMG CORBA IDL and C++). By default, each language contains a **Local Templates** node, which includes all the templates predefined and shipped with Together. They are divided into three categories: Link, Class, and Package. Each category contains subcategories related to the individual templates it contains.

Button	Description
Sort templates	Sorts the nodes in the view alphabetically.

Note: New projects will not be displayed in the Templates view until they contain a template.

Related Procedures

Working with the Templates
Editing Templates

Related Reference

Templates View Context Menus

Last Validation Results View

Window ▸ Show view ▸ Other ▸ Patterns and Templates ▸ Last Validation Results

The **Last Validation** view displays results of the latest validation of a pattern definition. This view opens automatically when the validation process reports errors.

Related Procedures

Validating Pattern Definition Projects

Patterns and Template GUI Components

This part describes GUI components of the Together interface you use for Together Pattern features.

In This Section

Pattern Explorer

Window ▸ Show view ▸ Other ▸ Patterns and Templates ▸ Pattern Explorer

The **Pattern Explorer** enables you to logically organize patterns (using virtual trees, folders and shortcuts), and manage recognized instances of patterns. You are working with shortcuts in Pattern Explorer, not with the actual patterns. Because of this, shortcuts to the same pattern may be included in several folders.

Context menu command	Description
Delete instance	Deletes a pattern instance from the model. When applied to a model folder, deletes all pattern instances.
Clear invalid instances	Deletes invalid instances of a pattern from the model.
Select on diagram	Sets highlight to the selected pattern instance in diagram. This command is available for pattern nodes.
Select in Model Tree	Sets highlight to the selected pattern instance in the Model Navigator. This command is available for pattern nodes.
Add as shortcut to diagram	Creates a shortcut to the pattern instance on the current diagram.
Verify pattern	Checks validity of the selected pattern.

Related Concepts

Patterns and Templates

Related Reference

Pattern Registry

Pattern Registry

Window ▸ Show view ▸ Patterns and Templates ▸ Pattern Registry

The **Pattern Registry** defines the virtual hierarchy of patterns. You can create virtual folders and group the patterns logically to meet your specific requirements. All operations with the contents of the **Pattern Registry** are performed in the **Pattern Explorer** and synchronized with the **Pattern Registry**.

Pattern Registry shows a tree of folders with shortcuts to patterns. The structure of the pattern registry is a simple tree with two separated subtrees with common root. These separated subtrees represent folder structures: one is taken from Eclipse extensions and another from workspace-specific local data. **Pattern Registry** allows several shortcuts to the same definition in different folders; therefore, internally, the registry keeps the plain list of definitions and a tree of folders and shortcuts to definitions.

The context menu of the pattern shortcuts in the Pattern Registry allows you to rename, copy, and delete the selected pattern. When attempting to delete a shortcut, you are prompted whether the shortcut should be deleted or the definition and all its shortcuts. In case the shortcut is the last one that refers to the corresponding pattern, the program warns you that deleting this shortcut will result in the loss of the corresponding pattern definition.

Command	Description
New	This command is available for the categories. You can choose to create a new nested category or a new pattern.
Rename	Opens the **Rename** dialog, where you can specify the new name of a category or a pattern.
Delete	Deletes the selected category or pattern. Requires confirmation.
Delete pattern definition	This command is available for the patterns. Deletes the pattern definition and all shortcuts to it from the Registry. Requires confirmation.
Copy	This command is available for the patterns. Copies the selected pattern to clipboard.
Cut	This command is available for the patterns. Cuts the selected pattern to clipboard.
Paste	This command is available for the categories. Pastes a pattern from the clipboard to the selected category.
Edit pattern definition	Opens a pattern definition project of the selected pattern.

Related Concepts

Patterns and Templates

Related Reference

Pattern Explorer

Templates View

Window ▸ Show view ▸ Other ▸ Patterns and Templates ▸ Templates

The **Templates** view displays currently available templates. These templates can be applied to open projects that have been template-enabled.

Together is shipped with a number of templates for the supported languages (Java, OMG CORBA IDL and C++). By default, each language contains a **Local Templates** node, which includes all the templates predefined and shipped with Together. They are divided into three categories: Link, Class, and Package. Each category contains subcategories related to the individual templates it contains.

Button	Description
Sort templates	Sorts the nodes in the view alphabetically.

Note: New projects will not be displayed in the Templates view until they contain a template.

Related Procedures

Working with the Templates
Editing Templates

Related Reference

Templates View Context Menus

Last Validation Results View

Window ▸ Show view ▸ Other ▸ Patterns and Templates ▸ Last Validation Results

The **Last Validation** view displays results of the latest validation of a pattern definition. This view opens automatically when the validation process reports errors.

Related Procedures

Validating Pattern Definition Projects

Quality Assurance GUI Components

Describes GUI components of the Together interface that you use for Together Quality Assurance features.

In This Section

Audit View

Use the Audits View to display and export audit results.

Audit View Toolbar

Item	Description
Quick Fix	Opens the **Quick Fix** dialog box and offers suggested quick fixes and suppression fixes as available.
Refresh	Recalculates the results that are currently displayed.
History...	Opens a submenu from which you can open a previously ran audit or clear the history.

Audit View Pull Down Menu

Item	Description
Show Suppressed Messages	Enables displaying suppressed messages.
Show Suppressed Messages Only	Filters the results table to show only suppressed messages.

Audit View Context Menu

Item	Description
Go To	Highlights affected code in the editor.
Fix	Opens the **Quick Fix** dialog box and offers suggested quick fixes and suppression fixes as available. Not all audit results have fixes. If no fix is available, the choices are dimmed. A lightbulb icon in the Audits View toolbar also indicates that a possible fix is available.
Detection Metrics	When a problem detection audit line is selected in the Audits View, the problem detection metric displays in the Metrics View.
Group By	Groups audits based on selection.
UnGroup	Ungroups audits.
Hide Selected	Hides specific audit results based on selection.
Show All Hidden	Shows hidden audit results.
Show Description	Displays a window with the full name and description of the selected audit.
Search in	Displays a dialog box for searching audit results.
Refresh	Reruns the same QA check on the selected resources.
Preferences	Displays the **Audit** tab of the **QA Source Preferences** dialog box.
Copy	Copies the currently displayed QA results into the clipboard, from which you can paste them into an external document.
Print	Displays the system print dialog allowing you to print the audit results set.
Load Audit Results	Load results of perviously run audits.
Export	Opens the **Export QA results to file** dialog box, where you can choose a format and destination for saving the displayed audit results.
Description	Displays the audit description in the help window.
Export Dialog	Opens the **Export QA results to file** dialog box.

Audit View Results Table

The results table displays only audit violations. For this reason, the results do not necessarily display all of the audits that you ran, or all the packages or classes that you processed.

Column	Description
Description	Describes why the audit flagged the item.
!	Indicates how serious, in general, violations of the audit are considered to be. This will help you sort the results and assess which violations are critical and which are not.
Resource	The source code file that was flagged by the audit.
In Folder	Displays the path to the folder that contains the file with a problem code.
Location	The line number in the file where the problem code is located.

Related Concepts

Quality Assurance

Related Procedures

Viewing Audit Results
Running Source Code Audits

Metric View

Use the **Metric** view to display and export metric results.

The metrics results report is displayed as a table in the Metrics View. The rows display the elements that were analyzed, and the columns display the corresponding values of selected metrics. Context menus of the rows and columns enable you to navigate to the source code, view descriptions of the metrics, and produce graphical output.

Metrics View Toolbar

Item	Description
Compare With...	Opens a submenu from which you can choose a previously ran metric to compare with currently displayed results.
Set Filter...	Opens a submenu from which you can choose a filter to narrow the list of currently displayed results.
Refresh	Recalculates the results that are currently displayed.
History...	Opens a submenu from which you can open a previously ran metric or clear the history.

Metrics View Context Menu

Item	Description
Go To	Highlights affected code in editor.
Load Metric Results	Opens a dialog box from which you can load a previously saved metric results file.
Export	Opens the **Export QA results to file** dialog box, from which you can choose a format and destination for saving the results displayed.
Kiviat Graph	Displays a Kiviat Graph for the selected resource. Graph displays in a new view.
Bar Graph	Displays a bar graph for the selected metric. Graph displays in a new view.
Search in	Displays a dialog box that lets you search metric results.
Refresh	Reruns the same QA check on the selected resources.
Preferences	Displays the **Metric** tab of the **QA Source Preferences** dialog box.
Copy	Copies the currently displayed QA results into the clipboard, from which you can paste them into an external document.
Print	Displays the system print dialog allowing you to print the metric results set.
Load Metric Results	Load results of previously run metrics.
Export	Opens the **Export QA results to file** dialog box, where you can choose a format and destination for saving the displayed metric results.
Description	Displays the metric description in the help window.
Export Dialog	Opens the **Export QA results to file** dialog box.

Related Concepts

Quality Assurance

Related Procedures

Running Source Code Metrics
Viewing Metric Results

Model Audits View

When you run model audits on your model, the **Model Audits** view with results table opens.

Model Audits View Elements

Option	Description
Description	Brief description of the audit item. For a full description, point to the audit in the list and see the description in the opened tooltip.
!	Severity of the audit item.
Parent Path	Name and path of the parent model item.
Entity Name	Name of the element that needs you attention.
Audit Text	Displays the code of the selected audit.

Model Audits View Toolbar

Option	Description
Refresh	Recalculates the results that are currently displayed.
History...	Opens a submenu from which you can open a previously ran audit or clear the history.

Model Audits View Context Menu

Option	Description
Go To	Highlights the affected element in the **Model Navigator** view and **Diagram Editor**.
Preferences	Displays the **Audit** tab of the **Model Quality Assurance Preferences** dialog box.
Group By	Groups audits based on selection. You can select **Group by Audit** or **Group by Entity**.
Ungroup	Ungroups audits.
Hide Current	Hides specific audit results based on selection. You can select from **Hide Current Row**, **Hide Current Audit**, or **Hide Current Entity**.
Show All Hidden	Shows hidden audit results.
Clean Up	Removes all audit results.
Export Results	Displays the **Export QA result to file** dialog box.
Load Audit Results	Loads results of previously run model audits.

Related Concepts

Model Audits

Related Procedures

Running Model Audits and Metrics

Related Reference

QA Model

Model Metrics View

When you run model metrics on your code, the **Model Metrics** view with results table opens.

Model Metrics View Elements

Option	Description
Entity	Name of the model element.
Metric Text	Displays the code of the selected metric. Each column with an abbreviation in its name represents a certain metric. Select a column and the corresponding metric code is displayed under the Metric Text.

Model Metrics View Context Menu

The **Export Results** command opens the Export QA result to a file dialog box.

Option	Description
Go To	Highlights affected element in the **Model Navigator** view and Diagram Editor.
Preferences	Displays the **Metric** tab of the **Model Quality Assurance Preferences** dialog box.
Kiviat Graph	Displays a Kiviat Graph for the selected resource. Graph displays in a new view.
Clean Up	Removes all metrics results.
Export Results	The **Export Results** command opens the **Export QA result to file** dialog box.
Load Metric Results	Loads results of previously run model metrics.

Related Procedures

Viewing Metrics as Graphs
Running Model Audits and Metrics

Related Reference

QA Model

Chart View

The Chart view displays a Kiviat or bar graph when you choose to view the metrics results as a graph.

Related Concepts

Quality Assurance

Related Procedures

Running Source Code Metrics
Viewing Metric Results

Together Projects

This part contains reference information about the supported Together project types and formats and project properties.

In This Section

Project Properties

Project ▸ Properties

You can modify the Together Project properties using the **Properties** dialog box. In the Model Navigator or in the Navigator, right-click a project and choose **Properties** on the context menu.

Among the general properties for a project, there are some properties specific to Together projects:

Open this property page...	to...
Model path	— Define the name of the model folder used to store diagrams and design elements. — Enable cross-project references in the list of imported projects.
QVT Settings	— Specify the Java source folder that is used for storing Java code generated by QVT Builder. This page is available for transformation projects only.
Profiles	— Select profiles to be used in the project.
QA Model	— View or modify the sets of audits and metrics to be used in the project. Note that you can only change sets of audits and metrics if you check the **Override workspace settings** check box.
QA Builder Properties	— View or modify the set of audits that is used during your source project build. Note that you can only change the set of audits if you check the **Enable project specific settings** check box.
Store package properties in package diagram files	— Preserves all properties of the package diagram, both visual and semantical, in the `default.txvpck` diagram file. If this option is not checked, only diagram-specific information (visual information, such as layout) is retained in the `default.txvpck` diagram file, while settings that you treat as package properties (semantical information, such as descriptions and custom properties) are moved from the `default.txvpck` file into the `default.txaPackage` file. Turning this option off allows you to track your package changes using version control. This option is on by default. Changing this option from this dialog converts the project files.
Create elements in separate files	— Create elements as standalone files. If this option is not checked, all design elements are stored in one file as file mates.
Sort elements in design files	— Enforces sorting of elements in the design and diagram files. Originally information saved without predefined order. Thus it was quite possible that minor change caused subsequent revisions of a file to look very different if compared as plain text or XML. If this option is checked information is ordered on save. Note: normally this is not needed and also it may take extra CPU time. Please don't change unless you clearly understand the implications.

Related Concepts

UML Profiles

Related Reference

QVT Settings
QA Builder Properties

C++ Projects

This section provides specific information related to C++ projects.

In This Section

Special Considerations for C++ Projects

In this section:

- Project configuration issues
- Header and implementation files
- Reverse engineering tips
- Processing syntax constructs

Project configuration issues

The task of creating and configuring C++ projects is important, especially for existing source code. The project must achieve a proper balance between the completeness of the model at runtime and resources used.

Two C++ language characteristics place limitations on C++ projects in Together:

- fully qualified class names in C++ need not correspond to the actual physical locations of the classes
- C++ uses a preprocessor

File Structure

The lack of correspondence between class names and source code files can impact memory demand. In order for Together to retrieve a class by name, all known classes must be visible when the project opens. Together must process all available files at the outset, thus increasing memory demand for C++ projects.

The memory demand becomes crucial when the project uses many libraries. The size of these libraries may be significant (for example, MFC, VCL, and OWL) even when the project itself makes use of only a small portion of the libraries.

Preprocessor

When working with the preprocessor, consider the following:

Macro definitions	The object model depends on resolving macro definitions. Because macros are used in the course of conditional compilation, they define portions of the source code that may or may not be displayed in the object model.
`#include` directives	Together projects are folder oriented. Header files and implementation (definition) files that reside in project folders are processed. Together parses the files that are specified in the `#include` directives, or in the entry points that are defined during project creation. A file that is an entry point is parsed once; an `#include` directive is parsed whenever it is encountered.

Entry points

In order to support the building of a correct model described by C++ code, Together introduces the concept of entry points. An entry point is a file explicitly defined as a point from which the parser will start.

There are certain differences in parsing between TCC and Together:

- The TCC parser processed all project files in an arbitrary order, taking into consideration the include directives. Each file was parsed only once.
- In Together the parsing order is explicitly defined by means of entry points and include directives. Each file is parsed as many times as it appears in the include directives that can be reached from the entry points.

Proper organization of entry points is critical for the correctness of the model and for the overall performance.

Header and implementation files

There are two kinds of C++ source code files: header files and implementation files. When you place a new class on a class diagram, Together creates a header file with a name that matches the class. An implementation file is created manually from an operation context menu (refer to “Creating Header and Implementation Files in C++ Projects”).

The C++ language options, specified in the Project Properties dialog, determine suffixes for each file.

#include

The files in the project path and in the include path determine the structure of a project.

Together parser exercises the same approach as all compilers do. While processing the `#include` directives, it parses header files as many times as they are included. Symbols found on multiple passes are merged according to their signatures in a manner similar to how a linker would operate when producing an executable from different object files.

In large-scale projects, all of the sources that are considered external (standard or third party libraries) with respect to the entire project structure should be added to the include path. This helps restrict the number of files handled within a project. Together parses all files in the include path.

#include conventions

Together uses the standard convention to discriminate between the project internal and external headers.

- External files are included via `#include <filename>` directives (with angle braces).
- Internal files are included via `#include "filename"` directives (with quotes).

Tip: `#include` directives automatically contain a relative path to the project root.

Reverse engineering tips

Existing C++ projects normally have an additional context, implicitly defined by the C++ dialect, and make/compile tools used. The Together parser cannot equally fit all possible combinations. As a result, opening an existing project may cause the parser to report unexpected errors that do not appear when the project is compiled.

To tune the Together parser to a specific project with its concrete combination, use the following:

- C++ dialect property in the project settings
- Skip standard includes property in the project settings
- Predefined macros property in the project settings
- `preinclude.inc`

Using `preinclude.inc` gives you more flexibility than using predefined macros. Together processes `preinclude.inc` before it processes any other C++ file available to the project. It treats `preinclude.inc` almost the same as other C++ files, except that it ignores the file's symbol declarations.

When a new project is created, an empty `preinclude.inc` file is created as a placeholder.

Particularly, you can use `preinclude.inc` to make some macros or additional reserved words defined for the Together parser explicitly by means of appropriate `#define` and/or `#include` directives.

Alternatively, you can make Together ignore all external `#include` directives and place the required directives into the `preinclude.inc` file. This option is available in the project properties.

Note: Normally you need just enough definitions to avoid unexpected parser errors. It may not be necessary to copy all of them, as they are in their original places. Instead you can use a shortened form.

Processing syntax constructs

Together parses syntax constructs and displays them in diagrams according to the C++ language options.

Union	If a union is encountered in the source code, it displays in the diagram as a class node with the stereotype `<<union>>`.
Structure	If a strucutre is encountered in the source code, it displays in the diagram as a class node with the stereotype `<<struct>>`. If the structure is anonymous, the name on the diagram node is the name of the structure variable (or the first such variable if there are more than one).
Typedef	Typedefs display in class diagrams as class nodes. The Model Navigator lists the types as ordinary model elements. Furthermore, if you use the Properties View of an attribute or an operation to select its type, typedefs are listed among the available model element types. If the typedef declaration simply creates another name for a known type, it is displayed on a class diagram as a rectangle with the stereotype `<<typedef>>` and the typedef identifier as the name. The notation does not support members or inheritance.
Forward declaration	When a forward declaration for a class or a type is encountered in the source code, it is displayed in the diagram as a class node with the stereotype `<<forward>>`, unless the definition of this class is available. Together lets you create declarations for the forward declarations using the context menu command.
Template specialization	When an explicit or implicit template specialization is encountered in the source code, it is displayed in the diagram as a class node and a template binding link to the template class. If a template specialization usage appears in the inheritance or as a type of attribute, the corresponding inheritance or association link goes to the model element that represents this specialization.
Global functions and variables	Currently not represented in diagrams.
Multi-Declarations	You can use multi-declarations of class members in the source code. For example: `class Class1 { private:int a,b;};` Such members are displayed in diagram elements as separate entries. However, deleting and editing in the Diagram Editor or in the Properties View are prohibited.

Related Concepts

Language Support

Related Procedures

Configuring C++ Projects
Creating, Editing and Opening Header and Implementation Files in C++ Projects

Related Reference

New project Wizard C++ Language-Specific Options

C++ Language-Specific Properties of the Model Elements

The Properties View displays these special fields for the C++ node elements, members and links:

Class and Interface properties	Description
definition file	Lists the filename of the associated definition file.
namespace	Text area for the class namespace.

Operation Properties	Description
return type	Combobox (**bool, char, double, float, int, long, long double, short, string, unsigned, unsigned long, and unsigned short.**) and file chooser to browse to your own type.
definition file	Text area and file chooser for the `*.cpp` file.
throw	Text area and file chooser button to help with exceptions.
visibility	**public**, **protected**, and **private**
virtual	Checkbox for adding a **virtual** modifier to a function.
const	Checkbox for adding a **const** keyword to a function.
inline	Checkbox for indicating an operation as **inline.**
volatile	Checkbox for adding the **volatile** keyword to an operation.

Attribute properties	Description
type	combobox (**bool, char, double, float, int, long, long double, short, string, unsigned, unsigned long, unsigned short**) and file chooser to browse to your own type.
visibility	**public, protected, private**.

Generalization link properties	Description
visibility	**public, protected, private**.
virtual	Checkbox for adding a **virtual** modifier to a function.

Related Concepts

Language Support

Related Reference

New project Wizard C++ Language-Specific Options

C++ Project Properties

Project ▸ Properties

Project properties are defined while the project is being created in the **New Project** wizard. You can modify these properties using the **Project Properties** dialog that displays the same fields.

Tip: Alternatively, choose **Properties** on the context menu of a C++ project node in the Model Navigator.

Tab	Description	
Project Source Path	Use this tab to define projects paths.	
	Use as a source folder	Use this button to add the selected package to the build path. Add the folder corresponding to the package to the build path if the package is the root of packages and source files. Entries on the build path are visible to the compiler and used for building.
	Remove from build path	Children of the folder will not be seen by the compiler anymore and will not be included when building the project.
	Toggle Read-Only Status	Use this button to make selected roots read-only or to clear the read-only attribute.
	Make Default Root	Use this button to choose the selected folder as the default root. This root is used as a target container when automatically creating new files.
	Exclude/Include	Use these alternative buttons to make the folder contents invisible or visible to the compiler.
	Configure inclusion and exclusion filters	Use this button to create the inclusion and exclusion filters instead of including and excluding each folder or file manually.
	Configure entry points	Use this button to add specific files as entry points. Usually, these are `*.cpp` files. You do not need to add header files, because they are processed from inclusion in `*.cpp` files. For more information, refer to “Special Considerations for C++ Projects.”
	Link additional source to project	Use this button to open the **Link Additional Source** dialog and add the sources that reside outside of the project.
Include Paths	Use this tab to include search paths to the project.	
	Include search paths	Folders in this area are included in the search path.
	Add	Click this button to add a folder to the project search path. Enter the path to the text field, or use the **Browse** button to locate the specific folder.
	Edit	Click this button to modify the include folder.
	Remove	Click this button to delete the selected folder from the path.
C++ Processing Settings	Use this tab to define C++ specific settings.	
	C++ generating class name prefix	Each new class name starts with the specified prefix.

	C++ generating definition file extension	Each new class has the specified extension.
	C++ generating file name prefix	Name of the file that contains C++ classes starts with the specified prefix.
	C++ generating file extension	File name has the specified extension.
	Recognize **wchar_t** as a data type	If this option is checked, the compiler recognizes the keyword **wchar_t** as a data type.
	Enable messenger	If this option is checked, the decorators describing compilation errors will be displayed in the **Problems View** and **Resource Navigator**.
	Package filter	Enter the names of the packages that you would like to filter out.
	Skip standard includes	If this option is checked, the `#include <filename>` directives (in angle brackets) are ignored. **Tip:** Do not confuse the name of this option with the standard C++ library.
	Predefined macros	Specify the list of predefined macros, which will be available for the whole project.
	Preinclude file name	Specify the name of the preinclude file (if any), to make its contents available for all participants of the project.
	New file default head comment	Specify the text that will be displayed in the generated C++ files.
	Recognize free comments as doc	If this option is checked, the Javadoc comments will be recognized.
	C++ dialect support	Select a C++ dialect from the list. The possible options are: GNU, MS, or pure C ++.
C/C++ Indexer	Together provides its own Borland indexer that reuses the results of project parsing. Using any other indexer results in increased memory consumption because the other indexers need to parse the Together project again.	

Related Concepts

Language Support

Related Reference

New project Wizard C++ Language-Specific Options
Special Considerations for C++ Projects

IDL Language-Specific Information

CORBA IDL inspectors have these special fields:

Item	Description	
Struct Properties	module	The text area to enter a module name for a struct.
Interface Properties	module	The text area to enter a module name for an interface.
	abstract	When this checkbox is checked, the Boolean property is set to `true`. If an interface is defined as local, its name is displayed in the diagram in italics.
	local	When this checkbox is checked, the Boolean property is set to `true`.
Interface and Valuetype Operation Properties	return type	**any, boolean, char, double, float, long, long double, long long, octet, short, string, unsigned long, unsigned long long, unsigned short**, and **void**, and a file chooser button to browse to your own return value.
	raises	The text area and file chooser button to associate an exception with an operation.
	oneway	Checkbox for adding a **oneway** modifier to an interface operation.
	context	The property editor for adding values to the **context** property. Multiple values are comma-delimited.
Interface and Valuetype Attribute Properties	type	**any, boolean, char, double, float, long, long double, long long, octet, short, string, unsigned long, unsigned long long, unsigned short**, and **wchar**, and a file chooser button to browse to your own type.
	read only	Checkbox for adding a **read only** modifier to an attribute.
	const	Checkbox for adding a **const** modifier to an attribute. If this property is set to `true`, the **initial value** property adds to the list of properties.
Valuetype Properties	module	The text area to enter a module name for a valuetype.
	supports	The text area and file chooser button for an interface name.
	abstract	Checkbox to add an **abstract** modifier to a valuetype.
	custom	Checkbox for adding a **custom** modifier to a valuetype.
	extends	The text area and a file chooser to define the parent valuetype.
Exception and Enumeration Properties	module	The text area to enter a module name for an exception or enumeration.
Exception and Struct Attribute (Member) Properties	type	**any, boolean, char, double, float, long, long double, long long, octet, short, string, unsigned long, unsigned long long, unsigned short**, and

	wchar, and a file chooser button to browse to your own type
Generalization Link Properties	A generalization link can be drawn between two interfaces or two valuetypes. It can also be drawn between an interface and a valuetype, where the client for the link is a valuetype and the supplier is an interface.
Union	Create unions with the **Class By Template** button of the Palette, choosing the Default Union template from the Templates list. Together provides limited support to the unions. For example, you cannot add an attribute from the diagram or create a link from the union module: Text area to enter a module name for a union. switch type: Shows union discriminator type. This is a read-only field.

Tip: CORBA IDL exception-type classes do not contain operations. These class types contain only attributes.

Related Concepts

Language Support

New Together Project Wizards

This section describes the common pages of the Wizards used to create new Together modeling projects, and language-specific pages for C++ and IDL projects.

In This Section

New Project Wizard Common Pages

File ▸ New ▸ Project ▸ Modeling ▸ < Project type>

These pages are common for the majority of project types provided by Together. For more information, refer to the topics that describe project settings for the specific project types.

Modeling Project page	
Project name	Use this text field to enter the project name.
Use default location	If this option is checked, the new project is created in the current workspace.
Location	Use this field to define the project location. This field is only available when the **Use default location** option is not checked.

Modeling Settings page	
Metamodel	These controls are only available for Java projects. You can choose the UML version to comply with. The default option is UML 2.0.
Start with Diagram	If this option is checked, the new project starts with the default package diagram. If this option is not checked, you can select the type of starting diagram from the drop-down list and specify its name.
Store package properties in package diagram files	If this option is checked, all properties of the package diagram, both visual and semantical, are preserved in the `default.txvpck` diagram file. If this option is not checked, only diagram-specific information (visual information, such as layout) is retained in the `default.txvpck` diagram file, while settings that you treat as package properties (semantical information, such as descriptions and custom properties) are moved from the `default.txvpck` file into the `default.txaPackage` file. Turning this option off allows you to track your package changes using version control. This option is on by default.
Create design elements in separate files	If this option is checked, the design elements are created as standalone. If this option is not checked, all design elements are stored in one file as file mates.

Profiles page	
Available Profiles	Displays the list of available profiles with checkboxes. The **Profile Description** field displays a brief description of the selected profile.
Select all	Checks all available profiles.
Deselect all	Unchecks all profiles.
Set Defaults	Resets profiles to the default settings

Related Procedures

Creating a Project

Related Reference

New project Wizard C++ Language-Specific Options
New project Wizard IDL Language-Specific Options

New project Wizard C++ Language-Specific Options

File ▸ New ▸ Project ▸ Modeling ▸ C++ Modeling Project

Access the properties for your existing project via **Project ▸ Properties** or **Project context menu ▸ Properties**.

Tab	Description	
Project Source Path	Use this tab to define projects paths.	
	Use as a source folder	Use this button to add the selected package to build path. Add the folder corresponding to the package to the build path if the package is the root of packages and source files. Entries on the build path are visible to the compiler and used for building.
	Remove from build path	Children of the folder will not be seen by the compiler anymore and will not be included when building the project.
	Toggle Read-Only Status	Use this button to make selected roots read-only or to clear the read-only attribute.
	Make Default Root	Use this button to choose the selected folder as the default root. This root is used as a target container when automatically creating new files.
	Exclude/Include	Use these alternative buttons to make the folder contents invisible or visible to the compiler.
	Configure inclusion and exclusion filters	Use this button to create the inclusion and exclusion filters instead of including and excluding each folder or file manually.
	Configure entry point	Use this button to add selected files from a package to the project in the **Configure Entry Points** dialog. The dialog displays a model tree with the check boxes for each file or folder. If a node is checked, it is considered an entry point. If a root is added to the project, all `*.cpp` files are automatically included in the project, but the header files should be added individually.
	Link additional source to project	Use this button to open the **Link Additional Source** dialog and add the sources that reside outside of the project.
Include Paths	Use this tab to include search paths to the project.	
	Include search paths	Folders in this area are included in the search path.
	Add	Click this button to add a folder to the project search path. Enter the path to the text field, or use the **Browse** button to locate the specific folder.
	Edit	Click this button to modify the include folder.
	Remove	Click this button to delete the selected folder from the path.
C++ Processing Settings	Use this tab to define C++ specific settings.	

	C++ generating class name prefix	Each new class name starts with the specified prefix.
	C++ generating definition file extension	Each new class has the specified extension.
	C++ generating file name prefix	Name of the file that contains C++ classes starts with the specified prefix.
	C++ generating file extension	File name has the specified extension.
	Support **wchar_t** as keyword	If this option is checked, the compiler recognizes the keyword **wchar_t** as a data type.
	Enable messenger	If this option is checked, the decorators describing compilation errors will display in the **Problems View** and **Resource Navigator**.
	Package filter	Enter the names of the packages that you would like to filter out.
	Skip standard includes	If this option is checked, standard includes are ignored.
	Predefined macros	Specify the list of predefined macros, which will be available for the whole project.
	Preinclude file name	Specify the name of the preinclude file (if any) to make its contents available for all participants of the project.
	New file default head comment	Specify the text that will be displayed in the generated C++ files.
	Recognize free comments as doc	If this option is checked, the Javadoc comments will be recognized.
	C++ dialect support	Select a C++ dialect from the list. The possible options are: GNU, MS, or pure C++.
Add CDT features to project	If this option is checked, CDT features become available in the project. You can access these features in the project properties dialog.	

Related Procedures

Creating a Project

Related Reference

C++ Projects

New project Wizard IDL Language-Specific Options

File ▸ New ▸ Project ▸ Modeling ▸ IDL Modeling Project

Access the properties for your existing project via **Project ▸ Properties** or **Project context menu ▸ Properties**.

Tab	Description	
Project Source Path	Use this tab to define projects paths.	
	Use as a source folder	Use this button to add the selected package to the build path. Add the folder corresponding to the package to the build path if the package is the root of packages and source files. Entries on the build path are visible to the compiler and used for building.
	Remove from build path	Children of the folder will not be seen by the compiler anymore and will not be included when building the project.
	Toggle Read-Only Status	Use this button to make selected roots read-only or to clear the read-only attribute.
	Make Default Root	Use this button to choose the selected folder as the default root. This root is used as a target container when automatically creating new files.
	Exclude/Include	Use these alternative buttons to make the folder contents invisible or visible to the compiler.
	Configure inclusion and exclusion filters	Use this button to create the inclusion and exclusion filters instead of including and excluding each folder or file manually.
	Link additional source to project	Use this button to open the **Link Additional Source** dialog and add the sources that reside outside of the project.
Include Paths	Use this tab to include search paths to the project.	
	Include search paths	Folders in this area are included in the search path.
	Add	Click this button to add a folder to the project search path. Enter the path to the text field, or use the **Browse** button to locate a specific folder.
	Edit	Click this button to modify the include folder.
	Remove	Click this button to delete the selected folder from the path.
IDL Processing Settings	Use this tab to define IDL-specific settings.	
	Preinclude file name	Specify the name of the preinclude file (if any) to make its contents available for all participants of the project.
	Show typedefs as classes	If this option is checked, typedefs display as classes in diagrams.
	Skip standard includes	If this option is checked, standard includes are ignored.
	Show natives as classes	If this option is checked, all types marked as natives display as classes in diagrams.
	Rename file when renaming class	If this option is checked, the container file is renamed together with its class.

Warn about not found include files	If this option is checked, a warning is displayed for the missing files.
Use preprocessor	If this option is checked, the existing macros are opened and includes are attached.
Predefined macros	Specify the list of predefined macros that will be available for the whole project.
Copy non-doc comments	If this option is checked, free comments are copied or moved together with the elements located next to them.
Recognize free comments as doc	If this option is checked, the Javadoc comments will be recognized.

Related Procedures

Creating a Project

New project Wizard Data Modeling Specific Options

File ▶ New ▶ Project ▶ Modeling ▶ Data Modeling Project

Access the properties for your existing project via **Project ▶ Properties** or **Project context menu ▶ Properties**.

Option	Description
Server	Choose the target database server to which the physical data model is bound.
Default schema	If this option is checked, the default schema with the specified name will be created during project creation. If this option is not checked, the project will be created without a schema. You can add a schema later using the **New** command of the project context menu.
Schema name	This field is only available when the **Default schema** option is checked. Use this text field to specify the name of the default schema.

Related Procedures

Creating a Data Modeling Project

Convert MDL Wizard

Use this wizard to create a design project around an existing IBM Rational Rose (MDL) model. The wizard is invoked by the **Design Projects** ▸ **Convert from MDL** template of the **New Project** dialog box.

Paths section

Button	Description
Add	Adds one model file to the Paths section. Press this button to open the **Select Model File** dialog box, navigate to the desired model file and click Open.
Add Folder	Adds all model files in the selected folder. Press this button to open the **Browse for Folder** dialog box, navigate to the desired folder that contains the model files and click OK.
Remove	Press this button to delete the selected entry from the Paths section.
Remove all	Press this button to delete all model files from the Paths section.

Options section

Option	Description
Scale factor	Specify the element dimensions coefficient. By default, the scale factor is 0.3.
Convert Rose default colors	If this option is checked, the default Rational Rose will be replaced with the default Together colors.
Preserve diagram nodes bounds	If this option is checked, user-defined bounds are preserved in the resulting diagrams. Otherwise, the default values are applied.
Convert Rose actors	This options enables you to choose mapping for the Rose classes with actor-like stereotypes (Actor, Business Actor, Business Worker, Physical Worker). If the option is checked, the Rose actors are mapped to Together actors. If the option is not checked, the Rose actors are mapped to the classes with the Actor stereotype.

Related Procedures

Importing a Project in IBM Rational Rose (MDL) Format

Related Reference

Together Projects

Import Together Project Wizard

File ▸ Import ▸ Modeling ▸ Together Project

Use this dialog box to migrate a legacy Together project to the current version of Together.

Migrate legacy Together project to Together <version>

Specify the Together project file and select the migrations type.

Item	Description	
Project Path	Click the Browse button to navigate to a specific source project.	
Diagram folders	This read-only area displays the folders of the legacy project that contain diagrams.	
Design elements storage policy	Use the radio-buttons in this section to define how to handle the design elements (as standalone or as file mates).	
	The same as in the original project	If this option is selected, the settings of the original project are preserved. The existing standalone design elements remain standalone. The new design elements are created according to the project settings.
	Force creating design elements in separate files	If this option is selected, all existing design elements are converted to standalone. All new design elements are created as standalone.
Migration type	Choose one of the possible ways to process the project roots.	
	Merge all roots contents into the new project	Click this radio-button to create a single project from a multi-rooted source project.
	Create a separate project for each root	Click this radio-button to create a Together project for each root.

Merged project name

This page will be displayed if the **Merge all roots contents into the new project** option is selected.

Item	Description
Project name	Enter the name of the resulting project. The default project name is constructed from the names of the last two folders of the source project file location.

Create a set of Together <version> projects

This page will be displayed if the **Create a separate project for each root** option is selected.

Item	Description
Root location	Displays the list of roots of the source project.
Together <version> project name	Displays the default name of the resulting project for the selected root. The default name is constructed from the package prefix, if any. If there is no

	package prefix, the project name is created from the names of the last two folders of the root location. Edit the project name as required.
Content type	Displays information about the type of contents in the selected root (design files or source code).
Diagram format	Displays information about the diagram format in the selected root, if any.
Decision	Select the way to handle information of the selected root. If the root contains design files, you can either copy them to the target location or skip the root. If the root contains source code files, you have the choice to copy it as is, copy and convert it to the design language, or skip the root.

Master project

This page is displayed when multiple projects are created.

Item	Description
Master Project Name	Specify the name of the master project that contains references to all projects created in the course of migration. The default name of the master project is based on the source project name. The master project is created to demonstrate the contents and structure of the source project. It is read-only and not intended for editing. Use the real projects to create or edit contents, and establish dependencies.

Related Concepts

Together Interoperability and Migration

Related Procedures

Importing Legacy Projects

Preferences

In This Section

Together Preferences

Window ▸ Preferences

The following pages of the **Preference** dialog enable you to modify the basic Together settings.

Tip: Some preference pages let you use keyboard shortcuts to set preferences. If you see a label with an underlined letter, you can use ALT + the underlined letter (for example, ALT + K) to quickly set the preference.

Generate Documentation Preferences

Use the **Generate Documentation** node to define options for documentation generated by Together.

Requirements Preferences

Use the **Requirements** node to customize global requirements management options for the products integrated with Together.

Modeling Preferences

On the **Modeling** page you can change deletion, ignored folders, and team sharing preferences.

Restore/Apply Buttons

All of the Together Preference dialogs have the following buttons:

Item	Description
Restore Defaults	Restores default selections for this dialog.
Apply	Applies selections made in this dialog.

Related Reference

Generate Documentation Preferences
Preferences

Generate Documentation Preferences

Window ▸ Preferences ▸ Generate Documentation

This node includes the following groups of options:

- Diagram Image Rotation
- Generate HTML
- HTML Output Options
- RTF Output Options

Related Concepts

Documentation Generation Overview

Related Reference

Diagram Image Rotation
Generate HTML Preferences
HTML Output Options
RTF Output Options
Preferences

Diagram Image Rotation

Window ▸ Preferences ▸ Generate Documentation ▸ Diagram Image Rotation

This node includes the following options:

Option/Button	Description
None	If this option is checked, the diagram image orientation is preserved.
90°	If this option is checked, the diagram image is rotated 90 degrees clockwise.
180°	If this option is checked, the diagram image is rotated 180 degrees.
270°	If this option is checked, the diagram image is rotated 270 degrees clockwise.

Related Reference

Generate HTML Preferences

Generate HTML Preferences

Window ▸ Preferences ▸ Generate Documentation ▸ Generate HTML

This node includes the following options:

Generate HTML

Option/Button	Description
Javadoc style	If this option is checked, the generated output corresponds to the Javadoc style (without navigation tree and diagrams).
Process line breaks	If this option is checked, line breaks are preserved.

Include Classes and Members options

Option/Button	Description
public	If this option is checked, elements with the public visibility modifier are included in the generated documentation.
protected	If this option is checked, elements with the protected visibility modifier are included in the generated documentation.
private	If this option is checked, elements with the private visibility modifier are included in the generated documentation.
package	If this option is checked, elements with the package visibility modifier are included in the generated documentation.
deprecated	If this option is checked, the deprecated elements are included in the generated documentation.

Include Tags options

Option/Button	Description
@author	If this option is checked, the `@author` tag is included in the generated documentation.
@version	If this option is checked, the `@version` tag is included in the generated documentation.
@param	If this option is checked, the `@param` tag is included in the generated documentation.
@return	If this option is checked, the `@return` tag is included in the generated documentation.
@see	If this option is checked, the `@see` tag is included in the generated documentation.
@since	If this option is checked, the `@since` tag is included in the generated documentation.
@throws	If this option is checked, the `@throws` tag is included in the generated documentation.
User-defined tags	If this option is checked, the user-defined tags are included in the generated documentation.

Javadoc options

Option/Button	Description
Window Title	Enter the window title.
Doc Title	Enter the title of the generated documentation.
Header	Enter the string that will be displayed in the header of each page of the generated output.

Footer	Enter the string that will be displayed in the footer of each page of the generated output.
Bottom	Enter the string that will be displayed at the bottom of each page.
Generate Tree	If this option is checked, the navigation tree is generated.
Generate Index	If this option is checked, index is generated.
Split index	If this option is checked, the index file is split into parts in alphabetical order.
Generate Use	If this option is checked, a table that shows usages of each class is included in the generated output.
Generate Deprecated List	If this option is checked, the list of deprecated elements is included in the generated output.
Generate Help	If this option is checked, a page that will be displayed on pressing the Help hyperlink is generated.
Generate Navigation bar	If this option is checked, a navigation bar is generated.
Stylesheet File	Specify stylesheet file.
Overview File	Specify path to a file that will be included in the generated documentation. A link to the overview file is placed at the top of the JavaDoc frame.

Related Concepts

Documentation Generation Overview

Related Reference

Generate Documentation Preferences
Preferences

HTML Output Options

Window ▸ Preferences ▸ Generate Documentation ▸ HTML Output Options

This node includes the following options:

Option/Button	Description
Process line breaks	If this option is checked, line breaks are preserved.

Related Reference

Generate HTML Preferences

RTF Output Options

Window ▸ Preferences ▸ Generate Documentation ▸ RTF Output Options

This node includes the following options:

Option/Button	Description
Render HTML tags	Check to translate HTML tags into appropriately formatted text in the printed documentation. The following tags are supported: `<b>,<i>,<u>,<h1> – <h6>, <code>, <tt>, <em>, <font color>, <pre>, <p>, <br>, <ol>, <ul>, <li>`
RTF Process line breaks	Check to preserve line breaks even if the Render HTML tags is checked on.
Store graphics in RTF	Check to embed all the graphics in a single RTF document.
Diagram image format	Select EMF or GIF format.
Color representation	Select RGB color or 16-bit color.
Included text formatting	Select whether to preserve the original formatting or apply the one from the Formatting template that is specified in the first option.

Related Concepts

Documentation Generation Overview

Related Reference

Generate HTML Preferences
Preferences

Modeling Preferences

Window ▸ Preferences ▸ Modeling

Use these preferences to change startup, deletion, error reporting, ignored folders, and team sharing options.

Copy/Paste Tab

Option	Description
Show warning about relationships when elements copied	Before elements are pasted into another package, prompts for a confirmation that relationships between elements will be mapped to the target package. This option is On by default.

Deletion Tab

Option	Description
Show confirmation when element is about to be deleted	Prompts for a confirmation before an element is deleted.
On pressing 'Delete' key always delete from:	
Model	Element is deleted from both model and view.
View only	Element is deleted from view, but remains in model.

Ignored Folders Tab

Use this tab to specify the folders you want Together to ignore. Usually this list contains **CVS, bin, lib** and **doc** directories. Ignored folders are not parsed, so no diagram will be generated for them.

Button	Description
Add	Opens the name field so you can enter a new folder.
Remove	Removes the selected folder.

Referenced projects Tab

Option	Description
Don't show referenced projects content under referring project node.	When this option is On, the content of the referenced project is not shown in the model tree of referring project. Note that in this mode it is impossible to copy content form the referenced project to referring one. This option is Off by default.

Team/Compare Tab

Use this tab to specify how you want to work with Team/Compare menus and version control in the Model Navigator.

Option	Description
Include diagram folders in Team/Compare	This option has an impact when the design and Java roots differ. When it is checked, Team/Compare (context menu) actions respect both folders (merged, seen as a single model node known as package in the Model Navigator). When it is unchecked, only folders from Java root are considered. If you do not store diagram elements in CVS, you should leave this option unchecked. When your diagram folders are in CVS alongside the folders from the Java root and you want to synchronize both of them with one action, you should check this option. A package in the Model Navigator is a logical view of two physical locations. One is the real directory in the project and the other is the directory under the model directory (named Together Model by default) where the Together diagrams are stored (these are updated automatically by Together and probably do not need to be shared). Default state is Off

Related Procedures

Diagrams

Related Reference

Preferences

Business Process Preferences

PreferencesBP

Use these preferences to change how Together checks types of events and activities in BPMN diagrams.

Check Activity and Event Type Options

Option/Button	Description
Auto update inconsistent element types on the diagram	If this option is selected, Together automatically fixes the type of the element when it becomes inconsistent. No coloring. The suggested event type is defined by the element input and output links. For Tasks, the presence of children is taken into account: if a Task contains at least one child, it automatically becomes an Embedded SubProcess. Auto update provides the fastest way to create a valid business process.
Highlight elements with inconsistent type	If this option is selected, the elements of inconsistent types are colored in red in the Diagram editor. This is a default option.
Prohibit creation of incorrect Sequence Flow links (between events only)	If this option is selected, Together does not allow events to be connected in the wrong order using a Sequence Flow link. No coloring.
Validation Only	If this option is selected, Together does not check events and activities as you draw a diagram. Provides constraint-free modeling, only with validation on. No checks beyond trivial ones (for example, you will not be able to create elements within containers that are not supposed to hold them). Constraint (specification compliance) checks are manual, using one of the Validate commands. No coloring.

Diagram Coloring

Option/Button	Description
Highlighting color	Opens the color chooser dialog to define a color to highlight invalid elements.
Color group	Uses “color coding” for group. Group color defines the background color for the elements it contains.

Simulation

Option/Button	Description
Inspector label for cost	Label of the property in the Properties view.
Report label for cost	Label that is used instead of `cost` in the simulation report. For example, if you type `CostName` in this field, the following names will be shown in the report: "Total CostName", "Total work CostName", and so forth.

Simulation Coloring

Option/Button	Description
Active status color	Color of the currently executing item on a diagram.
Wait status color	Color of the element waiting for an event (for example, message arrival) after which the element becomes active. Execution order is defined by the sequence flow links and if execution “pointer” does not reach a diagram element, the element is neither active nor waiting.

Related Concepts

Business Process Modeling

Related Procedures

Together Business Process Modeling

Related Reference

Preferences

Data Modeling Preferences

Window ▸ Preferences ▸ Modeling ▸ Data Modeling

Use these preferences to change database diagram notation and diagram display options.

Option	Description
Default diagram notation	Select a diagram notation (IDEF1x or IE) you want to use in physical data modeling.
Show icon for top-level elements	If this option is selected, top-level diagram elements are decorated with icons in the Diagram Editor .
Show owner	If this option is selected, the owner property is displayed in the name labels of tables and views in the Diagram Editor .
Show attribute datatype	If this option is selected, the value of the attribute type is displayed in the Diagram Editor .
Show attribute NOT NULL	If this option is selected, the value of the attribute not null is displayed in the Diagram Editor .

Related Concepts

Data Modeling

Related Procedures

Data Modeling Procedures

Related Reference

Preferences

Diagram Preferences

Window ▸ Preferences ▸ Modeling ▸ Diagram

Use these preferences to change how Together displays your diagrams:

Option/Button	Description	
Diagram	Font	Lists the current diagram font and size settings.
	Change	Click to open the system font dialog and change the font.
	Diagram toolbar visible	If this option is selected, the diagram toolbar is displayed in the Diagram Editor .
	Diagram background	Select to change the diagram background color.
	Antialiased graphics	If this option is selected, the diagram elements and graphics on the diagrams are displayed as anti-aliased.
	Antialiased text	If this option is selected, the text on the diagrams are displayed as anti-aliased.
	Show detailed feedback	If this option is selected, a translucent trace of the elements are displayed on drag-and-drop.
Rulers, Grid and Snapping	Show grid	Displays a background grid in the Diagram Editor .
	Snap to grid	Diagram elements, when dragged, will lock in place at the nearest grid point.
	Snap to geometry	Snaps selected objects for precise placement by aligning them with guides and other objects.
	Show snap feedback	Highlights guides and grids when an object aligns with them.
	Grid width	Determines width, in pixels, between grid points in a row.
	Grid height	Determines the height, in pixels, between grid points in a column.
	Grid color	Click the button to define the grid color.
	Grid type	Determines the grid type (line or dotted).
	Show rulers	Displays rulers with selected units.
	Store ruler guides	Saves created guides in your project upon exiting.
	Ruler units	Sets units of the rulers. You can select from centimeters, inches or pixels.
Palette	Show imported categories	Enables you to show the imported categories if required, or hide them if your diagram becomes overloaded.
Other	Show projection bars	If this option is checked, projection bars are displayed for the elements that can display their projections (Activity Partitions, Objects or Lifelines on Sequence diagrams, BPMN pool).
	Show chooser when link target is invalid	If this option is checked, the **Select target for the link** dialog is displayed when you drop the link on an invalid target location (for example, on the diagram background).
	Copy image along with diagram elements	If this option is checked, images of the copied diagram elements are also placed to the clipboard. You can paste these images to the other applications. In the large diagrams, this can slow down performance.

Synchronize source code editor to diagram	If this option is checked, the Editor highlights source code of the element selected in the diagram. Synchronization works for the elements already opened in the Editor.
If layout is changed on diagram open	With Together 2008 R3 it is possible to choose whether or not the changes to diagram layouts that are made automatically at open time should be saved. **Save** - Changes are silently saved (standard behavior up to version Together 2008 R2 SP1). **Ignore** - Changes are indicated with '*' in the name of diagram editor but not saved even when diagram is closed. Note: With the later choice any change to the diagram or underlying model explicitly initiated by the user will enforce saving of the changes (and the '*' marker will disappear). See Opening a Diagram procedure for details.

Tip: These selections become active on the current diagram when you click **Apply**.

Related Concepts

Together Diagram Overview

Related Procedures

Diagrams
Opening a Diagram

Related Reference

Preferences

EMF Model Compare Preferences

Window ▸ Preferences ▸ Modeling ▸ EMF Model Compare

Use these preferences to change how Together compares models.

EMF Model Compare

Use the **EMF Model Compare** tab to set global model comparison options.

Item	Description
Initially show containment references in difference overview	Specifies if you want to display references to containment features into the **Structure Compare** area of the **Compare Editor**.

ID Features

Use the **ID Features** tab to choose ID (key) EMF features for model comparison. By default, the `name` feature is set as ID throughout all metamodels (where applicable).

Item	Description
Metamodels	Displays the hierarchy of EMF metamodel classes with features that will be used as IDs during model comparison.
Home	Navigates to the root of the hierarchy.
Back	Navigates to the previously selected node.
Go Into	Navigates one level down from the selected node. The selected node becomes a root of the displayed view tree.
Features of class	Displays the list of features for the selected class. The checked features will be used as IDs during model comparison.
Reset local preferences	Resets ID features of the currently selected class to the default state.
Go To “Defined in”	Navigates to the feature where the selected containment feature is defined.

Ignored Features

Use the **Ignored Features** tab to choose EMF features that you want to ignore during model comparison.

Item	Description
Metamodels	Displays the hierarchy of EMF metamodel classes with features that will be ignored during model comparison.
Home	Navigates to the root of the hierarchy.
Back	Navigates to the previously selected node.
Go Into	Navigates one level down from the selected node. The selected node becomes a root of the displayed view tree.
Features of class	Displays the list of features for the selected class. The checked features will be ignored during model comparison.
Reset local preferences	Resets ignored features of the currently selected class to the default state.
Go To “Defined in”	Navigates to the feature where the selected containment feature is defined.

EMF Model File Compare

Use the **EMF Model File Compare** tab to set model file comparison options.

Item	Description
Use Model File Compare	Specifies if you want to enable the Model File compare feature.

Related Procedures

Comparing Models

Related Reference

Preferences

Export to UML2Tools Preferences

Window ▸ Preferences ▸ Modeling ▸ Export to UML 2 Tools

Use these preferences to change the options for exporting modeling projects to UML version 2.1.

Option	Description
Auto size converted nodes if size has not been modified by user.	Nodes that have not had their sizes modified in the legacy version will be automatically resized to a preferred size after the conversion. UML2Tools also has a different default font size from the legacy version. This preference sets the name label of converted node to nontruncated.
Always overwrite folder for diagram models.	If this preference is checked, the converter overwrites all diagrams silently. When the converter detects that the target folder is not empty, an alert is generated.
Always overwrite folder for model hyperlinks and requirement traces.	If this preference is checked, the converter overwrites all model hyperlinks and requirement traces. When the converter detects that the target folder is not empty, an alert is generated.
Invert Property's isUnique attribute meaning.	The default value of the MultiplicityElement's **isUnique** property is `false` in the legacy version and `true` in UML2Tools. When the converter detects that the value of the property is `false`, UML2T displays the corresponding label. This preference, on by default, avoids label complications by inverting the property labels to the UML2Tools default value for users who have never set the value in the legacy version.
Replace link name with its label.	This preference, on by default, replaces the name of legacy links that have a **label** property set with the value of the property.
Ignore synchronized package diagrams.	If this preference is checked, the converter overwrites all diagrams silently. When the converter detects that the target folder is not empty, an alert is generated. If this preference is checked, the converter prevents the diagram with the package contents from being converted.
Skip lnk* properties in source code elements.	When project links between source code elements are exported, the source association end gets duplicated. This preference, on by default, prevents this duplication by skipping the association's lnk* property during export.

Related Concepts

Model Import and Export Overview

Interaction Diagrams 2.0 Preferences

Window ▸ Preferences ▸ Modeling ▸ Interaction Diagrams 2.0

Use these preferences to customize view and editing options for the sequence and communication UML 2.0. diagrams.

Option/Button	Description	
Show	Sequence Numbers	If this option is selected, the sequence numbers are displayed in the Diagram Editor .
	Invocations	If this option is selected, method invocations are displayed in the Diagram Editor .
Edit	Delete preserves content of frame	If this option is selected, the deletion of a diagram element does not affect the contents of the deleted frame.
	Color Combined Fragments	If this option is selected, combined fragments display in color.
	Show “Select signature” dialog on message creation	If this option is selected, the **Select signature** dialog is displayed when a message is drawn to a lifeline. If the lifeline does not have any associated type, or this type does not contain any operation, the dialog is not displayed.
	Allow asynchronous call message to be straight	If this option is selected, the asynchronous call messages are drawn at the right angle to the target lifeline.

Related Concepts

About OCL Support in Together

Related Reference

Preferences

Java Preferences

Window ▸ Preferences ▸ Modeling ▸ Java

Use these preferences to define how Together handles your Java code.

Java tab

This tab contains Java-specific modeling options.

Option	Description		
Source code	This group contains the following options:		
	Update diagram immediately on changes in the text editor	If this option is checked, code changes made in the editor are reflected in the diagram without saving or rebuilding.	
	Save resources automatically on changes in the diagram	If this option is checked, changes made to diagram resources are saved automatically and immediately.	
	Optimize imports	If this option is checked, import statements are automatically optimized.	
	Call refactoring on rename of Java elements with default names (Class1, attribute1, etc.)	Renaming Java elements causes refactoring throughout the project.	
	Format source code on any changes	If this option is checked, source code is formatted upon making changes.	
Associations	This group contains the following options:		
	Association link name prefix	Specifies the letters to precede the field identifier in your code when you create links.	
	Show associations as attribute	The following options are available:	
		All	Shows all attributes representing association links.
		None	Shows no attributes representing association links.
		Automatic	Does not show attributes representing association links that have names starting with `lnk` (or whatever prefix you specify in the **Association link name prefix** field). Does show all other attributes representing association links.
Support comment-based associations (requires restart)	Enables Java comment-based associations. You can add comments to Java code to complete the concept that you are modeling, thereby		

	providing the information needed to properly render the model from source code. You must restart Together for this change to take effect.	
Link deletion	This group contains the following options:	
	Delete attributes on deletion of target types	If this option is checked, references are deleted from code when the target type is deleted.
	On deleting types, update extends and implements clauses	If this option is checked, extends and implements clauses are updated when changes are made to the associated element.
	On deleting types, update comment-based dependency links	If this option is checked, comment-based dependency links are updated when changes are made to the associated element.
Java Bean Properties Support	This group contains the following options:	
	Recognize Java Bean Properties	If this option is checked, Together recognizes Java Bean properties.
	Hide Java Bean Properties Participants	If this option is checked, Java Bean properties participants are hidden.

Name templates tab

In this tab you can define templates for the names of the new Java elements.

Option	Description
Class	Sets the default name for a new class element.
Interface	Sets the default name for a new interface element.
Field	Sets the default name for a new field element.
InnerClass	Sets the default name for a new inner class element.
InnerInterface	Sets the default name for a new inner interface element.
Method	Sets the default name for a new method element.
Enum	Sets the default name for a new **Enum** element.
Enum constant	Sets the default name for a new **Enum** constant element.

Members order tab

In this tab you can choose the order in which members are displayed in a diagram.

Option	Description
Order of creation of the new members	
Member order	Sets the display order for members. The default order is: **Field — Constructor — Method — Inner Class — Inner Interface**.
Visibility order	Sets the visibility display order for code elements. The default order is: **Public— Package — Local Protected — Private**.
Up Down	Click Up and Down to set visibility and member ordering.

Miscellaneous	
Maximum width of class element	Defines the maximum width of class elements in pixels.
Sort class elements alphabetically inside compartments	If selected, the class elements are sorted alphabetically inside compartments.
Open source editors	If selected, the Java editor opens when an element is added to the diagram.

Related Concepts

Roundtrip Engineering Overview

Related Procedures

Creating and Editing Properties

Layout Preferences

Window ▸ Preferences ▸ Modeling ▸ Layout

Use these preferences to define the alignment of diagram elements.

Option/Button	Description
Links Layout	Determines the shape of the links (direct or rectilinear).
Algorithm	Click the drop-down arrow to select a layout algorithm. UML diagrams can be thought of as graphs (with vertices and edges). Therefore, graph data structures (algorithms) can be applied to the UML diagrams for diagram layout. The various algorithms and their optional settings are described below. The algorithm that you specify executes when you lay out your diagram. See the detailed descriptions of the Algorithm-specific options in the subsections that follow.
Recursive	This option is available for all layout algorithms. Selecting this option lets you lay out all subelements within containers while laying out diagram nodes.

Algorithm-specific options are described in the following subsections.

<Autoselect>

Option/Button	Description
<autoselect>	Each of the layout algorithms contains internal information about the types of diagrams it will work with and the numeric characteristics for the final quality of the produced layout when applied to each applicable diagram type. Several algorithms can be available for the same diagram type. The <autoselect> option uses such internal information and picks the best layout algorithm for the current diagram type.

Hierarchical

The Hierarchical algorithm originates from the Sugiyama algorithm. The algorithm draws the UML diagram hierarchically according to the preferences that you select.

Option/Button	Description
Vertical and **Horizontal**	Minimal distance between elements in pixels. Here you can specify Vertical and Horizontal distance options.
Justification	This option defines the alignment of classes. The Justification setting is dependent on the Inheritance setting. Select from the following:

Top	If the Inheritance option is set as Vertical, then all nodes in a column are aligned at the left of the column. If the Inheritance option is set as Horizontal, then all nodes in a row are aligned at the top of the row.
Center	If the Inheritance option is set as Vertical, then all nodes in a column are aligned at the center of the column. If the Inheritance option is set as Horizontal, then all nodes in a row are aligned at the center of the row.
Bottom	If the Inheritance option is set as Vertical, then all nodes in a column are aligned at the right of the column. If the Inheritance option is set as Horizontal, then all nodes in a row are aligned at the bottom of the row.

Layer ordering	The heuristics are used to sort nodes within each layer to minimize edge-crossings.

Barycenter	The Barycenter heuristic reorders the nodes on node N according to the barycenter weight. The weight of node N is calculated as a simple average of all its successors/predecessors relative coordinates.

	Median	The Median heuristic reorders the nodes on node N according to the median weight. The weight of node N is calculated as a simple average of this nodes' relative positions dealing only with two central successors/ predecessors coordinates.
	Hybrid	The Hybrid heuristic combines the Median and Barycenter heuristics with the Proportion (see below) setting.
Inheritance	This option defines how classes are aligned with each other if they are connected by an inheritance link.	
	Horizontal	Classes connected by inheritance are aligned horizontally
	Vertical	Classes connected by inheritance are aligned vertically
Proportion	Used in conjunction with the **Hybrid ordering** option. The optimal setting for this value is 0.7.	

Tree

The algorithm draws the given graph in a tree layout according to its maximum spanning-tree.

Option/Button	Description	
Process non tree edges	If this option is selected, non-tree edges are bent to fit into the diagram layout.	
Horizontal and Vertical	Minimal distance between elements in pixels. Here you can specify Vertical and Horizontal distance options.	
Justification	This option defines the alignment of elements. The Justification setting is dependent on the Hierarchy direction setting. Select from the following:	
	Top	If the Hierarchy direction option is set to Vertical, then all nodes in a column are aligned at the left of the column. If the Hierarchy direction option is set to Horizontal, then all nodes in a row are aligned at the top of the row.
	Center	If the Hierarchy direction option is set to Vertical, then all nodes in a column are aligned at the center of the column. If the Hierarchy direction option is set to Horizontal, then all nodes in a row are aligned at the center of the row.
	Bottom	If the Hierarchy direction option is set to Vertical, then all nodes in a column are aligned at the right of the column. If the Hierarchy direction option is set to Horizontal, then all nodes in a row are aligned at the bottom of the row.
Hierarchy direction	This option defines the hierarchy direction of the elements	
	Horizontal	Elements are aligned horizontally.
	Vertical	Elements are aligned vertically.
Reverse hierarchy	Last in the hierarchy elements are laid out first in the diagram.	

Orthogonal

The Orthogonal algorithm uses heuristics to distribute diagram nodes within a lattice.

Option/Button	Description	
Node placement strategy	There are three strategies for node placement: Tree, Balanced, and Smart.	
	Tree	The Tree node placement strategy creates a spanning-tree diagram layout. The spanning-tree for the given graph is calculated and diagram nodes are placed on the lattice to minimize the tree edges length. This minimizes the distance between nodes that are linked with a tree-edge.

	Balanced	The balanced node placement strategy uses a balanced ordering of the vertices of the graph as a starting point. Balanced means that the neighbors of each vertex V are as evenly distributed to the left and right of V as possible.
	Smart	The Smart node placement strategy sorts all vertices according to the in/out degrees for each vertex and fills the lattice starting from the center with the vertices with the greatest degree.
Distance between elements	Specifies the minimum distance between diagram elements. Distance is in pixels.	

Spring Embedder

The Spring Embedder algorithm is force-directed layout algorithms that model the input graph as a system of forces and try to find a minimum energy configuration of this system. All edges are drawn as straight lines. This type of layout is especially suitable for projects with numerous diagram elements based on large amount of source code. When you lay out the graph according to the **Spring Embedder** layout algorithm, the program will simulate the graph as a physical model (masses and springs) and subject it to physical forces. The unnecessarily long edges will be the most tense and will try to contract the most. When the nodes and edges have pushed and pulled themselves to equilibrium, you will have a geometric representation of the graph.

Option/Button	Description
Movement	Specify the nodes movement factor. The more value you specify, the more distance will be between the nodes in the final graph. If you specify 0 as the movement factor, you will get a random layout of the nodes.
Spring force	Specify the rigidity of the springs. The greater value you specify, the less the length of edges will be in the final graph. **Tip:** Lay out your graph with the default spring settings first and then edit the spring options if necessary.

Together

The following layout options are used in the legacy versions of Together.

Option/Button	Description	
Layout inheritance	This option defines how classes are aligned with each other if they are connected by an inheritance link. Select either:	
	From left to right	Classes connected by inheritance are aligned horizontally from left to right.
	From top to bottom	Classes connected by inheritance are aligned vertically from top to bottom.
	From right to left	Classes connected by inheritance are aligned horizontally from right to left.
	From bottom to top	Classes connected by inheritance are aligned vertically from bottom to top.
Layout justification	This option defines the alignment of classes. The Justification setting depends on the Inheritance setting. The elements are aligned as summarized in the following table.	

Inheritance	Justification	
Left-right	Top	Right of the column
	Center	Center of the column
	Bottom	Left of the column
Right-left	Top	Left of the column

	Center	Center of the column
	Bottom	Right of the column
Top-bottom	Top	Bottom of the row
	Center	Center of the row
	Bottom	Top of the row
Bottom-top	Top	Top of the row
	Center	Center of the row
	Bottom	Bottom of the row

Related Reference

Preferences

OCL

Window ▸ Preferences ▸ Modeling ▸ OCL

Use the **OCL** preferences to specify auxiliary OCL operations and other OCL settings.

OCL Metamodels

Use the **OCL Metamodels** tab of the **OCL** preferences to choose which metamodels you want to use with OCL. The tab contains the list of all available metamodels; the required Together metamodels are listed under the **Together** node:

- UML 1.4 Project
- UML 2.0 Project
- Business Process Modeling Project
- Database Modeling Project

Warning: Do not deselect the required Together metamodels.

The user can select any metamodel and thus make it visible to the OCL processor. This means that metaclasses of the selected metamodel can be used as the contexts for the OCL operations created in the **OCL Operations** tab.

It is not possible to use the arbitrary metamodels as the contexts for the audits, metrics, documentation generation expressions and search, but you can use the features and operations of these metaclasses in the bodies of the audits and metrics.

For example, the user can create a metamodel for a profile. This metamodel is displayed in the list of available metamodels. If this metamodel is selected, it becomes possible to create operations with the respective context and create audits that will evaluate element properties specific for the selected profile.

OCL Operations Options

Use the **OCL Operations Options tab** of the **OCL** preferences to define custom OCL operations that you want to use in OCL queries throughout Together—in audits, metrics, search expressions, gendoc templates, and so on. As an example, you can use the predefined set of OCL operations created for Together audits. OCL operations are defined in the standard way, using `ocl def:` expressions.

Option	Description
Name	Displays names of auxiliary OCL operations defined upon the Borland metamodel.
New	Opens the **Edit Operation** dialog box, which lets you choose the context and provide the body of a new operation.
Remove	Removes the selected operation from the list.
Edit	Opens the **Edit Operation** dialog box, which lets you choose the context and edit the body of the selected operation.
Import...	Imports a text file containing auxiliary OCL operations. Use the **Export...** command to create the file. **Important:** During the import, OCL operations defined in the file replace the current list of operations.
Export...	Exports the current list of auxiliary OCL operations to the text file with the `.oclOperations` extension.

OCL Library Operations

Use the **OCL Library Operations** tab of the **OCL** preferences to view (not edit) the list of signatures of library operations that have been implemented as native extensions to OCL. The list includes a number of powerful `String` operations that are not defined in the OCL specification.

Option	Description
Operations	Displays a list of auxiliary OCL operations defined for the Borland metamodel in Java libraries.
Select All	Checks all operations in the list.
Clear All	Unchecks all operations in the list.

For a description of predefined library operations with native OCL extensions, refer to "Predefined OCL Library Operations."

Model Names Mappings

Use the **Model Names Mappings** tab of the **OCL** preferences to define mappings for keywords and other illegal symbols or words that you cannot use in OCL expressions directly.

Option	Description
Whole name	Specifies if you want to apply the mapping rule only to whole words matching the find substring. Thus, if the substring is "body" and the replacement is "_body", all occurrences of the word "body" will be replaced by "_body" (but, for example, "mybody" will not).
Substring	Defines a find substring. Before validating an OCL expression, if the substring is found in the expression body, it will be replaced by one defined in the **Replacement** field. To avoid a naming conflict, provide replacements for all OCL keywords that can appear in your model elements names.
Replacement	Defines a replace substring.
Select All	Checks all operations in the list.
Clear All	Unchecks all operations in the list.
New	Adds a new empty mapping rule to the list.
Remove	Removes the selected mapping rule from the list.
Edit	Opens a dialog box that allows you to change the Substring and Replacement values.

Related Concepts

OCL Support

Related Procedures

Working with Custom OCL Operations

Related Reference

Predefined OCL Library Operations
Java Source Generation Preferences

Predefined OCL Library Operations

Window ▸ Preferences ▸ Modeling ▸ OCL

Use the **OCL Operations Options tab** of the **OCL** preferences to define custom OCL operations to use in OCL queries. You can also use the following predefined set of OCL operations created for Together audits.

Library Operation	Description
OclAny::addStereotypeInstance(instance: OclAny): OclVoid	Adds a stereotype instance to the model element. For more information on OCL profile library operations, refer to “EMF API for Together Profiles.”
OclAny::allInstances(type: OclType): Set (OclAny)	Returns a Set of all instances of a datatype passed as an argument contained in the context. For example, `model.allInstances (uml20::classes::Class)` collects all classes in the model.
OclAny::getResourceContents(uri: String): Bag(OclAny)	Returns the contents of a resource created for the given URI string. See the description of the org.eclipse.emf.ecore.resource.Resource.Factory.createResource (URI uri) method in the Eclipse EMF API reference for more information.
OclAny::getStereotypeInstances(): OrderedSet(OclAny)	Returns all stereotype instances for the context model element. This operation can retrieve values of custom properties defined in profiles. The set returned is not live and does not automatically update when other clients perform changes affecting its contents, or when a profile is switched on or off for the model. For example, `getStereotypeInstances().oclAsType (My_Profile::My_Stereotype)->any(true).My_Tag` For more information on OCL profile library operations, refer to “EMF API for Together Profiles.”
OclAny::isStereotypeApplicable (stereotype: OclType): Boolean	Checks whether a given stereotype can be applied to the model element. For more information on OCL profile library operations, refer to “EMF API for Together Profiles.”
OclAny::removeStereotypeInstances (instances: OrderedSet(OclAny)): OclVoid	Removes stereotypes instances from the model element. For more information on OCL profile library operations, refer to “EMF API for Together Profiles.”
String::endsWith(suffix: String): Boolean	Compares the end of a string in a context to a specified suffix. This operation is similar to the `endsWith(String suffix)` method of the *java.lang.String* class.
String::indexOf(str: String): Integer	Searches forward in a string in a context for a substring passed as an argument. This operation is similar to the `indexOf(String str)` method of the *java.lang.String* class.
String::indexOf(str: String, fromIndex: Integer): Integer	Starting from the specified index, searches forward in a string in a context for a substring passed as an argument. This operation is similar to the `indexOf(String str, int fromIndex)` method of the *java.lang.String* class.
String::lastIndexOf(str: String): Integer	Searches backward in a string in a context for a substring passed as an argument. This operation is similar to the `lastIndexOf(String str)` method of the *java.lang.String* class.
String::lastIndexOf(str: String, fromIndex: Integer): Integer	Searches backward in a string in a context for a substring passed as an argument. This operation is similar to the `lastIndexOf(String str, int fromIndex)` method of the *java.lang.String* class.
String::replace(regex: String, with: String): String	Replaces the first substring of the string in a context that matches the given regular expression with the given replacement. This operation is similar to the `replaceFirst(String regex, String replacement)` method of the *java.lang.String* class.

String::replaceAll(regex: String, with: String): String	Replaces each substring of the string in a context that matches the given regular expression with the given replacement. This operation is similar to the `replaceAll(String regex, String replacement)` method of the *java.lang.String* class.
String::split(regex: String): Sequence (String)	The collection of Strings returned by this method contains each substring of the string in a context that is terminated by another substring that matches the given regular expression or is terminated by the end of the string. The substrings in the collection are in the order in which they occur in this string. This operation is similar to the `split(String regex)` method of the *java.lang.String* class.
String::split(regex: String, limit: Integer): Sequence(String)	The collection of Strings returned by this method contains each substring of the string in a context that is terminated by another substring that matches the given regular expression or is terminated by the end of the string. The substrings in the collection are in the order in which they occur in this string. The limit parameter controls the number of times the pattern is applied. This operation is similar to the `split(String regex, int limit)` method of the *java.lang.String* class.
String::startsWith(prefix: String): Boolean	Compares the start of a string in a context to a specified prefix. This operation is similar to the `startsWith(String prefix)` method of the *java.lang.String* class.
String::startsWith(prefix: String, toffset: Integer): Boolean	Compares the start of a string in a context to a specified prefix. This operation is similar to the `startsWith(String prefix, int toffset)` method of the *java.lang.String* class.
String::toLowerCase(): String	Converts all of the characters in the string in a context to lowercase using the rules of the default locale. This operation is similar to the `toLowerCase()` method of the *java.lang.String* class.
String::toUpperCase(): String	Converts all of the characters in the string in a context to uppercase using the rules of the default locale. This operation is similar to the `toUpperCase()` method of the *java.lang.String* class.
String::trim(): String	Returns a copy of the string in a context, with leading and trailing white space omitted. This operation is similar to the `trim()` method of the *java.lang.String* class.
uml::kernel::Element::getIncomingLinks ():Set(uml::together::BinaryLink)	Returns a collection (`Set`) of all links in the model whose target is an element in a context. For example, the following expression returns a Set of all incoming associations of an element: `self.getIncomingLinks()->select(oclIsTypeOf (uml20::kernel::KernelAssociation))`
uml::kernel::Element::isAssignableFrom (element: uml::kernel::Element): Boolean	Compares the metaclasses of an element in a context with the element passed as an argument. If the metaclasses are the same, or if the metaclass of the context element is a superclass of the argument's metaclass, this operation returns the Boolean value of true.

Related Reference

EMF API for Together Profiles
QA Source

Patterns Preferences

Window ▸ Preferences ▸ Modeling ▸ Patterns

Use the **Patterns Preferences** to define pattern conversion profiles.

Option	Description
Pattern conversion profiles	Lists available pattern conversion profiles.
Edit	Opens the **Edit Transformation Profile** dialog box, which lets you change the selected pattern conversion profile.
Remove	Removes the selected pattern conversion profile from the list.
Import...	Opens the **Import Pattern Conversion Profiles** dialog box, which lets you import one or more profiles stored in the `.xml` file.
Export...	Opens the **Export Pattern Conversion Profiles** dialog box, which lets you export one or more profiles to the `.xml` file.

Related Concepts

Patterns and Templates

Related Procedures

Patterns and Templates

Print Preferences

Window ▸ Preferences ▸ Modeling ▸ Print

Use this dialog box to define settings of the printed output.

Option	Description	
Size and Orientation	Paper size	Determines the size of the paper to be used. If you set the option to custom, you can use the Custom Paper Size tab to define the size. Default size is A4.
	Orientation	Sets the orientation of the paper. Default orientation is Portrait.
	Width (in.)	Determines the width, in inches, of the custom paper size. Default value is 8.5.
	Height (in.)	Determines the height, in inches, of the custom paper size. Default value is 11.
Margins (inches)	Top (in.)	Determines the size of the top margin in inches. Default value is 1
	Bottom (in.)	Determines the size of the bottom margin in inches. Default value is 1
	Left (in.)	Determines the size of the left margin in inches. Default value is 1
	Right (in.)	Determines the size of the right margin in inches. Default value is 1
Header and Footer	Print page header	Prints the header on each page. The header is a combination of the diagram's name and the page number. Use the adjacent field to specify the header text. The macros for this variable (%PAGE%, %PROJECT%, %ELEMENT%, %DATE%, AND %TIME%), are defined below. Default value is `On, %PAGE%, %ELEMENT%`
	Print page footer	Prints the footer on each page. The footer is a combination of the diagram's name and the page number. Use the adjacent field to specify the footer text. The macros for this variable (%PAGE%, %PROJECT%, %ELEMENT%, %DATE%, AND %TIME%), are defined below. Default value is `On, %PAGE%, %ELEMENT%`
	%PAGE%	Page number
	%PROJECT%	Project name
	%ELEMENT%	Element name
	%DATE%	Current date
	%TIME%	Current time
Diagram Print Options	Print zoom	Sets the zoom size of the document. This zoom size and its effect can be seen in the Preview dialog. This zoom option overrides any zoom size that may have been set with Zoom In or Out. The default value is 0.7.
	Fit to page	Fits the printed material on the paper size selected.
	Print border	Switches the border on or off for printing.
	Print empty pages	Determines whether empty pages are printed or ignored.
	Synchronize with Default Printer Options	Click to set Together print options to match your default printer options.

Related Concepts

Together Diagram Overview

Related Procedures

Printing Diagrams

Related Reference

Preferences

UML Profiles Preferences

Window ▸ Preferences ▸ Modeling ▸ Profiles

Use these preferences to define a default set of profiles available in your projects.

BPMN Tab

Profile	Description
BPEL Modeling	Adds BPEL Extensions for BPMN diagram.
BPMN Simulation	Adds simulation capabilities.
UML Links	Adds UML links for BPMN diagram.

ER Physical Tab

There are no ER Physical default profiles currently available.

UML14 Tab

Profile	Description	
Business Modeling	If selected, the Business Modeling profile is included into the default profile set. The UML Profile for Business Modeling is an example profile that describes how UML can be customized for business modeling.	
UML in Color	If selected, the UML in Color is included into the default profile set. This profile defines four interconnected archetypes that form a domain neutral component:	
	The moment-interval archetype	The first archetype in importance is a moment in or interval of time. It represents something that one needs to work with and track for business or legal reasons that occurs at a moment in time, or over an interval of time.
	The role archetype	The second archetype in importance is a role. A role is a way of participation by a person, place, or thing.
	The "catalog-entry-like description" archetype	The description archetype is a catalog-entry-like description. It is a collection of values that apply again and again. It also provides behavior across the collection of all things that correspond to its description.
	The "party, place or thing" archetype.	A party (meaning, a person or an organization), place or thing is someone or something who plays different roles.
Software Development Processes	If selected, the Software Development Processes profile is included into the default profile set. The UML Profile for Software Development Processes is	

	an example profile that is based on the Unified Process for software engineering.

UML20 Tab

Profile	Description
Business Modeling	If selected, the Business Modeling profile is included into the default profile set. The UML Profile for Business Modeling is an example profile that describes how UML can be customized for business modeling.
ER Logical Diagram Profile	If selected, the ER Logical Diagram Profile is included into the default profile set. This profile is intended for modeling an ER Logical Diagram.
Software Development Processes	If selected, the Software Development Processes profile is included into the default profile set.
Standard EJB Profile	If selected, the Standard EJB Profile is included into the default profile set. The EJB Profile for Standard EJB module complies with specification EJB 2.0.
Standard EJB Profile (ver. 2.1)	If selected, the Standard EJB Profile (version 2.1) is included into the default profile set. EJB profile for Standard EJB module complies with specification EJB 2.1. In order to work properly, this profile should be used along with the EJB profile for EJB 2.0 Specification.
UML in Color	If selected, the UML in Color profile is included into the default profile set.
WebLogic EJB Extension Profile	If selected, the WebLogic EJB Extension Profile is included into the default profile set. This profile should not be used together with the EJB profile for EJB 2.1 Specification.

Related Concepts

UML Profiles Basics

Related Reference

UML Profiles Preferences Constraints
UML Profiles Preferences View Management

UML Profiles Preferences Constraints

Window ▸ Preferences ▸ Modeling ▸ Profiles ▸ Constraints

Use this page to manage constraints to be run on elements if appropriate profiles are applied to the project.

Item	Description
BPMN, ER Physical, UML 1.4, UML 2.0	Each tab corresponds to one of the supported metamodels and contains the list of profiles with their constraints. If a constraint is checked, it will be applied to the model elements by the **Run Profile Constraints** command, after applying the parent profile. Checking or clearing a profile node results in checking or clearing all the nested constraints.
Select All	Checks all profile and constraint nodes in the current tab.
Deselect All	Clears all profile and constraint nodes in the current tab.

Related Concepts

UML Profiles Basics

Related Procedures

Verifying a Model Against Profile Constraints

UML Profiles Preferences View Management

Window ▸ Preferences ▸ Modeling ▸ Profiles ▸ View Management

Use this page to manage showing and hiding stereotypes after applying profiles to the projects.

Item	Description
BPMN, ER Physical, UML 1.4, UML 2.0	Each tab corresponds to one of the supported metamodels and contains the list of profiles with their stereotypes. If a stereotype is checked, it will be hidden on diagrams after applying the parent profile. If a stereotype is not checked, it will show up on diagrams after applying the parent profile. Checking or clearing a profile node results in checking or clearing all the nested stereotypes.
Select All	Checks all profile and stereotype nodes in the current tab.
Deselect All	Clears all profile and stereotype nodes in the current tab.

Tip: To manage the stereotype decoration in diagrams, refer to the Show Stereotype option (**Preferences ▸ Modeling ▸ View Management ▸ Text Decorations ▸ Show Stereotype**)

Related Concepts

UML Profiles Basics

Related Procedures

Working with Required Stereotypes

QA Model

Window ▸ Preferences ▸ Modeling ▸ QA Model

Use the **QA Model** preferences to define model audits and metrics.

Audits

Option	Description
Name	Displays names of the defined QA model audits. Select the checkbox next to the audit name to activate it.
Description	Provides an audit description.
Severity	Specifies the audit severity.
Select All	Selects all categories and all elements within categories. Selected categories will be included in the current QA set.
Clear All	Deselects all categories and all elements. Deselected categories will not be included in the current QA set.
New	Opens the **Edit Audit** dialog box, which lets you create a new audit.
Remove	Removes the selected audit from the list.
Edit	Opens the **Edit Audit** dialog box, which lets you change the selected audit.
Import...	Allows you to import a previously saved set of audits.
Export...	Allows you to export the current set of audits as a file.
Clone	Opens the **Edit Audit** dialog box, which lets you create a new audit identical to the audit that is currently selected in the list.

Metrics

Option	Description
Name	Displays names of the defined QA model metrics. Select the checkbox next to the metric to activate it.
Description	Provides a metric description.
Severity	Specifies the metric severity.
Select All	Selects all categories and all elements within categories. Selected categories will be included in the current QA set.
Clear All	Deselects all categories and all elements. Deselected categories will not be included in the current QA set.
New	Opens the **Edit Metric** dialog box, which lets you create a new metric.
Remove	Removes the selected metric from the list.
Edit	Opens the **Edit Metric** dialog box, which lets you change the selected metric.
Import...	Allows you to import a previously saved set of metrics.
Export...	Allows you to export the current set of metrics as a file.
Clone	Opens the **Edit Metric** dialog box, which lets you create a new metric identical to the metric that is currently selected in the list.

Related Concepts

Quality Assurance

Related Procedures

Running Model Audits and Metrics

QA Source

Window ▸ Preferences ▸ Modeling ▸ QA Source

Use the **QA Source** preferences to select source code audits and metrics and specify quality assurance default options.

QA Source

Item	Description
Show prompt dialog when QA preferences are changed	Specifies if you want to be notified every time when QA preferences are changed. You are also asked if you would like to refresh QA results.
Add QA Builder to new projects automatically	Specifies if you want to automatically enable the QA Builder for each new Java project that you create in the workspace.
Show QA Starter dialog	Specifies if you want to display the **Run QA** dialog box before running audits and metrics. The dialog box lets you specify the QA target and change QA preferences before running audits and metrics.
Show prompt dialog when resources are modified	Specifies if you want to display a warning dialog box every time the QA resources are changed. The dialog box also prompts if you would like to refresh QA results.

C++, C++ QA Builder, Java, and Java QA Builder Preferences

Displays the hierarchy of audit categories and the elements within categories that you can include in the current QA Builder set.

Item	Description
Configure Project Specific Settings	Opens the **Project Specific Configuration** dialog box, which lets you configure QA Builder properties for a specific project in your workspace.
Current set	Displays the name of the current QA Builder set.
Load set of options from the file	Opens the **Choose configuration file** dialog box, which lets you load a file containing a QA set (*.qa).
Save set of options to a file	Opens the **Choose configuration file** dialog box, which lets you save current QA set to a file (*.qa).
Expand all nodes	Expands all categories.
Collapse all nodes	Collapses all categories.
Select all	Selects all categories and all elements within categories. Selected categories will be included in the current QA set.
Clear all	Deselects all categories and all elements. Deselected categories will not be included in the current QA set.
Find an analyzer	Opens the **Find Analyzer** dialog box, which lets you quickly find a necessary analyzer in any category.
Property	Displays the properties of the selected element used in the calculation of the audit or metric. Note, that there is an `aggregation` property defined for each metric with the following values: `None`, `Sum`, `Average`, `Minimum`, `Maximum`, `Median`.
Value	Click a row under **Value** to edit the property's value.
Restore Defaults	Restores the selection to default settings.

Audits and Metrics tree context menu

Menu Item	Description
Group By	Provides methods for grouping audits and metrics (Category by default).
Ungroup	Removes grouping in the Audits or Metrics tree. All items are sorted alphabetically.
Description	Opens the Audit or Metric description in a default browser window.

Related Concepts

Quality Assurance

Related Procedures

Creating and Using Code QA Sets
Exporting and Importing Model Audits/Metrics

Related Reference

QA Model
Find Analyzer Dialog

Find Analyzer Dialog

Window ▸ Preferences ▸ Modeling ▸ QA Source ▸ Java ▸ QA Builder

Use this dialog box to find an analyzer by the specified string.

Item	Description
Choose an analyzer (* = any string)	Enter the search string in this text field, using wildcards if necessary.
Matching analyzers	This area displays the list of analyzers that match the specified search string. Note that the matching analyzers are selected by their full names rather than abbreviations.

Related Reference

QA Source

Requirements

Window ▸ Preferences ▸ Modeling ▸ Requirements

Use the **Requirements** preferences to define how Together processes traces between requirements and Use Case diagram elements.

Option	Description
Type of links	Specifies which type of links Together uses to link child objects to their parents. Choose one of the following options: **No Links**, **Generalization**, **Include**, or **Extend**.
Field with description	Specifies which Use Case property maps to the requirement description field (**Description** or **Explanation** (UML 1.4 only).
Create traces between Requirement and Use Case	Specifies if you want Together to create traces between requirements and Use Cases when you generate Use Case diagrams from requirements and vice versa.
Process links between Use Cases	Defines how Together treats links between Use Cases when you create requirements from Use Case elements. Choose one of the following options: **Ignore**, **As parent-child relationship**, or **As trace between requirements**.
Update requirement traces on element move	Defines how Together updates traces to requirements when a traced element is moved to another location in the model tree.

Related Concepts

Requirements Management

Related Reference

CaliberRM

CaliberRM

Window ▸ Preferences ▸ Modeling ▸ Requirements ▸ CaliberRM

Use the **CaliberRM** preferences to change how Together treats legacy traces imported from Together Control Center (TCC).

Option/Button	Description
Connection	Selects a connection to the CaliberRM server that stores legacy traces.
New...	Opens the **New CaliberRM Connection** dialog box where you can set up a new connection to a CaliberRM server.

Related Concepts

Requirements Management

Related Reference

Modifying Requirement Preferences
Requirements

Source Generation Preferences

Window ▸ Preferences ▸ Modeling ▸ Source generation

Use these preferences to define how Together generates source code from your projects.

Source generation tab

This tab contains source generation preferences.

Option	Description
Encoding	Selects character encoding used for the generated source code.
Line separator	Selects a platform-dependent separator used in the generated source code.

Name mapping tab

This tab contains model name mapping preferences.

Option	Description
Use mapping files	When this option is on, the `codegen__map.xml` file is generated in the model folder of a project being exported. This XML file contains name+alias (<design name>=<source name>) pairs for all packages, classes, members, and types of this project. This option uses the value of <source name> to replace the value of <design name> during generation. If the mapping file specified does not exist, this option creates one with a valid name. For example, the following name+alias pair: `<metaclass name="Class20"><map-entry name="My class with non-java name" alias="My class with non-java name" generated="true"/>` changes to: `<metaclass name="Class20"><map-entry name="My class with non-java name" alias="MyClassWithNonJavaName" generated="true"/>`.

Related Concepts

Roundtrip Engineering Overview

Related Reference

C++ Source Generation Preferences
Java Source Generation Preferences
Together Preferences

C++ Source Generation Preferences

Window ▸ Preferences ▸ Modeling ▸ Source generation ▸ C++

Use these preferences to change how Together generates C++ source code from your projects.

Option/Button	Description
Generate inherited abstract method skeleton	If checked, Together generates "skeletons" for inherited abstract methods.
Set `const` for query operations	If checked, Together generates `const` modifier for query operations.

Related Concepts

Roundtrip Engineering Overview

Related Reference

Java Source Generation Preferences
Together Preferences

Java Source Generation Preferences

Window ▸ Preferences ▸ Modeling ▸ Source generation ▸ Java

Use these preferences to define how Together generates Java source code from your modeling projects.

Java tab

Option	Description
Source compatibility	Selects the version of Java SDK. Together generates Java code compatible with the selected version.
Generate inherited abstract method skeleton	If checked, Together generates "skeletons" for inherited abstract methods.
Organize imports	If checked, Together generates import statements and short names.

OCL tab

Option	Description
Generate invariants	If this option is checked, the code is generated for the OCL invariants used in the source design project.
Generate pre/post conditions	If this option is checked, the code is generated for the OCL preconditions and post-conditions used in the source design project.

Related Concepts

Roundtrip Engineering Overview

Related Reference

C++ Source Generation Preferences
Together Preferences

View Management Preferences

Window ▸ Preferences ▸ Modeling ▸ View Management

Use these preferences to determine which elements are visible in the Diagram Editor .

Details Tab

Option	Description	
Detail level	Analysis	**(UML14)** Only the name of the operation assigned to a message is shown. If a return value is specified in the Message properties, the return value is displayed in the following form: `<return>:=<operation_name>` **(UML20)** Only the operation name is shown. Supplier/client end visibility is hidden while the role name remains visible.

	Design	**Design** is the default detail level.
		(UML14) The operation name and return type are shown. If a return value is specified in the Message properties, the return value is displayed in the following form:
		`<return>:=<operation_name>:<return_type>`
		(UML20) The operation name and parameter names are displayed in the following form:
		`<operation_name>([<param_name>‘,’<param_name>]*)`

	Implementation	**(UML14)** The operation name, parameters names, parameter types, and the return type are shown. if an arguments value is specified in the Message properties, the operation parameters are substituted. If a return value is specified in the Message properties, the return type is displayed in the following form:
		`<return>:=<operation_name>`
		`([params]) :<return_type>`
		(UML20) The full operation signature is shown, including the name, parameter names, parameter types, direction, multiplicities, unique, and ordered. If a return value and arguments are specified in the Message properties, the operation return type and parameters are replaced with these values

Show Icons for	This option enables you to use icons for better distinguishing metaclasses of elements with a similar look. Icons help recognize metaclasses in the following cases:	
	Element shown as a label inside another element	For example, class members; package or diagram children; internal transition and deferred event of a State element.
	Diagrams	An element that represents a diagram on diagram.
	Classifier shown as Class	Classifier element whose notation is the same as class notation (for example, interfaces and components)
Show Simple Java Interfaces	If selected, Java interfaces on class diagrams will be shown as circles. Default value: Off	

Make Classifiers look like Classes on diagram	If selected, the classifiers display on the diagram in a boxlike manner (like classes), regardless of their original notation. Default value: Off
Default Activity Partition orientation	Choose whether the activity partitions will have vertical or horizontal orientation by default.
Maximum auto-width of class element	If textual name of an element or a member exceeds the specified width in pixels, it will be truncated. This option becomes effective if the user has not resized the element manually.
Always show “Attributes” and “Operations” compartments	If selected, these compartments are always visible, even they are empty. Otherwise compartments only show when respective members exist. Default value: Off
Sort members alphabetically	Default value: Off
Mark link to member with a dot	Default value: On
Show shortcut sign	Default value: On
3D look	Default value: On

Text Decorations Tab

These options enable you to control text that is displayed on diagram elements.

Option	Description
Show stereotypes	If selected, the following options become enabled: Show members stereotypes Show <<communicate>> stereotype on Association Link on Use Case Diagram If selected, the <<communicate>> stereotype is displayed in UML 1.4 Use Case diagrams. Default value: Off
Show Diagram Types	If selected, a type of the diagram is displayed above the diagram name in the Diagram Editor. Default value: On
Show name of referenced Class in extending Class icon	If selected, the name of the base class is displayed in the icon of the extending class. Default value: Off
Use Fully Qualified Names in Shortcuts to classes from different packages	If selected, shortcut names on diagrams will contain both package name and class name. Default value: On
Show Aliases in Java Modeling Projects	If selected, the value of the alias property is displayed for the elements in Java Modeling projects. Default value: On
Show link names	If selected, the default non-empty link names are displayed on diagrams. Default value: On
Show names of Decision/Merge element	If selected, the ID names of **decision.merge** nodes are displayed on activity diagrams.

	Default value: Off
Show Message Link Signatures (Sequence Diagram 1.4)	If selected, the names of Message Link elements are shown in the Diagram Editor (available only for Sequence Diagrams 1.4). Default value: On
Show Class name below Object name (Sequence Diagram 1.4)	If selected, the names of instantiated classes are displayed under the respective object name in the Diagram Editor (available only for Sequence Diagrams 1.4). Default value: On
Show short instantiated Class names in Objects (Sequence / Collaboration diagram 1.4)	Default value: On

Show/Hide Elements Tab

All options in this dialog box are selected by default except **Elements marked as hidden**.

Option	Description	
Elements	Packages	Show all packages.
	Interfaces	Show all interfaces.
	Classes	Show all classes.
	Notes	Show note elements.
	Elements marked as hidden	Shows elements marked individually as hidden on the diagram. By default, this filter is off so that the Hide from View menu functions.
	Non-public Classes and Interfaces	Shows all package local classes and interfaces, including inner classes.
Members	Shows all attributes, operations and inner classes/interfaces. Either Interfaces or Classes must be selected for this check box to be active.	
	Attributes	Shows all attributes. Either Interfaces or Classes must be selected for this check box to be active. Members must also be selected for this check box to be active.
	Operations	Shows all operations. Either Interfaces or Classes must be selected for this check box to be active. Members must also be selected for this check box to be active.
	Non-public Members	Shows all private, local and protected members (members include attributes, operations, inner classes and inner interfaces). Either Interfaces or Classes must be selected for this check box to be active. Members must also be selected for this check box to be active.
Links	Associations	Shows all associations links.
	Generalizations	Shows all generalization links.
	Implementations	Shows all implementation links.
	Dependencies	Shows all dependency links.
Labels	Multiplicity	Shows multiplicity labels on links. In UML 2.0, multiplicity is a required property. If a multiplicity value is not specified explicitly, the default multiplicity value is 1 and the multiplicity value is displayed on diagrams. If this level of visible diagram detail is not needed, you can specify this option to disable the display of multiplicity labels. This option is currently supported only at the global level.

Related Concepts

Together Diagram Overview

Related Procedures

Diagrams

Related Reference

Preferences

Modeling Resources Team Preferences

Window ▶ Preferences ▶ Team ▶ Modeling Resources

Use these preferences to change the team sharing options for modeling resources.

Option	Description
AutoCheckout modeling resource on edit	Together checks out model files on any attempt of Together to change corresponding model entities. The actual checkout is performed when the edited resource is saved to disk (that is, potentially after you performed several modifications). This mode requires a VCS provider to mark files that are not checked out as read-only.
Checkout before model modification	Performs the checkout before any modification is performed.
Ignore default package diagrams	Does not check out or check in the default package diagrams.

Related Procedures

Diagrams
Sharing Projects

Related Reference

Preferences

XML

The **XML** preferences enable you to customize XML Editor options.

In This Section

XML Editor
Use the **XML Editor** preferences to change general XML Editor options.

Annotation
Use the **Annotation** preferences to change XML Annotation options.

Code Assist
Use the **Code Assist** preferences to change XML Code Assist options.

Folding
Use the **Folding** preferences to change XML elements folding options.

Formatter
Use the **Formatter** preferences to change XML formatting options.

Mark Occurrences
Use the **Mark Occurrences** preferences to change XML Mark Occurrences options.

References
Use the **References** preferences to change XML References options.

Relocation
Use the **Relocation** preferences to change XML Relocation options.

Syntax Coloring
Use the **Syntax Coloring** preferences to change XML Syntax Coloring options.

Templates
Use the **Templates** preferences to change XML templates options.

Typing
Use the **Typing** preferences to change the **XML Editor** typing options.

XML Editor

Window ▸ Preferences ▸ XML ▸ Editor

Use the **XML Editor** preferences to change general XML Editor options.

Item	Description
Attribute value delimiter	Lets you choose the delimiter for wrapping attribute values: **“** (quotation mark) or **'** (apostrophe).

Related Reference

Preferences

Annotation

Window ▸ Preferences ▸ XML ▸ Editor ▸ Annotation

Use the **Annotation** preferences to change XML Annotation options.

Item	Description
Enable XML well-formedness checking	Specifies if you want to enable automatic well-formedness check of your XML constructs.
Enable XML schema validation	Specifies if you want to enable automatic validation of your XML constructs against the schema.

Related Reference

Preferences

Code Assist

Window ▸ Preferences ▸ XML ▸ Editor ▸ Code Assist

Use the **Code Assist** preferences to change XML Code Assist options.

Item	Description
Insertion	Specifies how you want to insert code assist suggestions into your code. The following options are available: **Completion inserts**, **Completion overwrites**, and **Smart completion** (where code assist takes the decision depending on the current context).
Insert single proposals automatically	Specifies if you want to insert the code assist suggestion automatically when there are no other suggestions.
Insert common prefixes automatically	Specifies if you want to insert common XML prefixes (e.g., `xs`) automatically.
Add empty attribute value automatically	Specifies if you want to add an empty attribute value when you insert an attribute.
Fill required attributes on completion	Specifies if you want Together to automatically add the required attributes to a tag when you press CTRL+SPACE. If you added the tag attributes to the **Elements hints** area, these attributes are automatically added as well. This option requires the XML schema.
Present proposals in alphabetical order	Specifies if you want to sort code assist suggestions in alphabetical order.
Hide proposals not permitted in context	Specifies if you want to hide code assist suggestions that are inappropriate in the current context.
Hide templates not permitted in context	Specifies if you want to hide the suggested templates that are inappropriate in the current context.
Hide proposals already existing in context	Specifies if you want to hide the code assist suggestions that already exist in the current context.
Proposals from existing elements	Specifies if you want Together to provide the list of possible names for the tag or attribute as you type it. Suggestions are taken from the current XML document. The option does not require the XML schema.
Enable auto-activation	Specifies if you want code assist suggestions to automatically popup when you enter a trigger element.
Auto-activation delay	Specifies the popup delay in milliseconds.
Auto-activation triggers	Specifies the elements that trigger the code assist auto-activation.
Element	Specifies the element for which you define a specific code assist hint.
Tag creation	Specifies the tag creation option: **Empty tag**, **Tag pair**, or **As defined by Schema**.
Default attributes	Lists attributes the you want to insert by default when the **Fill required attributes on completion** check box is checked.
New	Opens the **New Element** dialog box, which lets you specify a new element for which you want to create a code assist hint.
Remove	Removes the selected hint from the list.

Related Reference

Preferences

Folding

Window ▸ Preferences ▸ XML ▸ Editor ▸ Folding

Use the **Folding** preferences to change XML elements folding options.

Item	Description
Enable folding	Specifies if you want to enable elements folding.
<![CDATA[]]>	Specifies if you want to enable CDATA sections folding.
<!-- Comments -->	Specifies if you want to enable comments folding.
<!DOCTYPE>	Specifies if you want to enable DOCTYPE sections folding.
<Element>	Specifies if you want to enable folding of a particular element.
Element folding	Check the check boxes next to the elements for which you want to enable folding.
New...	Opens the **New element** dialog box, which lets you specify the name of the element for which you want to enable folding.
Remove	Removes the selected element from the list.

Related Reference

Preferences

Formatter

Window ▸ Preferences ▸ XML ▸ Editor ▸ Formatter

Use the **Formatter** preferences to change XML formatting options.

Item	Description
Tab size	Specifies the number of spaces in a single tab position.
Use tab characters instead of spaces	Specifies if you want to use tab characters instead of spaces.
Maximum line width	Specifies the maximum line width.
Wrap long element tags	Specifies if you want to wrap element tags that exceed the maximum line width.
Align final '>' in multi-line element tags	Specifies if you want to align closing tags in multi-line elements.
Preview	Displays the sample XSL stylesheet with the current formatter rules applied.

Related Reference

Together Preferences

Mark Occurrences

Window ▸ Preferences ▸ XML ▸ Editor ▸ Mark Occurrences

Use the **Mark Occurrences** preferences to change XML Editor Mark Occurrences options.

Item	Description
Mark occurrences of the selected element in the current file	Specifies if you want to highlight occurrences of elements and attributes over which you move the caret.
Sticky	Specifies if you want to make the current marking persistent, even if you move the caret away from the currently marked tag, attribute name, or attribute value.
Elements	Specifies if you want to mark occurrences of XML elements.
Attribute names	Specifies if you want to mark occurrences of XML attribute names.
Same elements	Specifies if you want to mark occurrences of XML attribute names located within the same elements.
Attribute values	Specifies if you want to mark occurrences of XML attribute values.
Same elements	Specifies if you want to mark occurrences of XML attribute values located within the same elements.
References	Specifies if you want to mark element or attribute references.
Only references	Specifies if you want to mark only element or attribute references.
Inverse	Specifies if you want to use inverse highlighting.
Mark whole element	Specifies if you want to mark the whole contents of referenced element or attribute.

Related Reference

Preferences

References

Window ▸ Preferences ▸ XML ▸ Editor ▸ References

Use the **References** preferences to change XML References options.

Item	Description
Source	Displays the name of the source referencing element.
Target	Displays the name of the target referencing element.
New...	Opens the **New element** dialog box, which lets you specify the name of the referencing element.
Remove	Removes the selected referencing element from the list.

Related Reference

Preferences

Relocation

Window ▸ Preferences ▸ XML ▸ Editor ▸ Relocation

Use the **Relocation** preferences to change XML Relocation options.

Item	Description
URL	Specifies the relocation URL. Click the field to type in the URL.
Location	Specifies the relocation target. Click the field and then click the Browse button to open the **Location** dialog box, which lets you choose the target relocation file.
New	Opens the **New URL** dialog box, which lets you specify the URL that is the subject of relocation.
Remove	Removes the selected relocation from the list.

Related Reference

Preferences

Syntax Coloring

Window ▸ Preferences ▸ XML ▸ Editor ▸ Syntax Coloring

Use the **Syntax Coloring** preferences to change XML Syntax Coloring options.

Item	Description
Element	Displays current coloring for each element type.
Foreground	Opens the **Color** dialog box, which lets you change the current foreground color of the selected element. Check the associated check box to activate coloring for the selected element.
Background	Opens the **Color** dialog box, which lets you change the current background color of the selected element. Check the associated check box to activate coloring for the selected element.
Bold	Specifies if you want to display the selected element in a bold face type.
Italic	Specifies if you want to display the selected element in italics.
Namespaces (element coloring)	Displays current coloring for elements related to each of the defined namespaces. In the code, Together highlights elements within the namespace with the specified color.
New	Opens the **New namespace** dialog box, which lets you enter the URL of a new namespace.
Remove	Removes the selected namespace from the list of defined namespaces.
Preview	Displays a sample XSL stylesheet with the current coloring applied.

Related Reference

Preferences

Templates

Window ▸ Preferences ▸ XML ▸ Editor ▸ Templates

Use the **Templates** preferences to change XML templates options.

Item	Description
Name	Displays the name of the template.
Context	Displays the template context in which the template is applied (XML or XML Processing Instruction).
Description	Displays the template description.
Auto Insert	Specifies if you want the template to be applied automatically.
New	Opens the **Edit Template** dialog box, which lets you create a new custom template.
Edit	Opens the **Edit Template** dialog box, which lets you edit the selected template.
Remove	Removes the selected template.
Restore Removed	Restores predefined templates that have been removed.
Revert to Default	Restores predefined templates that have been modified.
Import	Lets you choose a file with previously saved templates.
Export	Lets you save current templates to a file.
Preview	Displays the body of the selected template.
Use code formatter	Specifies if you want to apply the Formatter rules to the template.

Related Reference

Preferences

Typing

Window ▸ Preferences ▸ XML ▸ Editor ▸ Typing

Use the **Typing** preferences to change the **XML Editor** typing options.

Item	Description
Add end </tags>	Specifies if you want the **XML Editor** to automatically add end tags when you type in your code.
Complete end </tags>	Specifies if you want Together to automatically add an appropriate closing tag name when you type "`</`".
'/' before '>' removes end tag of empty elements	Specifies if you want the **XML Editor** to automatically recognize and remove excessive end tags for empty elements.
Add attribute="values"	Specifies if you want Together to automatically add quotation marks when you type "`=`" after an attribute name.
Add required attributes	Specifies if you want the **XML Editor** to automatically add required attributes when you type in the element name.
Close <!-- Comments -->	Specifies if you want the **XML Editor** to automatically close comment sections when you type an opening character.
Close <![CDATA[]]>	Specifies if you want the **XML Editor** to automatically add end tags to <![CDATA[]]> constructs when you type an opening character.
Adjust indentation	Specifies if you want the **XML Editor** to automatically adjust indentation to the current indentation level when you paste XML code from the clipboard.
Escape text when pasting into a string literal	Specifies if you want to escape special characters in pasted strings when the strings are pasted into an existing string literal.

Related Reference

Preferences

XSL

The **XSL** preferences enable you to customize XSL Editor and Run/Debug options.

In This Section

XSL
Use the **XSL** preferences to specify the XSL version.

Annotation
Use the **Annotation** preferences to change XSL Annotation options.

OCL (Syntax Checking)
Use the **OCL** preferences to change XSL/OCL syntax checking options.

Code Assist
Use the **Code Assist** preferences to change XSL Code Assist options.

OCL (Syntax Coloring)
Use the **OCL** preferences to change OCL syntax coloring options.

Syntax Coloring
Use the **Syntax Coloring** preferences to change XSL Syntax Coloring options.

Run/Debug
Use the **Run/Debug** preferences to change XSL Run/Debug options.

XSL

Window ▸ Preferences ▸ XSL

Use the **XSL** preferences to specify the XSL version.

Item	Description
Default XSL version	Specifies which version of XSL you want to use in your stylesheets (1.0, 1.1, or 2.0).

Related Reference

Preferences

Annotation

Window ▸ Preferences ▸ XSL ▸ Editor ▸ Annotation

Use the **Annotation** preferences to change XSL Annotation options.

Item	Description
Enable XSL consistency checking	Specifies if you want to enable automatic consistency check of your XSL constructs.
Enable language checking	Specifies if you want to enable automatic language check of your XSL constructs.

Related Reference

Preferences

OCL (Syntax Checking)

Window ▸ Preferences ▸ XSL ▸ Editor ▸ Annotation ▸ OCL

Use the **OCL** preferences to change XSL/OCL syntax checking options.

Item	Description
Parsing (Syntax)	Specifies if you want to enable OCL syntax parsing. Syntax parsing detects only OCL structure violations (that is, misplaced operators, variables whose names do not comply with naming conventions, and so on) and cannot detect things like variables that are spelled correctly but do not exist in the context.
Analysis (Syntax and Semantics)	Specifies if you want to enable OCL syntax and semantics analysis. Analysis “knows” about the context of each OCL expression and can detect both the structure and the semantics violations, such as unavailable variables, operations, metaclasses, and so on.
Check boolean match	Specifies if you also want to check the `xsl:template@match` attribute values for a `Boolean` type. XSLT specifications do not require such matches to be strictly Boolean because there is a `boolean()` function that makes certain assumptions, such as an empty string is false while a non-empty string is true, an empty collection is false while a non-empty collection is true, and so on. Check this check box if you want to enforce Boolean results in matches. For example, you will be warned about match clauses such as `ecore::EClass`, which, otherwise, always return true (a non-undefined model is always true) even if the passed object is not of the used type. To eliminate the warning, you will need to use the `self.oclIsKindOf(ecore::EClass)` construct instead.

Related Reference

Preferences

Code Assist

Window ▸ Preferences ▸ XSL ▸ Editor ▸ Code Assist

Use the **Code Assist** preferences to change XSL Code Assist options.

Item	Description
Suggest suitable logic elements (XSLT)	Specifies if you want code assistant to suggest logical elements in XSLT that are appropriate in the current context.
Suggest suitable result elements (XSL-FO, XHTML, ...)	Specifies if you want code assistant to suggest result elements in XSL-FO, XHTML, and other markup languages that are appropriate in the current context.

Related Reference

Preferences

OCL (Syntax Coloring)

Window ▸ Preferences ▸ XSL ▸ Editor ▸ Syntax Coloring ▸ OCL

Use the **OCL** preferences to change OCL syntax coloring options.

Item	Description
Element	Displays current coloring for each element type.
Foreground	Opens the **Color** dialog box, which allows you to change the current foreground color of the selected element. Check the associated check box to activate coloring for the selected element.
Background	Opens the **Color** dialog box, which allows you to change the current background color of the selected element. Check the associated check box to activate coloring for the selected element.
Bold	Specifies if you want to display the selected element in a bold face type.
Italic	Specifies if you want to display the selected element in italics.
Preview	Displays a sample OCL script with the current coloring applied.

Related Reference

Preferences

Syntax Coloring

Window ▸ Preferences ▸ XSL ▸ Editor ▸ Syntax Coloring

Use the **Syntax Coloring** preferences to change how XSL code is rendered.

Note: Use **General ▸ Editors ▸ Text Editors** preferences to change general text editor settings, such as the background color. Use **General ▸ Appearance ▸ Colors and Fonts** preferences to change fonts.

Item	Description
Default language	Sets the default language for syntax coloring and code assist rules. Use the **Language** option to override these rules for individual elements and attributes.
Tag and Attribute	Lists tag/attribute pairs that you want to color. Use @ to separate tags and attributes.
Language	Lists tag/attribute pairs that override the **Default language** setting.
New	Opens the **New tag and attribute** dialog box, which lets you add a new tag/attribute pair to the list of colored elements and attributes.
Remove	Removes the selected tag/attribute pair from the list.
Preview	Displays the preview of XSL code respecting the currently applied colors and styles.

Related Reference

Preferences

Run/Debug

Window ▸ Preferences ▸ XSL ▸ Run/Debug

Use the **Run/Debug** preferences to change XSL Run/Debug options.

Item	Description
Verbousness	Specifies which messages you want to display when you debug your XSL code. The following options are available: **Quiet** (no messages), **Document** (display only document level messages), **Template** (display document and template level messages), **Instruction** (display document, template and XSLT instruction level messages).
Console output (Run/Debug, requires restart)	Specifies if you want to display debugging messages in the **Console** view.
Statistics (Run/Debug, requires restart)	Specifies if you want Together to collect statistical data on how much time the whole transformation takes and how much time it takes each call to complete an individual xsl instruction. The data displays in the **Console** view at the very end of the transformation log.
Collect messages (Run/Debug)	Specifies if you want to collect messages in the **Messages** view.
Collect traces (Debug)	Specifies if you want to collect traces in the **Traces** view.
Intercept results (Debug, requires restart)	Specifies if you want to make the result of the transformation available in the **Result** view of the **XSL Debugging** perspective. You can uncheck this check box for large style sheets.
Track contexts (Debug)	Specifies if you want to track contexts in the **Context** view.
Track template usage (Debug)	Specifies if you want to track template usage in the **Templates** view.
Debug instructions	Lists debug instructions for which you want to define an individual debugging/tracing behavior.
Stepping	Specifies the debugging/tracing behavior for the selected instruction when you use step debugger actions. **Normal** - specifies that when you press F5 (or use the Step-Into action), the debugger also steps through the child instructions of the current element. **Skip** - specifies that on any of the step actions, the selected instruction will always be stepped over, including children. **Don't go into** - is less restrictive than **Skip** as it suspends only execution of the element itself, its next sibling, or parent's sibling that are not set to **Skip**.
Tracing	Specifies whether you want the debugger to add a line about execution of the selected instruction to the **Traces** view or not.
New	Displays the **New Element** dialog box, which lets you add a new XSL element to the list of debugging instructions.
Remove	Removes the selected XSL elements from the list of debugging instructions.

Related Reference

Preferences

Profiles Reference

Contains reference information about Together profiles and profiles API.

In This Section

Profile Definition Properties

Describes Profile Definition and View properties of a profile, its nodes and links.

Profile Definition Properties of a Profile

Profile properties are accessible via the default package diagram of the Profile Definition project:

Context menu of the default diagram ▸ Properties ▸ Profile Definition

Property	Description
description	Gives a description of a profile. This text is displayed in the **Profiles** page of the **New Project** wizard and in the **Profiles** node of the **Preferences** dialog.
namespace	Provides external identification of the profile. This identification can be represented by a URI string, which will be used when exporting a model with the applied profile into an XMI file.

Profile Definition Properties of a Stereotype

Context menu of a stereotype ▸ Properties ▸ Profile Definition

The following properties define visual representation of a Stereotype extending a metaclass. They are available in the viewmap editor, which can be invoked from the viewmap field of the Properties View with the **Edit** button.

Property	Description
extended metaclass	Defines metaclasses extended by the stereotype. Metaclasses can be selected from those existing in the target metamodel. If a stereotype extends more than one metaclass or extends an abstract metaclass, it has no viewmap properties.
icon	Specifies the path to the icon file (.gif) that will represent this stereotype on the Tools Palette, in the Model Navigator, on diagrams, and on the diagram context menu. This property is available if the stereotype extends one metaclass that supports viewmapping.
viewmap	Specifies the selected Stereotype visual representation. This property extends one metaclass that supports viewmapping. You can specify color for this node or a figure from the selected .svg file. This property is available if the stereotype extends one metaclass that supports viewmapping.

Profile Definition Properties of an Attribute or Outgoing Association Link

inspector group	This property enables you to define the name of the group of properties of a stereotype. When the stereotype in question applies to an element, a tab with the specified name will add to the Properties View of this element.
icon	Specifies the path to the icon file (.gif).
viewmap	Specifies the visual representation of an attribute or association link.

Profile Definition Properties of a Palette Contribution

Property	Description
diagrams	Specifies diagrams whose Tools Palettes will contain a creation tool for the linked Stereotype.

icon	Opens the **Select Icon** dialog, where one can define an icon that will appear in the Palette Contribution node of the target project after applying the profile.

Properties of Extension link

Property	Description
isRequired	If this property is set to `true` for a stereotype, then any created instance of the base metaclass automatically gets its `stereotype` property assigned to the stereotype in question.
bindWithProfile	This property is only available if `isRequired` is set to `true`. If this property is set to `true`, then when you apply a profile to a project, all appropriate elements get the required stereotype but the diagram files are not modified. After removing the profile from the project, the stereotype is also removed from all elements to which it was assigned. If the property is set to `false`, then when you apply a profile to a project, all appropriate elements get the required stereotype and the diagram files are modified accordingly. After removing the profile, the stereotype persists in project.

Related Concepts

UML Profiles

UML Profile for Business Modeling

The UML Profile for Business Modeling is an example profile that describes how UML can be customized for business modeling. Note that UML can be used to model different kinds of systems (such as software systems, hardware systems, and real-world organizations).

The UML Profile for Business Modeling provides several new stereotype options for the following diagram elements:

Note: The profile-specific audits are available for this profile via **Model ▸ Profile ▸ Run Profile Constraints** command.

Stereotype set

The stereotypes that are defined by this profile are described in the following table.

A system modeled by the Unified Process consists of several different but related models. These models are characterized by the lifecycle stage that they represent, and each model makes use of one specific stereotype. Many of the stereotypes are used particularly to give the ability to structure and categorize models and systems during different stages of the development process. The UseCaseSystem, AnalysisSystem, UseCasePackage, AnalysisPackage and AnalysisServicePackage stereotypes can be set manually for a Package Diagram (and for uml20 projects, a Class Diagram).

Stereotype	Element
objectSystem	Subsystem
organizationUnit	Subsystem
workUnit	Subsystem
BM_worker	Class
BM_caseWorker	Class
BM_internalWorker	Class
BM_entity	Class
BM_communicate	Association
BM_subscribe	Association

Organization Stereotypes

Stereotype	Description
Object System <<objectSystem>>	An object system is the top-level subsystem in an object model, and may contain organization units, work units, classes, and relationships.
Organization Unit <<organizationUnit>>	An organization unit is a subsystem that may contain other organization units, work units, classes, and relationships.
Work Unit <<workUnit>>	A work unit is a subsystem that may contain one or more entities. It is a task-oriented set of objects that forms a recognizable whole to the end user, and may have a facade defining the view of the work unit's entities relevant to the task.

Class Stereotypes

Stereotype	Description
Worker <<BM_worker>>	A worker is a class that represents an abstraction of a human that acts within the system. A worker interacts with other workers and manipulates entities while participating in use case realizations.
Case Worker <<BM_caseWorker>>	A case worker is a special case of worker that interacts directly with actors outside the system.
Internal Worker <<BM_internalWorker>>	An internal worker is a special case of worker that interacts with other workers and entities inside the system.
Entity <<BM_entity>>	An entity is a passive class; that is, its objects do not initiate interactions on their own. An entity object may participate in many different use case realizations and usually outlives any single interaction.

Association Stereotypes

Stereotype	Description
Communicate <<BM_communicate>>	Communicate is an association used to specify that instances of the associated classifiers interact.
Subscribe <<BM_subscribe>>	A subscribe association between two classes states that objects of the source class (called the subscriber) will be notified when a particular event has occurred in objects of the target class (called the publisher). The association includes a specification of a set of events defining the event that causes the subscriber to be notified.

Constraint set

The UML Specification relies on the use of well-formedness rules to express constraints on model elements, and this profile uses the same approach. The constraints applicable to the profile are added to the ones of the stereotyped base model elements, which cannot be changed.

All the modeling elements in a generalization must be of the same stereotype; for example, a BM_worker class may only inherit from the other BM_worker classes.

Generalization

All the modeling elements in a generalization must be of the same stereotype; for example, a BM_worker class may only inherit from the other BM_worker classes.

Related Concepts

UML Profiles Basics

Related Reference

UML Profiles Preferences

Stereotype Options of UML Profile for Modeling In Color

The UML Profile for Modeling in Color provides several new stereotype options for the following diagram elements:

Diagram Element	Stereotype
Class	role
Class	moment-interval
Class	mi-detail
Class	party
Class	place
Class	thing
Class	description

When applying a stereotype to one of the diagram elements listed in the table, the view of the associated diagram element changes on the diagram. The stereotype field is displayed directly above the name field for the element, and the color of the element depends on the stereotype chosen.

This profile includes the four interconnected archetypes:

- moment-interval
- role
- catalog-entry-like description
- party, place or thing

For detailed information about modeling in color, refer to *Java Modeling in Color with UML: Enterprise Components and Process* by Peter Coad, Jeff De Luca, and Eric Lefebvre.

Related Concepts

UML Profiles

Related Procedures

Together Profiles

Related Reference

UML Profiles Preferences

UML Profile for Software Development Processes

The UML Profile for Software Development Processes is an example profile that is based on the Unified Process for software engineering. The profile is defined using the extensibility mechanisms of UML, which allow modelers to customize UML for specific domains, such as software development processes. This profile is not a complete definition of the Unified Process or how to apply it, but rather an example that shows how some of the profile terminology and notation is used.

Note: The profile-specific audits are available for this profile via **Model ▸ Profile ▸ Run Profile Constraints** command.

Stereotype set

The stereotypes that are defined by this profile are listed in the following table.

A system modeled by the Unified Process consists of several different but related models. These models are characterized by the lifecycle stage that they represent, and each model makes use of one specific stereotype. Many of the stereotypes are used particularly to give the ability to structure and categorize models and systems during different stages of the development process. The UseCaseSystem, AnalysisSystem, UseCasePackage, AnalysisPackage and AnalysisServicePackage stereotypes can be set manually for a Package Diagram (and for uml20 projects, a Class Diagram).

Stereotype	Element
designSystem	Subsystem
implementationSystem	Subsystem
designSubsystem	Subsystem
implementationSubsystem	Subsystem
designServiceSubsystem	Subsystem
SwDev_control	Class
SwDev_boundary	Class
SwDev_entity	Class
SwDev_communicate	Association
SwDev_subscribe	Association

Design Stereotypes

Stereotype	Description
Design System <<designSystem>>	A design system is a top-level subsystem that may contain design subsystems, design service subsystems, design classes, and relationships.
Design Subsystem <<designSubystem>>	A design subsystem is a subsystem that may contain other design subsystems, design classes, and relationships.
Design Service Subsystem <<designServiceSubsystem>>	A design service subsystem is a subsystem that may contain design service subsystems, components, and relationships.

Implementation Stereotypes

Stereotype	Description
Implementation System <<implementationSystem>>	An implementation model is a subsystem that may contain implementation subsystems, components, and relationships.
Implementation Subsystem <<implementationSubsystem>>	An implementation model is a subsystem that may contain other implementation systems, components, and relationships.

Class Stereotypes

Stereotype	Description
SwDev_Entity <<SwDev_Entity>>	An entity is a passive class; that is, its objects do not initiate interactions on their own. An entity object may participate in many different use case realizations and usually outlives any single interaction.
SwDev_Control <<SwDev_Control>>	A control is a class whose objects manage interactions between collections of objects. A control class usually has behavior that is specific for one use case, and a control object usually does not outlive the use case realizations in which it participates.
SwDev_Boundary <<SwDev_boundary>>	A boundary is a class that lies on the periphery of a system, but within it. It interacts with actors outside the system as well as with entity, control, and other boundary classes within the system.

Association Stereotypes

Stereotype	Description
SwDev_Communicate <<SwDev_Communicate>>	Communicate is an association between actors and use cases that is used to denote messages that may be sent between them. It may also be used between boundary, control, and entity, and between actor and boundary.
SwDev_Subscribe <<SwDev_subscribe>>	A subscribe association between two classes states that objects of the source class (called the subscriber) will be notified when a particular event has occurred in objects of the target class (called the publisher). The association includes a specification of a set of events defining the events that causes the subscriber to be notified.

Constraint set

The UML Specification relies on the use of wellformedness rules to express constraints on model elements, and this profile uses the same approach. The constraints applicable to the profile are added to the ones of the stereotyped base model elements, which cannot be changed.

All the modeling elements in a generalization must be of the same stereotype; for example, a `BM_worker` class may only inherit from the other `BM_worker` classes.

Generalization

All the modeling elements in a generalization must be of the same stereotype; for example, a `BM_worker` class may only inherit from the other `BM_worker` classes.

Related Concepts

UML Profiles Basics

Related Reference

UML Profiles Preferences

EMF API for Together Profiles

Provides a description of the EMF API, which you can use to programmatically access the Together profile data.

Introduction

Together provides a reflective EMF-compatible profile API supporting read/write access to Together profile data (stereotype instances). Profile API can be used programmatically from Java code, OCL, and QVT. The Model Compare/Merge utility displays stereotype instances under special `stereotypeInstances` containment reference, which also allows read/write access.

Profile metamodels

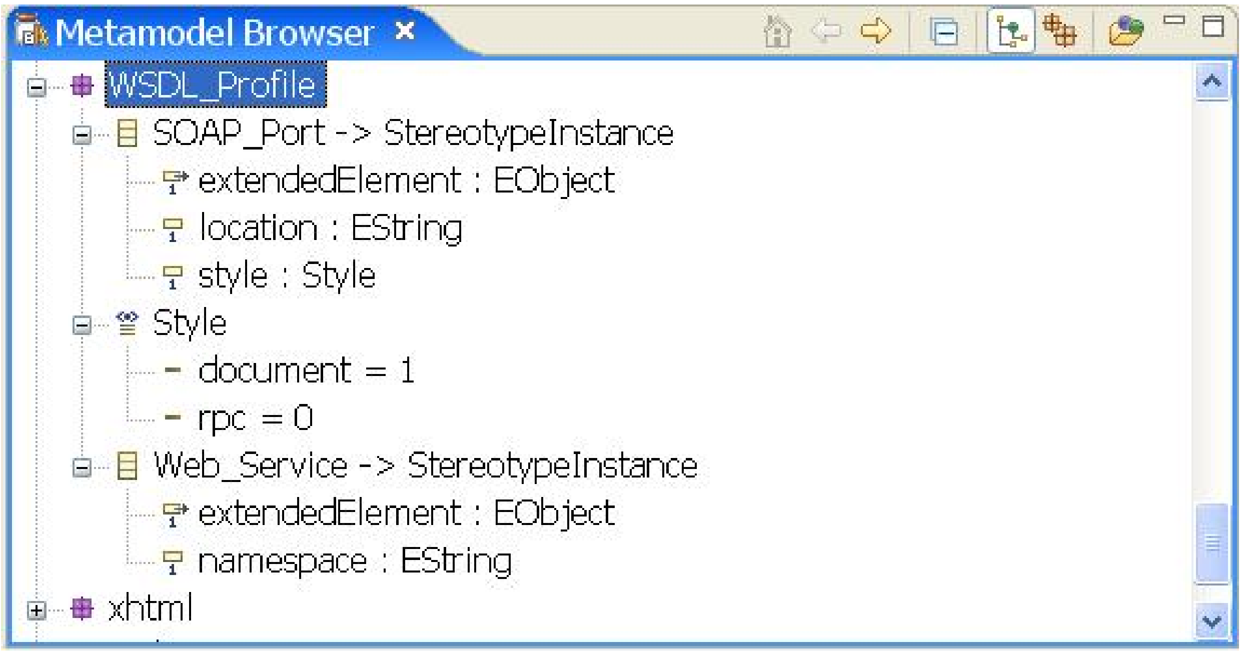
Profile metamodels are EMF-compatible metamodels (packages) that are built at runtime for all available (deployed) profile definitions. Each stereotype class has an `extendedElement` non-containment reference pointing to a model element, and typed features corresponding to properties and associations specified in the profile definition. Stereotype properties are represented as EMF attributes of the corresponding type and multiplicity. Stereotype associations are represented as non-containment references. Enumerations defined in the profile definition project become standard EMF enumerations.

Profile API does not support EMF notifications; that is, no notifications are sent after changes have been made to stereotype properties.

Each profile metamodel is uniquely identified by the profile nature identifier assigned to the profile definition project.

The screen shot below shows the **Metamodel Browser** view with the EMF metamodel generated for a WSDL profile taken from the "Define UML Profile" cheat sheet.

Note: When creating a metamodel, Together automatically replaces special characters found in profile names (spaces in our example) with underscores (`_`).



```
WSDL_Profile
  SOAP_Port -> StereotypeInstance
    extendedElement : EObject
    location : EString
    style : Style
  Style
    document = 1
    rpc = 0
  Web_Service -> StereotypeInstance
    extendedElement : EObject
    namespace : EString
xhtml
```

All Together stereotypes extend `StereotypeInstance`, which declares the `extendedElement` reference.

The profile metamodel is available during profile development, and you can use it in OCL constraints associated with extended metamodel elements. For example, the following OCL constraint can be written in the WSDL profile definition project.

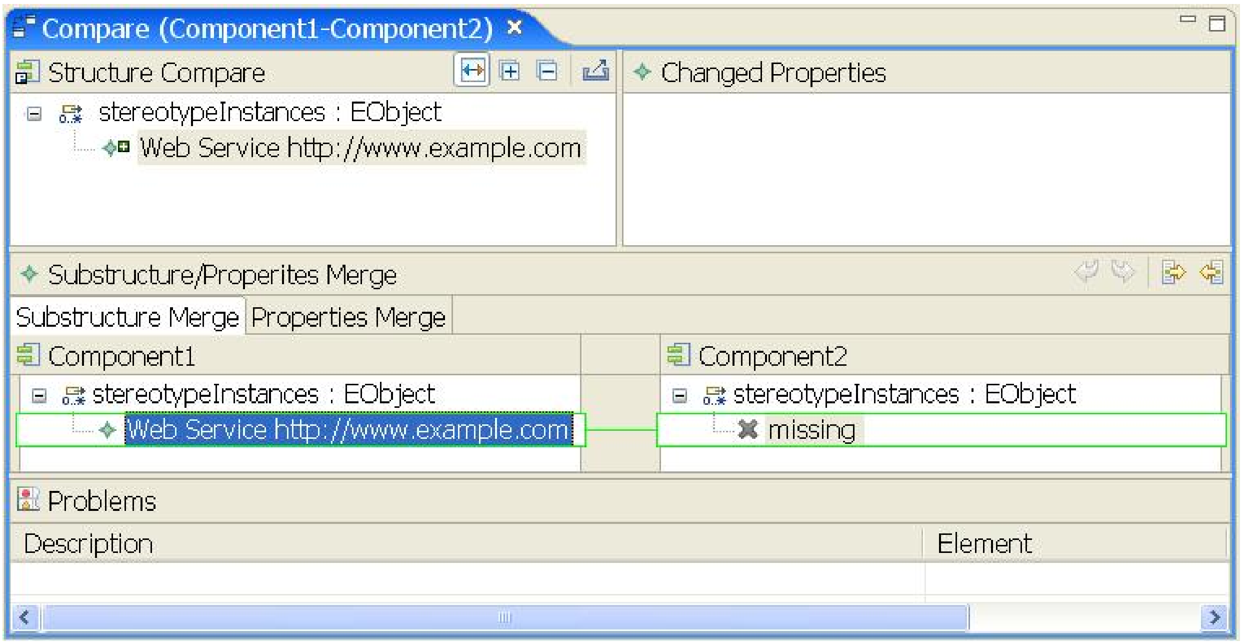
```
context uml20::components::Component
inv: let ns = self.getStereotypeInstances().oclAsType(
    WSDL_Profile::Web_Service)->any(true).namespace
in
    not ns.oclIsUndefined()
```

This invariant uses the OCL library function `getStereotypeInstances()`, which is shortly described further in the document.

Profiles in Model Compare/Merge

The **Model Compare/Merge** view displays the applied stereotypes under special `stereotypeInstances` containment reference. Note that this is a "virtual" reference, as it is not a part of the Together metamodel.

The screen shot below shows the **Compare** editor with two components, one of which has `Web Service` stereotype applied, and its namespace property is set to `http://www.example.com`.


Compare (Component1-Component2)
Structure Compare
stereotypeInstances : EObject
Web Service http://www.example.com
Changed Properties
Substructure/Properites Merge
Substructure Merge
Properties Merge
Component1
stereotypeInstances : EObject
Web Service http://www.example.com
Component2
stereotypeInstances : EObject
missing
Problems
Description
Element

When working with stereotype instances, you have full access to model compare/merge functionality. You can create and delete stereotype instances, and change their attributes and references.

Note that the comparing of arbitrary custom properties is not directly supported by Model Compare/Merge. To work with these properties, you need to define profile descriptions for them.

Profile API in OCL

You can use the Profile API as well as the metamodels in any OCL code written at the Together metamodel level: model audits/metrics, XSL/OCL transformations, documentation generation, model search, **OCL Expressions** view, and OCL constraints linked to metamodel elements. The OCL constraints are only supported inside a Profile Definition project, and the only profile metamodel that is available is the one being defined by the project profile.

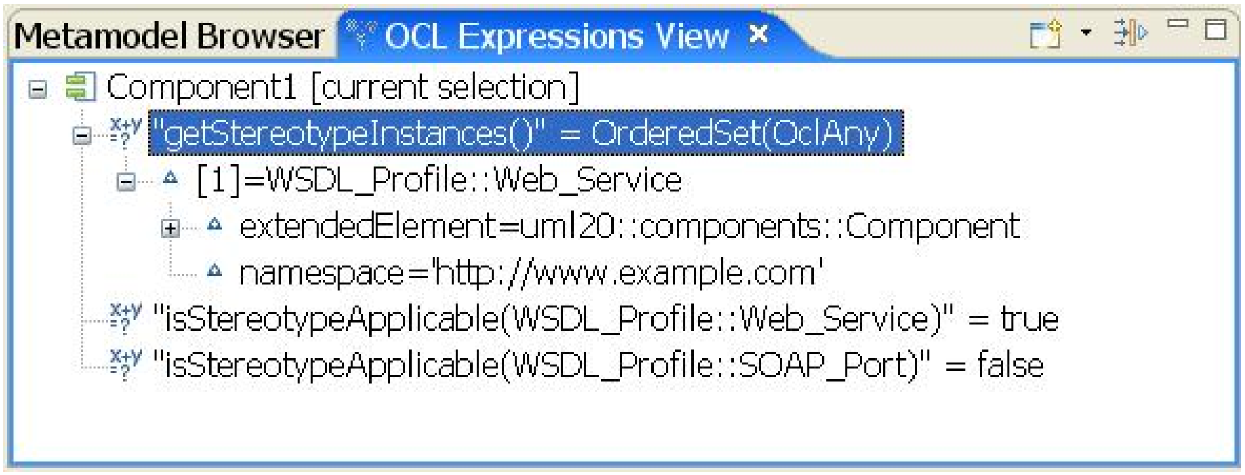
Use the **Metamodels** tab of the **Preferences** dialog box (**Window** ▶ **Preferences...** ▶ **Modeling** ▶ **OCL** ▶ **Metamodels**) to specify which metamodels you want to be visible to the OCL engine. The tab contains entries for all available metamodels, including profile metamodels. You need to check the check boxes next to each profile metamodel that you want to use in your OCL code.

The following profile library functions are available. To provide a more general example, the functions use `OclAny` as a type of stereotype instances.

Library Function	Description
`OclAny::getStereotypeInstances(): OrderedSet(OclAny)`	Returns all stereotype instances for the context model element. The set returned is not live and is not automatically updated when some other clients perform changes affecting its contents or when a profile is switched on/off for the model.
`OclAny::isStereotypeApplicable(stereotype: OclType): Boolean`	Checks whether a given stereotype can be applied to the model element.
`OclAny::addStereotypeInstance(instance: OclAny): OclVoid`	Adds a stereotype instance to the model element.
`OclAny::removeStereotypeInstances(instances: OrderedSet (OclAny)): OclVoid`	Removes stereotypes instances from the model element.

The `addStereotypeInstance` and `removeStereotypeInstances` functions modify the state of the model element and are therefore not used in OCL queries.

The screen shot below shows the **OCL Expressions** view with OCL expressions, which illustrate how to use profile library functions.



```
Component1 [current selection]
  "getStereotypeInstances()" = OrderedSet(OclAny)
    [1]=WSDL_Profile::Web_Service
      extendedElement=uml20::components::Component
      namespace='http://www.example.com'
  "isStereotypeApplicable(WSDL_Profile::Web_Service)" = true
  "isStereotypeApplicable(WSDL_Profile::SOAP_Port)" = false
```

Profile API in QVT

QVT uses the same profile functions, packaged in the `together.profiles` library. Profile metamodels are referenced using the standard `metamodel` statement.

Profile metamodels are created at runtime and do not have underlying Java implementation. QVT transformations that work with the profile data can only be run in interpreted mode when no Java code is generated from QVT sources.

The sample QVT transformation below uses profile API to modify the Web Service stereotype data for some input component.

```
transformation WebService;
```

```
import library together.profiles;

metamodel 'http://www.borland.com/together/uml';
metamodel 'http://www.borland.com/together/uml20';
metamodel 'WSDL_Profile.wsdl_profilenature';

mapping main(inout model: uml20::components::Component): uml20::components::Component {
    init {
        var webService := model.getStereotypeInstances().
            oclAsType(WSDL_Profile::Web_Service)->any(true);

        model.removeStereotypeInstances(OrderedSet{webService});

        var newNamespace := if webService.namespace.oclIsUndefined() then
                                'namespace' else webService.namespace + '*' endif;
        var newService := model.makeService(newNamespace);
        model.addStereotypeInstance(newService);

        result := model;
    }
}

mapping uml20::components::Component::makeService(ns: String): WSDL_Profile::Web_Service {
    namespace := ns;
}
```

First, it uses `getStereotypeInstances()` to read the web service stereotype instance. Then it removes the instance and creates a new one using the standard `object` expression. Finally, transformation calls `addStereotypeInstance()` to add a newly created stereotype to the component.

Traces to/from stereotype instances are stored in the transformation trace file and can be accessed programmatically (for example, using another QVT transformation). Currently, navigation from **Trace** view (**Show source/target**) is not supported for stereotype instances.

Profile API in Java

Java clients have to use reflective EMF APIs for working with stereotype instances, as profile metamodels have no Java implementations. Java Profile API methods are declared in `ProfileApi`, located in the `com.borland.tg.emfapi.profile` plugin.

The following sample Java code performs exactly the same transformation as the QVT code above.

```
package profileaccess;

import java.util.Collections;
import java.util.List;

import org.eclipse.emf.ecore.EAttribute;
import org.eclipse.emf.ecore.EClass;
import org.eclipse.emf.ecore.EObject;
import org.eclipse.emf.ecore.EPackage;

import com.borland.tg.emfapi.profile.ProfileApi;
import com.borland.tg.emfapi.profile.ProfilePackageRegistry;
import com.borland.tg.emfapi.uml20.components.Component;

public class WebService {
    public void generate(Component model, String outputPath) {
        main(model);
    }

    private void main(Component model) {
        List instances = ProfileApi.INSTANCE.getStereotypeInstances(model);
        String newNamespace;
        if(!instances.isEmpty()) {
            EObject webService = instances.isEmpty() ? null : (EObject) instances.get(0);
            ProfileApi.INSTANCE.removeStereotypeInstances(model, Collections.singletonList
(webService));

            newNamespace = (String)webService.eGet(WSDL_PROFILE.WEB_SERVICE_NAMESPACE) +
"*";
        }
        else {
            newNamespace =  "namespace";

        }

        EObject newService = makeService(newNamespace);
        ProfileApi.INSTANCE.addStereotypeInstance(model, newService);
    }

    private EObject makeService(String ns) {
        EObject service = WSDL_PROFILE.METAMODEL.getEFactoryInstance().create
(WSDL_PROFILE.WEB_SERVICE);
        service.eSet(WSDL_PROFILE.WEB_SERVICE_NAMESPACE, ns);
        return service;
    }

    static class WSDL_Profile {
        public final EPackage METAMODEL;
        public final EClass WEB_SERVICE;
        public final EAttribute WEB_SERVICE_NAMESPACE;

        public WSDL_Profile() {
            METAMODEL = ProfilePackageRegistry.INSTANCE.getProfilePackage(NS_URI);
            WEB_SERVICE = (EClass) METAMODEL.getEClassifier("Web_Service");
            WEB_SERVICE_NAMESPACE = (EAttribute)WEB_SERVICE.getEStructuralFeature
("namespace");
        }

        public static final String NS_URI = "WSDL_Profile.wsdl_profilenature";
    }

    private static final WSDL_Profile WSDL_PROFILE = new WSDL_Profile();
}
```

The code declares a helper `WSDL_Profile` class, which contains "Java-friendly" representation of the profile. This is a special metamodel, so its EPackage is obtained via a call to the `ProfilePackageRegistry.INSTANCE.getProfilePackage()` method.

Then, the code uses the `removeStereotypeInstances()` and `addStereotypeInstance` methods from `ProfileApi` to replace the stereotype instance of the component.

Related Concepts

Model Transformation Support
OCL Support

Related Procedures

Merging Models

Related Reference

QVT Language
EMF API for Together Models

Business Process Diagram

This section provides Business Process Modeling reference information.

In This Section

Mapping Elements

Business Process Mappings

Business Process Mappings are performed according to the OMG Adopted Business Process Modeling Notation Specification (specification). BPEL is generated for private processes. No generation is performed for the abstract processes.

Note: The final version of the specification can be found at the bpmn.org web site.

Events

Start and End events are mapped according to the specification.

Intermediate events are mapped according to the specification with the following exception: multiple intermediate events are mapped to pick, each child detail corresponds to onMessage activity.

Activities

Activities are mapped according to the specification with the following exceptions:

- Independent subprocesses are not supported. Service tasks should be used instead.
- Manual tasks and script tasks are not supported. Their implementation is server-dependent.

Gateways

Exclusive and parallel gateways are mapped according to the specification.

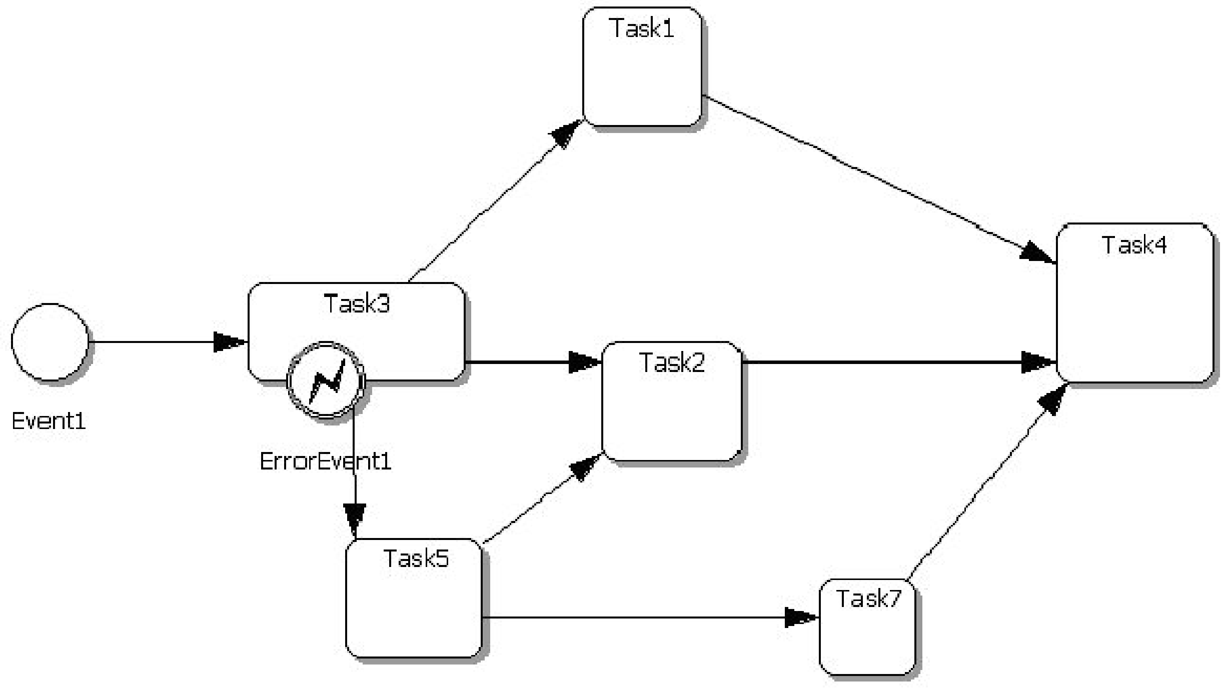
Mapping of the Inclusive and Complex gateways is not supported.

WebService

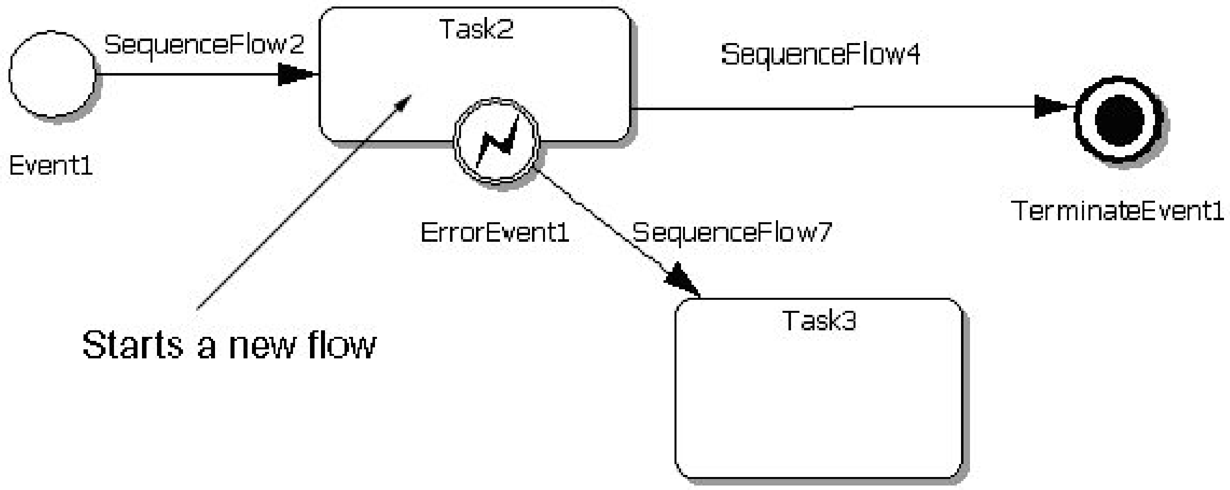
In the Together BPMN diagram, the WebService element is created inside the Participant element. The WebService element is used for the Interface and Operation information.

Mapping Exception Flow

In Together, mapping of an exception flow is completely different from the mapping described in the specification. The specification suggests mapping to switch, but it does not explain how to perform mapping if the exception flow contains tasks that must be executed in parallel. In this case, an exception flow can join normal flow at different points (see the figure below). Mapping that uses switch in such situations can hardly be suggested.



Any task or subprocess with boundary events starts a flow. All the elements from the exception flow or normal flow belong to that flow element, until all the paths from them merge (as in the figure below). If the paths do not merge explicitly, the program assumes that they merge in the end element.



For each boundary event, a variable with the name `event_name + "_completion"` is created. For the task that starts the flow, the variable with the name `task_name + "_completion"` is created (in the first example above, `Task2_completion` and `ErrorEvent1_completion`). A mapping of the task is enclosed into the sequence, and all these variables are initialized before executing the task. At initialization, `Task_name` completion is set to true and `EventName_completion` to false.

The sequence is put inside the scope and all the handlers of the events are defined inside this scope according to the specification of the mapping of the boundary events. If a boundary event occurs, inside handler `EventName_completion` is set to true and `TaskName_completion` to false.

All the links outgoing from boundary events have condition `"bpws:getVariableProperty (EventName_completion,completion)=true"` and from the original task have condition `"bpws:getVariableProperty(TaskName_completion,completion)=true"`. If exception flows are nested, the separate flow is not created for the inner elements, but they are considered as parts of the outer flow.

Mapping Pools and Message Flows

General Information

Web Services Description Language (WSDL) files are generated for all pools, including the default pool, that represent the abstract or private processes. The WSDL file is not generated in two cases:

- If the participant has attached the WSDL.
- If the pool represents the abstract process, which does not contain any web services (consumer processes). In other words, the pool does not contain any elements (except lanes) and does not have incoming message flows other than a reply for outgoing message flows.

All partner links are stored in WSDL files for the default process. For the name of the WSDL file, the process name is used. f there is no default process, any pool that the participant has no attached an WSDL file to will be used.

Attached WSDL

Participants can attach WSDL files. That WSDL file defines all possible interfaces and operations for the participant. It is impossible to define a new interface/operation for the participant with an attached WSDL file. Such a feature can be used to add side Web Services to a diagram and use them in the modeled Business Process.

Default process WSDL

The default pool is equal to other pools. If a default pool exists, the WSDL file is always generated for it. If a default pool does not exist, any other pool without an attached WSDL is used for this purpose.

Abstract processes with Web Services

For pools that represent abstract processes with Web Services (those that have incoming calls or asynchronous messages) and do not have associated WSDL files, Together generates an extended WSDL file that, along with necessary interface definitions, contains `<binding/>` and `<service/>` tags. These tags help to generate Java stubs for the defined web services.

Mapping

The generator creates the `<portType/>`, `<operation/>` and `<message/>` tags according to the Business Process Modeling Notation (BPMN) specification.

The only difference is in the mapping of property types. The specification assumes "xsd" prefix by default. In Together, the generator uses the prefix of the generated WSDL file by default. To use the XML Schema types, the "xsd" prefix must be specified directly (for example, `xsd:String`).

Element	Description
Activity Mapping	For each Task that has properties, the `<message/>` tag with the name of the activity and a standard suffix "_ActivityDataMessage" is generated. Each property maps to the `<part/>` tag.
Loop Activity Mapping	If the process contains an activity with a loop, the generator creates the `<message/>` tag with a "loopCounterMessage" name for the "simple" loop and with a "forEachCounterMessage" name for the "multiple instance" loop.

Boundary Event Mapping	If the process contains at least one boundary event, the `<message/>` tag with "completionMessage" name is created.
Link Event Mapping	For each intermediate Link Event that participates in the event-based decision (goes just after Event-Based Exclusive Gateway), the generator creates an `<operation/>` tag inside the "LinkPortType" port type. The fully qualified name of the Link Event will be used as the name of the operation.
Message Flow Mapping	For each incoming message flow, except the reply message flow, the generator creates the `<portType/>` and `<operation/>` tags as defined in the message flow. The input message is also generated from this one message flow. If the diagram contains a reply message flow, this one is used for the generation of the output message. The "InMessage" on the following figure is mapped to Web Service (`<portType/>` and `<operation/>` pair). The "InMessage" will be used as an outgoing message and the "OutMessage" as an incoming message of a created service.
Process Mapping	For each Process that has properties, the `<message/>` tag with the name of the process and standard suffix "_ProcessDataMessage" is generated. Each property maps to the `<part/>` tag inside the message.
Rule Event mapping	For each start Rule event, the generator creates an `<operation/>` tag in the "RulePortType" port type. The fully qualified name of the event will be used as the name of the operation.
Timer Event mapping	For each start Rule event, the generator creates an `<operation/>` tag in the "TimerPortType" port type. The fully qualified name of the event will be used as the name of the operation.

Implementations details

If the type property of the property contains prefix, the generator tries to find appropriated namespace URI from the existing (predefined and collected form associated WSDL files). If the URI cannot be found, the generator uses the default URI (see. The namespace URI above).

The namespace prefixes

In generated WSDL files generator tries to use original prefixes from the predefined WSDL file or from the associated WSDL file. If it is impossible (for example, some WSDL files use the same prefixes) the generator creates unique prefix in format "prcXX", where XX is a unique number (in scope of the diagram).

The namespace URI

The generator uses the following predefined namespace prefixes and URIs:

- prefix "xsd", URI " http://www.w3.org/2001/XMLSchema"
- prefix "plink", URI " http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
- prefix "bpws", URI " http://schemas.xmlsoap.org/ws/2003/03/business-process/"
- prefix "soap", URI " http://schemas.xmlsoap.org/wsdl/soap/"
- prefix "wsdl", URI " http://schemas.xmlsoap.org/wsdl/"

Such URIs cannot be used as namespace URIs for BPMN processes.

The property types

If the type property of the property contains prefix, the generator tries to find appropriated namespace URI from the existing (predefined and collected form associated WSDL files). If the URI cannot be found, the generator uses the default URI (see. The namespace URI above).

Mapping Process Structure: Flows and Sequences

Starting and Ending Business Process

In Together, each non-abstract business process must have only one event with the type start. If the start event is missing or there are more than one start events, the diagram is considered invalid and a validation error occurs.

Each event with the type end is considered as an end of a business process. There might be several events with type end and each of them will be considered as the end of the current BPEL container element (for example, sequence or flow). There may be no end element at all; each element without an outgoing sequence flow is considered as the end element.

Executing Tasks Sequentially

All the elements of the process are enclosed into one top-level sequence with the name main. If each of the elements has one incoming and one outgoing sequence flow, they form a sequence and sequence flow links are not mapped. Sequences can be nested into each other.

Executing Tasks in Parallel

If some element has more than one outgoing sequence flow, there must be a BPEL flow. There are three elements that can start the flow:

- Tasks
- Start Events
- Parallel gateway

Each flow starts with empty activity that has sources of the links outgoing from the first element. A name of the sequence flow becomes the name of the link. The condition type must be expression. The value of the expression property is mapped to the value of the link condition without any processing, as it is written. Therefore, the condition should be written with a prefix, for example:

```
bpws:getVariableData('request', 'amount')>=10000.
```

The flow starts and ends with an empty activity. The elements with several outgoing sequence flows (fork) and several incoming sequence flows (merge) are outside the BPEL flow.

Flow Inside Flow

Sometimes an element with several outgoing sequence flows is included inside another flow. In this case, Task1 does not start another inner flow but is considered as an activity in the outer flow. It is mapped as follows:

```
<flow>
 …
   <invoke name="Task1">
      <source linkName="SequenceFlow2"/>
      <source linkName="SequenceFlow5"/>
      <target linkName="SequenceFlow1">
   </invoke>
```

```
  …
</flow>
```

Exclusive gateways inside a flow are mapped to switch in the ordinary way.

Mapping of Exclusive Gateways

An exclusive gateway inside a flow is mapped into switch. Each sequence flow outgoing from the exclusive gateway starts a case with a sequence. Sometimes several paths merge before all the paths started from the gateway are merged. In this case, elements that are common to that case are duplicated. The same happens if several paths in nested switches are merged before the merge of all paths.

Unmapped Elements

To be mapped into BPEL properly, the process must be represented by a connected graph. Graph traversal starts from the start event and follows the sequence flow. If some elements cannot be reached in that way, they are not mapped to BPEL and a validation warning is provided.

Cycles Support

Currently Together does not support cycles that are formed by the sequence flow. Any cycle formed by the sequence flow leads to a validation error. However, each task or subprocess can be mapped to the cycle using the loop properties.

Elements That Are Not Transformed to BPEL

The following elements are not transformed to BPEL:

1. Artifacts. The mapping is not provided by specification.
2. Complex gateways. The mapping is not provided by specification.
3. Lane. The mapping is not required by specification.
4. Cancel events. The mapping is not provided by specification.
5. Inclusive gateway. Not supported in this version.
6. Manual tasks and script tasks are not supported. Their implementation is server-dependent.
7. Independent subprocesses are not supported. Service tasks should be used instead.

If an element that is not supported is present on the diagram, there can be two outcomes:

- If an element that is not supported is connected to other elements with sequence flow, a validation error occurs and no output is provided. Ignoring such an element can influence the behavior and lead to an incomplete process that does not correspond to the diagram. To get the BPEL process file, the user must remove all unsupported elements.
- An element is connected to other elements with an association or not connected at all. In this case, the element is simply ignored.

BPMN Validation View

Use this view to review the errors that occurred during validation. Validation view lists all BPMN diagram-related errors, including diagram errors, export, and simulation errors. An error in the list contains the name of the entity and a short description that helps you understand the cause of the error and how to fix it. The number of validation commands in the context menu depends on the selected profiles for the current project. For example, if the BPMN Simulation profile is on, you will see a **Validate for Simulation** command on the diagram context menu.

The following is a list of context menu commands.

Item	Description
Show in Model Navigator	Selects an entity in the Model Navigator view.
Select All	Selects all items listed in the BPMN Validation view.
Clean	Removes the selected items from the list.
Select on Diagram	Selects an entity on the BPMN diagram. If the diagram is not visible, this command will open it in the Diagram editor. Alternatively, you can double-click an item in the list to select the element on a diagram.

BPMN Simulation View

This view provides simulation run progress information and tools to control the simulation process.

Item	Description
Diagram	Displays the simulated diagram name.
Progress	Displays simulation run progress.
Execution	Provides simulation process statistics.

BPMN Simulation view buttons

Resume simulation	Resumes previously paused simulation.
Pause simulation	Pauses running simulation.
Next simulation step	Proceeds to the next simulation steps when a step-by-step simulation is running.
Stop simulation	Stops current simulation.

BPMN Diagram Context Commands

BPMN diagrams share common context menu commands with UML diagrams. To use the context menu for a diagram, right-click in the **Diagram Editor**.

To view the common context menu commands, see “Common Diagram Context Commands” topic.

Item	Description
Select Default Pool In Model Tree	Selects the default pool in **Model Navigator**.
Fix Element Type	Changes the type of the selected element if it is detected as incorrect.
Simulate	Starts simulation for the selected diagram with the default settings (Start time = 0, End time = 1000). The report is generated after choosing this command (report generation is off by default).
New	Offers a submenu with BPMN diagram elements available for the current context.
Validate BPMN Diagram	Validates the entire diagram.
Validate for BPEL4WS export	Validates the diagram to meet the BPEL4WS export requirements.
Validate for simulation	Validates the entire diagram to meet the simulation requirements.

Related Procedures

Together Business Process Modeling

Related Reference

Common Diagram Context Commands
Business Process Modeling

BPMN Simulation-specific Properties

This topic describes the BPMN diagram-specific properties. The composition of the Properties view changes depending on the element selected in the **Diagram** or **Model Navigator** view. The Properties lets you view and modify property values.

Use the BPMN simulation profile to add properties required to run a business process simulation.

Participant

Lifecycle Property	Description
None	Simulation is performed as is from Start to End.
Arriving	Instances of the process, for which this parameter is specified, periodically enter the system. An instantiation interval is defined by the arriving distribution property.

Additional property	Description
mean arriving interval	Mean time between arriving of the participants (tokens) that start the process.
arriving distribution	Different functions used to change the arriving intervals.
order	The parameter in an Erlang distribution.
standard deviation	The parameter in a Normal distribution.
maximum deviation	Deviation of the arriving interval in a Uniform distribution. The resulting range for a Uniform distribution is (mean-deviation to mean +deviation).

Lifecycle Property	Description
Cycle	Instances of the process for which this parameter is specified return to the beginning of the process after completion. The maximum number property specifies the number of instances executing in cycling order.

Additional Property	Description
maximum number	Specifies the maximum number of participants (entering tokens) performing the sequence.

Task

Properties	Description
mean duration	Mean duration of the task.
duration distribution	Functions that change the duration of the task. Can be Constant, Erlang, Exponential, Normal, and Uniform. The following additional values can be displayed when you change the duration distribution.

Additional Properties	Description
order	The parameter in an Erlang distribution
standard deviation	The parameter in a Normal distribution
maximum deviation	Deviation of the mean duration in a Uniform distribution

Properties	Description
cost per time unit	Cost of each time unit when the task is executed.

Sequence flow

Properties	Description
probability to take (in percent)	Specifies the probability of following the selected sequence flow when you specify two or more sequence flows. This property is taken into account only when the Sequence Flow property is set to `"condition type"="Expression"`.

Related Concepts

Business Process Modeling

BPMN Diagram Toolbar

The table below lists diagram elements available for a BPMN diagram in Together.

Element	Description
Process Pool	Represents a Participant in a Process. A Participant can be a specific business entity (for example, a company) or can be a more general business role (a buyer, seller, or manufacturer). There are three basic types of submodels within an end-to-end BPMN model: None (or undefined), the default business process Private (internal) business processes Abstract (public) processes Collaboration (global) processes All business process diagrams contain at least one Pool. In most cases, a business process diagram that consists of a single Pool will only display the activities of the Process and not display the boundaries of the Pool. Any Pool can have an invisible border, but only after you change border visibility of the Default Pool from `true` to `false`.
Lane	A Lane is a subpartition within a Pool that extends the entire length of the Pool. Lanes are used to organize and categorize activities within a Pool.
Task	A Task is an atomic activity that is included within a Process. A Task is used when the work in the Process is not broken down to a finer level of Process Model detail.
SubProcess	A compound task within a process that has a flow of other activities. A subprocess can be embedded, independent, or referenced. The subprocess can be in a collapsed view that hides its details or a subprocess can be in an expanded view that shows its details within the view of the Process in which it is contained. In the collapsed form, the subprocess object uses a marker to distinguish it as a subprocess rather than a Task.
None Event	An Event is something that "happens" during the course of a business process. These Events affect the flow of the Process and usually have a cause or an impact. BPMN has restricted the use of events to include only those types of events that will affect the sequence or timing of activities of a process. BPMN further categorizes Events into three main types: Start, Intermediate, and End. The Start Event indicates where a particular process will start. Intermediate Events occur between a Start Event and an End Event. It will affect the flow of the process, but will not start or (directly) terminate the process. The End Event indicates where a process will end.
Message Event	A Message Event occurs when a message with the exact identity is received by the process.
Timer Event	Sets a specific time-date or a specific cycle that will affect the process and trigger other events.
Error Event	An Error Event occurs when the process detects an error.
Cancel Event	Indicates that the process should be cancelled and triggers a Cancel Intermediate Event attached to the Subprocess boundary. In addition, it indicates that a Cancel message should be sent to any entities involved in the Process.
Compensation Event	Used for compensation handling—both setting and performing compensation. It calls for compensation if the Event is part of a Normal Flow. It reacts to a named compensation call when attached to the boundary of an activity.
Rule Event	Used only for exception handling. This type of event is triggered when a Rule becomes true. A Rule is an expression that evaluates some process data.
Link Event	A Link connects an End Event (Result) of one process to a Start Event (Start) or Intermediate Event (Trigger) in another process. Paired Intermediate Events can also be used as "Go To" objects within a process.

Multiple Event	Means that there are multiple ways of triggering the Event. Only one of them will be required. The attributes of the Intermediate Event will define which of the other types of Triggers apply.
Terminate Event	This type of End event indicates that all activities in the Process should be immediately ended. This includes all instances of Multi-Instances. The Process is ended without compensation or event handling.
Exclusive (XOR) Gateway	Exclusive Gateways (Decisions) are places within a business process where the Sequence Flow can take two or more alternative paths. For a given performance (or instance) of the process, only one of the paths can be taken.
Inclusive (OR) Gateway	This Decision represents a branching point where Alternatives are based on conditional expressions contained within an outgoing Sequence Flow. However, in this case, the True evaluation of one condition expression does not exclude the evaluation of other condition expressions. Because each path is independent, all combinations of the paths may be taken, from zero to all. However, it should be designed so that at least one path is taken.
Complex Gateway	Used to handle situations that are not easily handled through the other types of Gateways. Complex Gateways can also be used to combine a set of linked simple Gateways into a single, more compact situation. You can provide complex expressions that determine the merging and/or splitting behavior of the Gateway.
Parallel (AND) Gateway	Provides a mechanism to synchronize parallel flow and to create parallel flow.
Text Annotation	Provides additional information for the reader of a BPMN Diagram.
Data Object	Provides information about what the Process does; that is, how documents, data, and other objects are used and updated during the Process.
Group	Eases the division of a BPMN diagram into logical parts. A Group permanently keeps track of content, resizes on element move, colors elements with selected color, and so on.
Sequence Flow	Shows the order that activities will be performed in a Process.
Message Flow	Shows the flow of messages between two entities that are prepared to send and receive them. **Note**: Message Flow cannot connect to objects that are located within the same Process (Pool).
Association	Associates Artifacts with Objects within the process flow and with the flow itself. An Association is also used to show the activities used to compensate for an activity.

Related Concepts

Business Process Modeling

Related Procedures

Together Business Process Modeling

BPMN Simulation Report

Together generates a report on a BPMN diagram simulation run.

Use the **Simulation run** dialog box to specify the output folder for report file, start/end time, and time units. Start/End time and time units are shown in the report after its generation. For the entire BPMN process and all the report calculation decisions, a process is active if it contains at least one active task, and the process is waiting if it does not contain an active task but contains at least one waiting task.

The report provides the following information:

Setup:

Field	Description
Model name	Name of the simulated model.
Start time	Start of the simulation process.
End time	End of the simulation process.
Duration	Duration of the simulation process.
Actual End time	Time when the simulation process was stopped (the real End time).
Actual Duration	Actual duration of the simulation.

Token

Process statistics

Field	Description
Process	Name of the simulated process.
Started count	Number of starts of the process.
Completed count	Number of completed processes.

Activity statistics

Field	Description
Activity	Name of the simulated activity.
Completed count	Number of tokens completed for this activity during the simulation time.

Time

Process statistics

Field	Description
Process	Name of the simulated process.
Total time	Total process simulation time.
Average time	Average process simulation time.
Total work time	Total process work time.
Average work time	Average process work time.
Total wait time	Total process wait time.
Average wait time	Average process wait time.

Activity statistics

Field	Description
Activity	Name of the simulated activity.
Total time	Total activity simulation time.
Average time	Average activity simulation time.
Total work time	Total activity work time.
Average work time	Average activity work time.
Total wait time	Total activity wait time.
Average wait time	Average activity wait time.

Cost

Resource statistics

Field	Description
Activity	Name of the simulated activity.
Work cost	Work cost per time unit for the activity.

Process statistics

Field	Description
Process	Name of the simulated process.
Total cost	Total cost of the simulated processes.
Average cost	Average cost of the simulated processes.
Total work cost	Total cost of the work.
Average work cost	Average cost of the work.
Total wait cost	Total cost of the wait time.
Average wait cost	Average cost of the wait time.

Activity statistics

Field	Description
Activity	Name of the simulated activity.
Total cost	Total cost of the simulated activity.
Average cost	Average cost of the simulated activity.
Total work cost	Total cost of the work of the simulated activity.
Average work cost	Average cost of the work of the simulated activity.
Total wait cost	Total cost of the activity wait time.
Average wait cost	Average cost of the activity wait time.

Note: The report content is generated using the sim-doc.xsl template from the sim-data.xml file generated by the simulation run and simple sim-layout.xml template defining sections and tables within the report. The files are located in the **com.borland.tg.bpm.simulation.core** plugin under the **report** folder. It is possible to change the report view by editing the corresponding files. It is also possible to include some files in the simulation report by adding them into the /report/content folder. The included files will be present in the **content** folder of the generated report.

Related Concepts

Business Process Modeling

Related Procedures

Together Business Process Modeling

Related Reference

Launch BPMN Simulation

UML 1.4 Reference

This section contains reference material about UML 1.4 diagrams.

In This Section

UML 1.4 Class Diagrams

This section describes the elements of UML 1.4 Class Diagrams.

In This Section

UML 1.4 Class Diagram Elements

The table below lists the elements of UML 1.4 class diagrams that are available using the Palette.

Note that the available elements depend on the type of the project and selected profiles.

Package	node
Class	node
Interface	node
Enumeration	This element is available in the source code projects only.
Class by Template	Available in the source code projects. Opens the **Apply Template** wizard.
Association Class	node
Link by Template	Available in the source code projects. Opens the **Apply Template** wizard.
Object	Available in the design and source code projects.
Association end	link
Generalization/Implementation	link
Association	link
Dependency	link
Note	annotation
Note link	annotation link

Attribute Context Menu

All of the UML diagram elements share common context menu commands. To use the context menu for an element, right-click the element in the Diagram editor. To view the common context menu commands, see "Common Element Context Commands."

The context menu for an attribute shares the common element context commands as well as the following commands specific to it.

Open

Selecting Open from the context menu opens in the text editor the selected class that contains the attribute and highlights the attribute in the source code.

Show in Packages View

The Show in Packages View command highlights the node selected in the Packages tree-view. The Packages view will expand and highlight that element in the tree-view. If closed, the Packages view will open.

Show in Model Package Explorer

The Show in Model Package Explorer View command highlights the node selected in the UML Explorer tree-view. The Model Package Explorer view will expand and highlight that element in the tree-view. If closed, this view will open.

Modifiers

Static	Selecting Static from the context menu sets the static property for the attribute.
Public	Selecting Public from the submenu sets the visibility property for the attribute to public.
Protected	Selecting Protected from the submenu sets the visibility property for the attribute to protected.
Private	Selecting Private from the submenu sets the visibility property for the attribute to private.
Package Local	Selecting Package Local from the submenu sets the visibility property for the attribute to package local.

Apply Template

Selecting this command launches the Apply Template dialog, which displays the available templates. Select a template from the Templates list to apply.

Related Reference

Common Element Context Commands

Attribute Properties

This section describes the properties specific to attributes of classes, inner classes, and interfaces. Every element has general properties as well as specific properties. For more information, see “Properties View.” The composition of the Properties view changes depending on the element selected in the Diagram Editor or Model Navigator view. You can view and modify values of properties through the Properties View.

Attribute Properties

alias	Lets you give an element a name that would not normally be accepted for that element. For example, use the alias property to represent a class name with spaces to make it more readable. Using the alias property does not affect the original name. The Diagram editor shows the alias for an element.
associates	The associates property is displayed in the Properties view whenever an attribute has a non-primitive type. Click the Selection dialog button to open the Selection dialog for selecting model or java elements.
final	Set this value as true or false from the drop-down list.
initial value	Use this field to set the initial value of the attribute.
name	The name of the attribute.
static	Set this value as true or false from the drop-down list.
stereotype	Use this field to add your own stereotype property.
transient	Set this value as true or false from the drop-down list.
type	This field represents the attribute type (String, Boolean, double, float, and so on). Choose the appropriate type from the drop-down list.
visibility	Set the visibility for the attribute from the drop-down list. Choose from public, private, protected, or package local.
volatile	Set this value as true or false from the drop-down list. Final, static, transient and volatile properties only apply to Class attributes.

Javadoc Properties

The Properties View for source-generating elements also displays Javadoc properties. Using javadoc tags enables you to automatically generate a complete, well-formatted API from your source code. You can enter a description and specify values for Javadoc tags applicable to the selected element. These values are used when you generate Javadoc using the Documentation Generation feature of Together (**File** ▸ **Export** ▸ **UML Documentation**).

Filling in the Javadoc fields automatically generates appropriate tags in the Javadoc tags in the source code.

deprecated	Adds a comment to the generated documentation indicating that the API is deprecated and should no longer be used (even though it may continue to work).
see	Adds a hyperlinked "See also" entry when using the Documentation Generation feature.
since	Adds a "Since" entry with the specified since-text to the generated documentation. This tag indicates that this change or feature has existed since the software release specified by the since-text.

Related Concepts

UML 1.4 Class Diagram Definition

Related Reference

Properties View

Class Context Menu

Diagram Editor ▸ Select Class element ▸ Right-click

All of the UML diagram elements share common context menu commands. To use the context menu for an element, simply right click on the element in the Diagram Editor . To view the common context menu commands, see “Common Element Context Commands.”

The context menu for the class and interface elements share the common element context commands as well as the following commands specific to both.

New

The **New** command for the class or interface element offers a submenu with the following options:

Attribute	Selecting Attribute from the submenu adds an attribute to the class or interface.
Operation	Selecting Operation from the submenu adds an operation to the class or interface.
Constructor	Selecting Constructor from the submenu adds a constructor to the class or interface.
Property	Selecting Property from the submenu adds a property to the class or interface.
Inner Class	Selecting Inner Class from the submenu adds an inner class to the class or interface.
Inner Interface	Selecting Inner Interface from the submenu adds an inner interface to the class or interface.

Open

Selecting Open from the context menu opens the selected class or interface in the text editor.

Open Type Hierarchy

The Open Type Hierarchy command highlights the node selected in the Hierarchy view. The Hierarchy view will expand and highlight that element in the tree-view. If closed, the Hierarchy view will open.

QA Source

The **QA Source** command for the class element offers a submenu with the following options:

Load Audit Results	Use this command to load a set of audits results.
Load Metrics Results	Use this command to load a set of metrics results.
Audits	Selecting Audit from the submenu processes audits for only the selected class or interface. For more information, see “Running Model Audits and Metrics.”
Metrics	Selecting Metrics from the submenu processes metrics for only the selected class or interface. For more information, see “Running Model Audits and Metrics.”

Add linked

The **Add Linked** command is available for class and package diagrams, and provides search options for references, implementations, and inheritance according to the specified types and scopes. Search applies to a single object or to a group of selected objects.

Apply Template

Selecting **Apply Template** launches the Apply Template dialog, which displays the available templates. From the resulting dialog, select the template you want for the new class. Invoking **Choose Template** enables refactoring of the class according to the specific template. For more information, see “Creating Class By Template.”

Save As Template

Selecting the **Save As Template** command displays the Create Template dialog, which lets you save the class or interface as a template.

Add Bookmark

Bookmarks let you navigate to resources that are frequently used. You can set, remove, and view bookmarks using the Bookmarks view.

Hide / Show

The **Hide / Show** command for a class (or interface) element offers a submenu with the following options:

Attributes	Selecting **Attributes** from the submenu hides only the attributes of the class. The **Hide / Show** command for Attributes works as a toggle. To redisplay hidden attributes, right-click the class where the attributes are hidden and select **Hide / Show** ▸ **Attributes** from the context menu.
Operations	Selecting **Operations** from the submenu hides only the operations of the class. The **Hide / Show** command for Operations works as a toggle. To redisplay hidden operations, right-click the class where the operations are hidden and select **Hide / Show** ▸ **Operations** from the context menu.
Properties	Selecting **Properties** from the submenu hides only the properties of the class. The **Hide / Show** command for Properties works as a toggle. To redisplay hidden properties, right-click the class where the properties are hidden, and select **Hide / Show** ▸ **Properties** from the context menu.
Inner Classes	Selecting **Inner Classes** from the submenu hides only the inner classes of the class. The**Hide / Show** command for Inner Classes works as a toggle. To redisplay a hidden inner class, right-click the class where it was hidden and select **Hide / Show** ▸ **Inner Classes** from the context menu.
Inner Interfaces	Selecting **Inner Interfaces** from the submenu hides only inner interfaces of the class. The **Hide / Show** command for Inner Interfaces works as a toggle. To redisplay a hidden inner interface, right-click on the class where it was hidden and select **Hide / Show** ▸ **Inner** Interfaces from the context menu.
All	Selecting **All** from the submenu hides attributes, operations, properties, inner classes, and inner interfaces of the class. The **Hide / Show** command for **All** works as a toggle. To redisplay hidden class elements, right-click the class where the elements are hidden and select **Hide / Show** ▸ **All** from the context menu.

Hide

The **Hide** command hides a class in the Diagram Editor . To show classes hidden after using this command, right-click the diagram background, and select **Hide/Show**.

Compare With

Use **Compare With** to compare the resources in the Workbench with the resources held within the repository. The **Compare With** command offers a submenu with the following options:

Using the Each Other command simply compares two files. Select two diagrams in the Model Navigator view by using `CTRL+Click`, then invoking **Compare With Each Other**.

Related Procedures

Creating Class By Template
Running Model Audits and Metrics

Related Reference

Common Element Context Commands

Class Diagram Context Menu

All of the UML diagram types share common context menu commands. To use the context menu for a diagram, right-click in theDiagram Editor .

To view the common context menu commands, see “Common Diagram Context Commands.”

The class diagram context menu shares the common context menu commands as well as commands specific to it.

New

The New command for the class diagram offers a submenu with the following options:

Package	Adds a package element to the diagram.
Class	Adds a class element to the diagram.
Interface	Adds an interface element to the diagram.
Association class	Adds an association class element to the diagram.
Object	Adds an object element to the diagram.
Apply Template	Opens the Apply Template dialog, which displays the available templates.
Note	Adds a note element to the diagram.
Shortcut	To refer to an element located outside of the current diagram or to another diagram, you can use shortcuts. Invoking the Shortcut command displays a selection dialog, where you can choose the desired element (or diagram) from the appropriate location.

Quality Assurance

The Quality Assurance command for the class diagram offers a submenu with the following options:

Save Metrics Results	After you have run metrics on a project or part of a project, you can save those metric results and view them whenever you like. Use this command to load a set of metrics results. For more information, see “Saving and Loading Metric Results.”
Metrics	Selecting Metrics from the submenu processes metrics for only the specific diagram. For more information, see “Running Source Code Metrics.”
Audit	Selecting Audit from the submenu processes audits for only the specific diagram. For more information, see “Running Source Code Audits.”

Related Procedures

Running Source Code Audits
Running Source Code Metrics
Saving and Loading Metric Results

Related Reference

Common Diagram Context Commands

Class Diagram Properties

This section describes the properties specific to attributes of classes, inner classes, and interfaces. Every element has general properties as well as specific properties. For more information, see “Properties View.” The composition of the Properties view changes depending on the element selected in the Diagram Editor or Model Navigator view. You can view and modify values of properties through the Properties View.

diagram type	Shows the current diagram type.
name	The name of the class diagram.
stereotype	Choose the appropriate stereotype from the drop-down list or add your own stereotype. The available stereotypes are: ▪ data management ▪ facade ▪ framework ▪ human interaction ▪ problem domain ▪ stub ▪ subsystem ▪ system ▪ system interaction

Related Reference

Properties View

Class Diagram Relationships

There are several kinds of relationships for UML 1.4 and UML 2.0 Class diagrams.

Types of Relationships

Association		A relationship between instances of the two classes. There is an association between two classes if an instance of one class must know about the other to perform its work. In a diagram, an association is a link connecting two classes. Associations can be directed or undirected. A directed link points to the supplier class (the target). An association has two ends. An end may have a role name to clarify the nature of the association. A navigation arrow on an association shows which direction the association can be traversed or queried. A class can be queried about its Item, but not the other way around. The arrow also lets you know who "owns" the implementation of the association. Associations with no navigation arrows are bi-directional.
	Simple Association	A binary association in which `aggregationKind = none`.
	Aggregation	An association in which one class belongs to a collection. An aggregation has a diamond end pointing to the part containing the whole.
	Composition	An association that represents a composite aggregation (that is, a whole/part relationship). A composite aggregation is a strong form of aggregation that requires a part instance to be included in a maximum of one composite at a time. A composition has a filled diamond at the aggregate end.
Generalization/Implementation		An inheritance link indicating that a class implements an interface. An implementation has a triangle pointing to the interface.
Dependency		A supplier/client relationship between model elements in which modification of the supplier could impact the client model elements. A dependency implies that the semantics of the client are not complete without the supplier.
Part		An “owned” property that corresponds to a composition (that is, a composite aggregation). The syntax of such a property differs from a referenced part, or “shared” property. In the abstract syntax, which defines the model, a part is the role name for a relationship that a StructuredClassifier has with an owned property. In the concrete syntax, which defines the diagram, a part is the name of the graphical node that represents such an owned property.
Referenced Part		A “shared” property that corresponds to a shared association (that is, an aggregation). The syntax of such a property differs from a part, or “owned” property. In the abstract syntax, which defines the model, a referenced part is the role name for a relationship that a StructuredClassifier has with a shared property. In the concrete syntax, which defines the diagram, a referenced part is the name of the graphical node that represents such a shared property.
Required interface		Available in UML 2.0 class diagrams. Applying a provided interface link to a port on the client class creates a link in ball-and-socket notation.
Provided interface		Available in UML 2.0 class diagrams. Applying a provided interface link between a class and an interface creates a regular generalization/implementation link.
Instantiates		Available in UML 2.0 class diagrams. This link can be drawn between an instance specification and its instantiated class.

Multiplicities

Every class diagram has classes and associations. Navigability, roles, and multiplicities are optional items placed in a diagram to provide clarity.

The multiplicity of an association end is the number of possible instances of the class associated with a single instance of the other end. Multiplicities are single numbers or ranges of numbers.

0..1	Zero or one instance. The notation n . . m indicates n to m occurrences
0..* or	No limit on the number of occurrences (including none)
1	Exactly one occurrence
1..*	At least one occurrence
-1	No limit on the number of occurrences (note that this multiplicity value is displayed as `*` on the diagram)

Related Procedures

Changing Type of an Association Link

Related Reference

UML 1.4 Class Diagrams
UML 2.0 Class Diagrams

Class, Inner Class, and Interface Properties

This section describes the properties specific to classes, inner classes, and interfaces. Every element has general properties as well as specific properties. For more information, see “Properties View.” The composition of the Properties view changes depending on the element selected in the Diagram or Model Navigator view. You can view and modify values of properties through the Properties View.

abstract	(Class only) Set this value to true or false using the drop-down menu.
alias	Allows you to give an element a name that would not normally be accepted for that element. For example, use the alias property to represent a class name with spaces to make it more readable. Using the alias property does not effect the original name. The Diagram editor shows the alias for an element.
extends	If a class extends another class, use the drop-down arrow to select the appropriate class.
file	Lists the file name and its location in the project directory.
final	(Class only) Set this value as true or false from the drop-down list.
implements	(Class only) If a class implements an interface, use the drop-down arrow to select the appropriate interface.
invariants	This field allows you to enter text describing what is always true for all instances of a class.
name	The name of the class, inner class or interface.
package	The package the class or inner class belongs to.
persistent	Set this value as true or false using the drop-down list.
public	(Class only) Set this value as true or false using the drop-down list.
stereotype	Use this field to add your own stereotype property.

Javadoc

The Properties View or source-generating elements also displays Javadoc properties. Using Javadoc tags enables you to automatically generate a complete, well-formatted API from your source code. You can enter a description and specify values for Javadoc tags applicable to the selected element. These values are used when you generate Javadoc using the Documentation Generation feature of Together (**File** ▸ **Export** ▸ **Documentation Using Template**).

Filling in the Javadoc fields automatically generates appropriate tags in the Javadoc tags in the source code. The `@author` and `@see` tags allow multiple values. In this case, the values are separated by commas in the Properties field.

author	Creates an "Author" entry. This field can contain multiple `@author` tags.
deprecated	Adds a comment to the generated documentation indicating that the API is deprecated and should no longer be used (even though it may continue to work).
see	Adds a hyperlinked "See also" entry when using the Documentation Generation feature. For example, `@see java.lang.String`.
since	Adds a "Since" entry with the specified since-text to the generated documentation. This tag indicates that this change or feature has existed since the software release specified by the since-text.
version	Adds a "Version" entry to the class. A doc comment may contain at most one `@version` tag.

Related Reference

Properties View

Dependency Link Properties

This section describes the dependency links' specific properties. Every element has common properties as well as specific properties. For more information, see “Properties View.” The composition of the Properties View changes depending on the element selected in theDiagram Editor or Model Navigator view. You can view and modify values of properties through the Properties View.

Property	Description
client	This property field indicates the client for the link.
client role	Use client role to add a label to the dependency link. The label appears on the UML diagram towards the client side of the link.
label	Use label to add a label to the dependency link. The label is displayed on the UML diagram between the client and the supplier.
stereotype	Use this field to add your own stereotype property.
supplier	This property field indicates the supplier for the link.
supplier role	Use supplier role to add a label to the dependency link. The label is displayed on the UML diagram towards the supplier side of the link.

Related Reference

Properties View

Extend/Include Link Properties

This section describes the properties specific to extend and include links. Every diagram element has general properties as well. For more information, see “Properties View.” The composition of the Properties view changes depending on the element selected in the Diagram or Model Navigator view. You can view and modify values of properties through the Properties view.

Extend Link Properties

comment	Use this field to edit the comment property.
condition	Use this field to edit the condition property.
label	Use this property to indicate the label for the link.
metaclass	Use this property to indicate the metaclass for the link.
stereotype	Use this field to add your own stereotype property.

Include Link Properties

comment	Use this field to edit the comment property.
label	Use this property to indicate the label for the link.
metaclass	Use this property to indicate the metaclass for the link.
stereotype	Use this field to add your own stereotype property.

Related Reference

Properties View

Generalization/Implementation Link Properties

This section describes the properties specific to generalization/implementation links. Every element has general properties as well as specific properties. For more information, see “Properties View.” The composition of the Properties view changes depending on the element selected in the Diagram or Model Navigator view. You can view and modify values of properties through the Properties view.

Item	Description
client	This property field indicates the client for the link.
supplier	This property field indicates the supplier for the link.

Related Reference

Properties View

LiveSource Rules

The impact of changing a class, interface, or package on a logical class diagram varies according to the kind of change:

- Changing the name, adding a member, creating a new link, or applying a pattern makes the corresponding change in the actual source code.
- Choose **Delete from View** on the context menu of the element to remove the element from a current diagram and keep the element in the namespace (package).
- Choose **Delete** on the context menu to completely remove the element from the model.
- When you press DELETE on the keyboard, the **Delete from view** command is applied, if it is available in this particular situation. If it is not, the element is deleted completely.
- Direct changes in source code editor, such as renaming a class, cannot be tracked by Together. Use refactoring operations for this purpose.

Related Concepts

Roundtrip Engineering Overview

Related Procedures

Opening a Diagram Element in the Source Code Editor

Related Reference

UML 1.4 Class Diagrams
UML 2.0 Class Diagrams

Object Context Menu

All of the UML diagram elements share common context menu commands. To use the context menu for an element, right-click on the element in the Diagram editor. To view the common context menu commands, see “Common Element Context Commands.”

The object element offers the following additional context menu selections:

New

The object element has a **New** menu item, with a submenu for creating a new class, interface or a slot.

Item	Description
Class	Opens the **New Object's Class** dialog.
Interface	Opens the **New Object's Interface** dialog.
Linked note	Creates a note with the activated in-place editor.
Slot	Adds a slot to the object.

Select Class

Select Class expands to display a submenu with any classes that are "local" to the diagram and an option named More for browsing available classes that you wish to associate with the object.

Local Class List (Class1, Class2)	If there are classes that are "local" to the diagram, they will be displayed in this list. Select the class from the list to associate it with the object
More	Select More to open a file browsing dialog to select a class to associate with an object.

Show Class

After a class or interface has been associated with an object, the Show Class command is enabled. This command offers special context commands:

In Editor	Opens the source file in the Editor window.
In Navigator	Opens the Navigator, navigates to and highlights the source file in the view.
In Package Explorer	Opens the Package Explorer view, navigates to and highlights the source file in the view.
In Model Navigator	Opens the Model Navigator, navigates to and highlights the source file in the view.

Unlink Class

Use Unlink Class to unlink a previously linked class or interface from an object.

Related Reference

Common Element Context Commands

Object Properties

Diagram Editor ▸ Right-click Object ▸ Properties

This section describes properties specific to Object elements. Every element has general properties as well as specific properties. For more information, see "Properties View." The composition of the Properties View changes depending on the element selected in the Diagram Editor or Model Navigator view. You can view and modify values of properties through the Properties view.

active	Set this value as true or false from the drop-down list.
concurrent	Set this value as true or false from the drop-down list.
instantiates	Use the ellipsis button to open the Select Class dialog, and navigate to and select the instantiating class.
in	This field represents the package that the class resides in.
location	This field represents the physical location for the class.
multiple instance	Set this value as true or false from the drop-down list.
name	The name of the object.
persistence	Choose the appropriate persistence type from the drop-down list. Objects have the following persistence types: `transient, static, persistent` **Note:** When setting the persistence property for an object on class, collaboration, activity, state, or deployment diagrams the persistence property is displayed below the name of the object in curly braces.
state	Use this text field to describe the state of an object.
stereotype	Use this field to add your own stereotype property.

Related Reference

Properties View

Operation Context Menu

All of the UML diagram elements share common context menu commands. To use the context menu for an element, right-click on the element in the Diagram editor. To view the common context menu commands, see “Common Element Context Commands.”

The context menu for an operation shares the common element context commands as well as the following commands specific to it:

Open

Selecting Open from the context menu opens the selected class containing the operation in the text editor.

Show in Packages View

The Show in Packages View command highlights the node selected in the Packages tree-view. The Packages view will expand and highlight that element in the tree-view. If closed, the Packages view will open.

Show in Model Package Explorer

The Show in Model Package Explorer View command highlights the node selected in the UML Explorer tree-view. The Model Package Explorer view will expand and highlight that element in the tree-view. If closed, this view will open.

Modifiers

The Modifiers command for the operation offers a submenu with the following options:

Static	Selecting Static from the context menu sets the static property for the operation.
Abstract	Selecting Abstract from the context menu sets the abstract property for the operation.
Public	Selecting Public from the submenu sets the visibility property for the operation to public.
Protected	Selecting Protected from the submenu sets the visibility property for the operation to protected.
Private	Selecting Private from the submenu sets the visibility property for the operation to private.
Package Local	Selecting Package Local from the submenu sets the visibility property for the operation to package local.

Note: The visibility options are not available for Interface members.

Add Javadoc comment

Using this command, you are able to add Javadoc comments for the operation.

Generate Sequence Diagram

Use this command to generate sequence diagrams from your source code. For more information, see “Working with Operations in Sequence/Collaboration Diagrams.”

Related Procedures

Working with Operations in Sequence/Collaboration Diagrams
Roundtrip Engineering with Sequence Diagrams

Related Reference

Common Element Context Commands

Operation Properties

This section describes the properties specific to operations (methods) of classes, inner classes, and interfaces. Every element has general properties as well as specific properties. For more information, see “Properties View.” The composition of the Properties View changes depending on the element selected in the Diagram Editor or Model Navigator. You can view and modify values of properties through the Properties View.

Operation Properties

Property	Description
abstract	Set this value as true or false from the drop-down list.
alias	lets you give an element a name that would not normally be accepted for that element. For example, use the alias property to represent a class name with spaces to make it more readable. Using the alias property does not affect the original name. The Diagram editor shows the alias for an element.
final	Set this value as true or false from the drop-down list.
input	This field lets you enter text describing the inputs for an operation.
name	The name of the operation.
native	Set this value as true or false from the drop-down list.
output	This field lets you enter text describing the outputs for an operation.
parameters	Add the parameters for an operation using this field.
post-condition	This field lets you enter text describing the post-conditions for an operation.
precondition	This field lets you enter text describing the preconditions for a method an operation.
return value	Specify the return value of a method. Choose the appropriate return value from the drop-down list.
semantics	This field lets you enter text describing the semantics of an operation.
static	Set this value as true or false from the drop-down list.
stereotype	Use this field to add your own stereotype property.
synchronized	Set this value as true or false from the drop-down list.
throws	Use the drop-down list to select the appropriate throws statement.
time	This field lets you enter text for the time requirement of an operation.
visibility	Set the visibility for the operation from the drop-down list. Choose from public, private, protected, or package local.

Note: Abstract, final, native, static, synchronized and visibility properties apply only for Class operations.

Javadoc Properties

The Properties view for source-generating elements also displays Javadoc properties. Using javadoc tags enables you to automatically generate a complete, well-formatted API from your source code. You can enter a description and specify values for Javadoc tags applicable to the selected element. These values are used when you generate Javadoc using the Documentation Generation feature of Together (File > Export > UML Documentation). Filling in the Javadoc fields automatically generates appropriate tags in the Javadoc tags in the source code.

Property	Description
deprecated	Adds a comment to the generated documentation indicating that the API is deprecated and should no longer be used (even though it may continue to work).
exception	Use the `@exception` tag in place of the `@throws` tag. Use the file chooser button to designate the class-name of the exception that may be thrown by the method. A "Throws" heading is inserted in the generated documentation.
params	Adds a parameter to the "Parameters" section. You can insert multiple values in this field.

return	Use this field to add a "Returns" comment with the description text. The text should describe the return type and a permissible range of values.
see	Adds a hyperlinked "See also" entry when using the Documentation Generation feature.
since	Adds a "Since" entry with the specified since-text to the generated documentation. This tag indicates that this change or feature has existed since the software release specified by the since-text.

Related Reference

Properties View

Package Properties

This section describes package-specific properties. Every element has general properties as well as specific properties. For more information, see “Properties View.” Edit package properties through the Properties View. The composition of the Properties View changes depending on the element selected in the Diagram Editor or Model Navigator. You can view and modify values of properties.

diagram type	This read-only field displays the diagram type.
name	The name of the package.
package	This field lists the name of the package.
stereotype	Use this field to add your own stereotype property.

Related Reference

Properties View

UML 1.4 Use Case Diagrams

This section describes the elements of UML 1.4 Use Case Diagrams.

In This Section

UML 1.4 Use Case Diagram Elements

The following table lists the elements of UML 1.4 Use Case diagrams that are available using the Palette.

Actor	Draws an actor within the Diagram Editor .
Use Case	Draws a use case within the Diagram Editor .
Communicates	Draws a communication link between use case elements and actors.
Extend	Draws an extends link between use case elements.
Include	Draws an includes link between use case elements.
Generalization	Draws a generalization link between use case elements. A generalization link can be also created between a pair of Use Cases and a pair of Actors.
System Boundary	Draws a system boundary to separate a system from external actors.
Note	Creates an annotation.
Note Link	Creates an annotation link.

Actor Properties

This section describes properties specific to actor elements. Every element has general properties as well as specific properties. For more information, see “Properties View.” The composition of the Properties view changes depending on the element selected in the Diagram or Model Navigator view. You can view and modify values of properties through the Properties view.

explanation	Use this field to enter text to explain the actor.
name	Use this field to specify the name of the actor.
stereotype	Use this field to add your own stereotype property.

Related Reference

Properties View

Generalization Link Properties

This section describes the properties specific to generalization/implementation links. Every element has general properties as well as specific properties. For more information, see “Properties View.” The composition of the Properties view changes depending on the element selected in the Diagram or Model Navigator view. You can view and modify values of properties through the Properties view.

If the concept includes subtopics, introduce what the reader will encounter in the text.

label	Use this property to indicate the label for the link.
stereotype	Use this property to indicate the stereotype for the link.

Related Reference

Properties View

Use Case Diagram Context Commands

Diagram Editor ▸ Right-click

All of the UML diagram types share common context menu commands. To use the context menu for a diagram, right-click in the Diagram Editor .

To view the common context menu commands, see “Common Diagram Context Commands.”

The use case diagram offers a special context command **New** with a submenu for adding new elements to the use case diagram:

Option	Description	
New	The **New** command for the Use Case diagram offers a submenu with the following options:	
	Actor	Adds an actor element to the diagram.
	Use Case	Adds a use case element to the diagram.
	System Boundary	Adds a system boundary element to the diagram.
	Note	Adds an annotation to the diagram.
	Shortcut	To refer to an element located outside of the current diagram or to another diagram, you can use shortcuts. Invoking the Shortcut command displays a selection dialog in which you can choose the desired element (or diagram) from the appropriate location.

Related Reference

Common Diagram Context Commands

Use Case Diagram Elements Context Menu

All of the UML diagram elements share common context menu commands. To use the context menu for an element, right-click on the element in the Diagram editor. To view the common context menu commands, see “Common Element Context Commands.”

The context menus described in this section are specific to UML 1.4 use case diagram elements.

Use Case

The use case element offers a special context command named, **New**, with a submenu for adding new elements to the diagram:

Extension Point	Adds an extension point to a use case element. For more information, see “Creating an Extension Point.”

System Boundary

The system boundary element offers a special context command **New**.

Option	Description	
New	Opens a submenu for adding a Use Case element and a Note.	
	Use Case	Adds a use case within the system boundary.
	Note	Adds a note.

Related Procedures

Creating an Extension Point

Related Reference

Common Element Context Commands

Extension Point

An **extension point** refers to a location within a use case where you can insert action sequences from other use cases.

An extension point consists of a unique name within a use case and a description of the location within the behavior of the use case.

In a use case diagram, extension points are listed in the use case with the heading "Extension Points" (appears as bold text in the Diagram View).

Related Reference

UML 1.4 Use Case Diagrams
UML 2.0 Use Case Diagrams

Use Case Properties

Right-click Use Case ▸ Properties

This section describes the specific properties for use case elements. Every diagram and diagram element has general properties as well. For more information, see “Properties View.” The composition of the Properties view changes depending on the element selected in the Diagram Editor or the Model Navigator view. You can view and modify values of properties through the Properties View.

abstract	Set this value as true or false from the drop-down list. Setting this value true for a use case element displays the name of the element in italics on the diagram.
alternative flow	Use the text field to describe an alternate flow of events.
explanation	This field lets you enter text to explain the use case.
name	The name of the use case element.
normal flow	Use the text field to describe the normal flow of events.
post-conditions	This field lets you enter text describing the state of the system after the use case is performed.
pre-conditions	This field lets you enter text describing the necessary conditions that have to be met before the use case can be performed.
rank	Use the rank field to rank use case. For example, you might want to rank a use case for complexity and risk.
stereotype	Use this field to add your own stereotype property.

Related Reference

Properties View

UML 1.4 Interaction Diagrams

This section describes the elements of UML 1.4 Sequence and Collaboration diagrams.

In This Section

UML 1.4 Interaction Diagram Elements

The table below lists the elements of UML 1.4 Interaction (Sequence and Collaboration) diagrams that are available using the Palette, and context menus of the elements.

You can add shortcuts to the interaction diagrams, by using the **New ▸ Shortcut** command. However, referring to the elements of the other interaction diagrams is not allowed.

	UML 1.4 Interaction Diagram Elements
Object	Draws an object with its lifeline in the Diagram Editor .
Actor	Draws an actor in the Diagram Editor .
Message	Draws a message link between object lifelines.
Self Message	Draws a message link from an object lifeline back to itself.
Message with delivery time	Draws a message link with delivery time between object lifelines. Sequence diagram only.
Conditional Block	Creates a conditional block on an activation point. Sequence diagram only.
Return	Draws a return message.
Association link	Draws an association link. Collaboration diagram only.
Aggregation link	Draws an aggregation link. Collaboration diagram only.
Note	Draws a note.
Note link	Draws a note link.

Conditional Block

Conditional block statement is a flexible tool to enhance a sequence diagram. The following statements are supported:

- if
- else
- for
- foreach
- while
- do while
- try
- catch
- finally
- switch
- case
- default

UML 1.4 Message

By default, message links in a sequence diagram are numbered sequentially from top to bottom. You can reorder messages.

A “self message” is a message from an object back to itself.

Related Reference

UML 2.0 Message

Activation Bar

Together automatically renders the activation of messages that show the period of time that the message is active. When you draw a message link to the destination object, the activation bar is created automatically.

You can extend or reduce the period of time of a message by vertically dragging the top or bottom line of the activation bar as required. A longer activation bar means a longer time period when the message is active.

Nested Message

You can nest messages by originating message links from an activation bar. The nested message inherits the numbering of the parent message.

For example, if the parent message has the number 1, its first nested message is 1.1. It is also possible to create message links back to the parent activation bars.

Related Reference

UML 1.4 Message

Message Link Properties

This section describes the properties specific to message links in UML 1.4 interaction diagrams. The composition of the Properties View changes depending on the element selected in the Diagram Editor or Model Navigator. You can view and modify values of properties through the Properties View.

Property	Description
label	Use this field to add a label to the message link.
name	The name of the link.
operation	Use the text field or the file Edit button to select an operation to associate with the link.
in	A text field that displays the package and source file of the operation.
stereotype	Use this field to add your own stereotype for the link.
visibility	Select the visibility value from the list.
arguments	A text field to update the arguments property.
condition	A text field to update the condition property
creation	Use this field to set the creation property.
destruction	Use this field to set the destruction property.
iteration	Use this field to update the iteration property
non-atomic delivery	Use this field to set the non-atomic delivery property for a message link. By default, message links have atomic delivery.
return	Use this field to update the return property.
return message	Set the property value to `true` to create a return message.
sequence number	This read-only field displays the message sequence number.
synchronization	Use the drop-down list to set the synchronization property, which enables you to reduce or extend the time of invocation and execution simultaneously or independently.

UML 1.4 Statechart Diagrams

This section describes the elements of UML 1.4 Statechart diagrams.

In This Section

UML 1.4 Statechart Diagram Elements

The table below lists the elements of UML 1.4 Statechart diagrams that are available using the Palette.

Option	Description
State	node
Start State	node
End State	node
History	node
Object	node
Transition	link
Horizontal Fork/Join	node
Vertical Fork/Join	node
Note	annotation
Note Link	annotation link

State

A state models a situation during which some (usually implicit) invariant condition holds. The invariant can represent a static situation, such as an object waiting for some external event to occur. However, it can also model dynamic conditions, such as the process of performing an activity (for example, the model element under consideration enters the state when the activity commences and leaves it as soon as the activity is completed).

Actions

Entry and exit actions are executed when entering or leaving a state, respectively.

You can create these actions in statechart diagrams as special nodes or as stereotyped internal transitions.

Composite (nested) state

Create a composite state by nesting one or more levels of states within one state. You can also place start/end states and a history state inside of a state, and draw transitions among the contained substates.

Related Reference

UML 1.4 Statechart Diagrams
UML 1.4 Activity Diagrams
UML 2.0 State Machine Diagrams

Transition

A single transition comes out of each state or activity, connecting it to the next state or activity.

A transition takes an operation from one state to another and represents the response to a particular event. You can connect states with transitions and create internal transitions within states.

Internal transition

An internal transition is a way to handle events without leaving a state (or activity) and dispatching its exit or entry actions. You can add an internal transition to a state or activity element.

Self-transition

A self-transition flow leaves the state dispatching any exit actions, then reenters the state dispatching any entry actions.

Guard expressions

All transitions, including internal ones, are provided with the guard conditions (logical expressions) that define whether this transition should be performed. You can associate a transition with an effect, which is an optional activity performed when the transition fires. The guard condition is enclosed in the brackets (for example, "`[false]`") and displayed near the transition link on a diagram. Effect activity is displayed next to the guard condition. You can define the guard condition and effect using the **Object Inspector Properties Window**.

Guard expressions (inside `[ ]`) label the transitions coming out of a branch. The hollow diamond indicates a branch, and its subsequent merge indicates the end of the branch.

Related Reference

UML 1.4 Statechart Diagrams
UML 1.4 Activity Diagrams
UML 2.0 State Machine Diagrams

Deferred Event

A **deferred event** is a type of internal transition that handles the event and places it in an internal queue until it is used or discarded.

You can add a deferred event to a state or activity element.

Related Reference

UML 1.4 Statechart Diagrams
UML 1.4 Activity Diagrams
UML 2.0 State Machine Diagrams

UML 1.4 Activity Diagrams

This section describes the elements of UML 1.4 Activity Diagrams.

In This Section

UML 1.4 Activity Diagram Elements

The table below lists the elements of UML 1.4 Activity diagrams that are available using the Together Palette.

Activity	Activities are action states in an activity diagram. Action states are states with outgoing transitions that are triggered by the completion of an action associated with the state.
Decision	A decision element indicates possible transitions relative to Boolean conditions of the owning object. The decision represents a branch in the control flow of an activity diagram.
Signal	Signal receipt is an explicit symbol used on an activity diagram for certain kinds of information that can be specified on transitions.
Signal sending	Signal sending is an explicit symbol used on an activity diagram for certain kinds of information that can be specified on transitions.
State	A state is a condition during the life of an object or interaction during which it satisfies a condition, performs an action, or waits for an event.
Start State	You can use start state to indicate the initial state for a state or activity diagram.
End State	You can use end state to indicate the final states for a state or activity diagram.
History	A state region can contain history. History applies to the state element that directly contains it.
Object	An object represents an instance of a class.
Transition	A transition link can be drawn between the following elements: State, Activity, Decision, Signal receipt , Signal sending, Fork/Join, and History.
Fork/join	A transition can have multiple sources, meaning it is a join from several concurrent states; or it can have multiple targets, meaning it is a fork to several concurrent states. Use horizontal and vertical fork/joins to accomplish this task within activity and state diagrams.
Swimlane	Swimlanes are a UML grouping concept used to organize the responsibility for activities and subactivities in an activity diagram.
Object Flow	An object flow relationship can be drawn: from an Activity to an Object, from SignalSending to an Object, from an Object to SignalReceipt, from/to an Object to/from a Fork/Join.
Note	An annotation.
Note link	An annotation link.

Activity Diagram Context Commands

All of the UML diagram types share common context menu commands. To use the context menu for a diagram, right-click in the Diagram editor.

To view the common context menu commands, see “Common Diagram Context Commands.”

The activity diagram offers a special context command, **New**, with a submenu for adding new elements to the activity diagram:

New

The **New** command for the activity diagram offers a submenu with the following options:

Activity	Adds an activity element to the diagram.
Decision	Adds a decision element to the diagram.
Signal Receipt	Adds a Signal Receipt element to the diagram.
Signal Sending	Adds a Signal Sending element to the diagram.
Vertical Fork/Join	Adds a Vertical Fork/Join element to the diagram.
Horizontal Fork/Join	Adds a Horizontal Fork/Join element to the diagram.
State	Adds a state element to the diagram.
Object	Adds a object element to the diagram.
Start State	Adds a Start State element to the diagram.
End State	Adds an End State element to the diagram.
Swimlane	Adds a swimlane element to the diagram.
Note	Adds a note element to the diagram.
Shortcut	To refer to an element located outside of the current diagram or to another diagram, you can use shortcuts. Invoking the **Shortcut** command displays a selection dialog, where you can choose an element (or diagram) from the appropriate location.

Related Reference

Common Diagram Context Commands

History Properties

ActivityDiagram

This section describes properties specific to history elements. Every element has general properties as well as specific properties. For more information, see “Properties View.” The composition of the Properties View changes depending on the element selected in the Diagram Editor or the Navigator.

You can view and modify values of properties through the Properties View.

deep	Use the drop-down list to set the field to true or false. Setting the deep property as true for a history element displays the `*` symbol next to the history element on the diagram.
name	The name of the history element.
stereotype	Use this field to add your own stereotype.

Related Reference

Properties View

Horizontal and Vertical Fork/Join Properties

This section describes properties specific to horizontal and vertical fork/join elements. Every element has general properties as well as specific properties. For more information, see “Properties View.” The composition of the Properties View changes depending on the element selected in the Diagram Editor or the Navigator.

You can view and modify values of properties through the Properties View.

name	The name of the fork/join element.
orientation	Choose either horizontal or vertical for this property.
stereotype	Use this field to add your own stereotype property.

Related Reference

Properties View

Transition Link Properties

This section describes properties specific to the transition links. Every element has general properties as well as specific properties. For more information, see “Properties View.” The composition of the Properties View changes depending on the element selected in the Diagram Editor or the Navigator.

You can view and modify values of properties through the Properties View.

action expression	Document the action expression.
constraint	Add your own stereotype.
effect	Specify the effect property.
event arguments	Document event arguments.
event name	Document the event name.
guard condition	Document a guard condition
label	Indicate the label for the link
receive time	Document the receiving time.
send clause	Document the send clause.
send time	Document the send time.
stereotype	Add your own stereotype property.

Related Reference

Properties View

UML 1.4 Component Diagrams

This section describes the elements of UML 1.4 Component Diagrams.

In This Section

UML 1.4 Component Diagram Elements

The table below lists the elements of UML 1.4 component diagrams that are available using the Palette.

Element	Description
Subsystem	node
Component	node
Interface	node
Supports	link
Dependency	link
Note	annotation
Note link	annotation link

UML 1.4 Deployment Diagrams

This section describes the elements and context menus of UML 1.4 Deployment Diagrams.

In This Section

UML 1.4 Deployment Diagram Elements

Node	Draws a node element within the Diagram editor.
Component	Draws a component element within the Diagram editor.
Interface	Draws an interface element within the Diagram editor.
Supports Link	Draws a supports link between a component and interface.
Association Link	Draws an association link between elements.
Aggregation Link	Draws an aggregation link between elements.
Object	Draws an object element in the Diagram editor.
Dependency Link	Draws a dependency link between nodes, objects and components and from nodes, objects and components to interfaces.
Note	An annotation.
Note Link	An annotation link.

UML 2.0 Reference

This section contains reference material about UML 2.0 diagrams.

In This Section

UML 2.0 Class Diagrams

This section describes the elements of UML 2.0 Class diagrams.

In This Section

UML 2.0 Class Diagram Elements

The table below lists the elements of UML 2.0 class diagrams that are available using the Palette. Note that availability of the elements depends on the project type and profiles.

Package	A package groups elements of the diagram and provides a namespace for those grouped elements.
Class	A classifier whose behavior is described through the interaction of its parts. Within a class you can specify attributes, operations, and other classes.
Interface	Creates a new object whose classifier conforms to a static classifier. The new object resides on the output pin at runtime and is returned as the value of the action. An interface must have at least one class to implement it.
Enumeration	A data type with values enumerated in the model as user-defined literals.
Data Type	Available in design projects only. A classifier that is similar to a class except that its instances are identified only by their value.
Association Class	An element with both association and class properties that connects a set of classifiers. It has its own set of features that do not belong to any of the connected classifiers.
Port	Available in design projects only. Each of the small squares attached to classes that connect the behavior of classes with their internal parts and with the other parts of the system. Ports can specify which service a class provides to its environment and which service a class expects from its environment.
Instance Specification	Available in the source code projects. Specifies the existence of an entity in a modeled system. The entity can be a class, in which case the instance specification describes an object of that class. For example, an instance specification of the class Nation might be Brazil. An entity can also be an association, in which case the instance specification describes the link of the association.
Generalization/Implementation	If one classifier inherits all the behavior of another classifier and furthermore extends it with additional behavior, a generalization link results. The arrow points to the more general of the two.
Required Interface	A required interface specifies a usage dependency (which include the services needed to perform a required function) between instances of a classifier and their interfaces.
Provided Interface	A provided interface represents services that are offered by instances of a classifier to fulfill contractual obligations.
Association	The interaction between model elements, represented by a solid line between them. These interaction links can be either unidirectional (indicating that an actor initiates the interaction) or bidirectional (indicating that an actor can participate in the interaction without initiating it).
Directed Association	An association between a collection of source model elements and a collection of target model elements.
Aggregation	A binary association that specifies the literals for defining the aggregation type of a property.
Composition	A type of aggregation in which the composite object has responsibility for the existence and storage of its composed parts.
Association End	Connects the line depicting an Association Class and the icon depicting the connected classifier. The Association End defines the ends of the Association Class. Names of Association Ends are optional and can be suppressed.
Dependency	Elements whose semantics depend on the definition of a supplier element are in a dependency relationship with the supplier element. Dependency links are shown as dashed arrows between the two model elements with the arrowhead pointing to the supplier element.

Instantiates	A usage dependency between classifiers that specifies that operations on the client classifier create instances of the supplier classifier.
Constraint	A constraint is a Boolean expression that restricts the extension of an element. It restricts by imposing a value that specifies additional semantics beyond what is imposed by other language constructs applied to that element. The element that owns the constraint must have access to Constrained Elements.
Constraint Link	Links a constraint to a diagram element.
Template Signature	Bundles the formal template parameters into a set for the templated diagram element that owns it.
Template Binding	A relationship between a diagram element and a template. The binding specifies template parameter substitutions.
Note	An annotation.
Note Link	An annotation link.

Class Diagram Relationships

There are several kinds of relationships for UML 1.4 and UML 2.0 Class diagrams.

Types of Relationships

Association		A relationship between instances of the two classes. There is an association between two classes if an instance of one class must know about the other to perform its work. In a diagram, an association is a link connecting two classes. Associations can be directed or undirected. A directed link points to the supplier class (the target). An association has two ends. An end may have a role name to clarify the nature of the association. A navigation arrow on an association shows which direction the association can be traversed or queried. A class can be queried about its Item, but not the other way around. The arrow also lets you know who "owns" the implementation of the association. Associations with no navigation arrows are bi-directional.
	Simple Association	A binary association in which `aggregationKind = none`.
	Aggregation	An association in which one class belongs to a collection. An aggregation has a diamond end pointing to the part containing the whole.
	Composition	An association that represents a composite aggregation (that is, a whole/part relationship). A composite aggregation is a strong form of aggregation that requires a part instance to be included in a maximum of one composite at a time. A composition has a filled diamond at the aggregate end.
Generalization/Implementation		An inheritance link indicating that a class implements an interface. An implementation has a triangle pointing to the interface.
Dependency		A supplier/client relationship between model elements in which modification of the supplier could impact the client model elements. A dependency implies that the semantics of the client are not complete without the supplier.
Part		An “owned” property that corresponds to a composition (that is, a composite aggregation). The syntax of such a property differs from a referenced part, or “shared” property. In the abstract syntax, which defines the model, a part is the role name for a relationship that a StructuredClassifier has with an owned property. In the concrete syntax, which defines the diagram, a part is the name of the graphical node that represents such an owned property.
Referenced Part		A “shared” property that corresponds to a shared association (that is, an aggregation). The syntax of such a property differs from a part, or “owned” property. In the abstract syntax, which defines the model, a referenced part is the role name for a relationship that a StructuredClassifier has with a shared property. In the concrete syntax, which defines the diagram, a referenced part is the name of the graphical node that represents such a shared property.
Required interface		Available in UML 2.0 class diagrams. Applying a provided interface link to a port on the client class creates a link in ball-and-socket notation.
Provided interface		Available in UML 2.0 class diagrams. Applying a provided interface link between a class and an interface creates a regular generalization/implementation link.
Instantiates		Available in UML 2.0 class diagrams. This link can be drawn between an instance specification and its instantiated class.

Multiplicities

Every class diagram has classes and associations. Navigability, roles, and multiplicities are optional items placed in a diagram to provide clarity.

The multiplicity of an association end is the number of possible instances of the class associated with a single instance of the other end. Multiplicities are single numbers or ranges of numbers.

0..1	Zero or one instance. The notation n . . m indicates n to m occurrences
0..* or	No limit on the number of occurrences (including none)
1	Exactly one occurrence
1..*	At least one occurrence
-1	No limit on the number of occurrences (note that this multiplicity value is displayed as `*` on the diagram)

Related Procedures

Changing Type of an Association Link

Related Reference

UML 1.4 Class Diagrams
UML 2.0 Class Diagrams

Class Diagram Properties

This section describes the properties specific to attributes of classes, inner classes, and interfaces. Every element has general properties as well as specific properties. For more information, see “Properties View.” The composition of the Properties view changes depending on the element selected in the Diagram Editor or Model Navigator view. You can view and modify values of properties through the Properties View.

diagram type	Shows the current diagram type.
name	The name of the class diagram.
stereotype	Choose the appropriate stereotype from the drop-down list or add your own stereotype. The available stereotypes are: ▪ data management ▪ facade ▪ framework ▪ human interaction ▪ problem domain ▪ stub ▪ subsystem ▪ system ▪ system interaction

Related Reference

Properties View

Association Class and N-ary Association

Association classes are displayed in diagrams as three related elements:

- Association class itself (represented by a class icon)
- N-ary association class link (represented by a diamond)
- Association connector (represented by a link between both)

Association classes can connect to as many association end classes (participants) as required.

The Properties View of an association class, association link, and connector contains an additional Association tab. This tab displays the only label property, its value being synchronized with the name of the association class. For the association classes and association end links, the Custom node of the Properties View displays additional properties that correspond to the role of this part of n-ary association (`associationClass` and `associationEnd`, respectively).

You can delete each of the association end links or participant classes without destroying the entire n-ary association. However, deleting the association class results in deleting all the components of the n-ary association.

Related Procedures

Working with Association classes and n-ary associations

Related Reference

Class Diagram Relationships
UML 2.0 Class Diagrams
UML 1.4 Class Diagrams

Dependency Link Properties

This section describes the dependency links' specific properties. Every element has common properties as well as specific properties. For more information, see “Properties View.” The composition of the Properties View changes depending on the element selected in theDiagram Editor or Model Navigator view. You can view and modify values of properties through the Properties View.

Property	Description
client	This property field indicates the client for the link.
client role	Use client role to add a label to the dependency link. The label appears on the UML diagram towards the client side of the link.
label	Use label to add a label to the dependency link. The label is displayed on the UML diagram between the client and the supplier.
stereotype	Use this field to add your own stereotype property.
supplier	This property field indicates the supplier for the link.
supplier role	Use supplier role to add a label to the dependency link. The label is displayed on the UML diagram towards the supplier side of the link.

Related Reference

Properties View

Generalization/Implementation Link Properties

This section describes the properties specific to generalization/implementation links. Every element has general properties as well as specific properties. For more information, see “Properties View.” The composition of the Properties view changes depending on the element selected in the Diagram or Model Navigator view. You can view and modify values of properties through the Properties view.

Item	Description
client	This property field indicates the client for the link.
supplier	This property field indicates the supplier for the link.

Related Reference

Properties View

Operation Context Menu

All of the UML diagram elements share common context menu commands. To use the context menu for an element, right-click on the element in the Diagram editor. To view the common context menu commands, see "Common Element Context Commands."

The context menu for an operation shares the common element context commands as well as the following commands specific to it:

Open

Selecting Open from the context menu opens the selected class containing the operation in the text editor.

Show in Packages View

The Show in Packages View command highlights the node selected in the Packages tree-view. The Packages view will expand and highlight that element in the tree-view. If closed, the Packages view will open.

Show in Model Package Explorer

The Show in Model Package Explorer View command highlights the node selected in the UML Explorer tree-view. The Model Package Explorer view will expand and highlight that element in the tree-view. If closed, this view will open.

Modifiers

The Modifiers command for the operation offers a submenu with the following options:

Static	Selecting Static from the context menu sets the static property for the operation.
Abstract	Selecting Abstract from the context menu sets the abstract property for the operation.
Public	Selecting Public from the submenu sets the visibility property for the operation to public.
Protected	Selecting Protected from the submenu sets the visibility property for the operation to protected.
Private	Selecting Private from the submenu sets the visibility property for the operation to private.
Package Local	Selecting Package Local from the submenu sets the visibility property for the operation to package local.

Note: The visibility options are not available for Interface members.

Add Javadoc comment

Using this command, you are able to add Javadoc comments for the operation.

Generate Sequence Diagram

Use this command to generate sequence diagrams from your source code. For more information, see "Working with Operations in Sequence/Collaboration Diagrams."

Related Procedures

Working with Operations in Sequence/Collaboration Diagrams
Roundtrip Engineering with Sequence Diagrams

Related Reference

Common Element Context Commands

UML 2.0 Use Case Diagrams

This section describes the elements of UML 2.0 Use Case Diagrams.

In This Section

UML 2.0 Use Case Diagram Elements

The table below lists the elements of UML 2.0 Use Case diagrams that are available using the Palette.

Name	Type
Actor	An actor node is a role (usually a person or thing, depicted by a stick figure) outside of the system that interacts with the system through a use case to achieve an observable goal. Use the **Show as** context menu to optionally change the actor notation to display as a rectangle instead of a stick figure. Between actors, only a generalization relationship can exist.
Subject	A subject node represents a system under consideration with which the actors and other subjects interact. The required behavior of the subject is described by the use cases. When a subject is created on a Use Case Diagram, a component is created in the namespace for the diagram canvas. Then after a use case is created on the subject, a new use case element is added to the subject's namespace and a relationship is formed between the use case and the subject.
Use Case	The use case node is the action or sequence of actions that actors engage in to yield an observable goal. It can be any element that displays behavior, including a component, subsystem, or class. A use case is defined according to the needs of the actor. Relationships between use cases can be either extend, include, or generalization. Besides the use case's name and brief description, elements that describe use cases include flow or scenarios, special requirements, pre- and post-conditions, and extension points. Use the **Show as ▸ Classifier/UseCase** context menu to optionally display a use case as a Classifier rectangle.
Generalization	If one use case or actor inherits all the behavior of another use case or actor and furthermore extends it with additional behavior, a generalization link results. The arrow points to the more general of the two.
Association	The interaction between an actor and a use case, represented by a solid line between them. These interaction links can be either unidirectional (indicating that an actor initiates the interaction) or bidirectional (indicating that an actor can participate in the interaction without initiating it). Association links can further be refined into multiplicities (how often the use case and actor interact), labels (roles specified at each end of the association), and direction (who initiates communication, although not necessarily a sequential flow of events).
Extend	If a certain condition is met at a specific extension point, a use case can be extended to another use case. This results in an extend relationship between the use cases. For example, whenever the Repair use case in the diagram above reaches the value specified by the Mechanics Verification extension point, it is extended by the Repair Dispenser use case. An extended use case does not have a dependency on the use case it extends to. Extend links are indicated by a dashed arrow pointing from the use case providing the extension to the base use case.
Include	If one use case includes a basic behavior that other use cases show, you can separate the common behavior out into another use case and establish an include relationship. Include use cases are required in order for the original use case to execute successfully. For example, in order for the Initiate Transaction use case in the diagram above to complete, the actor must be verified through the Authenticate use case. Include links are indicated by a dashed arrow pointing from the base use case to the included use case.

Dependency	Elements whose semantics depend on the definition of a supplier element are in a dependency relationship with the supplier element. Dependency links are shown as dashed arrows between the two model elements with the arrowhead pointing to the supplier element.
Constraint	A constraint is a Boolean expression that restricts the extension of an element. It restricts by imposing a value that specifies additional semantics beyond what is imposed by other language constructs applied to that element. The element that owns the constraint must have access to Constrained Elements.
Constraint Link	
Template Signature	A template signature contains the list of template parameters that are defined for a package.
Template Binding	A relationship between an element and a template that specifies the substitution of actual parameters for the formal parameters of the template.
Note	Use a note to optionally show relationship conditions between diagram elements. Note: This element is not a UML element.
Note Link	Use a note link to optionally show relationship conditions between diagram elements. Note: This element is not a UML element.

Extension Point

An **extension point** refers to a location within a use case where you can insert action sequences from other use cases.

An extension point consists of a unique name within a use case and a description of the location within the behavior of the use case.

In a use case diagram, extension points are listed in the use case with the heading "Extension Points" (appears as bold text in the Diagram View).

Related Reference

UML 1.4 Use Case Diagrams
UML 2.0 Use Case Diagrams

UML 2.0 Interaction Diagrams

This section describes the elements of UML 2.0 Communication and Sequence diagrams.

In This Section

UML 2.0 Sequence Diagram Elements

The table below lists the elements of UML 2.0 sequence diagrams that are available using the Palette. For more information on these palette elements, refer to the other UML 2.0 Interaction Diagram help topics in this reference section.

Name	Type
Lifeline	Draws an object with its lifeline in an interaction. For each lifeline, its projection bar displays on top of the diagram. When scrolling down, these projection bars are always visible.
Interaction	Draws an interaction node in diagram.
Message	Draws a message link between the source and target lifelines.
Found Execution	Draws a message link to a target lifeline. The source of such a message is unknown.
State Invariant	Draws a state invariant node on a lifeline.
Action Execution	Draws an action execution node on a lifeline.
Combined Fragment	Draws a combined fragment node on a lifeline.
Interaction Use	Draws an interaction use node on a lifeline.
Constraint	A constraint is a Boolean expression that restricts the extension of an element. It restricts by imposing a value that specifies additional semantics beyond what is imposed by other language constructs applied to that element. The element that owns the constraint must have access to Constrained Elements.
Constraint Link	Links a constraint to a diagram element.
Template Signature	Bundles the formal template parameters into a set for the templated diagram element that owns it.
Template Binding	A relationship between a diagram element and a template. The binding specifies template parameter substitutions.
Note	An annotation.
Note Link	An annotation link.

Warning: Sequence diagram can contain shortcuts to the other diagram elements. However, shortcuts to the elements that reside in the other interaction diagrams are not supported.

Note: Interaction diagrams, represented in the Model Navigator, display a number of auxiliary elements that are not visible in the Diagram Editor . These elements play a supplementary role for representation of the diagram structure. Actually, these elements are editable, but it is strongly advised to leave them untouched to preserve the integrity of the interaction diagrams.

Related Reference

UML 2.0 Interaction Diagrams

UML 2.0 Communication Diagram Elements

The table below lists the elements of UML 2.0 communication diagrams that are available using the Palette.

Name	Type
Lifeline	node
Interaction	node
Message	link
OCL constraint	node
Constraint link	link
Template Signature	node
Template Binding	link
Note	annotation
Note Link	annotation link

Note: Interaction diagrams, represented in the Model Navigator, display a number of auxiliary elements that are not visible in the Diagram Editor . These elements play a supplementary role for representation of the diagram structure. Actually, these elements are editable, but it is strongly advised to leave them untouched to preserve the integrity of the interaction diagrams.

Interaction

By using Together, you can create interactions for the detailed description and analysis of inter-process communications.

Interactions can be visually represented in your Together projects by means of the two most common interaction diagrams: Sequence and Communication. On the other hand, interactions can exist in projects without visual representation.

Interaction use

Within an interaction, you can refer to the other interactions described in your project. So called “Interaction use” elements serve this purpose. Note that a referenced interaction can be explicitly defined from the model or just specified as a text string.

Each interaction use is attached to its lifeline with a black dot. This dot is an individual diagram element. If an interaction use is expanded over several lifelines, you can delete the attachment dots from all lifelines but one. An interaction use should be connected with at least one lifeline.

Lifeline

A lifeline defines an individual participant of the interaction. A lifeline is shown in a sequence diagram as a rectangle followed by a vertical-dashed line.

Lifelines of an interaction can represent the parts defined in the class or composite structure diagrams. If the referenced element is multivalued, then the lifeline should have a selector that specifies which particular part is represented by this lifeline.

If a lifeline represents a connectable element, has type specified, or refers to another interaction, the Select menu becomes enabled on the context menu of this lifeline. Using this menu, you can navigate to the part, type or decomposition associated with the lifeline. These properties are defined by using the Properties View. If the **represents** property is set, the **type** and **name** properties are disabled.

You can define these properties manually by typing the values in the corresponding fields of the Properties View If the specified values are not found in the model, they are displayed in single quotes. Such references are not related to any actual elements and the Select menu is not available for them. If the specified values can be resolved in the model, they are shown without quotes, and the Select menu is available for them.

State invariant

A state invariant is a constraint placed on a lifeline. This constraint is evaluated at runtime prior to execution of the next execution specification. State invariants are represented in the interaction diagrams in two ways: as OCL expressions or as references to the state diagrams. You can use the state invariants to provide comments to your interaction diagrams and to connect interactions with states.

It is important to note that Together provides validation of the state invariants represented as OCL expressions. If the syntax is wrong, or there is no valid context, the constraint is displayed in red. For example, to be a valid context, a lifeline should have **type** and **represents** properties defined.

Related Concepts

About OCL Support in Together

Related Reference

UML 2.0 Interaction Diagrams

UML 2.0 Message

Call messages are always visible in diagrams; reply messages normally are not displayed. However, you can visualize the reply message.

Messages on different diagram types

- **Messages in communication diagrams:** When you draw a message between lifelines, a generic link line is displayed between the lifelines and a list of messages is created under it. The link line is present as long as there is at least one message between the lifelines.
- **Messages in sequence diagrams:** Messages in sequence diagrams have the same properties as those in communication diagrams but allow you to perform more actions. The further discussion refers mainly to the sequence diagram messages.

Properties of the messages for both types of interaction diagrams can be edited in the Properties View.

Properties of the message links

Call messages have the following properties:

Property	Description
Signature	Use this field to specify the name of an operation or signal associated with the message. Note that changing the signature of a message call results in changing the signature of the corresponding reply.
Sort	Use this field to select the type of synchronization from the drop-down list. The possible values are: **asynchCall, synchCall, asynchSignal**. The message link changes its appearance accordingly. There are certain limitations related to the asynchronous calls: Sometimes it is impossible to create or paste an asynchronous call because of the frame limitations. Execution specification for an asynchronous call must always be located on a lifeline.
Name	Displays the link name. This field is editable.
Full name	Displays the fully qualified link name. This field is not editable.
Visibility	Use this field to select the visibility modifier from the drop-down list.
Stereotype	Use this field to define the message stereotype. The stereotype name displays above the link.
Metaclass	This read-only field displays the message metaclass.
Label	Use this field to define the link label.
Attribute	Use this field to define the link attribute.
Arguments	Displays actual arguments of an operation associated with a message call. This field is editable.
Return value	Use this field to enter the return value.
Commentary	Use this textual field to enter comments for a message link.
Show reply message	Use this Boolean option to define whether to draw a dashed return arrow.
Sequence number	Use this field to view and edit the sequential number of a message. When the message number changes, the message call changes respectively.
no duration	Use this Boolean option to make the invocation and execution specifications invisible in diagram. This option can be only specified for the link that has no operation.
creation	Use this Boolean option to define creation message. If this option is true, the message link points to the lifeline object node.

destruction	Use this Boolean option to define a destruction message. If this option is true, the message link points to the execution specification marked with a cross sign.

Reply messages have the following properties:

Property	Description
Stereotype	Use this field to define the message stereotype.
Attribute	Use this field to define an attribute to which the return value of the message will be assigned. This field can be edited.
Signature	Use this field to specify the name of an operation or signal associated with the message. Note that changing the signature of a message reply results in changing the signature of the corresponding call.
Arguments	Displays arguments of an operation associated with a message call. This field can be edited. Note that changing the list of arguments of a reply message results in changing the corresponding call.
Return value	Displays the return value of an operation associated with a message link. This field can be edited.
Sort	Use this field to select the type of synchronization from the drop-down list. The possible values are:**asynchCall, synchCall, asynchSignal.** The message link changes its appearance accordingly.
Commentary	Use this text field to comment the link.

Note: Properties of the call and reply messages, such as arguments, attribute, qualified name, return value, signature, and sort pertain to the invocation specification. You can edit these properties in the invocation specification itself, in the call or in the reply messages. As a result, the corresponding properties of the counterpart message and the invocation specification will change accordingly. Stereotype and commentary properties are unique for the call and reply messages.

Related Procedures

Working with a UML 1.4 Message

Related Reference

Execution Specification and Invocation Specification
UML 2.0 Interaction Diagrams

Execution Specification and Invocation Specification

In sequence diagrams, Together automatically renders invocation specification and execution specification of a message that shows the period of time when the message is active. When you draw a message link from the source lifeline to the destination lifeline, the invocation and execution specification bars are created automatically. You can extend or reduce the period of time of a message by vertically dragging the top or bottom line of the invocation or execution specification as required.

For an invocation or execution specification, you can define the **no duration** property. If this property is checked for one specification, it will be automatically checked for the other one. Also, you can define this property for the message. If the **no duration** property is set to **true**, the specification icons reduce to the minimal possible dimensions and become invisible. By default, the execution specification is synchronized with the invocation specification. You can make the invocation specification and execution specification asynchronous.

It is also possible to create an execution specification on a lifeline without creating an incoming message link. In this case, a found message is created, which is a message that comes from an object that is not shown in the diagram. Use theProperties View to hide or show the found messages.

Messages in sequence diagrams have their origin in an invocation specification. This is an area within an execution specification. The notion of an invocation specification is introduced in Together's implementation of UML 2.0 sequence diagrams. Though this element is not defined in the UML 2.0 specification, it is a useful tool for modeling synchronous invocations with the reply messages. In particular, an invocation specification marks a place where the reply messages (even if they are invisible) enter the execution context of a lifeline, and where submessages can reenter the lifeline.

Active and passive areas of the execution specification are rendered in different colors. The white execution specification bars denote active areas where you can create message links. The gray bars are passive and are not a valid source or target for the message links.

Related Reference

UML 2.0 Message

Operator and Operand for a Combined Fragment

In this section, the following topics are discussed:

- About combined fragment
- Operator
- Operand

About combined fragment

A combined fragment can consist of one or more interaction operators and one or more interaction operands. The number of interaction operands (just one, or more than one) depends on the last interaction operator of this combined fragment.

Use the Palette or context menus to create these elements. The operator type shows up in the descriptor in the upper-left corner of the design element. Note that you can define multiple operators in a combined fragment. In this case, the descriptor contains the list of all operators, which is a shorthand for nested operators.

When an operator is created, add the allowed operands using the combined fragment's context menu.

A combined fragment can be expanded over several lifelines, detached from and reattached to lifelines. In the Properties View, use the **Operators** field to manage operators within the combined fragment.

Each combined fragment is attached to its lifeline with a mounting link that is displayed in the diagram as a black dot. This mounting link is an individual diagram element, which can be selected or deleted. Deleting a mounting link means detaching a combined fragment from the lifeline. Note that a combined fragment cannot be detached from all lifelines and should have at least one attachment dot.

You can reattach a combined fragment later using the anchor tool.

Operator

When a combined fragment is created, the operator displays in a descriptor pentagon in the upper-left corner of the frame. You can change the operator type using the **operator** field of the Properties View, which is immediately reflected in the descriptor.

The descriptor can contain several operators. The UML 2.0 specification provides this notation for the nested combined fragments. In Together, you can use this notation or create nested combined fragment nodes.

Operand

Operands are represented as rectangular areas within a combined fragment, separated by the dashed lines. When a combined fragment is initially created, the number of operands is defined by the pattern defaults. You can create additional operands or remove the existing ones.

Note that the uppermost area of the operator is empty and does not contain any operands. It is reserved for the descriptor. Clicking on this area selects the entire operator; clicking on one of the dotted rectangles selects the corresponding operand. If a combined fragment contains only one operand, the entire combined fragment and the single existing operand are still separately selectable.

Related Concepts

About OCL Support in Together

Related Reference

UML 2.0 Interaction Diagrams

Clipboard operations with execution and invocation specifications

Clipboard operations are supported for the execution and invocation specifications. Cut, Copy, and Paste commands are available on the context menu of an execution specification and invocation specification. It is possible to copy or move these elements within the same diagram or to another diagram.

When an execution or invocation specification is copied, it means that the entire branch of messages is copied also. Pasting the clipboard contents to a target lifeline results in changing the message numbers according to the numbering of messages in the target lifeline.

If you paste an invocation or execution specification to another diagram, the entire outgoing bunch of messages will be pasted also, with all the respective lifelines. If the target diagram does not contain lifelines for this execution specification, they will be created automatically.

It is also possible to move and copy message branches using the drag-and-drop technique. To move an execution or invocation specification, drag-and-drop it to the target location. To create a copy, drag-and-drop while holding the CTRL key down.

Related Concepts

Interaction (Sequence and Communication) Diagrams

UML 2.0 State Machine Diagrams

This section describes the elements of UML 2.0 State Machine Diagrams.

In This Section

UML 2.0 State Machine Diagram Elements

The table below lists the elements of UML 2.0 State Machine diagrams that are available using the Palette.

Name	Type
State Machine	node A state machine describes the behavior of a part of a system. A state machine owns one or more regions.
State	node A state models a situation during which some invariant condition holds.
Entry Point	node Execution of the state starts at this point. It is possible to create several entry points for one state, which makes sense if there are substates.
Exit Point	node Execution of the state finishes at this point. It is possible to create several exit points for one state, which makes sense if there are substates.
Initial	Node at which flow starts when the activity is invoked.
Final	node Signifies that the enclosing region is complete.
Terminate	node A pseudostate that, when activated, terminates the execution of the object that owns the state machine.
Shallow History	node A pseudostate that restores the most recent active substate of the containing state (that is, the configuration state that was active when the enclosing composite state last exited). A composite state cannot have more than one shallow history vertex.
Deep History	node A pseudostate that restores the most recent active configuration state that was active when the enclosing composite state last exited. A composite state cannot have more than one deep history vertex.
Region	node Use regions inside the states to group the substates. The regions may have different visibility settings and history elements. Each state has one region immediately after creation (though it can be deleted.) In the regions, you can create all the elements that are available for the State Machine diagram. This element is only available on the state context menu.
Fork	node A fork node splits one incoming flow into multiple outgoing concurrent flows.
Join	node A join node synchronizes multiple incoming flows into one outgoing flow.
Choice	node A pseudostate that performs a dynamic branch within a single transition.
Junction	node A pseudostate that connects transition segments into a single transition.

Transition	link Draws a link from the exit point of a source state (or the state without exit points) to the entry point of the destination (or the state without points).
Internal Transition	link Internal transition elements are only available on the state context menu.
Dependency	Elements whose semantics depend on the definition of a supplier element are in a dependency relationship with the supplier element. Dependency links are shown as dashed arrows between the two model elements with the arrowhead pointing to the supplier element.
Constraint	A constraint is a Boolean expression that restricts the extension of an element. It restricts by imposing a value that specifies additional semantics beyond what is imposed by other language constructs applied to that element. The element that owns the constraint must have access to Constrained Elements.
Constraint Link	Links a constraint to a diagram element.
Template Signature	Bundles the formal template parameters into a set for the templated diagram element that owns it.
Template Binding	A relationship between a diagram element and a template. The binding specifies template parameter substitutions.
Note	An annotation.
Note Link	An annotation link.

State Machine Diagram Context Commands

Diagram Editor ▸ Right-click

All of the UML 2.0 diagrams share common context menu commands. To use the context menu for a diagram, right-click in the Diagram Editor .

To view the common context menu commands, see "Common Diagram Context Commands."

The State Machine diagrams offer the following context commands.

Diagram Context Menu

Option	Description
New	The **New** command for the State Machine diagram offers a submenu with the following options: State Machine Constraint Note Shortcut

State Machine Context Menu

Option	Description
New	The **New** command for the State Machine element offers a submenu with the following options: Entry point Exit point Region

Region Context Menu

Option	Description
New	The **New** command for the Region element offers a submenu with the following options: State Initial Final Shallow history Deep history Terminate Fork Join Choice Junction Note

State Context Menu

Option	Description
New	The **New** command for the State element offers a submenu with the following options: Internal Transition Reference To Entry Point Reference To Exit Point Region

Related Reference

Common Diagram Context Commands

State Machine Diagram Elements Properties

This section describes the properties specific to State Machine diagram elements. Every element has common properties as well as specific properties. For more information, see “Properties View.” The composition of the Properties View changes depending on the element selected in the Diagram Editor or Model Navigator view. You can view and modify values of properties through the Properties View.

The following elements are included in state machine diagrams:

- State Machine element
- Region element
- Entry Point element
- Exit Point element
- Reference to Entry Point element
- Reference to Exit Point element
- State element
- Initial element
- Final element
- Shallow History element
- Deep History element
- Terminate element
- Fork element
- Join element
- Choice element
- Junction element
- Transition link
- Dependency link

State Machine element

The following properties are associated with the State Machine element:

- abstract
- context
- extends
- final
- full name
- is reentrant
- metaclass
- name
- parameters

- specification
- stereotype
- visibility

Region element

The following properties are associated with the Region element:

- full name
- metaclass
- name
- stereotype
- visibility

Entry Point element

The following properties are associated with the Entry Point element:

- full name
- kind
- metaclass
- name
- stereotype
- visibility

Exit Point element

The following properties are associated with the Exit Point element:

- full name
- kind
- metaclass
- name
- stereotype
- visibility

Reference to Entry Point element

The following properties are associated with the Reference to Entry Point element:

- entry
- full name
- metaclass
- name
- stereotype
- visibility

Reference to Exit Point element

The following properties are associated with the Reference to Exit Point element:

- exit
- full name
- metaclass
- name
- stereotype
- visibility

State element

The following properties are associated with the State element:

- do activity link
- entry activity link
- exit activity link
- full name
- is composite
- is orthogonal
- is simple
- is submachine state
- metaclass
- name
- stereotype
- submachine
- visibility

Initial element

The following properties are associated with the Initial element:

- full name
- kind
- metaclass
- name
- stereotype
- visibility

Final element

The following properties are associated with the Final element:

- do activity link
- entry activity link
- exit activity link
- full name
- is composite
- is orthogonal
- is simple
- is submachine state
- metaclass
- name
- stereotype
- submachine
- visibility

Shallow History element

The following properties are associated with the Shallow History element:

- full name
- kind
- metaclass
- name
- stereotype
- visibility

Deep History element

The following properties are associated with the Deep History element:

- full name
- kind
- metaclass
- name
- stereotype
- visibility

Terminate element

The following properties are associated with the Terminate element:

- full name
- kind
- metaclass
- name
- stereotype
- visibility

Fork element

The following properties are associated with the Fork element:

- full name
- kind
- metaclass
- name
- stereotype
- visibility

Join element

The following properties are associated with the Join element:

- full name
- kind
- metaclass
- name

- stereotype
- visibility

Choice element

The following properties are associated with the Choice element:

- full name
- kind
- metaclass
- name
- stereotype
- visibility

Junction element

The following properties are associated with the Junction element:

- full name
- kind
- metaclass
- name
- stereotype
- visibility

Transition link

The following properties are associated with the Transition link:

- client
- effect
- full name
- kind
- label
- metaclass
- name
- stereotype
- supplier
- trigger

- visibility

Dependency link

The following properties are associated with the Dependency link:

- client named element
- full name
- label
- metaclass
- name
- stereotype
- supplier named element
- visibility

Related Reference

Properties View

Transition

A single transition comes out of each state or activity, connecting it to the next state or activity.

A transition takes an operation from one state to another and represents the response to a particular event. You can connect states with transitions and create internal transitions within states.

Internal transition

An internal transition is a way to handle events without leaving a state (or activity) and dispatching its exit or entry actions. You can add an internal transition to a state or activity element.

Self-transition

A self-transition flow leaves the state dispatching any exit actions, then reenters the state dispatching any entry actions.

Guard expressions

All transitions, including internal ones, are provided with the guard conditions (logical expressions) that define whether this transition should be performed. You can associate a transition with an effect, which is an optional activity performed when the transition fires. The guard condition is enclosed in the brackets (for example, "`[false]`") and displayed near the transition link on a diagram. Effect activity is displayed next to the guard condition. You can define the guard condition and effect using the **Object Inspector Properties Window**.

Guard expressions (inside `[ ]`) label the transitions coming out of a branch. The hollow diamond indicates a branch, and its subsequent merge indicates the end of the branch.

Related Reference

UML 1.4 Statechart Diagrams
UML 1.4 Activity Diagrams
UML 2.0 State Machine Diagrams

History Element (State Machine Diagrams)

The Shallow History and Deep History elements are placed on regions of the states.

There may be none or one Deep History, and none or one Shallow History elements in each region. If there is only one history element in a region, it may be switched from the Deep to Shallow type by changing its `kind` property.

Refer to UML 2.0 Specification for more information.

Related Reference

UML 2.0 State Machine Diagrams

UML 2.0 Activity Diagrams

This section describes the elements of UML 2.0 Activity Diagrams.

In This Section

UML 2.0 Activity Diagram Elements

The table below lists the elements of UML 2.0 Activity diagrams that are available using the Palette.

Activity	Node. Activities are action states in an activity diagram. Action states are states with outgoing transitions that are triggered by the completion of an action associated with the state.
Activity Parameter	Node component. An activity parameter node is an object node for inputs and outputs to activities.
Activity Partition	An activity partition is a kind of activity group for identifying actions that have some characteristic in common.
Action	An action is an executable activity node.
Dependency	Elements whose semantics depend on the definition of a supplier element are in a dependency relationship with the supplier element. Dependency links are shown as dashed arrows between the two model elements with the arrowhead pointing to the supplier element.
Initial	Node at which flow starts when the activity is invoked.
Activity Final	Node at which a flow in an activity stops.
Decision	Node. A decision element indicates possible transitions relative to Boolean conditions of the owning object. The decision represents a branch in the control flow of an activity diagram.
Merge	Node that brings together multiple alternate flows.
Join Node	A join node synchronizes multiple incoming flows into one outgoing flow.
Fork Node	A fork node splits one incoming flow into multiple outgoing concurrent flows.
Flow Final	Node that terminates a flow.
Control Flow	Link that starts an activity node after the previous one is finished.
Input Pin	An object node that permits inputs to actions.
Output Pin	Pin that holds input values to be consumed by an action.
Value Pin	Input pin that provides a value to an action that does not come from an incoming object flow.
Object Node	Node that is a part of a defining object flow in an activity.
Central Buffer	An object node that manages flows from multiple sources and destinations. Unlike other buffers, central buffers are not attached to actions or activities. They help manage queuing and competing object flows.
Data Store	A central buffer node that stores all incoming tokens and distributes select copies for movement downstream.
Object Flow	An object flow relationship can be drawn: from an Activity to an Object, from SignalSending to an Object, from an Object to SignalReceipt, from/to an Object to/from a Fork/Join.
Accept Event Action	An action that waits for the occurrence of an event that meets a specified condition.
Accept Time Event Action	When a specified condition is not yet met, an accept time event action (displayed as an hour glass) waits until the condition is met before the action can accept it.
Send Signal Action	Send Signal Action is an explicit symbol used on an activity diagram for certain kinds of information that can be specified on transitions.
Constraint	A constraint is a Boolean expression that restricts the extension of an element. It restricts by imposing a value that specifies additional semantics beyond what is imposed by other language constructs applied to that element. The element that owns the constraint must have access to Constrained Elements.
Constraint Link	Links a constraint to a diagram element.
Template Signature	Bundles the formal template parameters into a set for the templated diagram element that owns it.

Template Binding	A relationship between a diagram element and a template. The binding specifies template parameter substitutions.
Note	An annotation.
Note Link	An annotation link.

UML 2.0 Activity Diagram Context Commands

All of the UML diagram types share common context menu commands. To use the context menu for a diagram, right-click in the Diagram editor.

To view the common context menu commands, see “Common Diagram Context Commands.”

Diagram Context Menu

Activity	Adds an activity element to the diagram.
Constraint	Adds a constraint element to the diagram.
Note	Adds a note element to the diagram.
Shortcut	To refer to an element located outside of the current diagram or to another diagram, you can use shortcuts. Invoking the **Shortcut** command displays a selection dialog, where you can choose an element (or diagram) from the appropriate location.

Activity Context menu

The activity element offers a special context command named **New** with a submenu for adding the following elements:

Activity parameter	Adds an activity parameter element to the activity.
Activity partition	Adds a partition to the activity.
Action	Adds an action element to the activity.
Initial	Adds an initial element to the activity.
Activity Final	Adds an activity final element to the activity.
Decision	Adds a decision element to the activity.
Merge	Adds a merge element to the activity.
Fork	Adds a fork element to the activity.
Join	Adds a join element to the activity.
Flow Final	Adds a flow final element to the activity.
Object Node	Adds an object node element to the activity.
Central Buffer	Adds a central buffer element to the activity.
Data Store	Adds a data store element to the activity.
Accept Event Action	Adds an accept event action element to the activity.
Accept Time Event Action	Adds an accept time event action element to the activity.
Send Signal Action	Adds a send signal action element to the activity.

Action Context Menu

The action element offers the **New** context command with a submenu for adding the following elements:

Input Pin	Adds Input Pin to the action.
Output Pin	Adds Output Pin to the action.
Value Pin	Adds Value Pin to the action.

Accept Event Action Context Menu

The Accept Event Action element offers the **New** context command with a submenu for adding the following element:

Output Pin	Adds Output Pin to the Accept Event Action.

Accept Time Event Action Context Menu

The Accept Event Action element offers the **New** context command with a submenu for adding the following element:

Output Pin	Adds Output Pin to the Accept Time Event Action.

Send Signal Action Context Menu

The Accept Event Action element offers the **New** context command with a submenu for adding the following elements:

Input Pin	Adds Input Pin to the Send Signal Action.
Output Pin	Adds Output Pin to the Send Signal Action.
Value Pin	Adds Value Pin to the Send Signal Action.

Related Concepts

UML Modeling Overview

Related Reference

Common Diagram Context Commands

UML 2.0 Component Diagrams

This section describes the elements of UML 2.0 Component diagrams.

In This Section

UML 2.0 Component Diagram Elements

The table below lists the elements of UML 2.0 component diagrams that are available using the Palette.

Component	node
Port	node
Artifact	node
Interface	node
Instance specification	node
Delegation connector	link
Provided interface	link
Required interface	link
Association	link
Aggregation	link
Realization	link
Note	annotation
Note link	annotation link

UML 2.0 Deployment Diagrams

This section describes the elements of UML 2.0 Deployment diagrams.

In This Section

Deployment Diagram Context Commands

Diagram Editor ▸ Right-click

All of the UML 2.0 diagrams share common context menu commands. To use the context menu for a diagram, right-click in the Diagram Editor .

To view the common context menu commands, see “Common Diagram Context Commands.”

The deployment diagram offers the following context commands.

Diagram Context Menu

New	The **New** command for the deployment diagram offers a submenu with the following options: Node Device Execution Environment Artifact Deployment Specification Constraint Note Shortcut

Node Context Menu

New	The **New** command for the Node element offers a submenu with the following options: Attribute Operation Node Device Execution environment

Device Context Menu

New	The **New** command for the Device element offers a submenu with the following options: Attribute Operation Node Device Execution environment

Execution Environment Context Menu

New	The **New** command for the Execution Environment element offers a submenu with the following options: Attribute Operation

Node
Device
Execution environment

Deployment Specification Context Menu

New	The **New** command for the Deployment Specification element offers a submenu with the following options: Attribute Operation Artifact Deployment Specification

Related Reference

Common Diagram Context Commands

UML 2.0 Deployment Diagram Elements

The table below lists the elements of UML 2.0 deployment diagrams that are available using the Palette.

Name	Type
Node	A node is a computational resource upon which artifacts can be deployed for execution. Nodes can be interconnected through communication paths to define network structures.
Device	Node that represents a physical computational resource with processing capability upon which artifacts can be deployed for execution. Devices can be complex, i.e. they can consist of other devices.
Execution Environment	Node that offers an execution environment for specific types of components that are deployed on it in the form of executable artifacts.
Artifact	node An artifact represents a physical entity and is depicted in a diagram as a rectangle with the `<<artifact>>` stereotype. An artifact can have properties, which define its features, and operations, which can be performed on its instances. Physically, the artifacts can be model files, source files, scripts, binary executable files, a table in a database system, a development deliverable, a word-processing document, or a mail message. A deployed artifact is one that has been deployed to a node used as a deployment target. Deployed artifacts are connected with the target node by deployment links. Artifacts can include operations. You can create complex artifacts by nesting artifact icons.
Deployment specification	node A deployment specification specifies a set of properties that determine execution parameters of a component artifact that is deployed on a node.
Deployment	link
Generalization	link
Association	link
Dependency	Link used to model general dependencies. In Deployment diagrams, this notation is used to depict the following metamodel associations: (i) the relationship between an Artifact and the model elements that it implements, and (ii) the deployment of an Artifact (instance) on a Node (instance).
Manifestation	Link. A manifestation is the concrete physical of one or more model elements by an artifact.
Communication path	Link. A communication path is an association between two Nodes, through which Nodes are able to exchange signals and messages.
Note	annotation
Note Link	annotation link

UML 2.0 Composite Structure Diagrams

This section describes the elements of UML 2.0 Composite Structure Diagrams.

In This Section

UML 2.0 Composite Structure Diagram Elements

The following is a list of UML 2.0 composite structure diagram elements.

Name	Type
Class	A classifier whose behavior is described through the interaction of its parts. Within a class you can specify attributes, operations, and other classes.
Interface	Creates a new object whose classifier conforms to a static classifier. The new object resides on the output pin at runtime and is returned as the value of the action. An interface must have at least one class to implement it.
Collaboration	Collaborating elements that perform specialized tasks and are structured collectively to accomplish a function. Collaborations show how a collection of cooperating classes achieve something.
Collaboration Use	Describes one use of a collaboration that is applied in a given context involving specific classes or instances playing the roles of the collaboration.
Dependency	Elements whose semantics depend on the definition of a supplier element are in a dependency relationship with the supplier element. Dependency links are shown as dashed arrows between the two model elements with the arrowhead pointing to the supplier element.
Part	A role played by one instance of a classifier or by a set of instances.
Referred Part	The instance of the classifier that is referenced.
Port	Available in design projects only. Each of the small squares attached to classes that connect the behavior of classes with their internal parts and with the other parts of the system. Ports can specify which service a class provides to its environment and which service a class expects from its environment.
Provided Interface	A provided interface represents services that are offered by instances of a classifier to fulfill contractual obligations.
Required Interface	A required interface specifies a usage dependency (which include the services needed to perform a required function) between instances of a classifier and their interfaces.
Connector	A link that lets two or more instances communicate with each other.
Collaboration Role	A link between instances that play the roles of the collaboration.
Role Binding	A dependency that maps between features of collaboration types and features of a classifier or operation. The mapping determines which connectable element of the classifier or operation plays which role in the collaboration.
Note	An annotation.
Note Link	An annotation link.

Data Modeling Reference

In This Section

ER Logical Diagram Elements

Project ▶ Properties ▶ Profiles ▶ ER Logical Diagram Profile

The table below lists diagram elements that become available after activating the ER Logical Diagram Profile in a UML 2.0 project.

Tip: To create second-level elements, use context menus of the top-level elements.

Element	Description
ER Entity	Creates an entity.
ER View	Creates a view.
Subtype Cluster	Creates a subtype cluster.
ER Relationship	Creates an association between two entities.
ER Many To Many Relationship	Creates an association with the stereotype `erManyToManyRelationship`.
ER View Relationship	Creates an association with the stereotype `erViewRelationship` between two top-level elements.
Cluster Link	Creates a link with the stereotype `erClusterLink` between a subtype cluster and an entity.

Related Concepts

Data Modeling

Related Procedures

Data Modeling Procedures

Related Reference

Element Context Menu Commands of ER Logical Diagram

ER Physical Diagram Elements

The list below describes diagram elements available for a ER Physical diagram in Together.

Element	Description
Table	Creates a physical table.
View	Creates a physical view.
Foreign key	Creates ER relationship between a child table (link source or client) and a parent table (link destination or supplier).
View Relationship	Creates a view relationship between a physical view (link client or source) and a physical table (link destination or supplier).

Related Concepts

Data Modeling

Related Procedures

Data Modeling Procedures

Element Context Menu Commands of ER Logical Diagram

Element ▸ Right click

To view the common context menu commands, see “Common Diagram Context Commands.”

Using the tools Palette, you can only create top-level elements. To create the second-level elements, use context menus of the container elements. The context menus described in this section are specific to the elements that are displayed in the class diagram when the ER Logical Diagram profile is activated.

Table

The Table element offers a special context command named **New** with a submenu for adding the following subelements:

Element	Description
ER Entity Attribute	Adds a stereotyped attribute (`erAttribute`) to an entity.
ER Entity PK Attribute	Adds a stereotyped attribute (`erAttribute`) to the Primary Key section, and a stereotyped (erKeyGroup) key group to the Key Group of an entity.
ER Entity Key Group	Adds a stereotyped (`erKeyGroup`) key group to the Key Group of an entity.
ER Entity Check	Adds a stereotyped (`erEntityCheck`) entity check to an entity.

View

The View element offers a special context command named **New** with a submenu for adding the following subelement:

Element	Description
ER View Expression	Adds a stereotyped attribute (`erViewAttribute`) to a view.

Related Concepts

Data Modeling

Related Procedures

Data Modeling Procedures

Related Reference

Common Diagram Context Commands
ER Logical Diagram Elements

ER Physical Diagram Context Commands

To use the context menu for a diagram, right-click in the Diagram Editor . To view the common context menu commands, see “Common Diagram Context Commands.”

The ER Physical diagram offers a special context command named **New** with a submenu for adding new elements to the ER Physical diagram:

New

The **New** command for the ER Physical diagram offers a submenu with the following options:

Option	Description
Table	Adds a Table element to the diagram.
View	Adds a View element to the diagram.

Related Concepts

Logical and Physical Data Models

Related Reference

Common Diagram Context Commands

Element Context Menu Commands of ER Physical Diagram

Element ▸ Right click

To view the common context menu commands, see “Common Diagram Context Commands.”

The context menus described in this section are specific to ER Physical diagram elements.

Table

The Table element offers a special context command named **New** with a submenu for adding the following subelements:

Element	Description
Column	Creates a Column within the Table element.
Primary Key Column	Creates a Primary Key Column within the Table element.
PK Constraint	Creates a PK Constraint within the Table element.
Unique Constraint	Creates a Unique Constraint element within the Table element.
Check Constraint	Creates a Check Constraint within the Table element.
Index	Creates an Index within the Table element.

View

The View element offers a special context command named **New** with a submenu for adding the following subelements:

Element	Description
Column	Creates a Column within the View element.
Expression	Creates an Expression within the View element.

Related Concepts

Data Modeling

Related Procedures

Data Modeling Procedures

Related Reference

Common Diagram Context Commands

Links Context Menu Commands of ER Physical Diagram

Link ▸ Right click

The context menus described in this section are specific to ER Physical diagram links.

The Foreign Key Link and View Relationship Link offer the following special context commands:

Command	Description
Propagate Attributes	For view relationships, propagates table columns to view. For foreign keys, propagates columns included in parent table constraint (selected as parent key for foreign key link using Inspector) to child table.
Propagate Attributes to All	For view relationships, propagates table columns to the entire view hierarchy.
Unpropagate Attributes	Undoes the applied propagation.

Related Concepts

Data Modeling

Related Procedures

Data Modeling Procedures

MDA

This section provides reference information related to MDA.

In This Section

QVT Language

Provides a description of QVT Language syntax and semantics supported by Together.

Introduction

This topic provides a description of QVT Language syntax and semantics supported by Together. The language implementation is based on an OMG document ptc/05-11-01 (MOF 2.0 Query/View/Transformation final adopted specification).

This implementation of QVT does not support declarative relations. Only imperative, or operational, transformations are supported.

Operational transformations

An operational transformation contains a series of methods (mappings or queries) and defines an entry point (`main` method) for the execution of the transformation. Invoking an operational transformation implies executing its entry point. All methods in a transformation are stateless, and must pass all of the required information via parameters and return values.

The following example demonstrates a simple transformation that contains only one mapping, `main`. This transformation creates an instance of an RDB model from an instance of a SimpleUML model.

```
transformation samples.Simpleuml_To_Rdb;
metamodel 'http:///SimpleUML.ecore';
metamodel 'http:///rdb.ecore';
mapping main(in model: simpleuml::Model): rdb::Model {
object {
name := model.name;
}
}
```

The first line identifies the transformation by assigning a fully qualified name, **samples.Simpleuml_To_Rdb**. and **samples** specifies a package that the transformation belongs to and must match the physical directory path. **Simpleuml_To_Rdb** sets the name of the transformation and must be equal to the filename of the transformation (without extension). Files that contain a transformation must have a **.qvt** extension. Therefore, the above transformation must reside in the **Simpleuml_To_Rdb.qvt** file in the **samples** folder. Transformations are looked up starting from the root folder, which is usually the root of the containing project.

The next two lines reference two EMF metamodels using the metamodel statement. The QVT implementation operates on EMF model instances, and the corresponding EMF metamodels must be declared to provide the necessary type information to the QVT compiler. The type of compliance as defined by the QVT specification is always strict.

The only method declared by this sample transformation is **main**, which takes a single input parameter model of type **simpleuml::Model** and returns an instance of **rdb::Model** as its **result**. It is created using the **object** expression, which assigns a value for a single feature of **rdb::Model -- name**.

Transformations can operate on any model elements defined by the referenced metamodel(s). The entry point of the transformation is a specially named method. Its arguments are not globally accessible, but might be passed to other methods as parameters.

There are two modes of transformation execution: interpreted and compiled. Interpreted mode implies direct evaluation of the transformation semantic tree. This mode is used by the QVT debugger. Compiled mode runs Java 984 code generated for the QVT sources. Generated Java code is stored in Eclipse plug-in project and can be deployed as an Eclipse plug-in.

Transformation methods

Transformation consists of a number of methods. There are two kinds of methods: mappings and queries. Mappings usually create new model elements and populate trace data. Queries represent lists of expressions that are executed one by one.

Mappings

A mapping operation consists of a signature, a guard (when clause), and a mapping body.

A signature defines an optional context type of the mapping, an optional parameter `list`, and the `return` type. A single unnamed `return` value is supported and can be accessed inside the transformation via the `result` keyword. The context type is accessed using the `self` keyword. The return type of the mapping is always a `model` type.

An optional `guard` specifies a Boolean condition. The mapping is only executed if the condition evaluates to `true`. If the condition is `false`, the mapping body is skipped and the `undefined` value is returned.

The mapping body consists of an optional `initialization` section, containing variable declarations and assignments, and the `population` section, which contains an object expression creating the mapping result.

The following `toRdb` mapping is defined in the context of `simpleuml::Model` and takes a single input parameter `prefix` of type `String`. The `guard` checks the name of the context value. The `initialization` section declares and initializes a `newName` variable. This variable is then used in the `object` expression in the `population` section.

```
mapping simpleuml::Model::toRdb(in prefix: String): rdb::Model
when {
        self.name <> ''
}
{
        init {
                var newName := prefix + self.name;
        }
        object {
                name := newName;
        }
}
```

The `toRdb` mapping should be called in the context of `simpleuml::Model`. It might be viewed as a new operation defined for this context.

```
init {
        var model := object simpleuml::Model {};
        var rdbModel := model.toRdb('rdb');
}
```

Mappings that do not specify the context are defined in the `OclVoid` context and can be called either without passing any context value or with `undefined` context:

```
init {
    var model := contextless();
    var model2 := undefined.contextless();
}

...
```

```
mapping contextless(): simpleuml::Model {
}
```

The `initialization` section can contain the following kinds of expressions: variable declarations, feature assignments and assignment to `result`.

A variable declaration has the form of `var name[: type] := expression;`. An optional type can be specified. If the type is not specified, the variable gets the type of the right-hand expression.

Feature assignments let you modify features of the input-output (`inout`) parameters of the method. For example, the following code adds a class to the `model`.

```
mapping addClass(inout model: simpleuml::Model): simpleuml::Model {
        init {
                model.ownedMembers += object simpleuml::Class {
name := 'Customer'; };
                result := model;
        }
}
```

The last example also uses the `result` assignment. This prevents execution of the section responsible for mapping population, and the assigned value is returned as the mapping `result` value.

Queries

A query is a special kind of method that consists of a signature, and a list of expressions (which forms its body).

A query's signature is identical to the one used for the mapping. The only difference is there are no restrictions for the query `return` type.

The body of the query consists of a list of semicolon-separated expressions. Expressions are executed one by one. The value of the last expression is returned as the result of the query.

The following sample query `getName` extracts the name attribute from the `model` parameter.

```
query getName(in model: simpleuml::Model): String {
        model.name
}
```

Queries can also be defined in a context of some type. For example, the following query returns the container package of `simpleuml::Class`:

```
query simpleuml::Class::getContainerPackage(): simpleuml::Package {
        self.owner
}
```

QVT language constructs

QVT uses OCL extensively, adding a "write" capability to this "read-only" language through a number of special constructs. These QVT-specific constructs are described below.

The QVT implementation allows to call Java methods from within QVT sources. The mechanism used is identical to the one used for calling Java methods from OCL code. It allows to view specially written methods of a Java class as QVT operations.

Consider a simple example of defining a new **dumpErr** operation with the following signature: **OclAny::dumpErr (in prefix: String): OclAny**. This operation prints the string representation of **self**, prepended by the prefix to a standard error, and returns **self** as the result.

Note: The context parameter self is passed as the first parameter of the dumpErr operation. For OclVoid context, this parameter is omitted.

First, we need a Java class implementing the operation:

```
package qvtlib;
public class SampleLibrary {
            public Object dumpErr(Object self, String prefix) {
                        System.err.println(prefix + self);
                        return self;
            }
        public static class Metainfo {
            private static final String[] DUMP_ERR = new String[] {
"OclAny", "String", "OclAny" };
            public static String[] dumpErr(Object self, String prefix)
{
                    return DUMP_ERR;
            }
        }
}
```

Note that the context parameter **self** is passed as the first parameter of the **dumpErr** operation. In case of **OclVoid** context, this parameter is omitted.

Static **Metainfo** class provides information on the signatures of the exported operations. Each exported operation should provide metainformation through the static method of this class. The method should have the same signature as the operation being described, and the **String[]** return type. The returned array should contain OCL type names and is interpreted as follows.

- **array[0]** -- context type
- **array[1]..array[array.length-2]** -- parameter type(s)
- **array[array.length-1]** -- return type

SampleLibrary class must reside in the Eclipse plug-in, which should also provide a **com.borland.tg.ocl.emf.libraries** extension:

```
<extension point=”com.borland.tg.ocl.emf.libraries”>
        <library class=”qvtlib.SampleLibrary”
id="qvtlib.SampleLibrary">
                <inMetamodel
uri="http://www.eclipse.org/emf/2002/Ecore"/>
            <outMetamodel
uri="http://www.eclipse.org/emf/2002/Ecore"/>
        </library>
</extension>
```

Extension specifies a library id (**qvtlib.SampleLibrary**), and, optionally, metamodels required by the library (**http://www.eclipse.org/emf/2002/Ecore** in our case).

When the plug-in is deployed and available to the Workbench, the **qvtlib.SampleLibrary** library becomes available to QVT transformations and OCL scripts. It can be used with any QVT transformation whose transformation project has a dependency on the library plug-in. For example:

```
transformation Ecore_To_Ecore;
import library qvtlib.SampleLibrary;

metamodel 'http://www.eclipse.org/emf/2002/Ecore';

query dump(in model: ecore::EPackage): OclAny {
            model.dumpErr('model: ');
}
```

The above **dumpErr** call will be converted to the following Java code:

```
new qvtlib.SampleLibrary().dumpErr(model, "model: "));
```

object expression

****

The `object` expression lets you create an instance of the specified model type and initialize its features. For example, the following expression creates an instance of `rdb::Table` and sets its `name` attribute:

```

object rdb::Table {
        name := 'Customers';
}
```

The expressions within the `object` construct must be in the form `feature := expression;` or `feature += expression;`. The latter assignment is only applicable to collection types where it adds the value of the expression in the right part to the feature in the left part.

The `object` expression can be used in any OCL expression, as well as in the population section of a mapping. In the latter case, you can omit the reference to a type, as the type is identical to the return type of the mapping:

```

mapping makeTable(): rdb::Table {
        object {
                name := 'Customer';
        }
}
```

You can further simplify the above mapping by removing the `object` keyword:

```

mapping makeTable(): rdb::Table {
        name := 'Customer';
}
```

Omitting the type of the `object` expression is not possible if the return type of the mapping is abstract. In this case, a non-abstract derived type should be specified for the `object` expression.

ObjectExp is implemented according to the QVT specification. The variant of object { ... } without a variable specification is not allowed (as this involves performing implicit, complicated, non-standard resolution), and if only a single result is involved, an explicit object expression cannot be used without the population keyword (according to section 8.2.1.19 of the QVT specification).

resolve operations

Execution of a mapping automatically creates trace records that map input parameters to the mapping result. Trace data is available at runtime via a family of `resolve` library functions:

- `OclAny::resolve(in type: OclType): Set(type)`:Returns all target objects of the type `type` that were produced by a source object specified by the context parameter.
- `OclAny::resolveByRule(in rule:Mapping, in type: OclType): Set(type)` Returns all target objects of the type `type` that were produced by a source object using the specified mapping method.
- `OclAny::invresolve(in type: OclType): Set(type)` Returns all source objects of type `type` that were used to produce the target object.
- `OclAny::invresolveByRule(in type: OclType): Set(type)`Returns all the source objects of the type `type` that were used to produce the target object using the specified mapping method.
- `OclAny::lateResolve(in type: OclType): type` Identical to `resolve`, but the computation is performed after executing the transformation entry point. This allows you to make "future" references to objects that will be created (and traced) later in the transformation.
- `OclAny::lateResolveByRule(in rule:Mapping, in type: OclType): type` Identical to `resolveByRule`, but the computation is performed after executing the transformation entry point.

The example below shows the mapping that creates `rdb::Tables` from `simpleuml::Classes`:

```
mapping makeTable(in cls: simpleuml::Class): rdb::Table {
        object {
                name := cls.name;
        }
}
```

After executing this mapping for a number of classes, it is possible to retrieve the tables produced for a given class using the following code:

```
query getCreatedTables(in cls: simpleuml::Class): Set(rdb::Table) {
        cls.resolve(rdb::Table)
}
```

inout parameters

QVT methods can take two kinds of parameters: input-only parameters (default), specified by the `in` keyword, and input-output parameters, specified by the `inout` keyword. The `inout` parameters can be modified by the mapping/ query code. For example:

```
mapping patch(inout model: simpleuml::Model): simpleuml::Model {
        init {
                model.name := 'New name';
                result := model
        }
}
```

The `patch` mapping above assigns a new name to its `model` parameter. Then it assigns `model` to the special `result` variable, which prevents the execution of the population section, and returns the patched `model` from the mapping. This is an example of an inplace transformation.

Output-only (`out`) parameters are not supported by the current QVT implementation.

Transformation import

QVT implementation supports reuse of transformations via transformation import. Importing the transformation has the effect of making all its methods available to the current transformation. For example:

```
transformation utils();

modeltype simpleuml uses "http://www.eclipse.org/qvt/1.0.0/Operational/examples/simpleuml";

query simpleuml::Model::getName(): String {
        self.name
}

transformation main;

transformation NewTransformation1(inout model : simpleuml, out model1 : rdb);

import utils;

modeltype simpleuml uses "http://www.eclipse.org/qvt/1.0.0/Operational/examples/simpleuml";
modeltype rdb uses "http://www.eclipse.org/qvt/1.0.0/Operational/examples/rdb";

mapping main(inout inoutModel : simpleuml::Model) : rdb::Model {
    name := inoutModel.getName();
}
```

Transformation `main` imports transformation `utils` and uses the `getName` method declared by the `utils` transformation.

When you import a transformation and the imported transformation contains methods with identical signatures, the method from the importing transformation is used.

Virtual methods

QVT implementation uses real type information when resolving context operation calls at runtime. This provides virtual behavior for context operations:

```
query simpleuml::ModelElement::getName(): String {
        self.name
}

query simpleuml::Class::getName(): String {
        'class ' + self.name
}

query test(in element: simpleuml::ModelElement): String {
        element.getName()
}
```

Given the above declarations, the `test(object simpleuml::Class { name := 'Customer'; })` call returns `'class Customer'`.

Query libraries

It is possible to define query libraries using the following QVT syntax.

```
library queries;
modeltype simpleuml uses "http://www.eclipse.org/qvt/1.0.0/Operational/examples/simpleuml";

query getName(in model: simpleuml::Model): String {
        model.name
}
```

The `library` keyword is used to declare a QVT library. Libraries can contain the same QVT code as QVT transformations but are registered as a special kind of QVT transformation, and they can be reused not only within QVT transformations but in pure OCL code as well.

Traceability

Execution of a mapping method automatically creates trace records that point from the method parameters to the mapping result. Trace data is available at runtime via the `resolve` functions family. This data can also be saved to a file for later analysis.

The stack trace is now available in both runtime mode and debug mode. Because stack trace elements can be constructed with no impact on memory or performance, the readability of runtime failures is improved. A runtime failure is directly mapped to the corresponding QVT stack in the source code so that meaningful QVT output, such as the following, can be generated:

```
org.eclipse.m2m.qvt.oml.internal.ast.evaluator.QvtRuntimeException:
java.lang.IllegalArgumentException: Cannot instantiate type EClassifier
        at EClass.mapCreateInstaceFailure3(stacktrace.qvto:33)
        at EClassifier.mapCreateInstaceFailure2(stacktrace.qvto:23)
        at EClassifier.mapCreateInstaceFailure(stacktrace.qvto:17)
        at stacktrace.main(stacktrace.qvto:43)
        at stacktrace.<init>(stacktrace.qvto:39)
```

OCL support

QVT implementation supports a full range of OCL 2.0 expressions.

Debugging support

QVT implementation provides an Eclipse debugger, supporting the following functionality:

- Line breakpoints.
- Standard stepping commands: step over, step into, step out, run to line.
- The QVT-specific **Variables View**, which displays QVT variables from the context of the current frame. This view also provides type information for OCL complex types, including the following features:
  - Mapping for structural (`isMany=true`) features to corresponding OCL collection types, including element type and size information
  - Support for enumerations
  - Proper handling of `OclInvalid` value and type (`OclInvalid` literal value of `Invalid` type)
  - Values for Declared type and Actual type provided where possible (except in cases in which values are not accessible, such as with iterator variables)
- The QVT-specific **Expressions View**, which evaluates an arbitrary QVT expression in the context of the current frame.

QVT also provides the following library function: `OclVoid::dump(in o: OclAny): OclVoid`. This function prints the string representation of the `o` parameter to standard output, which is redirected to the **Console View** when transformation is executed in the Workbench.

Renaming features

In case a model element's feature name conflicts with a QVT keyword, contains spaces, or is otherwise invalid from the QVT point of view, it can be renamed using a special **rename** directive. The format of the directive is:

```
rename <type> = 'old_name'
```

This directive should be placed immediately after the metamodel declaration statements.

The sample QVT below uses the **rename** directive to assign a new name to the "name" feature of `ecore::Epackage`.

```
transformation EPackage_To_EPackage;

modeltype ecore uses "http://www.eclipse.org/emf/2002/Ecore";
rename ecore::EPackage.newName = 'name';

mapping main(in model: ecore::EPackage): ecore::EPackage {
    newName := 'NewPackage';
}
```

Note: In the current implementation, changing names of the model types and packages is not supported.

Appendix

This QVT implementation does not fully support the draft QVT specification as given by the ptc/05-11-01 OMG document. Incompatibilities include:

- No model parameters are supported. Transformation entry point is just a specifically named method (`main`). Parameters of an entry point must be explicitly passed to the methods that need to access them.
- Multiple return values are not supported. Each mapping should have a single return value, accessible in code via implicit `result` variable. The `main` mapping should have exactly one input or input-output parameter.
- Limited support for return statements in queries is provided. Support is provided for return statements that are used only with the last expression in the query body. The compiler produces a warning for a missing return that is used for the last expression in the query body, but the engine runs successfully.
- Output parameters are not supported.
- Input-output parameters are supported with the following semantics: they must be created by the caller, but it is possible to modify their features inside the method.
- Syntax for mapping method calls does not include the `map` keyword. Methods are invoked in the same way as ordinary methods, like operations defined in the model.
- The `where` clause (postcondition) is not supported.
- The `constructor` operations are not supported.
- Intermediate classes and properties are not supported.
- Supported mapping operation reuse mechanisms include *inherits*, *merges*, and *disjuncts*. Relation refinement is not supported because hybrid (mixed with QVT relational) parse/execution is not implemented.

- Dynamic instantiation and invocation of transformations is not supported.
- `typedefs` are not supported.
- Exceptions are not supported.

Related Concepts

Model Transformation Support

Related Procedures

Creating Model-To-Text Transformations
Creating a Model-To-Model Transformation

Related Reference

QVT Editor
EMF API for Together Profiles

QVTO Language

Provides a description of M2M.QVTO Language syntax and semantics. The language implementation is based on an OMG formal/08-04-03 QVT Specification.

Related Help

Please refer the following Help book **Help ▸ Help Contents ▸ QVT Operational Developer Guide**

Related Concepts

Model Transformation Support

Related Procedures

Manually Registering a Metamodel for Use with QVTO
Running an Operational QVT

Related Reference

QVT Operational Ant Tasks
QVT Operational Migration Notes
QVT Operational Imperative Iterators
QVT Operational Transformation Wizard Configuration Properties
QVTO/OCL Collections and Operations

XSL/OCL Language

Provides a description of XSL/OCL Language syntax and semantics supported by Together.

Introduction

This is an introduction into Extensible Stylesheet Language Transformations (XSLT) provided by Together. The language implementation is based on XSL Transformations (XSLT) Version 2.0 (W3C Candidate Recommendation 8 June 2006).

This implementation of XSLT is not used in conjunction with XPath 2.0. It uses the Object Constraint Language (OCL) as described in the UML 2.0 OCL Specification (OMG Final Adopted Specification ptc/03-10-14).

XSL transformations

An XSL transformation consists of a series of declarations, instructions, and result elements/text. It is organized into `xsl:template` elements that can be compared to procedures in other languages.

Invoking an XSL transformation executes one or more of the available templates. An example below demonstrates a simple transformation containing only one template. This transformation creates HTML output, listing the names of packages passed in as the source.

```
<xml version="1.0" encoding="UTF-8"?>
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0"
        xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
        xmlns:xta="http://www.borland.com/xta">
    <xta:metamodel uri="http://www.borland.com/together/uml"/>
    <xta:metamodel uri="http://www.borland.com/together/uml20"/>
    <xsl:output method="html"/>
    <xsl:template match="self.oclIsKindOf(uml::kernel::packages::Package)">
        <HTML>
            <HEADER>
            </HEADER>
            <BODY>
                <H1><xsl:value-of select="self.name"/></H1>
            </BODY>
        </HTML>
    </xsl:template>
</xsl:stylesheet>
```

Every stylesheet consists of one main element `xsl:stylesheet` or the synonymous `xsl:transform`. The most common XSLT namespaces, auxiliary instructions and result elements are defined within this element. The above example associates the URI `http://www.w3.org/1999/XSL/Transform` with the prefix `xsl` for XSLT and the URI `http://www.borland.com/xta` with a prefix `xta` for OCL add-ons.

To instruct the transformer to produce a valid HTML text, the `xsl:output` instruction has set its method attribute to `html`. This way, the resulting text will not have typical XML additions (like the XML processing instruction in the first line), which are not a part of HTML specs, and therefore, may startle HTML renderers.

The next two `xta:metamodel` child elements introduce the UML and UML 2.0 metamodels (referenced by their URIs `http://www.borland.com/together/uml` and `http://www.borland.com/together/uml20`) to the OCL engine.

The main entry point is an `<xsl:template>` element. Its match condition states that it can only be applied to objects currently referenced by the `self` variable in the OCL runtime context and which are directly or indirectly of the type `uml::kernel::packages::Package`. If the transformation source references a package like `together:/`

`Simple UML Model#model:project::Simple UML Model` that references a project (which is of the required kind), the match clause is evaluated to Boolean `true` and child elements of the template are executed.

The children of the template are a combination of result elements and XSLT elements—a combination of content and logic. In this case the result is HTML and the XSLT elements contribute text. The `xsl:value-of` element is not written to the output like the result elements `<HTML>` or `<BODY>`. The OCL expression in its `select` attribute is evaluated and the outcome of the evaluation written to the result.

In the above example, this would be the package name surrounded by header tags: `<H1>Simple UML Model</H1>`

Transformation Source

Because OCL is used instead of XPath, a different type of transformation source is required. The XML source is replaced by an ECore source, bringing model elements referenced by their URIs into the transformer. A `URI together:/Simple UML Model#model:project::Simple UML Model` enters the `Simple UML Model` project into a transformation.

Transformation Stylesheet

Stylesheets are a combination of content and logic. They combine content from the source with possible but not necessary content in the stylesheet (result elements and text) to some new result. The logic (XSLT) elements steer how the contents are combined.

Embedding stylesheets lets you reuse templates defined in other stylesheets. `<xsl:include href="someOtherStylesheet.xsl">` inserts the contents of the referenced stylesheet as if they were defined in the including stylesheet. `<xsl:import href="someOtherStylesheet.xsl">` uses the references stylesheet more in a library-like manner, introducing object-oriented concepts, like overriding. Both versions prohibit the direct or indirect embedding of a stylesheet into itself.

Transformation Result

The result of a transformation is stored generally in a file within the workspace.

Multiple results can be created by specifying relative or absolute URLs in the `href` attribute of an `xsl:result-document` directive. All results created by their child elements are redirected to the new output location. That also applies to results created by invoked templates. Result-document elements can be cascaded so it is possible to redirect results from within another redirection. The end-tag restores the result destination as it was before the start-tag.

Transformation Parameters

Pairs of string keys and values passed into a transformation are available within the stylesheet as global parameters. Those parameters need to be defined using `<xsl:param name="someParamName">` within the stylesheet to be able to access the passed-in values using for instance `<xsl:value-of select="someParamName">`. A transformation fails with an error if a `param` instruction with an attribute `required="yes"` exists and no key-value pair with the value of its name attribute was passed in.

Transformation Invocation

Without any further invocation settings, the best matching template is applied to the source element(s). A special case is when the mode attribute value of a template is specified before the transformation. Only templates with a fitting mode will be taken into consideration for application. If a named template should be called (similar to an entry point procedure in other languages), its name can be set for the transformation.

OCL

Utilizing XSL for model-to-text transformations predestines the usage of OCL in place of XPath to work directly against the model input. This eliminates intermediate layers to convert model elements into XML elements with the drawbacks of memory overhead, added processing cycles, and disconnecting the logic from the model.

Auto-context and stereotype

OCL constructs consist of three major parts. The context, stereotypes, and the actual expressions:

```
context Person::income : Integer -- context
init: parents.income->sum() * 1% -- 1st stereotype and expression
derive: if underAge -- 2nd stereotype and its expressions
then parents.income->sum() * 1% else job.salary
endif
```

Providing complete OCL statements would create considerable clutter in a stylesheet. In most cases, explicit naming of the context and stereotype is not necessary because they can be derived from the element in focus in the OCL runtime environment. If the `self` variable points to the Simple UML Model from the example above, the context can be derived from its type and set to: context `uml::together::Model`. The stereotype used for OCL attributes and embedded OCL is `inv`: so it can also be suppressed. By automatically computing context end stereotype, an element like `<xsl:value-of select="context uml::together::Model inv: self.name"/>` can be written much shorter as `<xsl:value-of select="self.name"/>`. Nonetheless, the full syntax is still supported to fine-tune OCL statements.

Embedded OCL

Attribute values are generally considered to have simple contents such as `xs:string`. Often, stylesheet writers want to access for instance the values of variables in the OCL runtime environment. This is possible by using embedded OCL. Embedded OCL is an OCL expression surrounded by braces `{` and `}`. Attributes with embedded OCL are converted at execution time into the combination of their literal parts and the results of the evaluations of the embedded OCL expressions. As an example, an attribute `att="type {self.type.name} - name {self.name}"` becomes `att="type SomeTypeName - name someName"`, assuming the context object was named "someName" and its type was named "SomeTypeName".

OCL attributes

Attributes of instructions that typically deal with OCL do not require embedded OCL. They presume that their values are a valid OCL expression. Meaningful conditional instructions need to test against non-literal values. Therefore, the condition of a simple conditional instruction is stated like `<xsl:if test="self.oclIsKindOf (SomeType)">`. To use literal strings within OCL attributes, they need to be in single quotes `<xsl:value-of select="'some string'">`.

Special characters

There are characters permitted in the OCL language, like comparative operators `<` or `=>` that collide with XML well-formedness. These characters need to be replaced by entity references such as `&lt;` or `=&gt;`, respectively. They are substituted by their referenced entities at parsing time of the stylesheet and the OCL statements will conform to the correct syntax.

Extension instructions

The OCL processor needs to be informed about certain things like which metamodels to expect or which libraries to use. The transformer implementation provides extensions that transcend XSLT and accommodate the OCL layer. `<xta:metamodel uri="someMetamodelURI">` makes a metamodel known to the OCL engine so that its types can be used in expressions. `<xta:import-library id="someLibraryID">` loads an OCL library in the OCL processor and allows the operations defined in there to be used in expressions.

Templates

Templates are the means of breaking down a transformation into smaller reusable chunks. There are two kinds of templates: named templates that can be called directly, and templates whose applicability is decided at runtime evaluating a match attribute clause.

Matches

`match` attributes are OCL expressions that determine whether the template element can be applied to the self object of the OCL runtime environment. Applicability is decided by passing the result of the evaluation to the `boolean()` function. Therefore, the OCL expression does not need to deliver a Boolean result by itself. It can return strings or numbers, which are then interpreted for their Boolean nature.

Undefined	always false
String	false if undefined or empty, else true
Number	false if undefined, 0, or 0.0, else true
Collection	false if undefined or empty, else true
Model element	false if undefined, else true
Type	always true

A check for an empty name can hence be as brief as `<xsl:template match="name">` and does not need a complex Boolean expression like `<xsl:template match="not name.oclIsUndefined() and name->size() > 0">`.

Parameters

Templates can be defining parameters that can or must be passed in at invocation by providing child elements of the type `xsl:param`. Parameters setting their attribute `required="yes"` force their invokers to provide these parameters or an error is reported. Since XSLT 2.0, parameters can be quietly propagated down an invocation hierarchy without the constant need of redefinition in templates by setting an attribute `tunnel="yes"`. Tunneling parameters are available in templates that define them even if there were templates invoked in between that do not define those parameters.

Template invocation

Templates can be called directly by name or applied using various strategies of determining which one to apply.

Calling

Templates in the form `<xsl:template name="someName" ...>` can be called directly by elements `<xsl:call-template name="someName">`. To avoid ambiguity, only one template with a given name can exist at a time with the same import precedence or an error stops the transformation. That is, a template of the same name can exist very well in an imported stylesheet and the importing stylesheet as the imported template would assume a lower precedence and hence not be called, allowing stylesheets to redefine imported behavior.

An important difference to template application is that the self object in the OCL runtime is the same. Caller and callee operate on the same object.

Applying

The most basic form of template application is `<xsl:apply-templates>`. This form finds the most applicable template whether it is imported or not and passes the child elements of the current self object in the OCL runtime environment as new self objects to the template **applied to each child**. Since the children could vary in their type and properties, different templates can be applied to each and every one of them.

That said, a template could basically apply itself recursively but doing that to the child elements of the self object rather than endlessly applying itself to the same object all over. The self object passed to the template can be overridden by using a `select` attribute OCL expression determining something other than the children of the current self.

Even if there are more templates to apply, **only one** is applied, actually. The candidate is determined by priority computations and precedence considerations.

Use the `xsl:next-match` element if you need to apply a similar template to the one currently applied. It enables templates to apply templates that they may have replaced in the match order in the first run. A **next match** is a template that would have been applied if the current one had not taken higher priority or precedence. This concept is related to the call to a super-implementation of virtual operations.

Another way of invoking a template that would otherwise be overridden by one with a higher import-precedence is `xsl:apply-imports`. This form of application will disregard non-imported templates and only consider match clauses, priory, and precedence within **imported templates**. This is also somewhat similar to virtuality in object-oriented development.

Priority

Priority of a template match is either determined by how exact or vague the criteria applies to the object or by setting it explicitly using the `priority` attribute and a decimal value, which can be negative.

If there is more than one matching template with the same priority, precedence takes over when deciding which template to apply.

Precedence

Precedence is determined by the order in which templates are defined. The farther down a template exists in a stylesheet, regardless of whether it was embedded, the higher its precedence is.

Mode

While priority and precedence are well-suited to control template traffic, they do not suffice to categorize templates. Categorizing templates provides a higher level of control to limit the set of templates coming into question for applicability. It reduces the amount of actual match evaluations and provides certain groups of functionality that do not interfere with each other. When specifying a mode attribute in an `xsl:template`, all `xsl:apply-templates`, `xsl:apply-imports`, and `xsl:next-match` will need to specify that very same attribute and value to be able to target the thereby marked templates.

With parameters

The counterpart to `xsl:param` elements within an `xsl:template` element are `xsl:with-param` elements within `xsl:call-template`, `xsl:apply-templates`, `xsl:apply-imports`, and `xsl:next-match` elements. The invoking element needs to provide parameters that were declared as required in their owning templates or the transformation is halted with an error. The tunneling parameter can be passed to potential recipients even if other templates are invoked before them in the hierarchy that do not define those parameters.

Constructing results

Style sheets let you create textual results in various means, from a plain text to elements composed at runtime.

Text

If the text to be added to the result is fairly simple, it can be placed at any position in a template that allows result content.

```
<?xml version="1.0" encoding="UTF-8"?>
    <xsl:stylesheet version="2.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="text"/>
        <xsl:template match="true">The quick brown fox jumps over the lazy dog.
        </xsl:template>
</xsl:stylesheet>
```

There may be cases in which preserving the line separation and avoiding replacement of special characters by entity references is necessary. Such texts can be enclosed in `xsl:text` elements with an attribute `disable-output-escaping="yes"` to avoid any conversion or reformatting. Also, if the content contains special characters, a `CDATA` section can be used so you do not have use entity references for them to conform to XML. This is useful for creating a JavaScript or source code of other languages.

```
<?xml version="1.0" encoding="UTF-8"?> <xsl:stylesheet version="2.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="text"/>
        <xsl:template match="true">
            <xsl:text disable-output-escaping="yes"><![CDATA[function matchwo(a,b) {
        if (a < b && a < 0) then {
            return 1
        } else {
            return 0
    } }]]></xsl:text>
        </xsl:template>
</xsl:stylesheet>
```

Elements and attributes

Elements that are not of the XSLT namespace or of an extension namespace are written to the result. If the result elements reside in a namespace, the latter has to be declared before or at the first result element. If the **result namespace** is the XSLT namespace, you can declare a namespace alias to avoid conflicts and still get a result with the right namespace and prefix.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0"
    xmlns:wxsl="http://www.w3schools.com/w3style.xsl"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:xta="http://www.borland.com/xta">
    <xta:metamodel uri="http://www.eclipse.org/emf/2002/Ecore"/>
    <xsl:namespace-alias stylesheet-prefix="wxsl" result-prefix="xsl"/>
        <xsl:template match="self.oclIsKindOf(ecore::EPackage)">
        <wxsl:stylesheet>
            <content/>
        </wxsl:stylesheet>
    </xsl:template>
</xsl:stylesheet>
```

If the name of an element can just be determined at runtime, the `xsl:element` instruction can be used. It has a `name` attribute that permits embedded OCL for runtime composition of the element name. In the example below, an element is dynamically composed from a prefix defaulting to `my` and a suffix that must be provided by the caller of the template.

```
<?xml version="1.0" encoding="UTF-8"?>
    <xsl:stylesheet version="2.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template name="createNamedElement">
        <xsl:param name="elementNamePrefix">my</xsl:param>
        <xsl:param name="elementNameSuffix" required="yes"/>
        <xsl:element name="{elementNamePrefix}-{elementNameSuffix}">
            <content/>
        </xsl:element>
    </xsl:template>
</xsl:stylesheet>
```

Not only elements can be constructed dynamically but also attributes. It does not matter if the element is a literal or using the XSLT instruction to compose elements dynamically. Dynamic attributes can even be grouped to reusable sets that can be applied repeatedly and in various locations, using the set name as reference in an `xsl:use-attribute-sets` attribute. The following example shows a callable template that creates an HTML paragraph element `P` having its `CLASS` attribute filled by an attribute set and allows overriding the value of one of its attribute name and value pairs by passed-in parameters that default to `ALIGN` and `LEFT`.

```
<?xml version="1.0" encoding="UTF-8"?> <xsl:stylesheet version="2.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:attribute-set name="paragraph">
        <xsl:attribute name="CLASS">MyParagraphs</xsl:attribute>
    </xsl:attribute-set>
    <xsl:template name="newParagraph">
        <xsl:param name="attName">ALIGN</xsl:param>
        <xsl:param name="attVal">LEFT</xsl:param>
        <P xsl:use-attribute-sets="paragraph">
            <xsl:attribute name="{attName}">
                <xsl:value-of select="attValue"/>
            </xsl:attribute>
            <xsl:apply-templates/>
        </P>
    </xsl:template>
</xsl:stylesheet>
```

Values

The `xsl:value-of` instruction is the most widely used one to contribute result content from model elements. A `select` attribute value being of OCL nature queries the model and returns the result as text. Alternatively, it can return contained text and elements as text. Similar to the `xsl:text` instruction, it supports the attribute `disable-output-escaping="yes"` to avoid conversion of special characters that are prohibited for content in XML to entity references.

Because of the differences from XPath and OCL, `xsl:copy-of` works differently as it cannot work against an XML node set and copy that one over to the result XML node set. Instead, it copies the XMI-form of the selected model element and its child elements into the result. This provides a very powerful yet simple way of serializing models and model parts into the most common storage format.

Controlling

In many cases the result to be created will depend on certain conditions. One may want to create different elements depending on the value of certain properties of model elements. XSLT provides various forms of conditional execution of stylesheet elements. The most versatile is the `xsl:chose` instruction with its `xsl:when` and xsl:otherwise child instructions. The next stylesheet excerpt depicts how an attribute can decide between two different values based on a property of a model element.

```
<xsl:attribute name="TITLE">
    <xsl:choose>
        <xsl:when test="self.getPropertyValue('$interface').oclIsUndefined()">
            <xsl:text>class in </xsl:text>
        </xsl:when>
        <xsl:otherwise>
            <xsl:text>interface in </xsl:text>
        </xsl:otherwise>
    </xsl:choose>
    <xsl:value-of select="self.owner.getPropertyValue('$fullName')"/>
</xsl:attribute>
```

For simple yes/no-assessments, the xsl:if instruction exists. The shown stylesheet snippet asks if a given UML 1.4 class instance has generalizations, and it executes the child elements of the conditional element only in such a case.

```
<xsl:template match="self.oclIsKindOf(uml14::kernel::classes::Class)"mode="class-tools-
extends">
    <xsl:param name="relativePathOfPackageRoot"/>
    <xsl:if test="self.generalizations->notEmpty()">
       <xsl:apply-templates select="self.generalizations"                mode="class-tools-
extends-generalizations">
            <xsl:with-param name="relativePathOfPackageRoot">
                <xsl:value-of select="relativePathOfPackageRoot"/>
            </xsl:with-param>
    </xsl:apply-templates>     </xsl:if> </xsl:template>
```

Iterating

xsl:apply-templates and its variations can be considered for iterating child elements and other selections. Sometimes it is difficult to separate a loop execution part into a template. The more simple xsl:for-each instruction helps iterating any collection and processing the contained objects; for example, the following stylesheet fragment iterates attributes of a class and puts their names and type names into an HTML table.

```
<xsl:template match="oclIsKindOf(ecore::EClass)">
    <table border="1" width="100%">
        <xsl:for-each select="eAttributes">
            <tr>
                <td width="20%">
                    <xsl:value-of select="name"/>
                </td>
                <td>
                    <xsl:value-of select="eType.name"/>
                </td>
            </tr>
        </xsl:for-each>
    </table>
</xsl:template>
```

Sorting

All iterating instructions like xsl:for-each or xsl:apply-templates can work against sorted data. xsl:sort instructions can be placed directly within looping instructions and thereby multiple sorting criteria can be implemented. If the sorting instruction is used without any attributes, the sorted object is converted to a text and this text is submitted to alphabetical ascending sorting. Attributions of sorted elements can be used to refine the sorting. Other attributes allow changing the order or the comparison method. The sorting illustrated below uses the name of the child elements other templates should be applied to as sorting criteria. It sorts them in reverse order based on textual comparison and determines an Austrian character set to be taken as the basis for the character sequence, placing Ä after A, Ö after O, Ü after U, ä after a, ö after o, ü after u, and ß after s.

```
<xsl:template match="self.oclIsKindOf(ecore::EPackage)">
    <xsl:param name="indent" select="'    '"/>
    <xsl:apply-templates>
        <xsl:sort select="self.name" data-type="text" lang="de-AU" order="descending"/>
        <xsl:with-param name="indent" select="indent + '    '"/>
    </xsl:apply-templates>
</xsl:template>
```

Numbering

Tables of contents and other directory-type document structures call for numbering. The `xsl:number` instruction can add numbering to single document entities as much as to repetitive entries.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:xta="http://www.borland.com/xta">
    <xsl:output method="text"/>
    <xta:metamodel uri="http://www.eclipse.org/emf/2002/Ecore"/>
    <xsl:template match="self.oclIsKindOf(ecore::EPackage)">
        <xsl:number format="1. " level="multiple"/>
        <xsl:text>ecore::EPackage:</xsl:text>
        <xsl:value-of select="self.name"/>
        <xsl:text>
        </xsl:text>
        <xsl:apply-templates/>
    </xsl:template>
    <xsl:template match="self.oclIsKindOf(ecore::EClass)">
        <xsl:number format="1.1. " level="multiple"
            count="self.oclIsKindOf(ecore::EPackage) or self.oclIsKindOf(ecore::EClass)"/>
        <xsl:text>ecore::EClass: </xsl:text>
        <xsl:value-of select="self.name"/>
        <xsl:text>
</xsl:text>
    </xsl:template>
</xsl:stylesheet>
```

The above stylesheet uses numbering in its most common form. It creates a hierarchical table of packages and their owned classes as shown in the next lines.

```
1. ecore::EPackage: epo3
1.1. ecore::EClass: Item
1.2. ecore::EClass: USAddress
1.3. ecore::EClass: PurchaseOrder
1.4. ecore::EClass: GlobalAddress
1.5. ecore::EClass: Address
```

The number instructions specify in a format attribute the way in which the numbers should be written. For packages it only uses a single number-period combination, while for classes it uses two of those to provide numbers for both the owning package and the owned classes.

In addition to that, level attributes set to multiple are supplied to enforce the number-period tuples for the classes. Note that the second numbering instruction also states a selecting count clause that makes sure that the internal grouping does not discard stored package information and thereby suppresses the leading number-period for the owning package. Numbering is very powerful and thus complex—you may need to experiment with it and read the "Numbering" chapter of the XSLT 2.0 specification.

Tracing

Even with XSL debugging, it is often useful to report to the console during stylesheet processing or even stop the process in case of inconsistent data. The `xsl:message` instruction can both print to the console or exit processing if it has an attribute `terminate="yes"`.

```
<xsl:template match="self.oclIsKindOf(uml14::kernel::classes::Class)">
<xsl:if test="verbose"><xsl:message>Processing UML14 class: <xsl:value-of
select="self.fullName"/><xsl:text>
</xsl:text></xsl:message></xsl:if>
```

Advanced techniques

Beautifying output

Especially when creating output that is to be read by people again, you should instruct the XSL transformer to format the result to comply to common standards—indentation is one of them.

```
<xsl:stylesheet version="2.0"
    xmlns:xalan="http://xml.apache.org/xslt">
    <xsl:output method="html" encoding="UTF-8" indent="yes" xalan:indent-amount="4"/>
    <xsl:template match="true">
        <html>
            <head>
                <title></title>
            </head>
            <body>
                <h1>
                </h1>
                <p>
                </p>
            </body>
        </html>
    </xsl:template>
</xsl:stylesheet>
```

You need to declare the Apache.org `Xalan` namespace and use the attribute `indent-amount` with its prefix `xalan` in combination with the attribute `indent="yes"` of the `xsl:output` instruction if you do not want to have the HTML code of this example lined up like pearls on a string but wrapped around into separate lines, neatly indented and easier to read and edit.

```
<html>
    <head>
        <title/>
    </head>
    <body>
        <h1/>
        <p/>
    </body>
</html>
```

Regular expressions

In many cases, models can consist of structures within structures that require further analysis and recombination. This can be in the form of properties of model elements whose text content follows a certain pattern. Regular expressions are a powerful means of breaking content following such patterns into its parts and reusing or recombining them. The `xsl:analyze-string` instruction with its two permitted child elements, `xsl:matching-substring` and `xsl:non-matching-substring`, introduces regular expressions into XSLT.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0">
    <xsl:variable name="abstract">
        <xsl:text>Whose woods these are I think I know.
His house is in the village, though;
He will not see me stopping here
To watch his woods fill up with snow.
My little horse must think it's queer
To stop without a farmhouse near
Between the woods and frozen lake
The darkest evening of the year.
He gives his harness bells a shake
To ask if there's some mistake.
The only other sound's the sweep Of easy wind and downy flake.
The woods are lovely, dark, and deep,
But I have promises to keep,
And miles to go before I sleep,
And miles to go before I sleep.</xsl:text>
    </xsl:variable>
    <xsl:output method="html"/>
    <xsl:template match="true">
        <HTML>
            <BODY>
                <P>
                    <xsl:analyze-string select="abstract" regex="\r\n">
                        <xsl:matching-substring><br/><xsl:text></xsl:text></xsl:matching-
substring>
                    <xsl:non-matching-substring>
                        <xsl:value-of select="self"/>
                    </xsl:non-matching-substring>
                </xsl:analyze-string>
            </P>
        </BODY>
    </HTML>
    </xsl:template>
</xsl:stylesheet>
```

The little stylesheet example above converts a poem in plain text and stored in a variable for demonstration purposes into HTML output, wrapping it in a paragraph. The matching-substring sections (that is, the carriage-return line-feed combinations specified in the regular expression for the analyze-string instruction) are replaced by the typical HTML line breaks `<BR/>` followed by enforced line breaks using the text-directives. For nonmatching substrings, that group part of the regular expression stored in the OCL self variable is transferred to the result unchanged, producing:

```
<HTML>
    <BODY>
        <P>
Whose woods these are I think I know.<br>
His house is in the village, though;<br>
He will not see me stopping here<br>
To watch his woods fill up with snow.<br>
My little horse must think it's queer<br>
To stop without a farmhouse near<br>
Between the woods and frozen lake<br>
The darkest evening of the year.<br>
He gives his harness bells a shake<br>
To ask if there's some mistake.<br>
The only other sound's the sweep<br>
Of easy wind and downy flake.<br>
The woods are lovely, dark, and deep,<br>
But I have promises to keep,<br>
And miles to go before I sleep,<br>
And miles to go before I sleep.
</P>
    </BODY>
</HTML>
```

Appendix

Priority computation

Expression	Priority	Examples
Computed priorities	Ranges between 0.25 and -0.5	
`oclIsTypeOf`	0	
`oclIsKindOf`	Ranges from -0.125 inclusively to -0.25 exclusively. The algorithm is: priority = (distance + 1) / (distance + 2) * -0.25, where distance is the number of "generalizes" relations between the actual argument type of the `oclIsKindOf` expression and the demanded type. That is, the priority starts at -0.125 when the types are identical and asymptotically nears -0.25, the farther an actual type is removed from the specified type.	Example: `oclIsKindOf(ecore::EPackage)`, `self` is of type `ecore::EPackage`, distance is 0, priority is ((0 + 1)/(0 + 2)*-0.25) or 1/2*-0.25 or -0.125 Example: `oclIsKindOf(ecore::EClassifier)`, self is of type `ecore::EClass`, distance is 1, priority is ((1 + 1)/(1 + 2)*-0.25) or 2/3*-0.25 or -0.1666666666 Example: `oclIsKindOf(ecore::EClassifier)`, self is of type `ecore::EEnum`, distance is 2, priority is ((2 + 1)/(2 + 2)*-0.25) or 3/4*-0.25 or -0.1875
Comparative expressions and functions	0.25	Example: `self.oclAsType (ecore::EClassifier).name = 'MyName'` Example: `self.oclAsType(ecore::EList)->notEmpty()`

		Example: `context uml::kernel::packages::Package inv: self.name <> 'SuspendedRoot'`
Unqualified collections	-0.5	Example: `context ecore::EPackage inv: self.eContents()`
Constrained collections respectively constraining collection operations	Depends on the constraint expression	Example: `context ecore::EPackage inv: self.eContents()->select(oclAny : OclAny | oclAny.oclIsKindOf(ecore::EClass))` The above selection operation has the priority of its constraint expression `oclIsKindOf`, which is -0.25 Example: `context ecore::EPackage inv: self.eContents()->select(oclAny : OclAny | oclAny.oclIsTypeOf(ecore::EClass))` The above selection has the priority of its constraint expression `oclIsTypeOf` , which is 0. Example: `context ecore::EPackage inv: self.eContents()->select(eClass : ecore::EClass | eClass.name <> 'HotPotato')` The above selection has the priority of its Boolean constraint expression, which is 0.25. Example: `context ecore::EPackage inv: self.eContents()->select(oclAny : OclAny | oclAny.oclIsTypeOf(ecore::EClass) and oclAny.oclAsType(ecore::EClass).name = 'PickMe')` The above selection has the priority of its composite constraint expression, which is 0.25 because of the higher prior comparison.
OR-ed expressions	The priority of the first striking part is used	Example: `self.oclIsTypeOf(ecore::EClass) or self.oclIsKindOf(ecore::EClassifier)` If context object is of type `ecore::EClass` the priority of `self.oclIsTypeOf(ecore::EClass)` is taken. If context object is of type `ecore::EClassifier` but not `ecore::EClass` the priority of `self.oclIsKindOf (ecore::EClassifier)` is taken. Example: `self.oclIsKindOf (ecore::EClassifier) or self.oclIsTypeOf (ecore::EClass)` If context object is of type `ecore::EClass`, the priority of `self.oclIsKindOf(ecore::EClassifier)` is taken. If context object is of type `ecore::EClassifier`, the priority of `self.oclIsKindOf (ecore::EClassifier)` is taken as well.
AND-ed expressions	The highest priority of all matching parts is taken	Example: `self.oclIsKindOf (ecore::EClassifier)` and `self.oclAsType (ecore::EClassifier).name = 'MyName'`

	If context object is of type `ecore::EClass` and has a name `'MyName'`, the priority of `self.oclAsType (ecore::EClassifier).name = 'MyName'` is taken. Example: `self.oclIsKindOf (ecore::EClassifier)` and `self.oclAsType (ecore::EClassifier).name = 'MyName'` If context object is of type `ecore::EClass` the priority of `self.oclAsType(ecore::EClassifier).name = 'MyName'` is taken.

Related Concepts

Model Transformation Support

Related Reference

XSL Editor

QVT Ant Tasks

Provides a description of QVT Ant tasks, which let you launch QVT transformations from the Ant build.

qvt.applyCompiledTransformation

This task applies a compiled (deployed) QVT transformation to the specified model.

Example:

```
<qvt.applyCompiledTransformation
transformation="package.Transformation"
sourceuri="platform:/resource/project/model.xmi"
targeturi="platform:/resource/project/result.xmi"
tracefile="platform:/resource/project/Transformation.trace"
targettype="EXISTING_MODEL"     feature="feature_name"
clearcontents="true"
resulturiproperty="result_uri" />
```

Attribute	Value	Description
`sourceuri`	String	Specifies the URI of the source model.
`targeturi`	String	Specifies the URI of the target model.
`tracefile`	String	Specifies the workspace-relative path to the trace file. If this attribute is omitted, Together does not generate a trace file.
`targettype`	String	Defines where to save the result of the transformation. Possible values: `NEW_MODEL` – (default). Creates a new model or overwrites the existing one.`EXISTING_MODEL` – Saves the result into the specified feature of the existing model.`INPLACE` – Overwrites the source model with the transformation result.
`clearcontents`	Boolean	Specifies whether you want to clear contents of the target feature before writing the result (when `targettype="EXISTING_MODEL"`).
`resulturiproperty`	String	Specifies the name of the Ant property where you want to save the result URI. Use this attribute to pass the transformation result to another transformation as the transformation input.

qvt.applyTransformation

This task applies a project (not compiled) QVT transformation to the specified model.

Note: Together generates and executes the transformation Java code.

Example:

```
<qvt.applyTransformation
    transformation="/project/Transformation.qvt"
    ...
/>
```

The attributes are the same as for `qvt.applyCompiledTransformation`, except the following:

Attribute	Value	Description
`transformation`	String	Specifies the workspace-relative path to the transformation file.

qvt.interpretedTransformation

This task applies a project (not compiled) QVT transformation to the specified model by interpreting the QVT code.

Example:

```
<qvt.interpretedTransformation
    transformation="/project/Transformation.qvt"
    ...
/>
```

The attributes are the same as for `qvt.applyTransformation`.

Related Concepts

Model Transformation Support

Related Procedures

Creating a Composite Transformation

Related Reference

Model-To-Text Ant Tasks
XSL/OCL Ant Tasks

QVT Operational Ant Tasks

Provides a description of Operational QVT Ant tasks, which let you launch QVTO transformations from the Ant script.

qvto.interpretedTransformation

This task applies a QVTO transformation (workspace located) to the specified model by interpreting the QVTO code.

Example:

```
transformation ecore2uml(in ecore : ECORE, out uml1 : UML, inout uml2 : UML);
```

```
<qvto.interpretedTransformation
    transformation="/project/ecore2uml.qvto"
    tracefile="/project/ecore2uml.trace"
    resulturiproperty="myResulturiproperty"
    >
    <targeturidef
        targeturi="platform:/resource/project/input.ecore"
    />
    <targeturidef
        targeturi="platform:/resource/project/result1.uml"
    />
    <targeturidef
        targeturi="platform:/resource/project/result2.uml"
        targettype="EXISTING_MODEL"
        feature="eOperations"
        clearcontents="true"
    />
    <configurationProperty name="prop" value="attr"/>
</qvto.interpretedTransformation>
```

The following table describes each script attribute and its value type.

Attribute	Value Type	Description
`transformation`	String	Specifies the workspace-relative path to the transformation file.
`tracefile`	String	Specifies the workspace-relative path to the trace file. If this attribute is omitted, trace file is not generated.
`resulturiproperty`	String	Specifies the name of the Ant property where you want to save the resulting URIs. Use this attribute to pass the transformation results to another transformation as the transformation input. Results are numbered starting from 1 and so forth in ascending order. Example: `targeturi="${myResulturiproperty1}"`
`targeturidef:targeturi`	String	Use both for specifying the URI of the source and target models.
`targeturidef:targettype`	String	Defines where to save the result of the transformation. Possible values:
`targeturidef:feature`	String	Specifies the target structural feature to save the result to (only when `targettype="EXISTING_MODEL"`).

`targeturidef:clearcontents`	Boolean	Specifies whether to clear contents of the target feature before writing the result (only when `targettype="EXISTING_MODEL"`).
`configurationProperty`	name:String value:String	Specifies configuration properties for the transformation.

Related Concepts

Model Transformation Support

Related Reference

QVTO Language

Model-To-Text Ant Tasks

Provides a description of Model-To-Text Ant tasks, which let you launch Model-To-Text transformations from the Ant build.

m2t.applyCompiledTransformation

This task applies a compiled (deployed) Model-To-Text transformation to the specified model.

Example:

```
<m2t.applyCompiledTransformation
    transformation="package.Transformation"
    sourceuri="platform:/resource/project/model.xmi"
    targetdir="platform:/resource/project/result"
/>
```

Attribute	Value	Description
`transformation`	String	Specifies the transformation ID.
`sourceuri`	String	Specifies the URI of the source model.
`targetdir`	String	Specifies the workspace-relative path to the folder where you want to store the transformation results (it can be a project root).

m2t.applyTransformation

This task applies a project (not compiled) Model-To-Text transformation to the specified model.

Example:

```
<m2t.applyTransformation
    transformation="com.foo.project.package.TransformationClass"
    ...
/>
```

The attributes are the same as for `m2t.applyCompiledTransformation`, except the following:

Attribute	Value	Description
`transformation`	String	Specifies a fully qualified name of the transformation class.

Related Concepts

Model Transformation Support

Related Procedures

Creating a Composite Transformation

Related Reference

QVT Ant Tasks
XSL/OCL Ant Tasks

XSL/OCL Ant Tasks

Provides a description of XSL/OCL Ant tasks, which let you launch XSL/OCL transformations from the Ant build.

xsl.applyTransformation

This task applies an XSL/OCL transformation to the specified model.

Example:

```
<xsl.applyTransformation
transformation="platform:/resource/project/index.xsl"
sourceuri="platform:/resource/project/model.xmi"
targetfile="/project/index.html"
applymode="index"
resulturiproperty="result_uri">
    <parameter name="title" value="Project Model Index"/>
</xsl.applyTransformation>
```

Attribute	Value	Description
`transformation`	String	Specifies the workspace-relative path to the XSL stylesheet file.
`sourceuri`	String	Specifies the URI of the source model.
`targetfile`	String	Specifies the workspace-relative path to the result file.
`applymode`	String	Specifies the `mode` attribute of the applied template.
`callname`	String	Specifies the `name` attribute of the applied template.
`resulturiproperty`	String	Specifies the name of the Ant property where you want to save the result URI. Use this attribute to pass the transformation result to another transformation as the transformation input.

<parameter> element

Use the contained `<parameter>` elements to pass parameters to the transformer.

Attribute	Value	Description
`name`	String	Specifies the parameter name that is used in your XSL stylesheet.
`value`	String	Specifies the parameter value.

Related Concepts

Model Transformation Support

Related Procedures

Creating a Composite Transformation

Related Reference

QVT Ant Tasks
Model-To-Text Ant Tasks

QVT Operational Migration Notes

Introduction

The legacy QVT engine is not fully compliant with the current QVT specification. Users who want to migrate from the proprietary QVT engine to M2M.QVTO should consider making the following code changes.

Use modeltype Expression to Reference Metamodels

Reference metamodels using the `modeltype` expression instead of the metamodel expression, as in the following example.

```
modeltype uml uses 'http://www.borland.com/together/uml';
```

The `metamodel` expression becomes deprecated.

```
metamodel 'http://www.borland.com/together/uml';
```

Transformation Signature

Use a transformation declaration to define model parameters.

```
transformation NewTransformation(inout model1 : rdb, out model2 : simpleuml);
```

A declaration without parameters becomes deprecated.

```
transformation samples.Simpleuml_To_Rdb;
```

The legacy QVT defines transformations using an identifier and a main mapping operation with parameters. In standard QVT, a transformation defines a signature that indicates the models that the transformation can process. The signature consists of a transformation identifier and a list of model parameters. A transformation accepting an Ecore model as input and producing a UML output model can be written as follows.

```
transformation Ecore2Uml(in inModel : ECORE, out outModel : UML);
```

The `ECORE` and `UML` model type identifiers denote the metamodels that are applicable to the transformation. There is no implicit metamodel resolution implemented, which could eventually resolve a metamodel uniquely identified by a name. You must declare a model type that refers to the model's metamodels by URI. To complete the preceding example, define `ECORE` and `UML` model types before the transformation signature.

```
modeltype ECORE uses "http://www.eclipse.org/emf/2002/Ecore";
modeltype UML uses "http://www.eclipse.org/uml2/2.1.0/UML";

transformation Ecore2Uml(in inModel : ECORE, out outModel : UML);
```

The `in`, `out` keywords used in the signature denote the direction kind of individual model parameters. In addition, the `inout` direction is available for definition of inplace transformations. Model parameters qualified as `in` require an existing model instance that represents a read-only input. Similarly, `inout` parameters refer to existing models, but modifications are allowed.

Finally, `out` parameters always result in the creation of new model instances that are initially empty and that are then populated during the transformation execution.

Note: In standard QVT, the `modeltype` declaration replaces the `metamodel` keyword used in legacy QVT.

In order to execute a transformation, all formal model parameters must be bound to actual contexts in which existing models are resolved and new model instances are created. This is done external to the transformation, typically in a runtime configuration, as when QVT Interpreter launches configurations in the Eclipse UI. The actual binding is realized by using an EMF resource referenced by URI. The contents of the resource form the logical MOF Extent associated with every model parameter of a transformation. Therefore, after a resource is associated with a model parameter, the transformation can load and eventually modify its contents and save new output there.

Standard main() Entry Point

The QVT specification defines the entry point of a transformation as a unique imperative operation named `main`. It has no arguments and return type, and its body is executed immediately after the owning transformation is instantiated.

Typically, the body contains the logic to query appropriate objects within the extents of `in` or `input` model parameters. These selected elements become the source objects for mapping calls producing the transformation output.

```
transformation Simpleuml_To_Rdb(in uml : UML, out rdb : RDB);

main() {
    uml.rootObjects()[UML::Model]->map model2RDBModel();
}
```

The standard signature-less main operation helps to define a flexible execution logic accepting input elements of various types, which belong to the declared model types. In many cases, no specific flow is required and a mapping operation between a top-level input type and its corresponding output type is sufficient. This mapping operation then invokes other necessary mappings on its child objects and composes a complete transformation result. This scenario is supported in legacy QVT and is also still valid in the new QVT. A mapping operation as the entry point is shown in the following example.

```
mapping main(in ePackage : EPackage, out umlPackage : Package)
```

Note: `mapping main(...)` is a legacy construct that originated from early versions of the QVT specification.

Collection Types

In legacy QVT, the data objects of Collection types cannot contain a `null` value. All related collection operations or literals do not allow undefined values to be added to resulting collections. The OCL 2.0 specification states that the `null` value is a legal element in a collection and the new QVT implementation follows this rule.

In addition, according to the OCL 2.0 specification, the Collection type no longer conforms to `OclAny`. Check for potential misuses of `OclAny` operations because some of these misuses might not be detected by a compilation error.

In the following example, legacy QVT takes collections as conforming to `OclAny`.

```
 var bag : Bag(String) := Bag { 'aString'};
-- calls oclIsKindOf() on 'bag', which conforms to OclAny
```

```
var b : Boolean := bag->oclIsKindOf(String);
var oclAny : OclAny := bag; -- This is legal
```

In the following example, standard QVT does not handle collection types as `OclAny` subtypes.

```
var bag : Bag(String) := Bag { 'aString'};
-- xcollect -> calls oclIsKindOf() on each element of 'bag'
var b : Bag(Boolean) := bag->oclIsKindOf(String);
var oclAny : OclAny := bag; -- compilation error
```

Mapping Structure

Mappings are now implemented according to specification and can contain `init`, `population` and `end` sections.

```
 mapping metamodel1::Metaclass1::mappingName(): metamodel2::Metaclass2{
    init{
    }
    population{
        object result:{
        }
    }
    end{
    }
}
```

Mapping Operation Call

Calls on mapping operations must be used in conjunction with the `map` keyword. The ordinary operation call variant is nonstandard and has been deprecated.

Because `null` can now be contained in collections, existing mapping calls on collection source objects or any `collect` of mapping calls results can cause the `null` value to be included in resulting collections. In fact, a mapping call can result in `null` if the mapping precondition fails.

```
-- sourceObjects is a collection and the mapping call causes to collect
-- the results of mapping calls on every collection element
-- The resulting collection may contain 'null'
sourceObjects.map foo();
```

The standard QVT uses the `xcollect` iterator (`->`) to skip irrelevant null results of mapping calls and to collect only the elements of interest.

```
-- the resulting collection will NOT contain 'null'
sourceObjects->map foo(); -- xcollect shorthand
```

Mapping Execution Semantics

In legacy QVT, every mapping operation call executes its mapping body even if the mapping operation is executed on the same source object repeatedly.

In standard QVT, only the first execution of a mapping operation on a given source object results in complete body execution, and a corresponding trace instance that records all parameters is created.

All subsequent calls made to the same mapping operation and the same source object check first available traces. If a trace record for a particular mapping and source already exists because it has been executed previously, the existing output value is fetched from the trace and returned to the caller without re-executing the operation body.

An exception to this unique mapping paradigm is the case of reentering a mapping operation. This is due to the fact that the trace record is created immediately after the mapping initialization section, so it is only from this point forward that the mapping result can be accessed using traces, even though the mapping might not yet have its result population step fully completed. If the execution within the init section again invokes the same mapping for the same source, no corresponding trace is available yet and the mapping body is re-executed for the same source.

It is the responsibility of the transformation writer to carefully consider the possible mapping calls from init sections.

The self Variable

Legacy QVT allows for implicit resolution of a `self` variable. This is applied if a property or operation call has no source object on which the call is performed.

```
/*
 * Legacy QVT implicit source resolution
 */
mapping simpleuml::Model::simple2Rdb() : rdb::Model {
    name := name; -- implicit resolution -> self.name
}
```

The QVT standard method is to refer to the contextual instance by a `self` variable explicitly in both mapping and query operations. An implicit source is resolved to the module (transformation or library) instance that defines the operation containing the implicit call expression.

```
/*
 * Standard QVT requires explicit 'self' variable
 */
mapping simpleuml::Model::simple2Rdb() : rdb::Model {
    name := self.name;
}
```

For explicit reference to the module instance, the predefined `this` variable can be used. Note that OCL implicit iterators take precedence over the module instance in implicit source object resolution, so `this` can be used for referring to the module scope.

object Expression

The explicit use of an `object` expression as a mapping body that was supported in legacy QVT is nonstandard and has been deprecated.

```
/*
 * obsolete object expression usage
 */
mapping simpleuml::Model::simple2Rdb() : rdb::Model {
    object {
        name := self.name;
    }
}
```

A mapping with an implicit or explicit population section can be used instead. The following example shows an `object` expression in an explicit population section, which updates a previously created `object` referenced by the `result` target variable.

```
/*
 * new construct with explicit population section
 */
mapping simpleuml::Model::simple2Rdb() : rdb::Model {
    population {
        object result: {
            name := self.name;
        }
    }
}
```

while Expression

The legacy QVT style of `while` expression is not compliant with the standard QVT and has been replaced by the following `while` constructs.
The following example shows a `while` loop with Boolean type condition only.

```
var i : Integer := 0;
while(i <= 10) {
    i := i + 1;
    -- do something
};
```

The following example shows a `while` loop with both variable declarator and Boolean type condition.

```
while(c : Integer = 0; c <= 10) {
    c := c + 1;
    -- do something
};
```

undefined and invalid Values

Legacy QVT supports only undefined values represented by the singleton value literal named `undefined`. This is obsolete and has been replaced by the `null` value literal. Besides the undefined value, the OCL 2.0 specification also defines the `invalid` value, which is of `Invalid` type (defined in the Standard Library). This has also been adopted in the new QVT implementation.

It has exactly one runtime instance named `OclInvalid`. By definition, any operation or attribute call performed on an undefined source object results in `OclInvalid`. Note that this is different from legacy QVT, where the result is also undefined.

In legacy QVT, equality operations involving undefined values result in `undefined`. The OCL 2.0 supports equality operations for both `undefined` and `invalid` values and always results in a Boolean type value.

Note: Always use an `OclAny::oclIsUndefined()` operation to test for undefined values in general, as this tests for both `undefined` and `invalid` cases. The operation `OclAny::oclIsInvalid()` only tests for invalidity.

Variable Initialization

Legacy QVT supports variable declarations without initial expressions and assigns the `null` value by default.

```
var str : Bag(String); -- resulting variable has the 'null' value
```

The QVT specification defines default values that are different from `null` for some predefined types such as String, Collection, and numerical types. However, the Boolean type is skipped and keeps a `null` value as the default.

The new QVT implementation always requires an initial expression in variable declarations. Missing initial expressions are marked as compilation errors. Because these are reasonably easy to fix, the initial expression makes for a safer migration step then new unexpected values, such as empty strings for the String type, that can silently change the execution logic.

Escaping Identifiers

Model element identifiers that conflict with QVT or OCL keywords can be escaped by prepending the underscore (_) character, as in the following example.

```
(self._abstract = false)
```

Resolve Expressions

Legacy QVT defines a limited trace ability by definition of `resolve` and `invresolve` operations, which have been replaced by OMG standard implementation of the `resolve` expression in the new QVT implementation. The `resolveByRule` legacy operation is now realized by the standard `resolveIn` equivalent.

Compare the following two examples of a `resolve` operation, the first using legacy QVT.

```
mapping simpleuml::Model::simple2Rdb() : rdb::Model {
    name := name;
    end {
        self.resolve(rdb::Model);
        self.resolveByRule('simple2Rdb', rdb::Model)
        result.invresolve(rdb::Model);
        result.invresolveByRule('simple2Rdb', rdb::Model);
    }
}
```

The following `resolve` operation example has been migrated to a standard QVT `resolve`.

```
mapping simpleuml::Model::simple2Rdb() : rdb::Model {
    name := self.name;
    end {
        self.resolve(rdb::Model);
        self.resolveIn(Model::simple2Rdb, rdb::Model)
        result.invresolve(rdb::Model);
        result.invresolveIn(Model::simple2Rdb, rdb::Model);
    }
}
```

The M2M.QVTO engine has a larger set of resolve call variants with more flexible filtering capabilities. In addition to the constraint on the type of resolved objects, further restrictions of the result can be defined.

You can accomplish type filtering by specifying the type name. For example, `source->resolve(Table);` selects only Table instances, and `source->resolve()` selects any object.

An optional Boolean type condition for further restriction can be applied as in the following example.

```
source->resolve(t : Table | t.name = 'nameValue')
```

where `t` is the target variable that provides access to the target in the condition expression and is initialized by applicable target objects. Only those target objects that satisfy the condition are included in the final result. It might appear that the same result could be retrieved by applying the condition on the result of the resolve. However, there might be a difference in performance if a built-in condition within the resolve avoids creating intermediate collections on which further filtering should be applied.

The result type of resolve calls is determined either by the type name condition or the target variable type (if specified). Otherwise, `OclAny` is the result type (or element type, if the result is a collection).

Note: The `Object` type (supertype of `OclAny`), described in the QVT spec is not currently supported by this QVT implementation.

The following modifiers can be used to select an execution strategy of resolve expressions:

one	The first applicable single result found is returned. If no suitable result was found, `null` is returned. If not applied, the result type of resolve is of the `Sequence` collection type. The resolve call variant used in the following example returns all targets created or updated by mappings already called on the source. In that case, a `Sequence` of the target type is returned. In order to limit the result to a single object, the resolveone variant can be used, which ensures that the first target found in traces is returned. `Sequence(Table) tables = source.resolve(Table); Table table = source.resolveone(Table);`
Inv	The inverse direction, resolving the source objects used in mappings to create or update the target objects on which the resolve call is applied.
late	Deferred resolution, the late resolving variant has different execution semantics and postpones the actual resolution until the end of the transformation. The actual late resolve call results in null during the regular transformation run and stores its current execution environment needed for the later re-execution. The environment contains the variables to the late resolve call that are available when reached at normal execution time as well as the source object of the call. The source is computed at normal execution time only and used to execute the late resolve call at deferred time. All late resolve calls should be used with assignments and should not be targeted to local variables. Instead, they should be targeted to object properties so that a permanent effect can be achieved after the transformation ends. The following `EClass2UMLClass` mapping performs mapping of objects referenced by the `EClass::eSuperTypes` property to `UML::Class` instances set into the `UML::Class::superClass` property by using the late resolve.When the assignment statement is reached during normal execution, the statement is not executed and its right side expression results in `null` without actual evaluation, which is performed only at the end of the transformation. Note, that no code referencing the left side property of deferred assignments is ever re-executed along with deferred assignments. Such code will retain its last value assigned in the normal execution time. In the following example, the variable c does not get assigned the late value of the `superClass` property.`mapping ECORE::EClass::eClass2UMLClass() : UML::Class { superClass := self.eSuperTypes.late resolve(UML::Class)->asSet(); end { var c := result.superClass->asOrderedSet(); } }` All late resolutions are executed sequentially in the order they were encountered by the normal execution. The implementation does not perform any reordering to guarantee that any condition expressions used in late resolve calls and referencing late assigned properties will receive the late assigned values during its evaluation.

The preceding modifiers are not mutually exclusive and can be combined as one of the following:

- resolveone
- late resolve
- late resolveone

- invresolve
- invresolveone

The late resolve variant with invresolve is also allowed but is not useful because the source object is always available through a non-late resolve.

resolveIn	resolves target objects created or updated by a specific mapping operation, which is referred to by its qualified identifier (`<context class>::name`). This modifier is not currently supported by QVT concrete syntax. If there are multiple mappings of the same name but with different signatures, an ambiguity error is reported.

Strings Library

A number of operations on the String type are defined in the new QVT Standard Library, while some of these are duplicated in the custom Strings native library available in legacy QVT. If it is available, refer to QVT `StdLib` in the migrated QVT source code.

Logging

The QVT standard `log` expression can be used to replace usages of the proprietary `dump()` operation available in legacy QVT. A textual message along with optional object data can be sent to the log. Eventually, this can be disabled or enabled by a Boolean type condition as shown in the following example.

```
log('a message');
log('a message', self);
log('a message', self.name) when self.name <> null;
```

Helpers

Helpers are very similar to queries, but have the advantage of being able to modify passed arguments.

Explicit Return for Queries

In legacy QVT, the last expression of an operation body is considered the result value. The new QVT implementation introduces a dedicated `return` statement to return a result from a query or helper operations. Additionally, this can be used to change the execution flow as the `return` statement causes the current operation to quit immediately and return the result to the caller.

```
helper EClass::getUpperName() : String {
    return self.name.toUpper();
}
```

Related Concepts

Model Transformation Support

Related Reference

QVTO Language
QVT Language

QVT Operational Imperative Iterators

Provides a description of QVT imperative iterators and their shorthand notation.

An imperative iterate expression is an imperative loop expression that iterates over a source collection and builds a given result by using iterator variables, a target variable, a body, and a condition expression.

QVT imperative iterators are implemented in DSL Toolkit in accordance with section 8.2.2.7 of the MOF QVT Final Adopted Specification (http://www.omg.org/docs/ptc/05-11-01.pdf), with the following exceptions.

`xcollect`	Unlike `collect`, `xcollect` (`collectOne`) does not flatten results. A flat collection cannot contain other collections as elements, and the elements of all nested collections are xcollect extracted to the top level. For example,
	 ``` `{{1, 2}, 3, {4 , {5, 6}}}->flatten()` ```
	 results in
	 ``` `{1, 2, 3, 4, 5, 6}` ```

`collectOne`	DSL Toolkit implements six imperative operators, not five as specified in section 8.2.2.7 of the MOF QVT Final Adopted Specification.The QVT specification does not document the collectOne iterator. The EBNF section of the QVT specification documents the collectOne iterator.
`selectOne` and `collectselectOne`	The pseudocode for selectOne and collectselectOne in section 8.2.2.7 of the QVT specification does not make sense. According to this pseudocode, the evaluation of selectOne and `collectselectOne` the `selectOne` and `collectselectOne` operators produces a collection that contains a single element. However, the apparent intention is to return the element itself, not a collection that contains the element.
`xcollect`, `collectset`, and `xselect` behavior	Imperative iterators in DSL Toolkit are implemented by using the behavior of collect and select in OCL. For more information, see section 7.6.2 of the OCL Specification (http://www.omg.org/docs/ptc/05-06-06.pdf). Therefore, for `xcollect` and `collectselect`, Sets and Bags result in Bags, while `OrderedSets` and Sequences result in Sequences. This approach allows for duplicates and preserves the source collection's original type for xselect.

Full notation forms	Two notation forms exist for `xselect`, two forms for `xcollect`, and only one form for `collectset`. The following forms are the only ones permissible.
	 `<source> –> xselect (<iterator-list> | <condition>) ;`
	 `<source> –> xselect (<condition>) ;`
	 `<source> –> xcollect (<iterator-list> | <body>) ;`
	 `<source> –> xcollect (<body>) ;`
	 `<source> –> collectselect (<iterator-list>; <target> = <body> | <condition>)`
	
	 For the `selectOne`, `collectOne`, and `collectselectOne` operators, see the `xselect`, `xcollect`, and `collectselect` notations, respectively.

Related Concepts

Model Transformation Support

Related Reference

QVTO Language

QVT Operational Transformation Wizard Configuration Properties

Use the **Configuration Properties** page of the **Apply Transformation** wizard to specify values for the properties defined in your QVT script. The page is available only for Model-To-Model transformations.

A transformation can define configuration properties. These configuration properties receive their actual values at execution. You can use QVT launch configurations to pass these configuration properties, which let you assign a string representation of values to individual properties. Because raw string values are specified, the launch configuration validates each passed value according to its related property type and rejects invalid values. This check is not always sufficient, however, so an additional validation is performed at execution.

By modifying configuration properties in QVT scripts, the transformation writer can eventually change the type of a previously defined configuration property. Therefore, an existing launch configuration can contain an invalid stringified value for the configuration property. In this case, a QVT runtime exception is thrown at the point of the configuration property initialization and the execution is terminated. This behavior is preferred over a silent execution using undefined or invalid values, because values that are explicitly set are received. If no value is available (no value is set) for a configuration property at execution time, the configuration property is initialized to a null value. To test this behavior, verify that the following list of assert statements remain true:

- `assert (myConfigProperty == null);`
- `assert (myConfigProperty.oclIsUndefined());`
- `assert (not myConfigProperty.oclIsInvalid`

Note: Currently, only QVT-recognized primitive types are supported for configuration properties.

Item	Description
Property	Specifies the name of the property that is passed to the transformation.
Type	Specifies the property type.
Value	Specifies the property value. You can edit this field.

Related Concepts

Model Transformation Support

Related Procedures

Applying Model-To-Model Transformations
Applying Model-To-Text Transformations

Related Reference

QVT Language
QVT Editor

QVTO/OCL Collections and Operations

Most relationships in object-oriented systems occur between an object and a collection of other objects. OCL predefines a number of collection types and collection operations to allow the manipulation of collections. The different types of collections influence OCL expressions.

OCL Collection Types1

A *collection* is an abstract superclass with concrete collection types as subclasses. OCL specifies the following four subclasses of collections.

Set: A collection of unordered, unduplicated elements, typically resulting from a single navigation.
Ordered Set : A Set, but with a collection of *ordered*, unduplicated elements.
Bag: A collection of unordered elements that might be duplicated, typically resulting from a combined navigation.
Sequence: A Bag, but with a collection of *ordered* elements that might be duplicated.

Collections often result from navigating from objects. However, collections can also result from specifying literals. Elements within these collections can be enumerated within the syntax by placing them within curly brackets following the type of collection that contains them.

OCL Collection Operations

Another way of creating collections is through the use of collection operations, which efficiently project new collections from existing ones.

In syntax, an arrow between a collection and the operation represents a collection operation, as in the following example.

```
collection->operation
```

The arrow indicates the collection operation, which reveals the properties of a collection. The following types of operations are available:

- *standard operations* - operations available for all four collection types.
- *variant operations* – operations that provide different functions when applied to different collection types.
- *loop operations* or *iterators* – operations that loop over a collection, take an OCL expression as a parameter, and evaluate each element in a collection for that expression.

The following table gives a brief description for operations applicable to the four collection types.

Operation	Collection Type	Description
`any`	All	A loop operation and variant of the select operation that returns any random element of the source collection whose expression is `true`.
`append`, `prepend`	OrderedSet, Sequence	Variant operations that put an element at the end of (append) or at the beginning of (prepend) a Sequence or OrderedSet.
`asBag`, `asOrderedSet`, `asSequence`, `asSet`	All	Use one of these variant operations to transform one concrete collection type into an instance of another concrete collection type. Depending on which operation is applied to which collection type, the

		result changes either the order of the elements or the duplication properties of the elements. For example, if you apply the `asBag` operation on a Sequence collection, the order of the Sequence is lost in the resulting collection.
`at`	OrderedSet, Sequence	A variant operation that results in the element at the current position.
`collect`	All	A loop operation that returns the set of all values for a certain attribute of all objects in a collection. For example, in the context `stores.numberOfCustomers`, the integer value of *numberOfCustomers* is added to each element in a collection of stores.
`collectNested`	All	A loop operation that returns the set of all values for a certain attribute of all nested collections in a collection.
`count`	All	A standard operation that returns the number of occurrences of an element in a collection.
`equals` (or `=`)	All	A variant operation that evaluates to `True` if all elements in two collections are the same. For orderedSets and Sequences, the elements must not only be the same in both collections but the order must also match.
`excludes` (or `excluding`)	All	A standard operation that results in a new collection with one fewer object than was in the original collection.
`excludesAll`	All	A standard operation that results in a new collection in which all the objects from the original collection are absent.
`exists`	All	A loop operation that identifies the presence of at least one element in a collection for which a certain condition is `True`.
`first`	OrderedSet, Sequence	A variant operation that results in the first element of a collection.
`flatten`	All	A variant operation that changes a collection of collections into a collection of objects.
`forAll`	All	A loop operation that always takes a Boolean expression as a parameter. Evaluates to `True` if all elements comply, and to `False` if at least one does not.
`includes` (or `including`)	All	A standard operation that results in a new collection with one more object than was in the original collection. The element is added to sets and ordered sets only if it does not already reside in them.
`includesAll`	All	A standard operation that results in a new collection in which all the objects from the original collection are present.
`indexOf`	OrderedSet, Sequence	A variant operation that returns an integer value specifying the first position of an element in a collection.

insertAt	OrderedSet, Sequence	A variant operation that results in the insertion of an extra element at the specified position of a sequence or ordered set.
intersection	Set, Bag	A variant operation that results in a collection of objects that holds all elements in both collections.
isEmpty	All	A standard operation that must be true when a collection has no elements.
isUnique	All	A loop operation that returns true if the value of the evaluated parameter for every element in the source collection is unique.
iterate	All	The most basic and complex loop operation, the iterate operation is used to build a value by accumulation over a collection. For example, in the expression `collection->iterate( element : Type; accumulator : Type = <expression> | expression-with-element-and-accumulator )`, element is the iterator operation that iterates over a collection. expression-with-element-and-accumulator s evaluated for each element. After each evaluation, the value is assigned to accumulator. In this way, the value of accumulator is built up during the iteration of a collection.
last	OrderedSet, Sequence	A variant operation that results in the last element of a collection.
minus (or -)	Set, OrderedSet	A variant operation that results in a new set that contains all the elements of the set that called the operation but none of the elements in the parameter set.
notEmpty	All	A standard operation that evaluates to true when a collection has at least one element.
notEquals (or <>)	All	A variant operation that evaluates to true if all elements in two collections are not the same.
one	All	A loop operation and variant of the exists operation that returns true if a certain condition for one and only one element in the source collection is true.
reject	All	A loop operation that is like the select operation except it specifies every element from a collection for which the Boolean expression is false.
select	All	A loop operation that specifies a subset in the resulting collection if a certain condition for the subset is true. For example, if customers must have a card for every service in a collection that they use, their total number of cards is a subset of the total number of cards or services available. The select operation picks among the services.
size	All	A standard operation that indicates the predefined operation size of a collection.

`sortedBy`	All	A variant operation that uses a property of the type of the elements in a collection as a parameter to sort the ordered (OrderedSet, Sequence) or unordered (Set, Bag) elements in a collection. The resulting collection's first element is the lowest element.
`subOrderedSet`	OrderedSet	A variant operation that results in an OrderedSet with elements from the lower to upper index in the original order.
`subSequence`	Sequence	A variant operation that results in a Sequence with elements from the lower to upper index in the original order.
`sum`	all	A standard operation that results in the addition of all of the elements in a collection.
`symmetricDifference`	Set	A variant operation that results in a Set containing all the elements in either the Set that called the operation or in the parameter.
`union`	All	A variant operation that results in a new collection of objects that hold the elements in both sets. For example, a Set combined with a Set makes another Set without duplicates. A Set combined with a Bag makes a Bag. An OrderedSet combined with a Sequence, neither of which can be combined with a Set or Bag, results in a collection in which the elements of the calling operation are sequenced before the elements of the parameter collection.

For examples of how collection operations are used, refer to the *UML 2.0 OCL Specification*.

Mapping OCL Collections

OCL collection types can be mapped to collections in one of the libraries in the target language. If the target language does not provide a collection type, you can either define your own class of collection types based on a standard collection type or use a Java collection type that is already closely mapped to OCL, such as Tree, Set, or List. For more information, refer to *The Object Constraint Language – Getting Your Models Ready for MDA*, by Jos Warmer and Anneke Kleppe.

Related Concepts

Model Transformation Support

Related Reference

QVTO Language

MDA Example Projects

The following example transformation projects are available in Together

Transformation Project	Description
Data Modeling to UML	A QVT Transformation that maps an ER Physical model to UML 2.0 model.
Documentation Generation (via XSL)	An XSL transformation that produces Javadoc style documentation in HTML format from a modeling project.
Ecore to UML	A QVT transformation that produces a UML 2.0 model from an Ecore model.
ER Logical to Data Modeling	A QVT transformation that transforms an UML 2.0 project with an ER profile applied into a Data Modeling project using the EMF Profile API.
RDB to DDL	A Model-To-Text transformation that produces a DDL script from an Ecore RDB model.
SimpleUML to RDB	A QVT transformation that produces an RDB Model from a simple UML model.
UML Diagrams	A QVT transformation that demonstrates how to use the EMF API for Together models. The transformation enumerates all the classes, interfaces, data types and enumerations in the input project and creates a Class diagram that contains references to these elements. The references are displayed in different colors and styles.
UML to Data Modeling	A QVT transformation that produces a Data Modeling (ER Physical) project from a Together UML 2.0 project.
UML to J2EE	A collection of transformations (including QVT, Model-To-Text, and XSL transformations) chained to each other using the Composite transformation script. The transformation chain produces a J2EE application from a Together UML 2.0 model.
UML to Java (via XSL)	An XSL transformation that produces Java code from a UML 2.0 model.
UML to WSDL	A QVT transformation that produces a WSDL description of a web service from a UML 2.0 model.
UML to XHTML	A QVT transformation that produces XHTML markup from a UML 2.0 model.
UML to XSD	A QVT transformation that produces an XML Schema from a UML 2.0 model.
WSDL to UML	A QVT transformation that produces a UML 2.0 model from a WSDL file (using reverse engineering).

Related Concepts

Model Transformation Support

Related Procedures

Creating an Example MDA Transformation Project

Related Reference

MDA

EMF API for Together Models

Provides a description of EMF API for Together models.

Introduction

EMF API provides EMF-like access to the public part of Together models. The obtained EMF model is synchronized with the Together model for which is it built for; any changes to one of them immediately affect the other.

EMF API is a Java framework generated from metamodels existing as EMF `.ecore` files. It is organized as a set of EMF packages rooted at `com.borland.tg.emfapi`. The API provides metamodels for UML modeling (UML 1.4 and UML 2.0), Business Process modeling and Data modeling. Together generates a separate EMF API plug-in for each type of modeling. The original `.ecore` model (along with the `.genmodel` file) is located in the `/model` folder of each plug-in.

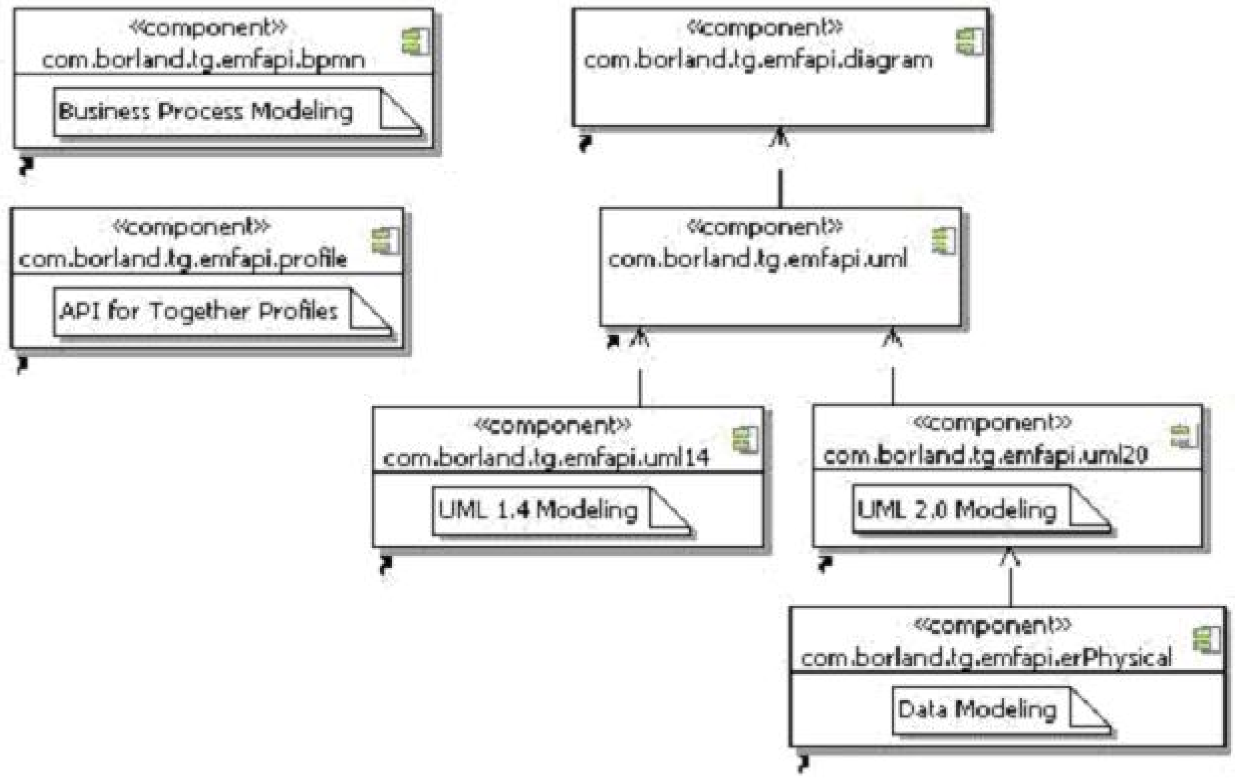
In the **EMF Editor**, using these `.ecore` models, you can create a new EMF model that uses types from Together metamodels.

For every feature of a Together model element there is a named `get<Feature>()` method. For writable features, there is a `set<Feature>()` method.

EMF API supports:

- read/write access to model structure (containment hierarchy and links)
- read/write access to attributes of model elements
- creation/deletion of model elements
- conversion of model change notifications to EMF notifications

Component diagram for existing EMF API plug-ins


«component»
com.borland.tg.emfapi.bpmn
Business Process Modeling
«component»
com.borland.tg.emfapi.diagram
«component»
com.borland.tg.emfapi.profile
API for Together Profiles
«component»
com.borland.tg.emfapi.uml
«component»
com.borland.tg.emfapi.uml14
UML 1.4 Modeling
«component»
com.borland.tg.emfapi.uml20
UML 2.0 Modeling
«component»
com.borland.tg.emfapi.erPhysical
Data Modeling

Creating and accessing instances

EMF API is built on top of the `BCore` metamodel, which is the abstraction layer between API and EMF. Classes from the `BCore` model decorate appropriate EMF classes (namely: `BProxyFactory -> EFactory`, `BProxyObject -> EObject`, `BProxyPackage -> EPackage`).

These classes are roots in the EMF API hierarchy. For example, inheritance hierarchy for an UML Element looks like: `Element (UML) -> BProxyObject (BCore) -> EObject (EMF)`.

`BCore` tracks all generated EMF API plug-ins using the `com.borland.tg.emfapi.bcore.generated_emfapi` extension point. It provides a uniform way to obtain an EMF object for any Together element:

```
Entity modelEntity = … EObject model=BProxyObject.Registry.INSTANCE.getBProxyObject
(modelEntity, ProxySession.DEFAULT_MODEL_SESSION);
```

The second parameter is a `ProxySession` instance. It is used to specify modes that EMF API uses to interact with a Together model. The predefined `ProxySession` values are listed below:

Value	Description
`DEFAULT_SESSION`	Specifies that the obtained EMF object will track model change notifications
`DEFAULT_NONSYNC_SESSION`	Specifies that the obtained EMF object will not track model change notifications, and each request to EMF API will generate a different set of objects
`DEFAULT_MODEL_SESSIONS`	Specifies that the obtained EMF object will track model change notifications and will perform model operations in the “thread-safe” manner

In most cases, the predefined `ProxySession.DEFAULT_MODEL_SESSION` is suitable. A new session can be created using the session factory:

```
ProxySession session = ProxySession.REGISTRY.createSession(..);
```

The main difference between EMF API and the pure EMF is how it obtains the element factory. For an EMF model, something like the following is used:

```
LibraryFactory factory = LibraryFactory.eINSTANCE; Book book = factory.createBook();
```

For EMF API, `IProject` and `ProxySession` must be specified to get the element factory:

```
PackagesFactory factory = PackagesFactory.REGISTRY.getFactory(getIProject(),
ProxySession.DEFAULT_MODEL_SESSION);
Package pack = factory.createPackage();
```

Accessing Together specific properties

The `Bcore` model provides the way to get EMF classes and features using original Together model names:

```
EClass clazz = BProxyObject.Registry.INSTANCE.getBProxyClass("Classifier20");
// get attribute structural feature
EStructuralFeature feature = BProxyObject.Registry.INSTANCE.getBProxyFeature
("Classifier20", "$abstract");
// get reference structural feature
```

```
EStructuralFeature feature = BProxyObject.Registry.INSTANCE.getBProxyFeature
("Classifier20", "generalization");
```

For UML modeling projects, a Together model object is obtained using the `getModel()` method in a `TogetherFactory` class. This method returns a singleton model object that corresponds to `IProject` specified during the factory creation.

```
ProxySession session = ProxySession.DEFAULT_MODEL_SESSION;
TogetherFactory factory = TogetherFactory.REGISTRY.getFactory(getIProject(), session);
Model model = factory.getModel();
```

To distinguish between different UML modeling types (UML 1.4 or UML 2.0), a model object has an `namespaceURI` attribute. This feature contains `NsURI` of an appropriate `Epackage`:

```
ProxySession session = ProxySession.DEFAULT_MODEL_SESSION;
TogetherFactory factory = TogetherFactory.REGISTRY.getFactory(getIProject(), session);
Model model = factory.getModel();
if (model. getNamespaceURI().equals(com.borland.tg.emfapi.uml20.Uml20Package.eNS_URI)) {
    // it's Uml20 model
}
if (model. getNamespaceURI().equals(com.borland.tg.emfapi.uml14.Uml14Package.eNS_URI)) {
    // it's Uml14 model
}
```

`BCore` provides operations for get/set custom properties of the corresponding Together model element:

```
Entity tgClass = …
Class cls = (Class) BProxyObject.Registry.INSTANCE.getBProxyObject(tgClass);
String metaclass = cls. getPropertyValue("$metaclass");
cls.setPropertyValue("custom", "foo");
```

Accessing Together diagrams

The EMF API provides read/write access to the contents of Together diagrams. For the given package, the list of diagrams is obtained like:

```
Package pack = …
EList diagrams = pack. getOwnedDiagrams();
```

The EMF API diagram metamodel is built on the GMF metamodel. This implies that a diagram consists of elements of two types: `Node` (the diagram representation of Together elements) and `Edge` (the diagram representation of Together links). View styles of the diagram elements are represented as the list of instances of the `Style` interface. To modify view styles of the diagram element, you need to add or remove an appropriate instance of `Style`. For example:

```
ProxySession session = ProxySession.DEFAULT_MODEL_SESSION;
TogetherFactory factory = TogetherFactory.REGISTRY.getFactory(getIProject(), session);
Model model = factory.getModel();
```

To distinguish between different UML modeling types (UML 1.4 or UML 2.0), a model object has an `namespaceURI` attribute. This feature contains `NsURI` of an appropriate `Epackage`:

```
ProxySession session = ProxySession.DEFAULT_MODEL_SESSION;
DiagramFactory diagramFactory = DiagramFactory.REGISTRY.getFactory(getIProject(), session);
Node node = diagramFactory.createNode();
FontStyle fontStyle = diagramFactory.createFontStyle();
fontStyle.setFontHeight(20);
node.getStyles().add(fontStyle);
FillStyle fillStyle = diagramFactory.createFillStyle();
fillStyle.setBlue(100);
node.getStyles().add(fillStyle);

Package pack = ...
Class clazz = ...
Diagram defaultDiagram = pack.getDefaultDiagram();
node.setElement(clazz);
defaultDiagram.getChildren().add(node);
```

Saving and loading resources

EMF API objects, like any other EMF objects, can be persisted as XMI:

```
Entity modelEntity = …
ProxySession session = ProxySession.DEFAULT_MODEL_SESSION;
Model model = (Model) BProxyObject.Registry.INSTANCE.getBProxyObject(modelEntity, session);
// Create a resource for the file URI
Resource resource = resourceSet.createResource(fileURI);
// Add the model objects to the contents
resource.getContents().add(model);
// Save the contents of the resource to the file system
try {
  resource.save(Collections.EMPTY_MAP);
} catch (IOException e) {}
```

The `BCore` model provides a special type of `Resource` that is used to store cross-references to EMF API objects. When they are loaded, these objects resolve to Together model elements.

```
Entity entity = ...
URI uri = TogetherResourceFactory.createUri(entity.getModel());
TogetherResource resource = new TogetherResource(uri);
EObject eObject = resource.getEObject(TogetherResource.makeUriFragment(entity));
```

Adapting EMF objects

When an EMF API object is obtained for a synchronized session, the API ensures that the listener is registered for Together model notifications. While dispatching the Together model delta, the API collects all generated EMF notifications and fire them after the delta is processed. The generated EMF notifications are very similar to those generated by the pure EMF for the similar action.

`BCore` provides a set of adapters for adapting `org.eclipse.core.resources.IProject`, `com.tssap.selena.model.elements.Model`, and `com.tssap.selena.model.ui.IElementWrapper` to `EObject`:

```
IProject project = ...
EObject eModel = (EObject) project.getAdapter(EObject.class);
Entity gdmModel = (Entity) Platform.getAdapterManager().getAdapter(eModel, Model.class);
```

This can be used when defining extension points:

```
<extension point="org.eclipse.ui.popupMenus">
  <objectContribution
      adaptable="true"
      id="elementContributions"
      objectClass="org.eclipse.emf.ecore.EObject">
    <action   ...
    </action>
  </objectContribution>
</extension>
```

Using the reflective API

Because the EMF API is based on EMF, you can manipulate with a generated model class using the reflective API defined in the `EObject` interface: `eGet()`, `eSet()`, `eIsSet()`, `eUnset()`.

In the EMF API, `eIsSet()` returns `true` for references and multi-valued properties. For single valued properties, it returns `true` only when the value differs from the default.

In order to create an arbitrary EMF API object using `EClass`, you can use the following pattern:

```
EClass eClass = ...;
EFactory eFactory = BProxyFactory.REGISTRY.getFactory(
   eClass.getEPackage().getEFactoryInstance().getClass(),
   getIProject(), ProxySession.DEFAULT_MODEL_SESSION);
EObject eObject = eFactory.create(concreteClass);
```

Support for the EMF.Edit framework

For every model plug-in, the EMF API provides an `.edit` plug-in that contributes to the `org.eclipse.emf.edit.itemProviderAdapterFactories` extension point. These plug-ins delegate to a Together model the task of providing the label and images for EMF objects.

Samples

The `Uml2Ecore` example (`com.borland.tg.samples.api` plug-in) illustrates how to transform a Together UML 2.0 model static structure into an `EPackage` instance. The generated `EPackage` contains all packages, classes, enumerations, datatypes, and other core elements.

Related Concepts

Model Transformation Support

Related Reference

EMF API for Together Profiles

Model Compare/Merge

Provides reference information on Together Model Compare/Merge facility.

Introduction

Together provides a generic Model Compare/Merge facility that works with EMF models. It supports Together proprietary models (UML 1.4, UML 2.0, and so on) via the EMF API. Model Compare/Merge is designed to be consistent with the standard Eclipse Compare/Merge functionality and uses similar terms.

The term *model* below means the whole containment tree rooted in an EMF object (`EObject`). Together elements are adapted to `EObject` by EMF API. This definition implies that in a model, each object but the root has an unique container. Model Compare/Merge only processes the objects contained in a model and objects that are referenced by an object contained in the model.

A reference is called *internal* when both referencing and referenced object are contained in the same model; a reference is called *external* when it crosses model boundaries. The external references are references to standard UML types (such as Integer or String) that are not contained in any user model.

The term *resource* denotes either a basic EMF resource or a Together project.

Comparing Models

It is possible to compare two or three models. In a two-way compare, case models are called Left and Right. When Model Compare/Merge is used with a VCS like CVS or StarTeam, the *Left* model represents a local version and the *Right* model represents a remote version. In a three-way compare, an *Ancestor* model is added. It represents a common ancestor version when used with VCS.

To activate Model Compare/Merge, first either select elements of the same type on a diagram or in the Model Navigator view, or select Together projects or files with saved EMF resources in any resource view (such as the Navigator view). Then choose **Compare With Each Other (as Models)** in the context menu. Note, that the compare action is disabled if less than two or more than three elements are selected.

During comparison, Model Compare/Merge traverses the models, going level-by-level down the containment tree. On each level, objects are matched using ID features that are set on a preference page. A tuple of ID features values should uniquely identify the object in the list of its container's direct contents. When all objects on all levels of the models are matched, Model Compare/Merge compares values of attributes and non-containment references.

Exporting Compare Results

The model compare results can be exported to the EMF model (`http://www.borland.com/tg/emf/compare/2006/Change` metamodel). The exported model is saved as EMF XMI and has enough information to generate a difference report (by means of XSL).

Merging Models

After the models are compared they can be merged. Note that changes in the Model Compare editor will not be applied to the models until you click **Save**. This behavior implies that if changing some feature value has side effects, then they cannot be observed until the models are saved. This is why derived features are ignored by default.

The most basic operations used when merging are copying the feature value and copying an object (together with its containment tree) from one model to its proper place in another model. When an object is copied, Model Compare/Merge ensures that its container is copied too or has a matching object in another model.

After objects are copied, each non-containment reference is set to an object that matches its original setting or to null (in case there is no such an object). If an object in the containment tree of the copied object references an object that is not in the tree and belongs to the same resource as the copied object, the referenced object is copied too, provided it has no matching object already. It is not possible to copy the referenced object if it is not checked in the **Elements to Copy** dialog box.

Copy operations can put models to an invalid state. Model constraint violations (errors and warnings) are listed in the bottom of the **Model Compare** editor. Errors prevent models from being saved. For example, it is impossible to save a Together model when it has external references to another resource.

Related Concepts

Model Transformation Support

Related Procedures

Comparing and Merging Models

Requirements Management

In This Section

Element Traces View

Window ▸ Show View ▸ Other... ▸ Requirements ▸ Element Traces

This topic provides information about the Element Traces view. You can use this view to display existing traces to CaliberRM or RequisitePro requirements.

Columns:

Column	Description
Name	DIsplays the traced element or requirement name.
Project	Displays the requirement project name.

Context menu commands:

Command	Description
Open	Selects the traced requirement in the CaliberRM or **RequisitePro Navigator**, depending on the requirement type.
Remove	Removes the trace.
Convert	Converts the legacy CaliberRM trace to the current format. The option is available only if a trace imported from a Together ControlCenter 6.1 project is selected.

Related Concepts

Requirements Management

Trace Synchronizer View

Window ▸ Show View ▸ Other... ▸ Requirements ▸ Trace Synchronizer

This topic provides information about the **Trace Synchronizer** view. You can use this view to find and fix desynchronized traces to CaliberRM or RequisitePro requirements.

Toolbar buttons and context menu items

Synchronize Traces	Opens the **Trace Synchronizer** dialog box.
Refresh trace synchronization information	Refreshes the trace information displayed in the **Trace Synchronizer** view.
Save as HTML	Opens the **Save As** dialog box, where you can export the current content of the **Trace Synchronizer** view to an HTML file.
Update Trace	Discards local changes and updates the selected traces from the repository.
Restore Trace	Discards changes in the repository and restores the requirement information stored in the model.
Delete Trace	Deletes the trace.
Navigate to Trace Source	Opens the trace source (requirement) in the CaliberRM or RequisitePro Navigator depending on the requirement type.
Navigate to Trace Target	Opens the trace target (model element) in the appropriate editor.

Columns

Status	Displays the status of the trace source.
Trace from	Displays the name of the trace source.
Trace from project	Displays the name of the source CaliberRM or Together project.
Status	Displays the status of the trace target.
Trace to	Displays the name of the trace source.
Trace to project	Displays the name of the target CaliberRM or Together project.
Status summary	Displays the summary information about the current trace status.

Status items

Not Found	Information about the object is not found.
Current	Information about the object is up to date.
Missed	Information about the object is missing.
New	The object is new.
Modified	The object has been modified.
Outdated	The object becomes outdated.

Related Concepts

Requirements Management

Patterns and Templates

Together includes a number of predefined templates that you can apply to your projects. Customize templates using one of the three template editors. Use the **Templates view** to see and manage your templates.

Related Reference

- Apply Template Wizard
- Save As Template Wizard
- Templates View
- Templates View Context Menus
- Template Editors
- Template Properties
- Syntax and Conditions in Templates
- Supported Templates

Patterns and Template GUI Components

This part describes GUI components of the Together interface you use for Together Pattern features.

In This Section

Pattern Explorer

Window ▸ Show view ▸ Other ▸ Patterns and Templates ▸ Pattern Explorer

The **Pattern Explorer** enables you to logically organize patterns (using virtual trees, folders and shortcuts), and manage recognized instances of patterns. You are working with shortcuts in Pattern Explorer, not with the actual patterns. Because of this, shortcuts to the same pattern may be included in several folders.

Context menu command	Description
Delete instance	Deletes a pattern instance from the model. When applied to a model folder, deletes all pattern instances.
Clear invalid instances	Deletes invalid instances of a pattern from the model.
Select on diagram	Sets highlight to the selected pattern instance in diagram. This command is available for pattern nodes.
Select in Model Tree	Sets highlight to the selected pattern instance in the Model Navigator. This command is available for pattern nodes.
Add as shortcut to diagram	Creates a shortcut to the pattern instance on the current diagram.
Verify pattern	Checks validity of the selected pattern.

Related Concepts

Patterns and Templates

Related Reference

Pattern Registry

Pattern Registry

Window ▸ Show view ▸ Patterns and Templates ▸ Pattern Registry

The **Pattern Registry** defines the virtual hierarchy of patterns. You can create virtual folders and group the patterns logically to meet your specific requirements. All operations with the contents of the **Pattern Registry** are performed in the **Pattern Explorer** and synchronized with the **Pattern Registry**.

Pattern Registry shows a tree of folders with shortcuts to patterns. The structure of the pattern registry is a simple tree with two separated subtrees with common root. These separated subtrees represent folder structures: one is taken from Eclipse extensions and another from workspace-specific local data. **Pattern Registry** allows several shortcuts to the same definition in different folders; therefore, internally, the registry keeps the plain list of definitions and a tree of folders and shortcuts to definitions.

The context menu of the pattern shortcuts in the Pattern Registry allows you to rename, copy, and delete the selected pattern. When attempting to delete a shortcut, you are prompted whether the shortcut should be deleted or the definition and all its shortcuts. In case the shortcut is the last one that refers to the corresponding pattern, the program warns you that deleting this shortcut will result in the loss of the corresponding pattern definition.

Command	Description
New	This command is available for the categories. You can choose to create a new nested category or a new pattern.
Rename	Opens the **Rename** dialog, where you can specify the new name of a category or a pattern.
Delete	Deletes the selected category or pattern. Requires confirmation.
Delete pattern definition	This command is available for the patterns. Deletes the pattern definition and all shortcuts to it from the Registry. Requires confirmation.
Copy	This command is available for the patterns. Copies the selected pattern to clipboard.
Cut	This command is available for the patterns. Cuts the selected pattern to clipboard.
Paste	This command is available for the categories. Pastes a pattern from the clipboard to the selected category.
Edit pattern definition	Opens a pattern definition project of the selected pattern.

Related Concepts

Patterns and Templates

Related Reference

Pattern Explorer

Apply Template Wizard

Together provides a wizard to help you make use of templates. The following context menu commands open the Apply Template Wizard:

View	Element/Member and Context Command
Model Navigator	Project. **New** ▶ **Class by Template**
Package Diagram	Package. **New** ▶ **Class by Template**
Model Navigator or Navigator	Class. **Apply Template**
Model Navigator or Navigator	Interface. **Apply Template**
Model Navigator or Navigator	Various elements as applicable. **Apply Template**
Diagram Editor	**Class by Template** button, **Link by Template** button.
Package	.java. **Apply Template**

Note: Templates that are out of the scope are not shown (for example, link and class templates are not shown when invoking the wizard on a package). But templates with the right scope that can't make use of the current selection are grayed.

Options

Option	Description
Templates list	The top left area displays the list of templates in Together. Click a template to display its parameters.
Property name	Displays the properties of the selected template.
Value	Displays property values.
Description	Provides a template description.

Related Reference

Patterns and Templates

Create Pattern from Elements

File ▸ Export ▸ Modeling ▸ Pattern Definition.

Use this wizard to create a pattern from the selected model elements.

Customize page

Item	Description
Pattern name	Enter the name of the new pattern definition.
Select category	Use this field to select the target category for the new pattern definition.
Show existing patterns	Check this option to display the available patterns.
Transformation profile	Check this option to display the transformation profile.
Open pattern definition project after pattern is finished	If this option is checked, the new profile definition project opens for editing.

Set Role Names page

Use this page to edit the role names of the elements involved in the pattern definition

Item	Description
Element name	Displays the name of the element participating in the pattern definition.
Role name	Use this column to edit the name of the pattern participant. By default, the role name is the same as the element name.

Set Default Values page

Use this page to specify the default values for the properties of the roles. Each property has the following set of parameters:

Parameter	Description
Customize value on application	If this option is set to `true`, you can modify the value of this property in the **Model element by pattern** wizard to be set on element creation. This property should be `false` if `Use property on application` is `false`.
Use for recognition	Controls whether to use this property on recognition.
Use property on application	Controls whether to set this property on creating elements by pattern.
Value	If **Use for recognition** is `true`, this value is compared to the property of the element against which the pattern is recognized. If **Use property on application** is `true`, this field defines the default value of the property.

Related Concepts

Patterns and Templates

Related Procedures

Patterns and Templates

Save As Template Wizard

Accessed from the context menu of an element in the Model Package Explorer, Model Navigator or Package Explorer views, this wizard lets you save a template from an existing element.

Name	The name of the new template. The default value is <blank>.
Project List	List of projects for which templates can be created. "Local Templates" contains the templates included with Together. Templates stored here are stored locally. You can also save templates to shared directories for use by a team. For more information, see “Using Version Control and Teams in Together.” For a project to be displayed in the list, it must be template-enabled. (See “Working with the Templates.”) The default value is Local Templates.
Finish	When clicked, generates the basic structure of your template.

Related Procedures

Using Version Control and Teams in Together
Working with the Templates

Templates View

Window ▸ Show view ▸ Other ▸ Patterns and Templates ▸ Templates

The **Templates** view displays currently available templates. These templates can be applied to open projects that have been template-enabled.

Together is shipped with a number of templates for the supported languages (Java, OMG CORBA IDL and C++). By default, each language contains a **Local Templates** node, which includes all the templates predefined and shipped with Together. They are divided into three categories: Link, Class, and Package. Each category contains subcategories related to the individual templates it contains.

Button	Description
Sort templates	Sorts the nodes in the view alphabetically.

Note: New projects will not be displayed in the Templates view until they contain a template.

Related Procedures

Working with the Templates
Editing Templates

Related Reference

Templates View Context Menus

Templates View Context Menus

Project Level

The following context menu options are available at the project level.

New	Opens the **Save As Template** wizard through which you may create a new template of the type selected.
Properties	Opens the **Properties** dialog for the selected resource.

Class Level

The following context menu options are available at the class level.

New	Opens the **Save As Template** wizard through which you may create a new template of the type selected.
Create Category...	Opens a dialog through which you may create a new template category.
Properties	Opens the Properties dialog for the selected resource.

Category Level

The following context menu options are available at the category level.

New	Opens the **Save As Template** wizard through which you may create a new template of the type selected.
Rename	Opens a dialog through which you may rename the selected category.
Delete	Opens a confirmation dialog through which you may delete the selected category.
Restore deleted...	Allows you to restore a deleted resource with a saved version from your local history. For more information, refer to the documentation set provided with your IDE. From the menubar, choose **Help** ▸ **Help Contents**.
Properties	Opens the Properties dialog for the selected resource.

Template Level

The following context menu options are available at the template level.

New	Opens the **Save As Template** wizard through which you may create a new template of the type selected.
Open	Opens the template editor for the selected template. For more information, see “Editing Templates.”
Copy	Opens a dialog through which you may save the selected template to another location.
Rename	Opens a dialog through which you may rename the template.
Delete	Opens a confirmation dialog from which you may delete the selected template.
Properties	Opens the Properties dialog for the selected resource.

Related Procedures

Editing Templates

Related Reference

Save As Template Wizard
Preferences

Template Editors

Each type of template has its own template editor in Together. Use the template editor to alter template values. Double-click on any editor in the Templates view to open the appropriate editor. The tabs on each editor give you access to different parts of the template's code.

Class Template Editor

Use this editor to make changes in class templates. This editor is displayed when you double-click a class template in the Templates view.

Overview

This tab, which is used only with the interface option that lets you designate the template for interfaces, includes:

- The class template Name.
- Description field in which you can enter the description of the template. This text is displayed in the **Apply Template...** dialog description area when you select this template. The description of a template usually describes what the template does and what the parameters (if any) expect and do.
- Help Context ID field for linking to a help page (for use with the extension point `org.eclipse.help.contexts`) if you want to provide F1 context help for the template.

Variables

Use this tab to specify text in the code that the template will generate or for names that are displayed on the various tabs.

Columns include:

- Name
- Label
- Type, for example: `int, float, java.lang.String`
- Value

Imports

Use this tab to specify import statements to add to the class created by this template. Double-click an import line to open the Edit Import dialog.

Attributes

Use this tab to specify the attributes to generate.

Operations

Use this tab to create operations.

Buttons

The Buttons menu options are described in the following table.

Item	Description
Add	On the **Variables**, **Imports**, **Attributes**, and **Operations** tabs. Opens a wizard or new dialog to add new elements. The fields in the wizard correspond to those on the tab.
Edit	On the **Variables** and **Imports** tabs. Opens a wizard (for Variables) or new dialog (for Imports) to change elements. The fields in the wizard correspond to those on the tab.
Remove	On the **Variables** and **Imports** tabs. Removes the selected element. **Note:** Consider this option carefully. You will not be prompted before an element is removed.
Delete	On the **Attributes** and **Operations** tabs. Removes the selected element. **Note:** Consider this option carefully. You will not be prompted before an element is removed.
Rename	On the **Attributes** and **Operations** tabs. Opens a dialog with an editable name field.
Variable	On the **Attributes** and **Operations** tabs. Opens a popup with `className, fieldName, packageName` choices for adding code snippets.

Related Reference

Templates View

Link Template Editor

Use this editor to make changes in link templates. It is displayed when you double-click a link template in the Templates view.

Related Reference

Templates View

Package Template Editor

The Package Template Editor allows you to modify the structure of package templates. This editor is displayed when you double-click a package template in the Templates view.

Overview

This tab includes:

Field	Description
Name	The package template Name.
Description field	Description field in which you can enter the description of the template. This text is displayed in the **Apply Template...** dialog description area when you select this template. The description of a template usually describes what the template does and what the parameters (if any) expect and do.
Help	Help Context ID field for linking to a help page (for use with the extension point org.eclipse.help.contexts) if you want to provide F1 context help for the template.

Variables

Use this tab to specify text in the code that the template will generate or for names that are displayed on the various tabs.

Columns include:

- Name
- Label
- Type, for example: `int, float, java.lang.String`
- Value

Units

Use this tab to create the individual classes that the package template should generate. In the area below the list, you can enter the syntax for each class, or enter comments. As you select different classes in the list at top, the area below displays the syntax for that class. You can also use conditional statements based on the variables you create on the Variables tab.

Buttons

The Buttons menu options are described in the following table.

Button	Description
Add	Opens a wizard to add new variables or units. The fields in the wizard correspond to those on the tab.
Remove	Removes the variable or unit selected. Consider this option carefully. You will not be prompted before an element is removed.
Rename	Opens a dialog with an editable name field.
Edit	Opens the dialog box to change the selected element. The fields in the dialog box correspond to those on the tab.

Delete	Removes the selected element. Consider this option carefully. You will not be prompted before an element is removed.

Related Reference

Templates View

Template Variable Types

Variable Type	Description
String	Simple String entry field.
Boolean	True or false drop-down list.
Class	Enter the fully qualified class name or use the **Browse** button in this field to use the Class Selector dialog.
ClassArray	As the Class type, but supports multiple classes separated by a comma. The **Class Selector** dialog for this type supports multiple class selections.
Fieldname	String name for entry for an attribute.
Interface	Enter the fully qualified interface name or use the **Browse** button in this field to use the **Interface Selector** dialog.
InterfaceArray	As the Interface type, but supports multiple interfaces separated by a comma. The **Interface Selector** dialog for this type supports multiple interface selections.
Methodname	String name for entry of a method.
Packagename	String name for entry of a package.
SimpleTypeName	The value is a (java.lang.)String instance representing a simple Java type name. For example, "MyApplet".
Type	The value is a (org.eclipse.jdt.core..)IType instance representing an existing type/array-of-types from the workspace.
TypeArray	The value is a (org.eclipse.jdt.core..)IType[] instance representing an existing array-of-types from the workspace.
Typename	The value is a (java.lang.)String instance representing a qualified Java type name.

Template Properties

The Properties view can be used alongside the template editors and the Templates view to change the properties of a template.

Property

This column displays the names of the properties of a selected resource.

Property Value

This column displays the values of the properties of a selected resource. Double-click on a value to edit.

Buttons

The Buttons menu options are described in the following table.

Button	Description
Show/Hide Categories	This button groups lines under their appropriate categories.
Filter Properties	This button determines whether advanced properties are displayed in this view. Basic properties are always shown.
Restore Default Value	If you make changes to a value, this button restores the selected property to its default value.
Menu	Displays some additional commands.
Minimize	Minimizes current properties view to the view title. To show the entire view, click the Restore button.
Maximize	Maximizes the current view to the entire window. To restore the view, click the Restore button.

Syntax and Conditions in Templates

Syntax

Together uses Velocity to generate templates. Velocity is a Java-based template engine. This page describes commonly used Velocity syntax.

#foreach($ref **in** arg) statement #end

$ref	The first variable reference.
arg	Can be a reference to a list (i.e. object array, collection, or map), an array list, or the range operator.
statement	The output when Velocity finds a valid item in the arg list. Output is any valid Velocity Template Language statement. Rendered each iteration of the loop.
Reference	#foreach ($part in $whole)
Array list	#foreach ($part in ["Follow", $my, "lead"])
Range operator	#foreach ($part in [1..3])

This example of a foreach loop is on the Units tab in the Unit Test package template in Together.

For a thorough syntax list and explanations, visit: `http://jakarta.apache.org/velocity/vtl-reference-guide.html`

Note: VTL directives (beginning with "#") are displayed in the snippets area as comments.

Variables

Notation:

$ [!][{][a..z, A..Z][a..z, A..Z, 0..9, -, _][}]

Notation examples:

Normal: $long-Thrower_4 Silent: $!long-Thrower_4 Formal: ${long-Thrower_4}

If /ElseIf /Else Conditionals

Format:

#if([condition]) [output] [**#elseif**([condition]) [output]]* [**#else** [output]] **#end**

Usage:

condition: If Boolean, considered true when its value is true. If not Boolean, considered true when not null. output: May contain Velocity Template Language.

Examples:

Equivalent Operator: #if($foo == $bar) Greater Than: #if($foo > 34) Less Than: #if($foo < 34) Greater Than or Equal To: #if($foo >= 34) Less Than or Equal To: #if($foo <= 34) Equals Number: #if($foo == 34) Equals String: #if($foo == "bar")

Foreach Loop

Reference: #foreach ($part in $whole) Array list: #foreach ($part in ["Follow", $my, "lead"]) Range operator: #foreach ($part in [1..3])

This example of a foreach loop is on the Units tab in the Unit Test package template in Together.

For a thorough syntax list and explanations, visit: `http://jakarta.apache.org/velocity/vtl-reference-guide.html`

Note: VTL directives (beginning with "#") are displayed in the snippets area as comments.

Format:

#foreach($ref **in** arg) statement #end

Usage:

$ref: The first variable reference. arg: May be a reference to a list (that is, object array, collection, or map), an array list, or the range operator. statement: The output when Velocity finds a valid item in the arg list. Output is any valid Velocity Template Language statement. Rendered each iteration of the loop.

Examples:

Reference: #foreach ($part in $whole) Array list: #foreach ($part in ["Follow", $my, "lead"]) Range operator: #foreach ($part in [1..3])

This example of a foreach loop is on the Units tab in the Unit Test package template in Together.

For a thorough syntax list and explanations, visit: `http://jakarta.apache.org/velocity/vtl-reference-guide.html`

Note: VTL directives (beginning with "#") are displayed in the snippets area as comments.

When using the template editor to write the code snippets and syntax for fields, methods and classes, you can write conditional logic to control what is created when the template is applied.

The typical format for conditional statements is as follows:

#if([${reservedVariable}] | [${variable from Variable Tab}] [operator] [operand]) ...template code to be generated is inserted here... #end

When you type the `$` character as you write a condition, a pop-up window is displayed allowing you to select one of the current variables, including the reserve variables that apply to this type of template. This variable selected is inserted into the template snippet at the cursor.

An `if` condition can be anywhere within the template code; `if` conditions are granular down to specific lines of code.

Note: Variable names declared on the Variables tab must be preceded by a `$` symbol. Otherwise, they are treated as normal text.

Conditions

When using the template editor to write the code snippets and syntax for fields, methods and classes, you can write conditional logic to control what is created when the template is applied.

The typical format for conditional statements is as follows:

#if([${reservedVariable}] | [${variable from Variable Tab}] [operator] [operand]) ...template code to be generated is inserted here... #end

When you type the `$` character as you write a condition, a pop-up window is displayed allowing you to select one of the current variables, including the reserve variables that apply to this type of template. This variable selected is inserted into the template snippet at the cursor.

An `if` condition can be anywhere within the template code; `if` conditions are granular down to specific lines of code.

Note: Variable names declared on the Variables tab must be preceded by a `$` symbol. Otherwise, they are treated as normal text.

Last Validation Results View

Window ▸ Show view ▸ Other ▸ Patterns and Templates ▸ Last Validation Results

The **Last Validation** view displays results of the latest validation of a pattern definition. This view opens automatically when the validation process reports errors.

Related Procedures

Validating Pattern Definition Projects

Supported Templates

Together includes a collection of predefined templates. You can customize these using the template editors, or create your own templates and share them with team members.

Predefined GoF patterns supplied with the product are only used in the Java modeling projects.

Link Templates

Miscellaneous

Composition

Aggregation

Association

Aggregations

Aggregation as AbstractCollection

Aggregation as AbstractList

Aggregation as AbstactMap

Aggregation as AbstractSequentialList

Aggregation as AbstractSet

Aggregation as ArrayList

Aggregation as Collection

Aggregation as HashMap

Aggregation as HashSet

Aggregation as Hashtable

Aggregation as LinkedList

Aggregation as List

Aggregation as Map

Aggregation as Set

Aggregation as SortedMap

Aggregation as SortedSet

Aggregation as Stack

Aggregation as TreeMap

Aggregation as TreeSet

Aggregation as Vector

Aggregation as WeakHashMap

Associations

Association as AbstractCollection

Association as AbstractList

Association as AbstactMap

Association as AbstractSequentialList

Association as AbstractSet

Association as ArrayList

Association as Collection

Association as HashMap

Association as HashSet

Association as Hashtable

Association as LinkedList

Association as List

Association as Map

Association as Set

Association as SortedMap

Association as SortedSet

Association as Stack

Association as TreeMap

Association as TreeSet

Association as Vector

Association as WeakHashMap

Class and Package Templates

Class Templates

Standard
- Main Method
- Parse XML Document
- String Representation

Enterprise
- Lookup EJBHome

Package Templates

Standard
- Class
- Interface
- Main Class
- Bean
- Applet
- Exception
- RemoteObject
- Logging
- Assertion
- Locale Message
- Unit Test

J2EE Connector
- Connection
- ConnectionFactory
- ManagedConnectionFactory

J2EE Servlet
- ServletFilter
- ServletContextListener
- ServletContextAttributeListener
- HttpServlet
- HttpSessionListener
- HttpSessionAttributeListener>

J2EE ServletTags
- Tag
- TagExtraInfo

TagSupport

BodyTag

BodyTagSupport

J2EE JMS

JMS Queue

JMS Topic

J2EE, TagLibs, J2EE JMS Templates

J2EE Templates

- App Event Listeners
- ServletFilter
- ServletContextListener
- ServletContextAttributeListener
- HttpServlet
- HttpSessionListener
- HttpSessionAttributeListener
- Lookup EJBHome
- Connection
- ConnectionFactory
- ManagedConnectionFactory

TagLibs Templates

- Tag
- TagExtraInfo
- TagSupport
- BodyTag
- BodyTagSupport

J2EE JMS Templates

- JMS Queue
- JMS Topic

GoF Templates

GoF Patterns

GoF patterns are predefined. These patterns can be applied and recognized in Java-modeling projects only. Modification of the GoF patterns is not allowed.

Creational
Abstract Factory
Builder
Factory Method
Prototype
Singleton
Behavioral
Command
Chain Of Responsibility
Interpreter
Iterator
Mediator
Memento
Observer
State
Strategy
Template Method
Visitor
Structural
Adapter
Bridge
Composite
Decorator
Facade
Flyweight
Proxy

Quality Assurance

In this part you will find reference information about the command line QA tools and their syntax, and API information that enables you to create audits and metrics of your own.

In This Section

Model Audits and Metrics Descriptions

Use QA Model Preferences (**Window** ▸ **Preferences** ▸ **Modeling** ▸ **QA Model**) for viewing and editing model audits and metrics. Together includes the following predefined audits and metrics:

Model Audits

Name	Short Description	Long description
AAFC	Avoid (Weak) Aggregation, Favor Composition (Strong Aggregation)	In general, aggregation is not well defined and leads to confusion in modeling. Specifically, it is the hollow diamond weak aggregation that is to be discouraged, in preference to the filled-in diamond strong aggregation element commonly called composition. Fowler, 68
AIM	Always Indicate Multiplicity	In many modeling environments, it is possible to create an association without specifying multiplicity on the association ends. This is advantageous during the beginning stages of modeling, but may confuse or complicate generation facilities. Ambler #88 Frankel, 186 Frankel, 178
AIMAE	Always Indicate Multiplicity	In many modeling environments, it is possible to create an association without specifying multiplicity on the association ends. This is advantageous during the beginning stages of modeling, but may confuse or complicate generation facilities. Ambler #88 Frankel, 186 Frankel, 178
AIN	Always Indicate Navigability	A lack of navigability in UML indicates either bidirectional, or non-specified navigability. This is fine for human interpretation with a common understanding, but may confuse or complicate generation facilities. Frankel, 168
AMIMM	Avoid Multiplicities Involving Minimums and Maximums	Fixed range multiplicities are not recommended as they are less flexible than 1..* or 0..* and represent commonly used collections as found in most object-oriented languages today. Ambler #102
AMIMMAE	Avoid Multiplicities Involving Minimums and Maximums	Fixed range multiplicities are not recommended as they are less flexible than 1..* or 0..* and represent commonly used collections as found in most object-oriented languages today. Ambler #102
AMM	Avoid '*' Multiplicity	A multiplicity of * is ambiguous and should be avoided as it is not clear whether 0..* or 1..* was intended. Ambler #89
AMMAE	Avoid '*' Multiplicity	A multiplicity of * is ambiguous and should be avoided as it is not clear whether 0..* or 1..* was intended. Ambler #89
ANA	Always Name Associations	Generators will require names on all associations and association ends. Those names provided by the modeler are likely to be more readable/useful than those generated. Frankel, 186
IRNOAE	Indicate Role Name on Association Ends	Generators will require names on all associations and association ends. Those names provided by the modeler are likely to be more readable/useful than those generated. Frankel, 186 Ambler #98

IRNOAEAE	Indicate Role Name on Association Ends	Generators will require names on all associations and association ends. Those names provided by the modeler are likely to be more readable/useful than those generated. Frankel, 186 Ambler #98
IRNORA	Indicate Role Names on Recursive Associations	-
AUD	Avoid Using Dependencies	Manually added semantic dependencies in UML diagrams are problematic to generators and cannot be enforced due to insufficient meaning. Dependencies do not exist in MOF and are to be discouraged in the UML. Frankel, 159
NPGOIT	Never Place Guard on Internal Transition	A guard placed on an initial transition that evaluates to false renders the diagram useless and should be avoided. Ambler #186
UPNAEMS	Use Plural Names on Association Ends with Multiplicity > 1	A plural name on association ends that have multiplicities > 1 allow for improved model readability. Fowler, 39
UPNAEMAE	Use Plural Names on Association Ends with Multiplicity > 1	A plural name on association ends that have multiplicities > 1 allow for improved model readability. Fowler, 39
UPNAEMC	Use Plural Names on Association Ends with Multiplicity > 1	A plural name on association ends that have multiplicities > 1 allow for improved model readability. Fowler, 39
AGBUC	Avoid Generalization Between Use Cases	The use of generalization between use cases is rare and not commonly understood. This relationship should be avoided in favor of <<include>> and <<extend>>. Ambler #45
AUIE	Avoid <<uses>>, <<includes>>, and <<extends>>	These stereotypes are deprecated and should not be used. Ambler #46
AUIEAE	Avoid <<uses>>, <<includes>>, and <<extends>>	These stereotypes are deprecated and should not be used.&#10;Ambler #46
AAC	Avoid Association Classes	Association classes can be decomposed into a separate class that associates two others. Association classes are not supported by MOF and may confuse some generators. Some generators may support them, but will likely decompose them anyway. Frankel, 159
ACD	Abstract Class Declaration	If a class marked abstract does not provide any abstract methods, or if a class that is marked as abstract contains one or more public constructors, this audit will be flagged.
ONAMAM	Overriding Non-Abstract Method with Abstract Method	A child class should not override with an abstract method a non-abstract method found in a parent class.
ASTP	Always Specify Type on Parameters	Many UML modeling tools default to a primitive type assignment. Others may leave a <<null>> in place, which may prove problematic for generators. It is therefore recommended that type information be provided for all attributes and parameters. Frankel, 185
ASTA	Always Specify Type on Attributes	Many UML modeling tools default to a primitive type assignment. Others may leave a <<null>> in place, which may prove problematic for generators. It is therefore recommended that type information be

		provided for all attributes and parameters. Frankel, 185
HIA	Hiding Inherited Attribute	A child class should not declare an attribute of the same name and type as is found in its parent.
HISM	Hiding Inherited Static Method	Inherited static methods should not be hidden by same-named methods in child classes.
SHSA	Subclasses Have the Same Attribute	If two or more direct subclasses of a class or interface define a field of the same signature, a refactoring may be in order to pull up the field. These cases are identified by this audit.
SHSO	Subclasses Have the Same Operation	If two or more direct subclasses of a class or interface define a field of the same signature, a refactoring may be in order to pull up the field. These cases are identified by this audit.
CSODOI	Components Should only Depend on Interfaces	Many UML drawing tools allow for dependencies to be drawn from one component to another. This is discouraged in favor of indicating only dependency relationships between the interfaces of a component. Ambler #226
AESHD	All elements should have descriptions	This audit checks not all elements but only classifiers and states. Users can adopt it to their own needs if necessary
SSBTR	States should belong to Regions	In valid models, all states should be owned by regions of StateMachine or other states.
ATEDMHG	All Transitions Exiting a Decision Must Have Guards	To ensure all cases are covered, each outgoing transition should have a guard indicated. Ambler #195
ABHS	Avoid "Black Hole" States	Other than End states, no state should have an incoming transition without an outgoing transition. Ambler #169
FSHOOET	Forks Should Have Only One Entry Transition	Some UML tools allow for the drawing of multiple incoming transitions to a fork. A fork should only have a single incoming transition and more than one exiting transition. Ambler #202
FSHMTOET	Forks Should Have More Then One Exiting Transition	Some UML tools allow for the drawing of multiple incoming transitions to a fork. A fork should only have a single incoming transition and more than one exiting transition. Ambler #202
JSHOOET	Joins Should Have Only One Exit Transition	Some UML tools allow for the drawing of multiple outgoing transitions from a join. A join should only have a single outgoing transition and more than one incoming transition. Ambler #203
JSHMTOET	Joins Should Have More Then One Entry Transition	Some UML tools allow for the drawing of multiple outgoing transitions from a join. A join should only have a single outgoing transition and more than one incoming transition. Ambler #203
AMS	Avoid "Miracle" States	Other than Start states, no state should have an outgoing transition without an incoming transition. Ambler #170
ICKJ	Identifier Conflicts with Keyword	Java language keywords should not be used as a model elements names.

AUA	Avoid Unassociated Actors	An actor with no association to a use case provides no value to a diagram and should therefore be avoided. Ambler #35
CSI	Class Should be Interface	An abstract class that contains only abstract methods and final static fields should be declared as an interface.
CWSCJ	Conflict With System Class	Classes should be given names that will not cause potential conflicts with standard Java API classes.
BSMSNHCP	A state machine as the method for a behavioral feature cannot have entry/exit connection points	-
CPSBTSM	Pseudostates of kind entryPoint can only be defined in the topmost regions of a StateMachine	-
SSSHSM	If state is submachineState, submachine should be defined for it	-
SSSNHR	A state is not allowed to have both a submachine and regions	-
FSMNHGT	A fork segment must not have guards or triggers	-
JSMNHGT	A join segment must not have guards or triggers	-
FSMTS	A fork segment must always target a state	-
JSMOFS	A join segment must always originate from a state	-
TFPMNHT	Transitions outgoing pseudostates may not have a trigger	-
OSSCHCPR	Only submachine states can have connection point references	-
PIPSBPT	Provided Interface of a Port Should Be the Port Type or one of the interfaces realized by the port type	-
IMVMBP	The visibility of all features owned by an interface must be public	-
POEVMBPOP	If an element that is owned by a package has visibility, it is public or private	-
SMHFD	Slot must have defined feature	-
SFDMBU	One structural feature (including the same feature inherited from multiple classifiers) is the defining feature of at most	-
DFSBSFOC	The Defining Feature of each slot Should Be a Structural Feature (directly or inherited) Of a Classifier of the instance specification	-
ACHATUCC	An actor can only have associations to use cases, components and classes; furthermore, these associations must be binary	Only binary associations between Actor and Use Case, Class or Component are valid from the point of UML.
UCNHATSSU	UseCases cannot have Associations to UseCases specifying the same subject	-
UCNIUTII	A use case cannot include use cases that directly or indirectly include it	-
ACIIC	Avoid cyclic inheritance in classifiers	Cyclic inheritance is nonsense and reflects an incorrectness of used model.

UCNEUTEI	A use case cannot extend use cases that directly or indirectly extend it	-
AQA	Avoid qualified associations	Qualified associations are not popular among most UML modelers and can be decomposed to a class representing the association with an attribute representing the qualifier. Frankel, 159
AQAAE	Avoid qualified associations	Qualified associations are not popular among most UML modelers and can be decomposed to a class representing the association with an attribute representing the qualifier. Frankel, 159
CORTDNMOT	<<Create>> Operation Return Type Does Not Match Owner's Type	For operation with stereotype <<create>>, return type should be either an operation's owning classifier or not specified.
IAWNPV	Interface Association With Not Public Visibility	-
ROFSHIV	Read-Only Field Should Have Init Value	-
AMIU	Avoid Multiple Inheritance Usage	Multiple inheritance is not supported in some OO languages (like Java). Aggregation or Implementation links could compensate this limitation.
PTSOC	Primitive Type Should be OCL-Compatible	This audit contains a list of primitive types supported by OCL. All the rest of primitive types should not be used together with OCL constraints referencing them.
AMRA	Avoid Modeling Return Arrows	To reduce clutter on diagrams, the explicit modeling of return arrows on messages is discouraged.
AMOOD	Avoid Modeling Of Object Destruction	To reduce clutter on diagrams, the explicit modeling of object destruction is discouraged. This is particularly the case in languages such as Java where the actual destruction of an object is up to the virtual machine. Ambler #131
ICKC	Identifier Conflicts with keyword	C++ language keywords should not be used as a model elements names.
ICIC	Identifier Contains Incorrect Character	These characters are not a valid part of an identifier for most of the programming languages.

Model Metrics

Name	Short Description
NOOA	Number Of Owned Attributes
NOOO	Number Of Owned Operations
NOOPO	Number Of Owned Public Operations
NLOC	Nesting Level Of Class
NOA	Number Of Ancestors
NODA	Number Of Direct Ancestors
NOIO	Number Of Inherited Operations
NOIA	Number Of Inherited Attributes
NOOAS	Number Of Outgoing Associations

NOCDO	Number Of Classes this one Depends On
NOOPT	Number Of Operation Parameter Types
NOCIP	Number Of Classes In Package
NOCIPR	Number Of Classes In Package Recursively
NOOICOP	Number Of Operations In Classes Of Package
NOAC	Number Of Abstract Classes In the Package
AR	Abstractness Ratio
NLOP	Nesting Level Of Package
NOEPUC	Number Of the Extension Point of this Use Case
NOIUC	Number Of Included Use Cases
NOEUC	Number Of Extended Use Cases
NOSISM	Number Of States In State Machine
NOOT	Number Of Outgoing Transitions
NOCABAC	Number Of Classes Associated by Association Class

Related Procedures

Viewing and Finding QA Descriptions
Using OCL in Model Audits and Metrics

Related Reference

Quality Assurance
QA Model

Audit and Metric Sample Project

This topic provides a sample Audit and Metric project, including its structure, configurations, and considerations for running it.

Project structure

Item	Description	
sample.properties	Auto-generated file that contains a short sample description.	
Plugin.xml	Class declaration of the plug-in used by the QA extension of JUnit test framework.	
	List of dependencies from required plugins	
	com.borland.sapient.core	Contains API for writing source code audits and metrics.
	com.borland.sapient.audit	Required for writing audits to avoid errors during execution.
	com.borland.sapient.metric	Required for writing metrics to avoid errors during execution.
	com.borland.sapient.test	Contains QA extension of JUnit test framework required for writing unit tests.
	org.eclipse.core.runtime	Required for UserPlugin, because it extends org.eclipse.core.runtime.Plugin.
	Extension point	
	com.borland.sapient.core.plugins	Shows that the plug-in contains extensions for QA framework.
sapient.xml	Describes extensions for QA framework.	
sapient.xsd	XSD for sapient.xml.	
sapient.properties	Resources file for sapient.xml.	
cases folder	Contains test cases for unit test in the sample.	
pre_build.xml	ANT file with jar task that creates cases.jar from files located in the **cases** folder. You should run “Ant Build” using this file every time you change the contents of the **cases** folder.	
com.borland.sapient.examples.audit package	UserNC	A class that represents an example of a user audit.
	UserAuditMessages.properties	Contains messages from the UserNC audit.
com.borland.sapient.examples.metric package	UserNOO class that represents an example of a user metric.	
com.borland.sapient.examples.audit.test package	UserAuditTestSuite	A class that represents an example of using QA extension of JUnit test framework.
	UserPlugin (extends org.eclipse.core.runtime.Plugin)	A class required for initializing QA extension of JUnit test framework.

docs/java folder	Contains HTML descriptions for audits and metrics.

sapient.xml

sapient.xml contains description of audits and metrics provided by this plug-in. Information in this file is used for loading audits and metrics as well as for specifying additional parameters thus; user audits and metrics are included into the general list of existing audits and metrics. sapient.xml is localized according to the rules in Eclipse (attributes with the `name` name are usually localized). Localized resources should be located in the sapient.properties file. The sapient.xml file structure is described with sapient.xsd.

The following are some explanations of the peculiarities that are not described in the schema.

description element	An optional auxiliary element.
`library` attribute of the `deployment` element	Specifies a library (jar) that contains audit and metric classes.
`category` attribute of the `inspector` element	Must be set to `audit` to describe audits and to `metric` to describe metrics.
category element	Corresponds to the displayed node in the tree of audits and metrics.
analyzer element	Describes an audit or metric.

Attribute	Description
id	An abbreviation of the audit or metric name that is used to open the description HTML file. The id must match the file name.
implementation	Correspond to a full audit or metric class name.

parameter element	Describes the audit or metric parameter. There is a set of standard parameters that can be applied to all audits or metrics. All parameters are available in the interface except the language parameter.
Standard `id` attribute values of the `parameter` element for metrics	

Value	Description
Aggregation	Method to calculate a metric for internal objects.
PackageUpperLimit	Upper limit for a package (for Java).
PackageLowerLimit	Lower limit for a package (for Java).
NamespaceUpperLimit	Upper limit for namespace (except Java).
NamespaceLowerLimit	Lower limit for namespace (except Java).
ClassUpperLimit	Upper limits for a class.
ClassLowerLimit	Lower limits for a class.

`type` attribute values of the `parameter` element	

Value	Description
boolean	`value` attribute can be true or false.
integer	`value` attribute can be any integer.
string	`value` attribute can be any string.
list	A composite parameter composed of subparameters. All subparameters must be of the same type. The structure of the list can be changed via UI. The value of the list parameter is the set of values specified in the list.
enum	A composite parameter composed of subparameters. All subparameters must be of the same type. The structure of the list can be changed via UI. The value of the enum parameter is one of the values specified in the list.

Using API for creating your Audits and Metrics

Together is shipped with a modified API to simplify the creation of audits, metrics, and audit test cases.

Note: A previous method of creating audits can be used as well.

When using the new API, a class should be inherited from `com.borland.sapient.core.audit.AuditRule`.

To create an audit, redefine necessary methods of the base class using the API and write a code that performs analysis and displays messages.

To create TestSuite for the JUnit test, extend the `AuditTestSuite` class, which is documented in the Test API Overview.

Metric creation differs in the base class that can be one of the following:

- com.borland.sapient.core.metric.ClassMetric
- com.borland.sapient.core.metric.MethodMetric
- com.borland.sapient.core.metric.PackageMetric
- com.borland.sapient.core.metric.ProjectMetric

Running the Sample

To see how a custom audit and metric is added to the list of audits and metrics in Together, create an Eclipse Application launch configuration to run the Audit and Metric Sample Project. The simplest way to create this launch configuration is to right-click the Audit and Metric Sample project node in the **Navigator** and select **Run As** ▶ **Eclipse Application**.

Note: After you create a launch configuration, you can run it from the drop-down menus of the **Run** or **Debug** toolbar buttons.

Borland recommends you add the following values to your launch configuration in the **VM arguments** field of the **Arguments** tab: `-Xms128M -Xmx1024M -XX:MaxPermSize=256M`.

To run the unit test included in the sample, create a JUnit Plug-in Test launch configuration. The simplest way to create this launch configuration is to right-click the Audit and Metric Sample project node in the **Navigator** and select **Run As** ▶ **JUnit Plug-in Test**. Edit your launch configuration to choose **Run an application** ▶ **[No Application] – Headless Mode** on the **Main** tab.

Project Documentation

In This Section

Documentation Generation

This section contains reference information about command line utilities.

In This Section

Gendoc Utility Syntax

The following is a syntax for the `gendoc.cmd` command:

```
gendoc.cmd [project filename [package name]] [options]
```

where:

Parameter	Description
[project filename]	Path to the project you want to export to documentation. Note that the `<-sourcepath>` and `<-classpath>` options are ignored, as are package names.
[package name]	Specifies package to export to documentation.
[options]	See the list of available options below.

Option	Description
–help	Displays command line options
–d <directory>	Specifies the destination directory for output files
–data	Locates the project's workspace when it differs from the default workspace
–template	Specifies the name of the default template or path to the template file
–format	Specifies the documentation format: HTML, TXT, RTF, PDF or XSL-FO
–nodiagrams	Specifies that diagram pictures are not created
–hyperlinks	Includes the contents of hyperlinked files into documentation
–audits	Includes the contents of the audits results into documentation
–browser	Launches HTML browser

Related Concepts

Project Documentation

Related Procedures

Generating Project Documentation from Command Line

Genhtml Utility Syntax

The following is a syntax for `genhtml.cmd` command:

```
genhtml.cmd [project name [package name]] [options]
```

where:

[project filename]	Path to the project you want to export to documentation. Note that the `<-sourcepath>` and `<-classpath>` options are ignored, as are package names.
[package name]	Specifies packages to export to HTML documentation.
[options]	See the list of available options below.

Option	Description
–overview <file>	Reads overview documentation from HTML file
–public	Shows only public classes and members
–protected	Shows protected/public classes and members (default)
–package	Shows package/protected/public classes and members
–private	Shows all classes and members
–help	Displays command line options
–d <directory>	Specifies the destination directory for output files
–data	Locates the project's workspace when it differs from the default workspace
–use	Creates class and package usage pages
–version	Includes `@version` paragraphs
–author	Includes `@author` paragraphs
–splitindex	Splits index into one file per letter
–windowtitle <text>	Specifies the browser window title for the documentation
–doctitle <html code>	Includes title for the package index (first) page
–header <html code>	Includes header text for each page
–footer <html code>	Includes footer text for each page
–bottom <html code>	Includes bottom text for each page
–nodeprecated	Does not include `@deprecated` information
–nodeprecatedlist	Does not generate deprecated list
–notree	Does not generate class hierarchy
–noindex	Does not generate index
–nohelp	Does not generate help link
–nonavbar	Does not generate navigation bar
–recurse	Creates output for packages specified in `[packagenames]` and their subpackages
–javadoc	Creates the same output as `javadoc.exe` produces
–audits	Includes audits results in the generated documentation
–browser	Launches HTML browser
–nodiagrams	Does not create diagrams' pictures
–nonavtree	Does not generate navigation tree

Related Concepts

Project Documentation

Related Procedures

Generating Project Documentation from Command Line

Documentation Template Designer

This part contains information about the Documentation Designer tool, structure and controls of documentation templates. You can also find reference information about the variables, functions and OCL expressions used in custom templates.

In This Section

Area Properties

Template Designer ▸ Static section, header or footer ▸ Details pane ▸ context menu ▸ Area Properties

Use this dialog box to view or edit area properties of a static section, header or footer.

Settings tab

Contains check boxes for page settings and for suppressing formatting.

Hypertext Target tab

Any generated output that contains an anchor or bookmark can be a link target. Use this tab to insert anchors at the "main documentation" of model elements.

Option/Button	Description
Expression for Target Bookmark Selector	Inserts a bookmark into a file used as File Link target. Result of evaluating this expression should match the result of Bookmark Name Expression set for File Link reference. Click the **Edit Expression** button to create the expression in OCL or legacy notation.
Start of the current element's specific documentation	Identifies the output of this section as the "main documentation" for the current element. When DocGen processes the section, it inserts a hypertext anchor or bookmark into the output, automatically generating its name. DocGen recognizes this section as the element's main documentation.
Expression for Documentation Subject Selector	This option should be used in conjunction with Start of the current element's specific documentation option. It marks the location of the current element's specific documentation with the appropriate Documentation Subject Selector. Result of evaluating this expression should match the Expression for Documentation Subject Selector set for link reference. Click the **Edit Expression** button to create the expression in OCL or legacy notation.

Other tab

Use this tab for associating formatting styles with an area, setting style name expressions, and using a control delimiter.

Option/Button	Description	
Formatting style	Select the desired formatting style from the list of available styles.	
Reset all controls in the area with this style	Applies the selected formatting style to all controls in the area. The individual style of each control will be suppressed.	
Style Name Expression	Enter the style name expression in the text field.	
Control Delimiter	View or edit control delimiter options.	
	Default	If this option is checked, the default delimiter is used, and the font settings fields are disabled.

Related Procedures

Setting Area Properties

Call to Stock Section Properties

Template Designer ▸ Call to Stock Section context menu ▸ Properties

Use this dialog to view or edit properties of a call to stock section.

Call To tab

This tab lists the available stock sections and highlights the name of the stock section that is actually called.

Other tab

Use this tab to define enable condition and template parameters.

Option/Button	Description
Left indent (mm)	Specify indentation.
Parameter Expression	Lets you specify string parameter of the stock section call. Within stock section, this parameter can be retrieved using getDGVariable('stockParam'). Click the **Edit Expression** button to create an expression in OCL or legacy notation.
Enable condition	An enable condition for turning this section on or off. Use the Expression editor to specify enable condition using the OCL or legacy notation.
Disabled	Check this option to skip the section.

Related Concepts

Enable Conditions

Related Procedures

Creating Stock Sections

Call to Template Properties

Template Designer ▸ Call to Template section context menu ▸ Properties

Use this dialog to view or edit properties of a call to template section.

General tab

Option/Button	Description
Template	Assigns a template that is invoked by the Call to Template section. Click the **Browse** button to choose the actual template to be called.
Output Settings	Gives a choice of where the output for the called template goes.

Separate file	This is important for generating multiframe HTML documentation consisting of separate HTML documents that are extensively linked together. If this option is selected, the following fields are displayed:

Output Filename Expression	Enter the name of the document. This expression should not include the file path. If this field is blank, the generated document is named according to the name of the called template. Click the **Edit Expression** button to create the expression in OCL or legacy notation.
	
	Example:
	
	`context uml::kernel::NamedElement`
	
	`name.concat('.Dia')`

Output Directory Expression	Enter the path to the destination directory of the generated document. If the expression contains directories that do not yet exist, they will be created when the template is processed. Click the **Edit Expression** button to create the expression in OCL or legacy notation. This path is always relative. Define it according to the following conventions: 1. If the calling template is a frameset template, the path is relative to the destination directory for the entire documentation. 2. If the calling template is a document template, the path is relative to the location of the document that is generated by the calling template. 3. The right slash character (`/`) is the name-separator for the path.
Output Image Subdirectory Expression	Enter the path to the directory for the images files of the generated document. Click the **Edit Expression** button to create the expression in OCL or legacy notation. **Example:**

		`context OclAny` `'../doc-images'`
	Do not create file with empty output	Check this option to skip empty files.
Common stream	The called template behaves like a stock section. If this option is selected, the following field is available:	
	Left indent (mm)	The called template provides output to the same file as the calling template, and you can only specify indentation if required.

Parameters tab

A calling template can pass additional information to the called template through template parameters. The parameter value can be obtained in a template body using the `String getParam(String paramName)` function.

Parameter	Enter parameter name.
Expression	Displays the parameter expression. Click the **Edit Expression** button to create the expression in OCL or legacy notation.
Set	Adds parameter to the list. This button is only enabled when the **Parameter** field is not empty.
Delete	Removes the selected parameter from the list. This button is only enabled when a parameter is selected in the list.

Other tab

Use this tab to define enable condition.

Item	Description
Enable condition	An enable condition for turning this section on or off. Use the Expression editor to specify enable condition using the OCL or legacy notation.
Disabled	Check this option to skip the section.

Related Concepts

Enable Conditions

Control Properties

Template Designer ▸ Section's details pane ▸ Control context menu ▸ Properties

This dialog enables you to define properties of the various template controls. Composition of the tabs depends on the selected control.

Label Control Properties

Tab	Description	
Label tab	Use this tab to specify the label text.	
Font tab	Use this tab to define family, size, style and alignment.	
Color tab	Use this tab to define foreground and background color.	
Border tab	Use this tab to define foreground and background color, and line styles of the borders.	
Hyperlinks to Elements tab	Use this tab to create link references.	
	Option/Button	**Description**
	Type	Click radio button to select the hyperlink type.
	Link Settings	Fields in this section depend on the selected link type.
	Add Hyperlink2	Creates a compound hyperlink for a single control. Adds a **Hyperlinks to Elements 2** tab.
	Delete Hyperlink2	Deletes the second hyperlink from the current control and deletes its tab from the Properties. This button is displayed if the second hyperlink exists.
Other tab	Formatting Style	Select a formatting style from the list.
	Render embedded HTML tags	If this option is checked, HTML tags encountered in the text are recognized and visualized appropriately.
	Render embedded Javadoc tags	If this option is checked, Javadoc references in the `{@link}` tags are converted into hypertext links.

Image Control Properties

For the descriptions of the **Border** and **Hyperlinks to Elements** tabs, refer to the *Label Control Properties* section.

Option / Button	Description	
Image tab	Choose an image type from the drop-down list. The other controls of this tab depend on the selected image type.	
	Static (URL)	**URL**: Enter the fully qualified path in the URL field, or click the Browse button and navigate to the image. Note that GenDoc functions/variables cannot be used in expressions for Static (URL).
	Static (Resource)	**Image Resource Expression**: Enter the expression that describes the relative path to the resource. This option is deprecated. Note that GenDoc functions/variables cannot be used in expressions for Static (Resource).
	Diagram	Use this option to output the diagram image. Note that Documentation Generator, while processing the section, will create an image only if the

	current model element represents a model diagram.
Element's Small Icon / Element's Large Icon	If one of these options is selected, the respective element icon is displayed in the generated documentation.

Panel Control Properties

For the descriptions of the **Border**, **Color** and **Other** tabs, refer to the *Label Control Properties* section.

Option / Button	Description
Fill panel with the output of stock section	Select the stock section, which will produce its output to the panel.
Parameter Expression	Specify the parameter for stock section call. You can query the specified parameter using the `getDGVariable('stockParam')` function.
Passed Model Element Expression	Use this field to define an element that will be used as a root element for the stock section called from this panel

Formula Control Properties

For the descriptions of the **Border**, **Color**, **Hyperlinks to Elements** and **Other** tabs, refer to the *Label Control Properties* section.

Data Control Properties

For the descriptions of the **Border**, **Color**, **Hyperlinks to Elements** and **Other** tabs, refer to the *Label Control Properties* section.

Option / Button	Description
Source tab	Data Source: Select data source from the drop-down list.

Include Text Control Properties

For the descriptions of the **Border**, **Color**, and **Font** tabs, refer to the *Label Control Properties* section.

Option / Button	Description
Text File URL Expression	Specify the relative path to the file that contains the desired text.

Related Concepts

Documentation Template Controls

Related Procedures

Creating Controls

DG functions in Formulae Expressions

This section gives a brief description of the major legacy functions used in the doc generation module.

getDGVariable

`String getDGVariable(String variableName)`

Returns the specified DG variable of **String** type.

Parameter: name of the variable

Returns: value of the variable or an **empty** string if the variable is not defined in the given place

getDGRwiElement

`RwiElement getDGRwiElement(String variableName)`

Returns the specified DG variable of RwiElement type.

Parameter: name of the variable

Returns: value of the variable or **null** , if the variable is not provided in the given place

getDGRwiProperty

`RwiElement getDGRwiProperty(String variableName)`

Returns RwiProperty type DG variable with the specified name.

Parameter: name of the variable

Returns: value of the variable or **null** if this variable is not in place

getDGOption

`String getDGOption(String optionName)`

Returns the specified DG option.

Parameter: name of the option

Returns: value of the option or empty string if the option is not defined

Features:

The option can be specified for an object of Report Generator (descendant of the `..gendoc.docgenerator.Generic.GnrReportGenerator` class, for example: class `..gendoc.docgenerator.txt.TXTReportGenerator`) using the method: `addReportOption (String optionName, String optionValue)`

Default values for some options can be defined in the template file. This definition persists even though the Template Designer subsequently modifies the template. However, the `addReportOption` method overwrites the options values.

Example:

default values for the options `inclSubpackages", "inclDoc", "DTLAdapter"`

```
DEFAULT_OPTIONS=
{inclSubpackages='yes';inclDoc='yes';DTLAdapter='com.togethersoft.modules.doorslink
.DTLAdapter'}
```

getParam

`String getParam(String paramName)`

Returns the value of the specified template parameter.

Parameter: parameter name

Warning:

The requested parameter should be declared in the Template Parameters tab of Template Properties. If the parameter declaration is not defined, calls to this function will cause an error message and stop the generator.

Since: Together 5

invokeForName

`String invokeForName(String className, String methodName)`

`String invokeForName(String className, String methodName, String param1)`

`String invokeForName(String className, String methodName, String param1, String param2)`

Invokes specified method of the user-provided class.

Parameters:

- `className`: Fully qualified name of the user-provided class. This class should not be abstract. The Documentation Generator creates an instance of the `className` class and calls the `methodName` method with this instance. Note that this instance object is created for each entry of `invokeForNamecall` within each particular expression of the template where this function is used. However, the object is created only during the first call from such an entry and will be used for the next calls unless the `className` parameter is changed.
- `methodName`: name of the method in the class to be executed. The method should have the following signature: `String methodName (..gendoc.api.GenDocContext)`

Parameter is an instance of the class `..gendoc.api.GenDocContext` that provides the following methods:

- `RwiReference getRwiReference`: Returns RwiReference if the current DG iteration element is an RWI reference within a diagram. Otherwise, the method returns **null.**
- `RwiElement getRwiElement`: Always returns the RwiElement. If the current DG iteration element is an RWI reference, the returned element is rwiReference.getElement(). Otherwise, returned element is the current DG iteration RWI element.
- `String getParameter1()`: Returns the value of the first optional parameter passed to the invokeForName function, or null if the parameter is omitted.
- `String getParameter2()`: Returns the value of the second optional parameter passed to the `invokeForName` function, or null if the parameter is omitted.

Returns: Value calculated by the user-provided `methodName` method.

getContainingDiagram

`RwiDiagram getContainingDiagram()`

Returns the RWI diagram containing the primary reference to the current element.

Example: `rwiElement-> getContainingDiagram()`

Returns: RWI diagram containing the primary reference to the current element.

isDiagram

`boolean isDiagram()`

Tests if the current RWI element is a diagram. Call this function to test any RWI element accessible in your expression.

Examples:

`rwiElement->isDiagram()`

`getDGRWIElement("diagramMapElement")->isDiagram()`

This function may be useful when you design a Multi-Frame documentation and need to program some special behavior when clicking hyperlinks. For example, if a hyperlink references to a diagram you may want when clicking it to reload one frame with a document describing the diagram and another frame with the graphic chart of this diagram. Whereas, if the hyperlink references to any other model element only document frame should be reloaded. See also: Creating compound Hyper-References.

Returns: **True** if the element is a diagram; **False** otherwise.

isImported

`boolean isImported()`

Checks if the current element in the diagram is presented by a shortcut.

Returns: True if the element is a shortcut; **False** if the element is not a shortcut.

getSubproperty

`String getSubproperty(RwiProperty rwiProperty, String subpropertyName)`

The function returns the value of the `subpropertyName` subproperty contained in the `rwiProperty_` RWI property. See the description of the `curPropertyInstance` DG variable for an example of using this function. A possible call is: `rwiProperty->getSubproperty(subpropertyName)`

Parameters:

`rwiProperty` The element property

`subpropertyName` The name of its subproperty

Returns: Value of the specified subproperty

hasSubproperty

`String hasSubproperty(RwiProperty rwiProperty, String subpropertyName)`

Checks if the `rwiProperty_` RWI property contains the `subpropertyName` subproperty.

A possible call is: `rwiProperty->hasSubproperty(subpropertyName)`

Parameters:

- `rwiProperty` The element property
- `subpropertyName` The name of subproperty to be checked

Returns: True if the property has the specified subproperty; **False** otherwise.

getJDRefType

`String getJDRefType(String jdref)`

Returns type of the JavaDoc Reference specified as the parameter.

Returns: "element" if `jdref` references a model element (that is, if it has the form `package.class#member` label);

"url" if jdref references a URL (that is, if it has the form `<a href="URL#value">label</a>`).

"text" if jdref has the form "string"

Since: Together 5

getJDRefDisplayName

`String getJDRefDisplayName(String jdref)`

Returns a text to be displayed in place of the specified JavaDoc Reference.

Returns:

- if jdref is an "element" reference (that is, it has the form `package.class#member` label, where `package.class#member` represents a model element) the returned text is the label. If the label is omitted, returns the name of the referenced element.
- if jdref is a "url" reference (that is, it has the form `<a href="URL#value">label</a>`) the returned text is the label.
- if jdref has the form "string", the returned text is the string.

Since: Together 5

getJDRefElement

`RwiElement getJDRefElement(String jdref)`

If the specified JavaDoc Reference is an "element" reference (that is, it has the form `package.class#member` label, where `package.class#member` represents a model element) and the referenced element exists in the model, the function returns this element; otherwise, it returns null.

Since: Together 5

getJDRefURL

`String getJDRefURL(String jdref)`

If the specified JavaDoc Reference is a "url" reference (that is, it has the form `a href="URL#value">label</a>`) the function returns the text URL#value; otherwise, it returns an empty string.

Since: Together 5

findElement

`RwiElement findElement(String uniqueName)`

Passes the call to the RwiModel.findElement() method, which finds an element by its unique name.

Parameter: String with the unique name of an RWI element that needs to be found

Returns: An element found by its unique name

getCodeElement

`Object getCodeElement(RwiElement rwiElement)`

Passes the call to the `rwiElement.getCodeElement()method` method declared in the `com.togethersoft.openapi.rwi.RwiElement` interface.

This function is used in the template expressions together with one of the following functions: `findMember(), findNode(), findLink(), findPackage()`.

Since: Together 5

getCodeElements

`Enumeration getCodeElements(RwiElement rwiElement)`

Passes the call to the `rwiElement.getCodeElements()` method declared in the `com.togethersoft.openapi.rwi.RwiElement` interface.

This function is used in the template expressions together with one of the following functions: `findDocumentedMember(), findDocumentedNode(), findDocumentedLink(), findDocumentedPackage()`. These functions may be helpful when you need to provide hyperlinks from some specific elements on a diagram chart.

For example, if you have set the Recognize Java Bean / C++ properties option in Together's View Management, each JavaBean/C++ property is presented by a single element on a class diagram, whereas actually, it consists of 2 elements: property's attribute and setter/getter methods. When you generate the documentation for such a class, you will get for every JavaBean/C++ property all those 2 elements documented (or at least, docs for accessor methods if you have specified to skip private members). The corresponding element on the diagram chart associates with a certain RWI element, and you can obtain this RWI element via the `diagramMapElement` variable. But actually, this RWI element is a kind of a proxy. It will not be identical to any of those 2 elements your JavaBean/C++ property consists of, those elements which you can see in Java/C++ code and which will be documented by the template's iterators.

Thus, in the case of the JavaBean/C++ property, you cannot directly use the RWI element representing it on the diagram to establish a hyperlink to anything contained in the generated documentation. Instead of this, use the following expression: `findDocumentedMember(getCodeElements(getDGRwiElement(" diagramMapElement")))`.

The `findDocumentedMember()` function returns one of the RWI elements associated with the JavaBean/C++ property and that is definitely presented in the generated documentation.

Since a diagram contains ordinary elements as well, your expression for diagram hyperlinks connecting an RWI element is more complicated:

```
if(getDGRWIElement("diagramMapElement")->hasPropertyValue
("$shapeType","BeanProperty"), findDocumentedMember(getCodeElements(getDGRWIElement
("diagramMapElement"))), getDGRWIElement("diagramMapElement"))
```

Since: Together 5

findMember

`RwiElement findMember(Object codeElement)`

Passes the call to the `com.togethersoft.openapi.rwi.RwiModel.findMember()` method. This function should be used together with the `getCodeElement()` function.

Since: Together 5

findNode

`RwiElement findNode(Object codeElement)`

Passes the call to the `com.togethersoft.openapi.rwi.RwiModel.findNode()` method. This function should be used together with the `getCodeElement()` function.

Since: Together 5

findLink

`RwiElement findLink(Object codeElement)`

Passes the call to the `com.togethersoft.openapi.rwi.RwiModel.findLink()` method. This function should be used together with the `getCodeElement()` function.

Since: Together 5

findPackage

`RwiElement findPackage(Object codeElement)`

Passes the call to the `com.togethersoft.openapi.rwi.RwiModel.findPackage()` method. This function should be used together with the `getCodeElement()` function.

Since: Together 5

findDocumentedMember

- `RwiElement findDocumentedMember(Enumeration codeElements)`
- `RwiElement findDocumentedMember(Enumeration codeElements, String subjectSelector)`

This function should be used together with the `getCodeElements()` function. It utilizes the `com.togethersoft.openapi.rwi.RwiModel.findMember()` method and seeks the model for an RWI element that matches the following conditions:

- it is associated with the passed codeElements
- it is an `RwiMember`
- it will definitely be presented among all generated documents by its Main Documentation or, if subjectSelector is specified, by its "specific" documentation associated with the passed subjectSelector.

Returns: Found RwiElement, or null if the requested element does not exist in the model

Since: Together 5

findDocumentedNode

- `RwiElement findDocumentedNode(Enumeration codeElements)`
- `RwiElement findDocumentedNode(Enumeration codeElements, String subjectSelector)`

This function should be used together with the `getCodeElements()` function. It utilizes the `com.togethersoft.openapi.rwi.RwiModel.findNode()` method and seeks the model for an RWI element that matches the following conditions:

- it is associated with the passed `codeElements`
- it is an `RwiMember`
- it will definitely be presented among all generated documents by its Main Documentation or, if `subjectSelector` is specified, by its "specific" documentation associated with the passed `subjectSelector`.

Returns: Found RwiElement, or null if the requested element does not exist in the model

Since: Together 5

findDocumentedLink

`RwiElement findDocumentedLink(Enumeration codeElements)`

`RwiElement findDocumentedLink(Enumeration codeElements, String subjectSelector)`

This function should be used together with the `getCodeElements()` function. It utilizes the `com.togethersoft.openapi.rwi.RwiModel.findLink()` method and seeks the model for an RWI element that matches the following conditions:

- it is associated with the passed `codeElements`

- it is an `RwiMember`

- it will definitely be presented among all generated documents by its Main Documentation or, if `subjectSelector` is specified, by its "specific" documentation associated with the passed `subjectSelector`.

Returns: Found `RwiElement`, or null if the requested element does not exist in the model

Since: Together 5

findDocumentedPackage

`RwiElement findDocumentedPackage(Enumeration codeElements)`

`RwiElement findDocumentedPackage(Enumeration codeElements, String subjectSelector)`

This function should be used together with the `getCodeElements()` function. It utilizes the `com.togethersoft.openapi.rwi.RwiModel.findPackage()` method and seeks the model for an RWI element that matches the following conditions:

- it is associated with the passed codeElements

- it is an `RwiMember`

- it will definitely be presented among all generated documents by its Main Documentation or, if `subjectSelector` is specified, by its "specific" documentation associated with the passed `subjectSelector`.

Returns: Found `RwiElement`, or null if the requested element does not exist in the model

Since: Together 5

findDocBySubjectSelector

`String findDocBySubjectSelector(String subjectSelectorList)`

Returns the first generated document that contains an area marked with one of the specified subject selectors from the list.

This is how it works. The function takes the first passed subject selector from the list and checks if there are any generated documents that contain areas marked with this subject selector. If such documents exist, the function returns the one that has been generated first. Otherwise, it iterates to the next subject selector from the list and repeats examination. When all subject selectors are passed and no document is found, the function returns an empty string.

Parameter: List of subject selectors separated with semicolons.

Note: Blank subject selector is allowed and will refer to the Main Documentation of an element.

Returns: Path of the found document relative to the documentation's root directory. Subdirectories are delimited with a slash (`/`). If no document is found, the function returns an empty string.

Example:

```
findDocBySubjectSelector("package-summary;summary")
```

returns the fist generated document for one of the subject selectors: "package-summary", "summary"

Warning:

This function can be used only inside the Source File Name Expression of the node in FrameSet Structure definition.

Since: Together 5

findDocByTemplate

`String findDocByTemplate(String templateList)`

Returns the first generated document produced by one of the specified templates.

The function takes the first passed template name and checks if there are documents generated by this template. If such documents exist, it returns the one which has been generated first. Otherwise, it iterates to the next template from the passed list and repeats examination. When all templates are passed and no document is found, the function returns an empty string.

Parameter: List of template names (without file name extensions) separated with semicolons.

Returns: Path of the found document relative to the documentation's root directory. Subdirectories are delimited with a slash `/`. If no document is found, the function returns an empty string.

Example:

```
findDocByTemplate("all-classes;all-diagrams")
```

returns the fist document produced by one of the templates: "all-classes.tpl" and "all-diagrams.tpl"

Warning:

This function can be used only inside the Source File Name Expression of the node in FrameSet Structure definition.

Since: Together 5

checkStockSectionOutput

```
boolean checkStockSectionOutput(String stockSectionName, RwiElement rwiElement)
```

Tests if a Stock Section with the name stockSectionName will produce a nonempty output, provided that it is invoked from a Stock Section Call and `rwiElement` is passed to it as the current model element. When this function is called, no actual output is produced.

Parameters:

`stockSectionName` – name of the Stock Section to be tested. If no Stock Section with the specified name is found in the template, the function call issues an error message and stops the generator.

`rwiElement` – RWI element passed to the Stock Section as the current model element.

Returns: true, if the tested Stock Section would have a non-empty output; false, otherwise.

Example:

```
checkStockSectionOutput("Included Diagram List", getDGRwiElement("curElement"))
```

Since: Together 5

getPropertyExt

```
String getPropertyExt(String propertyName)
```

This function gets any element property available in DG for the metatype to which this element belongs. It includes the properties provided by RWI and the properties calculated only by DG (names of such properties start with `%`. See the **MetaModel.mm** file).

A possible call is `rwiElement->getPropertyExt(propertyName)`. In this case, the RWI element whose property should be obtained is specified before the arrow.

Parameter: Name of the required property

Returns: Value of the property or **empty** string if the element has no such property

See also: getProperty()

Utility functions provided by Documentation Generation

substring

```
String substring(String str, int beginIndex) String substring(String str, int
beginIndex, int endIndex)
```

Returns a new string that is a substring of the string str. Parameters are the same as in the standard Java String.substring() methods.

replace

```
String replace(String str, String oldStr, String newStr)
```

Returns a new string produced by replacing all occurrences of oldStr in the string str with newStr. Operation is case-sensitive.

Example:

```
replace("str-oldStr-newStr", "Str", "S")
```

Returns: "str-oldS-newS"

This function is especially helpful when you create a Call to Template section; the location of the document, generated by the called template, should be derived from some properties of the current model element (for example, from the full nume of the package where the current element belongs). In such a case you can write in the field "Output Directory Expression" something like this:

```
replace(getContainingPackage()->getProperty("$fullName"), ".", "/")
```

See also: Linking document templates when designing Multi-Frame documentation.

Since: Together 5

duplicate

```
String duplicate(String str, int num)
```

Returns a new string resulting from duplication of the specified string `str` `num` times. If `num` is 0, returns an empty string.

Example:

```
duplicate("abc", 2)
```

Returns: " abcabcabc"

Since: Together 5

length

```
int length(String str)
```

Returns the length of string str.

str

`String str(Numeric N)`

Converts numeric value to a string.

val

`Numeric val(String str)`

Converts numeric value represented as String into Numeric format. If conversion is impossible, returns 0.

Functions used in queries

The following functions, commonly provided in Together formulae queries, are also very useful in DG expressions.

getProperty

`String getProperty(String rwiPropertyName)`

Returns the value of the specified RWI property the current element has.

A possible call is `rwiElement->getProperty(rwiPropertyName)`. In this case, the RWI element whose property should be obtained is specified before arrow.

Parameter: name of the required property

Returns: value of the property or **empty** string if the element has no such property

See also: getPropertyExt()

hasProperty

`boolean hasProperty(String rwiPropertyName)`

Checks if the current element has the specified property.

A possible call is: `rwiElement->hasProperty(rwiPropertyName)`

In this case, the RWI element whose property should be checked is specified before arrow.

Parameter: name of the property being checked

Returns: true, if the element has such property; false, otherwise

hasPropertyValue

`boolean hasPropertyValue (String rwiPropertyName, String value)`

Checks if the current element has the property with the specified value.

A possible call is: `rwiElement->hasPropertyValue (rwiPropertyName, value)`

In this case, the RWI element whose property should be checked is specified before arrow.

Parameter:

rwiPropertyName – name of the property being checked

value – required property value

Returns: true, if the element has specified property with the required value; false, otherwise

if

type if(**boolean** condition, **type** value1, **type** value2)

If the parameter condition is **true** , the function returns value1. If the condition is **false** , the function returns value2.

The **type** can be any data type allowed in queries.

getContainingNode

`RwiNode getContainingNode()`

Returns the `RwiNode` element that contains the current element. Can be called for RWI member or node current element.

A possible call is: `rwiElement->getContainingNode()`

Example:

The following expression calculates visibility modifier for the class/interface member:

```
if (hasProperty("$private"), "private",   if (hasProperty("$protected"),
"protected",     if (hasProperty("$public") && !getContainingNode()-> hasProperty
("$interface"),        "public", "")))
```

In this case, the public modifier is printed only when the containing node is not an interface, because all interface members are public implicitly.

DG Variables

When using the Documentation Designer to develop custom documentation templates for Documentation Generator building block (DocGen), you have to reference the internal variables and functions to specify formulae expressions, and provide section flow control.

When the Documentation Generator executes a template and generates a report, it produces some specific internal information, which may be interesting to include in the report. This includes information such as the project name and current date/time. Moreover, there are special internal temporary data that get displayed when DG executes some particular parts of the template.

Documentation Generator variables enable access to this information and its insertion in the report. Each variable has a specific name and represents a particular kind of internal Documentation Generator information available at any particular moment.

Internal variables are not all accessible at any instant. Most of them get displayed only in special areas or inside special sections. Documentation Generator variables belong to one of the following types: `String`, `RwiElement`, `RwiProperty`. Access to these variables is provided by appropriate functions in formulae expressions: getDGVariable, getDGRwiElement, getDGRwiProperty.

Variable	Availability	Accessible via
`curItemNo` **String** The current iteration item number (starting with 1)	inside any Property Iterator and Element Iterator	`getDGVariable`
`curPropertyName` : **String** Name of the current property	inside Property Iterator	`getDGVariable`
`curPropertyFullName` : **String** Full name of the current property (specified in DG MetaModel File)	inside Property Iterator	`getDGVariable`
`curPropertyType` : **String** Type of the element property. Since RWI-interface does not provide property types, they should be specified in the DG MetaModel File. Possible values:**"string"** for String property; **"boolean"**for Boolean property	inside Property Iterator	`getDGVariable`
`curPropertyValue` : **String** value of the current property	inside Property Iterator	`getDGVariable`
`curPropertyInstance` **RwiProperty** The RwiProperty object of the current property instance. This variable is useful when you have to get a subproperty of the current property instance. For example, if the current model element is a class node and you need to list information about all interfaces implemented by this class, you have to create a template section that iterates by instances of the `IMPLEMENTS` property of the current class element. After that, within this iteration section you can use the `curPropertyInstance` variable to access the subproperty `REFERENCED_ELEMENT` that lets you obtain all information about the implementing class. This lets you get the full names of the implemented interfaces. Then the required expression should be: `findElement(getDGRwiProperty ("curPropertyInstance")-> getSubproperty`	inside Property Iterator while iterating by instances of the specified property	`getDGRwiProperty`

`("$referencedElement"))-> getProperty ("$fullName")`

See also DG Functions: `getDGRwiProperty, getSubproperty, findElement, getProperty`

`curPropertyInstance` : **String** value of the current property instance	inside Property Iterator while iterating by instances of the specified property	`getDGVariable`
`curElement` : **RwiElement** the current model element	inside Element Iterator	`getDGRwiProperty`
`prevElement` : **RwiElement** previous element in the current iteration scope. Possible values: **null**, if it is the beginning of the scope.	inside Element Iterator	`getDGRwiProperty`
`diagramMapElement` : **RwiElement** This variable should be used to create hyperlinks from image elements of a diagram chart.	Inside Image Control	`getDGRwiElement`
`projectName` : **String** The Project name	in the report or page header and footer areas	`getDGVariable`
`nowDateTime` : **String** The current date/time	in the report or page header and footer areas	`getDGVariable`
`outputFormat` : **String** returns output type of the generated documentation. Use this variable to control behavior of your templates depending on the output format type selected for the generator. Possible Values: "RTF", "HTML", "TXT"	in any place	`getDGVariable`
`reportScope` : **String** shows the specified report scope. Possible values: **"all_model"** – the scope is the whole model **"current_package"** – the scope is the current package only **"current_package_recursive"** – the scope is the current package with subpackages **"current_diagram"** – the scope is the current diagram only	in any place	`getDGVariable`
`stockParam` : **String** parameter of the stock section call	inside stock sections	`getDGVariable`

Documentation Template Designer

File (in the main menu) ▸ New ▸ Other ▸ Modeling ▸ Documentation Template

The **Template Designer** is a tool for creating custom documentation templates. The **Template Designer** displays two panes:

Scope	pane on the left, reveals the template structure
Details	pane on the right shows the contents of the zones.

Each template is presented in its own tab.

Most manipulations with the template sections and controls are performed by means of the Template Designer toolbar or executing right-click (or context) menu commands on the selected elements. Each header, footer, and static section has two different context menus, one for the *scope pane* (on the left) and the other for the *details pane* (on the right). The context menus on the details pane are for setting area style and format characteristics and for inserting controls. The right-click menus on the scope pane vary among sections. Items on those menus include the following:

Context menu command	Description
Delete	Deletes the selected section. This command applies for all sections, except for static sections without siblings and for the root iterator.
Insert Sibling Section	Inserts a new section immediately below this header or section. This command applies to all sections except for the footers.
Insert Nested Section	Inserts a new section immediately below the parent section. This command applies to the folder sections and iterators only.
Move Up, Move Down, Copy Section	**Move Up** and **Move Down** commands change the position of a section among its siblings. **Copy Section** creates a copy of the section in the clipboard. You can paste from the clipboard when you insert a new section. This command does not apply to the headers and footers.
Copy into Stock	Copies the whole section into a new stock section. This command applies to the folder sections and element iterators only.
Properties	Opens the **Properties** dialog. This command does not apply to the headers and footers.

Buttons on the Template Designer toolbar include the following:

Item	Description
Show Template Properties	Opens the **Template Properties** dialog.
Insert Nested Element	Inserts a nested element (static section, element property iterator, folder section, call to stock section, call to template) . This action is enabled for iterator sections only.
Insert Sibling Element	Inserts a sibling element (static section, element property iterator, folder section, call to stock section, call to template).
Insert Label Control	Opens the **Label Control** dialog that enables you to insert a label into a static section, header or footer of a template.
Insert Image Control	Opens the **Image Control** dialog that enables you to insert an image into a static section, header or footer of a template.
Insert Panel Control	Opens the **Panel Control** dialog that enables you to insert a panel into a static section, header or footer of a template.
Insert Formula Control	Opens the **Formula Control** dialog that enables you to insert a formula into a static section, header or footer of a template.
Insert Data Control	Opens the **Data Control** dialog that enables you to insert a data control element into a static section, header or footer of a template.

Insert Include Text Control	Opens the **Include Text Control** dialog that enables you to insert a text control element into a static section, header or footer of a template.
New Stock Section	Opens the **New Stock Section (Element Iterator)** dialog that enables you to create a new stock section in a separate tab.
New Stock Section (Folder)	Opens the **New Stock Section (Folder)** dialog that enables you to create a new folder section in a separate tab.
Generate Documentation Using Template	Opens the **Generate Documentation Using Template** dialog that enables you to generate project documentation with the current template.

Related Concepts

Documentation Generation Overview

Related Reference

Organization of a Documentation Template
Documentation Template Properties

Documentation Template Properties

Template Designer toolbar ▶ Show Template Properties button

Use this dialog box to view and modify the properties of a documentation template. The dialog box contains the following tabs:

- General
- Page Settings
- Formatting Styles
- Template Parameters

General tab

Option / Button	Description
Model	This read-only field displays the metamodel defined on template creation.
Template type	This read-only field displays the template type defined on template creation.
Template Description	Enter commentary information in this text field.
Report Title Expression	The field displays report title expression created in the Expression Editor. Click the editor button to the right to open the Edit Expression dialog, choose notation and enter a title expression.
Root Object Metatype	Select root object metatype from the drop-down list.
Formatting Template	Enter the path to the formatting template, or click the Browse button and choose an MS Word document.
Headers/Footers	Choose to generate specified headers and footers.

Page Settings tab

Use this tab to specify page size, margins, and orientation.

Option / Button	Description	
Page Size (mm)	Use this section to define page dimensions:	
	Page Type	Select page type from the list.
	Width/Height	These fields are read-only for the predefined page types. If “User” type is selected, the fields are editable.
Page Margins (mm)	Use this section to define page margins.	
Page Orientation	Click one of the radio buttons to select a page orientation. Default page size and margins display automatically.	

Formatting Styles tab

Use this tab to change formatting styles used in a template.

Option / Button	Description
New	Opens **Style** dialog, where you can define properties of a new formatting style.
Delete	Removes the selected style from the **Styles** list.

Edit	Opens **Style** dialog for the selected formatting style.

Template Parameters tab

Use this tab to specify the formal parameters that will be used for calling this template from another template.

Option / Button	Description
Parameter	Use this field to enter the name of a formal parameter that will be used for calling this template from another template.
Description	Enter optional parameter description.
Default Value	Enter optional string value. The parameter value can be obtained in a template body using the `String getParam(String paramName)` function.
Set	Saves the specified parameter in the list of formal parameters. This button is only enabled when the **Parameter** field is not empty.
Delete	Deletes the selected parameter from the list of formal parameters. This button is only enabled when the list is not empty.

Related Concepts

Organization of a Documentation Template

Related Procedures

Setting Template Properties

Element Iterator Properties

Template Designer ▸ Element Iterator section context menu ▸ Properties

Use this dialog box to access properties of the element iterators. The dialog displays the following tabs: **Metatype**, **Scope Options**, **Sorting**, **Output Style** and **Other**. This topic describes the **Scope Options** tab.

For the description of the **Sorting** tab, refer to the “Property Iterator Properties” topic. For the description of the **Metatype**, **Output Style** and **Other** tabs, refer to the “Folder Section Properties” topic.

Scope Options tab

Choose iteration scope from the list.

Iteration scope	Description	
Collecting elements	Use this section to define how the elements are collected. Depending on the selected radio-button, the dialog displays a different set of controls.	
	Default	If this option is selected, the **Search Options** section is displayed. Defined iteration by the elements of the metatype, selected in the **MetaType** tab. There are five different options to find out which elements of the subtree should be included in the generated documentation: **– 'Recurse subpackages'**: traverses the packages tree searching for elements of the current metatype. **– 'Recurse Subnodes':** searches inside elements. **– 'Include Parent Element'**: Visits the parent element (the current element, from which the iteration was initiated). Normally, an iterator goes through the contents of an element, not the element itself. This option is for documenting the containing element as well. **– 'Visit Diagrams':** Searches for element's references on diagrams as well as for elements themselves. Used in conjunction with 'Include Shortcuts'. **– 'Include Shortcuts':** Searches shortcuts contained in elements.
	Customized	If this option is selected, the **Customized Iteration Scope** section is displayed. Enables you to specify the expressions that return the first and the subsequent elements of the iteration: the first expression defines an element to begin with; the other describes how to get the subsequent element from the current one. Expressions can be defined in the Expression Editor using OCL or legacy notation.
	Programmed	If this option is selected, the **Programmed Iteration Scope** section is displayed. You can specify either class and method that returns Collection, or create an expression in the Expression Editor using the OCL or legacy notation.
Filter expression	Use this field to restrict the search scope to satisfy the filter condition. The filter expression can contain properties of the element by which the iteration is currently performed, as well as calls to DG functions returning DG options and template parameters (see the list of DG functions and variables in the “Documentation Template Designer” section).	

Related Concepts

Documentation Template Sections

Related Procedures

Setting Section Properties

Related Reference

Documentation Template Designer
Property Iterator Properties
Folder Section Properties

Frameset Template Properties

Template Designer toolbar ▸ Show Template Properties button

Use this dialog box to view and modify the properties of a documentation template. The dialog box contains the following tabs:

- General
- Frameset Structure
- Template Parameters

General tab

Option / Button	Description
Model	This read-only field displays the metamodel defined on template creation.
Template type	This read-only field displays the template type defined on template creation.
Template Description	Enter commentary information in this text field.
Report Title Expression	The field displays the report title expression created in the Expression Editor. Click the editor button to the right to open the Edit Expression dialog, choose notation and enter a title expression.
Root Object Metatype	Select a root object metatype from the drop-down list.
Formatting Template	Enter a path to the formatting template, or click the **Browse** button and choose a Microsoft Word document.

Frameset Structure tab

Use this tab to define properties of a frame or frameset.

Note: The availability of controls depends on the selected node.

Frameset option / button	Description
Layout	This is the topmost property of a frameset. The possible options are: Columns: A frameset with a column layout divides its window into columns, with one frame per column for each child. Rows: A frameset with a row layout divides its window (HTML frame) into rows, with one frame per row for each of its children.
Percent Size	You can assign a Percent Size to each frameset child to determine the percentage of the frameset's total space to be allocated to the child. The total of the sizes of a frameset's children should be 100%. Otherwise, the browser will decide the sizes for the children when it displays the documentation. This property is not available for the root frameset.
Scrolling	Choose the scrolling option from the drop-down list. This property is not available for the root frameset.
Enable Condition	This property determines if the frameset is to be skipped or included when the frameset file is generated. This condition is identical to that for body sections. This property is not available for the root frameset.
Add Frame	Adds a new frame under the current frameset node.
Add Frameset	Adds a new frameset under the current frameset node.

Delete	Deletes the current frameset node from the structure.

Frame Option / Button	Description
Frame Name	The Frame Name value is translated into the name parameter of the corresponding `<frame>` tag. You can use that name in a hyperlink to load the referenced document into the frame window. The tree in the left pane of the Frameset Structure tab shows the Frame names.
Percent Size	You can assign a Percent Size to each frameset child to determine the percentage of the frameset's total space to be allocated to the child. The total of the sizes of a frameset's children should be 100%. Otherwise, the browser will decide the sizes for the children when it displays the documentation. This property is not available for the root frameset.
Scrolling	Choose the scrolling option from the drop-down list. This property is not available for the root frameset.
Source File Name Expression	The Source File Name Expression determines the name of the HTML file that will be initially loaded into the frame.
Enable Condition	This property determines if the frameset is to be skipped or included when the frameset file is generated. This condition is identical to that for body sections. This property is not available for the root frameset.
Delete	Deletes the current frame from the structure.

Template Parameters tab

Use this tab to specify the formal parameters that will be used for calling this template from another template.

Option / Button	Description
Parameter	Use this field to enter the name of a formal parameter that will be used for calling this template from another template.
Description	Enter optional parameter description.
Default Value	Enter an optional string value. The parameter value can be obtained in a template body using the `String getParam(String paramName)` function.
Set	Saves the specified parameter in the list of formal parameters. This button is enabled only when the **Parameter** field is not empty.
Delete	Deletes the selected parameter from the list of formal parameters. This button is enabled only when the list is not empty.

Related Concepts

Multi-frame Documentation Templates

Related Procedures

Setting Frame and Frameset Properties

Folder Section Properties

Template Designer ▸ Folder section context menu ▸ Properties

Use this dialog box to access properties of the folder sections. The dialog displays three tabs: **Metatype**, **Output Style** and **Settings**.

Metatype tab

Displays the list of available metatypes.

Output Style tab

The Output Style tab is for specifying whether the documentation is to be in paragraph, text, or table format.

Style	Description
Paragraph Flow	In this default format, documentation for each element constitutes a single paragraph.
Delimited Text Flow	Delimiter separates the documentation for different elements. The following fields are available:

Formatting Style	Choose the formatting style from the list. Refer to the section “Creating Formatting Styles for Documentation Templates” for details.
Delimiter	Enter the delimiter character.
Font	Specify font size and style.
Always print section's header / footer	If this option is checked, section header and footer are always printed, even though the section is empty.
Suppress all GenDoc formatting	This option relates to HTML documentation only. If the option is checked, formatting options in controls are ignored, and all output text is printed in default font.

Style	Description
Table	Documentation for different elements is written to a table, with one row per element. Within each row, the different pieces of documentation are in different table cells. You can set border styles and cell padding. There are two check boxes for RTF documentation: print a separate table header on each page and allowing breaking a table over successive pages.

Settings tab

Item	Description
Left Indent (mm)	Specify indent. Note that indents are relative to the indentation for the containing sections rather than the physical paper border.
Commentary	Enter a descriptive string to identify the folder section in the template.
Enable condition	An enable condition for turning this section on or off. Use the Expression editor to specify an enable condition using the OCL or legacy notation.
Disabled	Check this option to skip the section.

Related Concepts

Documentation Template Sections

Related Procedures

Setting Section Properties
Creating Formatting Styles for Documentation Templates

OCL Functions in formulae expressions

The functions listed below are intended for the usage of expressions with OCL syntax. Refer to the list of legacy functions for comparison. Note that same-named functions return the same values.

Name	Return Type	Context	Parameter Type	Notes
getContainingPackage	uml::kernel::NamedElement	uml::kernel::Element		Returns the package containing the element in context or an undefined value if there is no such package.
getContainingNode	uml::kernel::NamedElement	uml::kernel::Element		Returns the node containing the element in context or an undefined value if there is no such node.
getContainingEntity	uml::kernel::NamedElement	uml::kernel::Element	String metaclass	Returns an entity containing the element in context with the specified metaclass.
getUin	String	uml::kernel::Element		Returns the model element uin.
getFriendlyMetaclassName	String	uml::kernel::Element		Returns human readable metaclass name for the element in context.
getDGVariable	String	OclAny	String dgVariableName	Returns the value of the variable or an empty string if the variable is not defined in the given place.
getDGOption	String	OclAny	String dgOptionName	Returns the value of the option or empty string if such an option is not defined.
getParam	String	OclAny	String param	Returns the value of the specified template parameter.
getDGRwiElement	uml::kernel::Element	OclAny	String dgVariableName	Returns the specified DG variable of Element type or null if the variable is not provided in the given place.
getPropertyExt	String	uml::kernel::Element	String propertyName	Gets any element property available in DG for the metatype to which this element belongs. Returns the value of the property or an empty string if the element has no such property.
findDocByElement	String	uml::kernel::Element	uml::kernel::Element element, String subjectSelector	Returns an area marked as start of passed Element's specific documentation and marked with the specified subject selector.
findDocBySubjectSelector	String	OclAny	String subjectSelectorList	Returns the first generated document that contains an area marked with one of the specified subject selectors from the list.
findDocumentedMember	uml::kernel::Element	uml::kernel::Element		Returns found Element or null if the requested element does not exist in the model.
findElement	uml::kernel::Element	OclAny	String uniqueName	Returns an element found by its unique name.

getJDRefElement	uml::kernel::Element	OclAny	String jdref	If the specified JavaDoc Reference is an "element" reference (that is, it has the form `package.class#member` label, where `package.class#member` represents a model element) and the referenced element exists in the model, the function returns this element, otherwise returns null.
getJDRefDisplayName	String	OclAny	String jdref	Returns a text to be displayed in place of the specified JavaDoc Reference.
substring	String	OclAny	String str Integer beginIndex	Returns a substring of a string starting from the specified position.
substring	String	OclAny	String str, Integer beginIndex, Integer endIndex	Returns substring of a given string starting from the first index, excluding the last one.
substring	String	OclAny	String string, String startFrom, Boolean include, Boolean index	Returns substring of a given string that starts from the specified string, searching from the start, if Boolean parameter is `true`, or from the end otherwise.
duplicate	String	OclAny	String str Integer number	Returns string that concatenates the given string for the specified number of times.
replace	String	OclAny	String oldString String newString	Replaces all occurrences of `oldString` with the `newString` in the string.
val	Integer	OclAny	String str	Parses the given string and returns an integer value or 0 in case of an error in the string.
str	String	OclAny	Integer value	Returns string representation of the given integer value.
isDiagram	Boolean	uml::kernel::Element		Returns `true` , if the element is a diagram; `false` otherwise.
getSubpropertyValue	String	uml::kernel::Element	String propertyName, String subpropertyName	Returns the value of subproperty of the given property of an element in context.
getSubproperty	String	OclAny	String propertyName, String subpropertyName	Returns the value of subproperty subpropertyName contained in the passed property. See the description of the curPropertyInstance DG variable for an example of using this function.
getAuditMessage	String	uml::kernel::Element		Returns audit result for the given source code element in context.

getSessionId	String	OclAny		Returns ID of the current Caliber session; can only be used when Caliber session is open. This is a service function, that is normally passed as a parameter of other GenDoc functions used for retrieving requirements properties, like getRequirementSystemProperties (RequirementTrace trace, String sessionID).
getLocalWorkspaceURL	String	OclAny	String url	For a hyperlink to a file from the workspace, returns the hyperlinked file path.
getRequirementDescription	String	OclAny	String requirement String sessionId	Returns a description of a requirement linked to the currently iterated model element.

Related Concepts

DG functions in Formulae Expressions

Property Iterator Properties

Template Designer ▸ Property Iterator section context menu ▸ Properties

Use this dialog box to access properties of the property iterators. The dialog displays the following tabs: **Iteration Scope**, **Sorting**, **Output Style** and **Other**. This topic describes the **Iteration Scope** and **Sorting** tabs. For the description of **Output Style** and **Other** tabs, refer to the “Folder Section Properties” topic.

Iteration Scope tab

Choose an iteration scope from the list. Depending on your choice, the dialog displays different controls.

Iteration scope	Description	
All User-Defined Properties	Iterates over the properties that are not described in the metamodel.	
	Exclude already iterated properties	If this option is checked, all properties that were already iterated for the current element are skipped.
	Iterate only unknown properties	If this option is checked, only those properties that were not included in the metamodel are included.

	Filter expression	Use this field to restrict the search scope to satisfy the filter condition. Expressions can use DG Variables available inside Property Iterators. Refer to “DG Variables” reference for details.
		
		Example of a filter expression for Property Iterator:
		
		`context: OclAny`
		
		`body: (getDGVariable ('curPropertyFullName') <> 'Stereotype') and (getDGVariable ('curPropertyFullName') <> 'Visibility')`
		
		Click the editor button to open the Expression editor.

Set of properties	These are properties that belong to the metatype of the parent element iterator. A property iterator can iterate over multiple properties. Use CTRL + CLICK to select multiple properties from the Available properties list, and then use the double-arrow button to move these properties to the Selected properties list. You can change the order in which the properties are documented by arranging the properties in the Selected properties list.	
Instances of a single property	These are properties that can have multiple values, for example, `@see` or `@author`. Use the Filter expression to restrict the search elements that satisfy the filter properties.	
All properties	Iterates by all properties defined for the current element metatype (only properties described in the metamodel are iterated). Use the Filter expression to restrict the search elements that satisfy the filter properties.	

Sorting tab

Use this tab to specify the order in which the elements are to be searched and thus documented.

Option	Description
Sorting mode	The following options are available: none, by name, by value, by key expression.
Reverse order	Elements are always documented in ascending order. Check the Reverse scope order box to list elements in descending order.

Related Concepts

Documentation Template Sections

Related Procedures

Setting Section Properties

Related Reference

Folder Section Properties
DG functions in Formulae Expressions
DG Variables

Static Section Properties

Template Designer ▶ Static section context menu ▶ Properties

Use this dialog box to access properties of the static sections. The dialog displays the only **Settings** tab.

Item	Description
Enable condition	An enable condition for turning this section on or off. Use the Expression editor to specify an enable condition using the OCL or legacy notation.
Disabled	Check this option to skip the section.

Related Concepts

Documentation Template Sections

Related Procedures

Setting Section Properties

Model Import and Export

In This Section

Import Together Project Wizard

File ▸ Import ▸ Modeling ▸ Together Project

Use this dialog box to migrate a legacy Together project to the current version of Together.

Migrate legacy Together project to Together <version>

Specify the Together project file and select the migrations type.

Item	Description	
Project Path	Click the Browse button to navigate to a specific source project.	
Diagram folders	This read-only area displays the folders of the legacy project that contain diagrams.	
Design elements storage policy	Use the radio-buttons in this section to define how to handle the design elements (as standalone or as file mates).	
	The same as in the original project	If this option is selected, the settings of the original project are preserved. The existing standalone design elements remain standalone. The new design elements are created according to the project settings.
	Force creating design elements in separate files	If this option is selected, all existing design elements are converted to standalone. All new design elements are created as standalone.
Migration type	Choose one of the possible ways to process the project roots.	
	Merge all roots contents into the new project	Click this radio-button to create a single project from a multi-rooted source project.
	Create a separate project for each root	Click this radio-button to create a Together project for each root.

Merged project name

This page will be displayed if the **Merge all roots contents into the new project** option is selected.

Item	Description
Project name	Enter the name of the resulting project. The default project name is constructed from the names of the last two folders of the source project file location.

Create a set of Together <version> projects

This page will be displayed if the **Create a separate project for each root** option is selected.

Item	Description
Root location	Displays the list of roots of the source project.
Together <version> project name	Displays the default name of the resulting project for the selected root. The default name is constructed from the package prefix, if any. If there is no

	package prefix, the project name is created from the names of the last two folders of the root location. Edit the project name as required.
Content type	Displays information about the type of contents in the selected root (design files or source code).
Diagram format	Displays information about the diagram format in the selected root, if any.
Decision	Select the way to handle information of the selected root. If the root contains design files, you can either copy them to the target location or skip the root. If the root contains source code files, you have the choice to copy it as is, copy and convert it to the design language, or skip the root.

Master project

This page is displayed when multiple projects are created.

Item	Description
Master Project Name	Specify the name of the master project that contains references to all projects created in the course of migration. The default name of the master project is based on the source project name. The master project is created to demonstrate the contents and structure of the source project. It is read-only and not intended for editing. Use the real projects to create or edit contents, and establish dependencies.

Related Concepts

Together Interoperability and Migration

Related Procedures

Importing Legacy Projects

MDL Import Wizard

File ▸ Import ▸ Modeling ▸ Project from MDL file

The MDL Import Wizard is used to import MDL projects created in another application for use in Together.

Option/Button	Description
Add/Add Folder	Specifies the name (or names) of the Rational Rose project file (or files) to be imported (several model files can be imported at once).
Remove	Deletes the selected file or files from the Paths list.
Remove all	Deletes all files from the Paths list.
Scale factor	Specifies the element dimensions coefficient. Default value is 0.3.
Convert Rose default colors	If this option is selected, the default Rational Rose colors will be replaced with the default Together colors. Deselected by default.
Preserve diagram nodes and bounds	If this option is selected, user-defined bounds are preserved in the resulting diagrams. Otherwise, the default values are applied. Deselected by default.
Convert Rose actors	If the option is selected, the Rose actors are mapped to Together actors. Deselected by default.
Generate source code	If this option is selected, a new Java Modeling project is created; otherwise, a Modeling project is created from imported MDL.

Related Concepts

Model Import and Export Overview

Related Reference

MDL Projects Import Options

MDL Projects Import Options

This topic provides information necessary for importing MDL projects, and the list of command line parameters.

MDL Import Notes

- A single state/activity element in a Rose model can be put into several different swimlanes. However, state/activity elements in Together can belong to only one swimlane; therefore, when importing a Rose project with a single state/activity element placed into several different swimlanes, Together places the state/activity element in one swimlane only.
- Using Rose, it is possible to create nested diagrams for class, use case, activity, and state elements. When using MDL import, the relationship between the element and the nested diagram is shown by a hyperlink that is created from the element to the diagram.

Path Aliases

Rational Rose model files may contain path aliases that need to be converted to real paths. Together recognizes path aliases and displays the Virtual Path Map dialog box that enables you to supply a real path for each path alias. To specify the actual path, click the **Browse** button. This opens the **Select Actual Path** dialog box. Navigate to the desired path and click **OK** when ready.

Tip: If new aliases are encountered in course of the file or subunit parsing, the **Virtual Path Map** dialog will be displayed again.

Parameters available for MDL command line import

Option	Description
–d <directory>	Name of the target directory where the output files are placed.
–model <directory name>	Name of the target directory where the generated diagram files are placed.
–src <directory name>	Name of the target directory where the generated source files are placed.
–project <project name>	Project name for generating hyperlinks to inner diagrams.
–scale <value>	Scale factor between Rose and Together diagrams. Default value is "0.3".
–colors	Converts Rose default colors to Together default colors.
–bounds	Preserves diagram nodes bounds.
–logfile <log file>	Path to log file.
–gensource	Enables source generation.
–log[:none|errors|debug]	Defines the level of logging. Default value is "errors".
–A<name>=<value>	This option is used to convert path aliases used in the Rose model file to the real paths; here <name> is the path alias and<value> is the real path for it.
–modelfile <file name>	Name of the Rose model file (`*Mdl, *.ptl, *Cat, *Sub`).
–f <file name>	Subunit files or paths to subunits (`*Cat` or `*Sub` files).
–v	Enables validation of the source code elements' names. Validation means that the symbols restricted in Java are replaced with the '_' characters.
–p	Enables creating diagrams that comply with the new containment metamodel.
–actors	Maps classes with the Actor stereotype (and the other similar stereotypes) to actors. The supported stereotypes are: Actor, Business Actor, Business Worker, Physical Worker.

–ac	Maps association classes to the simple text properties of association links. This option is intended for the legacy Together versions that did not support association classes.

Related Procedures

Importing a Project in IBM Rational Rose (MDL) Format
Importing a Project in IBM Rational Rose (MDL) From the Command Line

MDX Import Wizard

File ▸ Import ▸ Modeling ▸ Project from MDX file

The MDX Import Wizard is used to import MDX projects created in another application for use in Together.

MDX Import Options page

Option	Description
Path to the MDX file	Specifies the name of the IBM® Rational® XDE .mdx file.
Scale factor	Specifies the element dimensions coefficient. The default value is 0.03.
Preserve diagram nodes and bounds	If this option is selected, user-defined bounds are preserved in the resulting diagrams. Otherwise, the default values are applied. Deselected by default.
Convert Rose XDE colors	If this option is selected, the Rational Rose XDE colors will be replaced with the default Together colors. Deselected by default.

MDX Projects Import Options

The following parameters are available for importing MDX models.

- `-modelfile`
- `-d`
- `-model`
- `-project`
- `-scale`
- `-bounds`
- `-logfile`
- `-p`

Related Procedures

Importing a Project in IBM Rational Rose (MDL) Format
Importing a Project in IBM Rational Rose (MDL) From the Command Line

XMI Export Wizard

File ▸ Export ▸ Modeling ▸ XMI file

The XMI Export Wizard is used to export projects or sections of projects created in Together for use by other applications and languages.

Export Project to XMI File

Open projects list box	Displays currently open Together projects, which you can export as XMI data. Click on the plus sign to expand a project and select only a portion of it for export. You can select only one project at a time.
Select XMI type	Select an XMI type for export. Options: XMI for UML 1.3 (Unisys Extension) XMI for UML 1.3 (Unisys Extension, Recommended for TCC) XMI for UML 1.3 (Unisys Extension, Recommended for Rose) XMI for UML 1.4 (OMG) XMI for UML 2.0 Note: TCC stands for Together ControlCenter. XMI for UML 2.0 compliant with OMG standard XMI for UML 2.1
XMI Version	Specifies the version of XMI to be exported.
XMI Encoding	Specifies the XMI encoding setting.
XMI file	Specifies the path and file name to be used. Together will create these if they do not exist. You may enter a name and path or accept the default
Use prefix of imported root	Enabled for UML 1.4 projects that have imported roots. If this option is checked, a top-level package with the same name as the imported project prefix (specified in **Project ▸ Properties ▸ Model Path**) is created for each imported root.

Run Audits on Exported Project

The **Part-Port Audit** is provided for UML 2.0 projects. This audit provides the possibility to resolve problems that occur in Together 2006 models, where it was possible to add a port to a part. Such ports can be moved to the chosen classifier when one decides to fix the problem found by audit before the XMI export.

The **Required/Provided Interface Audit** is provided for UML 2.0 projects. It searches for Required/Provided Interface links with a null supplier. When the problem found by this audit gets fixed before the XMI export (by clicking **Fix All**), the link target can be changed to the chosen interface.

Related Concepts

Model Import and Export Overview

Related Procedures

XMI Export and Import of the Models with Cross-Project References

XMI Import Wizard

File ▸ Import ▸ Modeling ▸ XMI file

The XMI Import Wizard is used to import XMI projects or sections of projects created in another application for use in Together.

Note: For UML 1.4 and Java Modeling projects, only XMI 1.1/1.2 import is supported. Attempting to import an XMI 1.0 file results in an empty project. After selecting XMI File from the Import wizard, the XMI Import wizard's only dialog opens.

Select the Source File	Specifies the full path to the .xml, .xmi, or .uml2 file you are importing.
Open projects list box	Displays a list of open Together projects into which you can import the XMI data.
Select XMI Type of input file	The radio-buttons in this read-only section turn on corresponding to the type of the chosen project.

Related Concepts

Model Import and Export Overview

Version Control

In This Section

Sharing Design Elements: Special Considerations

When sharing design elements there are several points to consider.

- By default, Together stores diagram elements in a single file per package. While creating a project, choose to create design elements in separate files. To create design elements in separate files, select the **Create design elements in separate files** option.
- Use the locking mechanism of your version control system to ensure that only one user is allowed to edit the diagrams for each package.
- Standalone design elements have their own Team menu, which you can use to interact with the VCS. You need to add and commit standalone design elements individually or use **Team ▸ Synchronize with Repository** to get a list of outgoing changes (make sure the **Outgoing** button is selected). Outgoing elements are under the **Together Model** branch.
- Borland recommends that the *txa* files that store the design elements themselves (both txaPackage that store multiple nodes and standalone element files) are kept in synch with the diagram files that store the references to those elements.

Related Concepts

Version Control in Together

Related Procedures

Using Version Control and Teams in Together

Sharing Packages: Special Considerations

When using the Team menu to share packages through the Model Package Explorer, Model Navigator and the Diagram editor, there are several points to consider.

- Package elements on diagrams represent the diagram for that package, not the physical package directory. Therefore, the Team menu commands for these elements represent actions for the package diagram only.
- Class and interface elements on diagrams represent the source file and design elements represent the corresponding *txa* file they are located in; therefore, you can use their Team menu to Add, Commit or Update.
- To commit your packages and source code, you can use the Team menu accessed through the Model Package Explorer view. When you commit a package, it will (recursively, through subpackages) commit all the classes that are already part of the repository. New classes need to be added separately by either right-clicking on the Class in the Model Package Explorer or on the element in the diagram. Committing your packages through the Model Package Explorer will commit only the source and subpackages.
- The Model Navigator behaves similarly to the Model Package Explorer, but note that by default, while committing a package in Model Navigator, you commit all the source and design resources. To disable such behavior, use the Together modeling preferences dialog. To open the preferences dialog, from the menubar, select **Window** ▶ **Preferences**. In the options list on the left, expand the Modeling node, and select the Team/Compare tab on the right. Clear "Include diagram folders in Team/Compare actions." With this option disabled, the **Commit** command will not commit source and design resources. Note that Borland does not recommend committing package diagrams in source code projects.
- To find out how to prevent package diagrams being committed, refer to “Sharing Projects,” especially the last item, Recommendations and Tips.
- To prevent files synchronization problems and to ensure you can use the compare tools (including Model Compare provided by Together) to observe the differences, use **Team** ▶ **Synchronize with Repository...**, and wait for the **Synchronize** view to open. Click the outgoing and incoming buttons to see a list of outgoing and incoming changes. With this view you can commit your packages and source. If you commit a package with new resources, you are asked to confirm that you want to add the new files.

Related Procedures

Sharing Projects

Sharing QA Sets and Audits and Metrics Results

This topic describes what to consider when sharing QA sets and Audits and Metrics results.

Note: You can version both source code and model audits and metrics sets and results.

Sharing QA Sets

When deciding which audits and metrics you want to run on your project, you have the option to create a customized set, or load an existing set. You can save the QA sets you want to use in your project, possibly under a "sets" directory that you create, and use VCS to distribute them among your team. Each team member can then use the Load button on the QA Preferences page to select the agreed upon set. QA sets can be configured on the project level and the settings can be version controlled if saved with the project.

Sharing Audits and Metrics Results

Just as you can save QA sets in your project, you can save the audits and metrics results in your project if you want to distribute them. This gives your team a central place to store the results. Then, by using either Load Audits Results or Load Metric Results context commands in the Audits or Metric view, you can easily compare current and past results.

Related Concepts

Version Control in Together

Related Procedures

Using Version Control and Teams in Together

Dialogs

This part contains reference information about various Together dialogs.

In This Section

Frameset Template Properties Dialog Box
Describes the dialog that lets you view and modify properties of a documentation template.

Generate HTML Documentation dialog box
Describes the wizard that lets you generate HTML documentation for your projects.

Generate Documentation Using Template dialog box
Describes the wizard that lets you use predefined or custom templates to generate documentation for your projects.

Generate Sequence Diagram dialog box
Describes the dialog that lets you choose classes and namespaces to display on generated sequence diagrams.

Import Wizard: DB Schema from ER Logical Diagram Profile UML 2.0 Project
Use this wizard to create a DB Schema from a UML 2.0 project with ER Logical Diagram Profile enabled.

Import Wizard: DB Schema from JDBC
Use this wizard to create a DB Schema from a JDBC connection.

Import Wizard: DB schema from SQL script

Import Pattern Conversion Profiles
Use the **Import Pattern Conversion Profiles** dialog box to specify which profiles you want to import to a file, and the location of the file.

Import Together Project Wizard
Use this dialog box to migrate a legacy Together project to the current version of Together.

Manage Traces Dialog
Describes the options for defining traces from a model element selected in the Diagram editor or Model Navigator.

Modeling Preferences
Use these preferences to change startup, deletion, error reporting, ignored folders, and team sharing options.

New MDA Ant Task
Use the **New MDA Ant Task** wizard to add a new Ant task to your Composite transformation.

MDL Import Wizard
The MDL Import Wizard is used to import MDL projects created in another application for use in Together.

MDX Import Wizard
The MDX Import Wizard is used to import MDX projects created in another application for use in Together.

Model Search and OCL Model Search
Describes the dialog used to search the current diagram or all opened diagrams for the specified string in a certain scope.

New Together Project Wizards
This section describes the common pages of the Wizards used to create new Together modeling projects, and language-specific pages for C++ and IDL projects.

Print Audit dialog box
Describes the dialog that lets you print selected sets of audit report results to a specified printer.

Print Diagram Dialog Box
Describes the dialog that lets you print selected diagrams to a specified printer.

Print Dialog
Describes the options for printing diagrams.

Project Properties
Use this dialog box to modify your project's properties.

Project Specific Configuration
Use the **Project Specific Configuration** dialog box to select a C++ or Java project for which you want to configure QA Builder properties.

Property Iterator Properties
Use this dialog box to access properties of the property iterators.

QA Builder Properties
Use the **QA Builder Properties** dialog box if you want to customize the workspace-level set of audits for the QA Builder to suit your C++ or Java project needs.

QA Search
Use the **QA Search** dialog box to search within Audits View or Metrics View.

QVT Settings
Use the **QVT Settings** page of the **Project Properties** dialog box to specify if you want to generate Java code for your MDA transformation project, and choose a Java container where you want to store the code.

Run
Use the **Run** dialog to create, manage, and run configurations.

Run QA
Use the **Run QA** dialog box to choose resources that you want to process when running quality assurance.

Requirement Traces Search Dialog Box
Use this dialog box to search for specific elements that have traces to requirements, or find all traced elements in the defined search scope.

Select element dialog box
This dialog box displays a tree view of the available contents within your project.

Selection Manager
Describes the options for selecting elements from the available contents and adding them to a certain destination scope.

Static Section Properties
Use this dialog box to access properties of the static sections.

Template Properties Dialog Box
Describes the options for viewing and modifying properties of a documentation template.

Trace Synchronizer Dialog Box
Use this dialog box to search for traced requirements or model elements that become desynchronized.

XMI Export Wizard
The XMI Export Wizard is used to export projects or sections of projects created in Together for use by other applications and languages.

XMI Import Wizard
The XMI Import Wizard is used to import XMI projects or sections of projects created in another application for use in Together.

Apply Transformation

Use the **Apply Transformation** wizard to apply a QVT transformation to your model or model element.

In This Section

Select Destination

Use the **Select Destination** page of the **Apply Transformation** wizard to select the output model for Model-To-Model transformations.

Select Transformation

Use the **Select Transformation** page of the **Apply Transformation** wizard to select the source transformation file for Model-To-Model transformations.

Select Destination

Use the **Select Destination** page of the **Apply Transformation** wizard to select the output model for Model-To-Model transformations.

Item	Description	
Target type	Selects where you want to store the transformation results:	
	New model	Saves transformation output to a new EMF resource with URI specified in the **URI** field.
	Existing container	Adds transformation output to a containment feature (reference) of an existing model element.
	Inplace	Directly modifies the transformation input and returns it as the transformation result (inplace transformation). The URI specified in the **Source model URI** field is used as the URI of the transformation output.
URI	Specifies the URI of the target model element when you select **New model** or **Existing container** as the target type.	
Feature	Specifies the name of a container within the selected target model or model element where you want to store the transformation results. Enabled when you select **Existing container** as the target type.	
Select...	Opens a dialog box that allows you to choose the feature from the list of all containment references of the selected element.	
Clear contents	Specifies if you want to remove elements specified in the chosen reference before you store the transformation results. For the references with multiplicity 1, this option has no effect.	
Generate trace file	Specifies if you want to generate the trace file for the transformation. If selected, lets you specify the name and location of the file.	
Open result in editor	Specifies if you want to open the transformation result in the corresponding editor.	

Related Concepts

Model Transformation Support

Related Procedures

Applying Model-To-Model Transformations
Applying Model-To-Text Transformations

Related Reference

QVT Language
QVT Editor

Select Transformation

Use the **Select Transformation** page of the **Apply Transformation** wizard to select the source transformation file for Model-To-Model transformations.

Item	Description
Select Transformation	Displays the list of transformation projects available at the specified location.
Current selection	Displays the selected transformation file (*.qvt).
Run in interpreted mode	Specifies if you want to run the selected transformation using QVT Interpreter.

Related Concepts

Model Transformation Support

Related Procedures

Applying Model-To-Model Transformations
Applying Model-To-Text Transformations

Related Reference

QVT Language
QVT Editor

BPEL4WS Export Wizard

The BPEL4WS Export wizard allows you to export your BPMN diagram to BPEL and WSDL files.

Item	Description
List of BPMN projects	Select the project you want to export to BPEL and WSDL files.
Destination	Type the path to the created BPEL and WSDL files.
Browse	Specify the path to the created BPEL and WSDL files.
Open file in Active BPEL Designer	Opens generated BPEL file in the new view as the Active BPEL Designer file; the BPEL structure is shown in a special diagram. You can export BPEL/ WSDL files to the current workspace project when the Active BPEL Designer is already installed. The checkbox becomes available when the folder inside your workspace project is created.

After you export your project to BPEL4WS, you can use Active BPEL Designer to work with BPEL files.

Warning: If you want to open the export result in Active BPEL Designer, make sure your export result is located within one project opened in the workspace.

Related Concepts

Business Process Modeling

Related Procedures

Together Business Process Modeling

BPEL4WS Import Wizard

The BPEL4WS Import wizard allows you to import BPEL and WSDL files and create a BPMN diagram based on the imported files.

Item	Description
BPEL File	Type the path and the name of the imported BPEL File. Lists BPEL files opened earlier using the Browse button.
Browse	Browse for a BPEL file to import.
WSDL fIles	Lists WSDL files selected for import together with a BPEL file.
Add	Selects WSDL files to add to the list.
Add folder	Selects all WSDL files in a folder.
Diagram name	Name of the created BPMN diagram.
Project	Selects a project, where the new BPMN diagram will be created.

Related Concepts

Business Process Modeling

Related Procedures

Together Business Process Modeling

Call to Stock Section Properties

Template Designer ▸ Call to Stock Section context menu ▸ Properties

Use this dialog to view or edit properties of a call to stock section.

Call To tab

This tab lists the available stock sections and highlights the name of the stock section that is actually called.

Other tab

Use this tab to define enable condition and template parameters.

Option/Button	Description
Left indent (mm)	Specify indentation.
Parameter Expression	Lets you specify string parameter of the stock section call. Within stock section, this parameter can be retrieved using getDGVariable('stockParam'). Click the **Edit Expression** button to create an expression in OCL or legacy notation.
Enable condition	An enable condition for turning this section on or off. Use the Expression editor to specify enable condition using the OCL or legacy notation.
Disabled	Check this option to skip the section.

Related Concepts

Enable Conditions

Related Procedures

Creating Stock Sections

Call to Template Properties

Template Designer ▸ Call to Template section context menu ▸ Properties

Use this dialog to view or edit properties of a call to template section.

General tab

Option/Button	Description
Template	Assigns a template that is invoked by the Call to Template section. Click the **Browse** button to choose the actual template to be called.
Output Settings	Gives a choice of where the output for the called template goes.

Separate file	This is important for generating multiframe HTML documentation consisting of separate HTML documents that are extensively linked together. If this option is selected, the following fields are displayed:

Output Filename Expression	Enter the name of the document. This expression should not include the file path. If this field is blank, the generated document is named according to the name of the called template. Click the **Edit Expression** button to create the expression in OCL or legacy notation.
	
	Example:
	
	`context uml::kernel::NamedElement`
	
	`name.concat('.Dia')`

Output Directory Expression	Enter the path to the destination directory of the generated document. If the expression contains directories that do not yet exist, they will be created when the template is processed. Click the **Edit Expression** button to create the expression in OCL or legacy notation. This path is always relative. Define it according to the following conventions: 1. If the calling template is a frameset template, the path is relative to the destination directory for the entire documentation. 2. If the calling template is a document template, the path is relative to the location of the document that is generated by the calling template. 3. The right slash character (`/`) is the name-separator for the path.
Output Image Subdirectory Expression	Enter the path to the directory for the images files of the generated document. Click the **Edit Expression** button to create the expression in OCL or legacy notation. **Example:**

			`context OclAny` `'../doc-images'`
		Do not create file with empty output	Check this option to skip empty files.
Common stream	The called template behaves like a stock section. If this option is selected, the following field is available:		
		Left indent (mm)	The called template provides output to the same file as the calling template, and you can only specify indentation if required.

Parameters tab

A calling template can pass additional information to the called template through template parameters. The parameter value can be obtained in a template body using the `String getParam(String paramName)` function.

Parameter	Enter parameter name.
Expression	Displays the parameter expression. Click the **Edit Expression** button to create the expression in OCL or legacy notation.
Set	Adds parameter to the list. This button is only enabled when the **Parameter** field is not empty.
Delete	Removes the selected parameter from the list. This button is only enabled when a parameter is selected in the list.

Other tab

Use this tab to define enable condition.

Item	Description
Enable condition	An enable condition for turning this section on or off. Use the Expression editor to specify enable condition using the OCL or legacy notation.
Disabled	Check this option to skip the section.

Related Concepts

Enable Conditions

Create Pattern from Elements

File ▸ Export ▸ Modeling ▸ Pattern Definition.

Use this wizard to create a pattern from the selected model elements.

Customize page

Item	Description
Pattern name	Enter the name of the new pattern definition.
Select category	Use this field to select the target category for the new pattern definition.
Show existing patterns	Check this option to display the available patterns.
Transformation profile	Check this option to display the transformation profile.
Open pattern definition project after pattern is finished	If this option is checked, the new profile definition project opens for editing.

Set Role Names page

Use this page to edit the role names of the elements involved in the pattern definition

Item	Description
Element name	Displays the name of the element participating in the pattern definition.
Role name	Use this column to edit the name of the pattern participant. By default, the role name is the same as the element name.

Set Default Values page

Use this page to specify the default values for the properties of the roles. Each property has the following set of parameters:

Parameter	Description
Customize value on application	If this option is set to `true`, you can modify the value of this property in the **Model element by pattern** wizard to be set on element creation. This property should be `false` if `Use property on application` is `false`.
Use for recognition	Controls whether to use this property on recognition.
Use property on application	Controls whether to set this property on creating elements by pattern.
Value	If **Use for recognition** is `true`, this value is compared to the property of the element against which the pattern is recognized. If **Use property on application** is `true`, this field defines the default value of the property.

Related Concepts

Patterns and Templates

Related Procedures

Patterns and Templates

Create Requirement(s) Dialog Box

Use case context menu ▸ Requirements ▸ Create Requirement(s)...

Use this dialog box to create CaliberRM or RequisitePro requirements from use case elements selected in the Diagram Editor or **Model Navigator**.

Tab	Description
CaliberRM	Lets you select a requirement or requirement type in the CaliberRM projects tree under which the requirements created from model elements will be inserted.
RequisitePro	Lets you select a project, package or a requirement in the RequisitePro projects tree under which the requirements created from model elements will be inserted.

Related Concepts

Requirements Management

Related Procedures

Creating Requirements Based on Use Case

Edit Audit

Window ▸ Preferences ▸ Modeling ▸ QA Model ▸ Audits tab ▸ New, Edit, or Clone button

Use the **Edit Audit** dialog box to create or edit a model audit.

Option	Description
Name	Displays the name of the QA model audit.
Description	Specifies the audit description.
Severity	Specifies the audit severity.
Context	Selects the context for the OCL operation.
Body	Displays the body of the selected OCL operation in the built-in OCL expression editor. The editor provides OCL code sense and auto-complete (CTRL+SPACE) options.

Related Concepts

Model Audits

Related Procedures

Running Source Code Audits

Related Reference

QA Model

Edit Hyperlinks for Diagram dialog box

Context menu ▸ Hyperlinks ▸ Edit

This dialog box is invoked from the context menus in the Diagram Editor or the **Model View**. It contains two tabbed pages that enable you to create hyperlinks to the model elements and external resources.

Dialog title	The title of the dialog box varies depending on the way it is invoked. It displays the string that corresponds to the invoking object.
Model Elements tab	The pane on the left of the dialog box displays the content available in your project. You can use the explorer to navigate to the element and select it for inclusion in the pane of values returned by the dialog to the invoking object.
External Documents tab	The Recently Used Documents pane displays the external contents that you make available for your project. Such contents may be represented by the file system resources or by URLs. Use the Browse and URL buttons to specify these resources.
	Browse: Click this button to invoke the Open dialog box. Navigate to the desired file and click OK.
	URL: Click this button to invoke the Documents URL dialog box. Type a URL in the text field and click OK.
	Clear: Click this button to remove all entries in the list of the Recently Used Documents.
Selected pane	This pane displays two kinds of data: it displays values already existing and passed from the invoking object, if any; and it displays values of the selections you have added from the left-hand pane, if any.

Buttons	
Add	Enabled when an element is selected in the left-hand pane. Adds the selected element to the right-hand pane.
Remove	Enabled when you select an item in the right-hand pane. Removes the selected item from the pane. All removed values or objects are removed from the invoking property or diagram upon clicking OK.
Remove All	Enabled when items are present in the right-hand pane. Removes all items from that pane. All removed values or objects are removed from the invoking property or diagram upon clicking OK.

Related Concepts

Model Hyperlinking Overview

Related Procedures

Hyperlinking Diagrams

Edit Metric

Window ▸ Preferences ▸ Modeling ▸ QA Model ▸ Metrics tab ▸ New, Edit, or Clone button

Use the **Edit Metric** dialog box to create or edit a model metric.

Option	Description
Name	Displays the name of the QA model audit.
Lower limit	Defines the lower limit of the constraint in the metric.
Upper limit	Defines the upper limit of the constraint in the metric.
Description	Specifies the audit description.
Severity	Specifies the audit severity.
Context	Selects the context for the OCL operation.
Body	Displays the body of the selected OCL operation in the built-in OCL expression editor. The editor provides OCL code sense and auto-complete (CTRL+SPACE) options.

Related Concepts

Model Metrics

Related Procedures

Running Source Code Metrics

Related Reference

QA Model

Edit Operation

Window ▸ Preferences ▸ Modeling ▸ OCL ▸ OCL Operations tab ▸ New (or Edit) button

Use the **Edit Operation** dialog box to create or edit an OCL operation.

Option	Description
Context	Selects the context for the OCL operation.
Body	Displays the body of the selected OCL operation in the built-in OCL expression editor. The editor provides OCL code sense and auto-complete (CTRL+SPACE) options.

Related Concepts

OCL Support

Related Procedures

Together Object Constraint Language (OCL)

Related Reference

OCL

Edit Transformation Profile

Window ▸ Preferences ▸ Patterns ▸ Edit...

Use the **Edit Transformation Profile** dialog box to specify which metaclass properties you want to filter or omit when you apply patterns to your models.

Option	Description
Metaclass	Displays the hierarchy of metaclasses available in Together.
Show only filtered	Displays only metaclasses that contain filtered properties.
Expand All	Expands the hierarchy of metaclasses.
Collapse All	Collapses the hierarchy of metaclasses.
Properties to filter out	Unchecks the check boxes against all profiles in the list.
Omit element entirely	Omits the selected metaclass.
Unfilter	Removes filter from the properties selected in the **Properties to filter out** field.
Omit properties with default values	Specifies if you want to omit the properties whose values have not been specifically defined.
Omit properties with derived values	Specifies if you want to omit the properties whose values derived from values of other properties.
Save profile as	Specifies the name under which you want to save the profile.

Related Concepts

Patterns and Templates

Related Procedures

Patterns and Templates

Element Iterator Properties

Template Designer ▸ Element Iterator section context menu ▸ Properties

Use this dialog box to access properties of the element iterators. The dialog displays the following tabs: **Metatype**, **Scope Options**, **Sorting**, **Output Style** and **Other**. This topic describes the **Scope Options** tab.

For the description of the **Sorting** tab, refer to the “Property Iterator Properties” topic. For the description of the **Metatype**, **Output Style** and **Other** tabs, refer to the “Folder Section Properties” topic.

Scope Options tab

Choose iteration scope from the list.

Iteration scope	Description	
Collecting elements	Use this section to define how the elements are collected. Depending on the selected radio-button, the dialog displays a different set of controls.	
	Default	If this option is selected, the **Search Options** section is displayed. Defined iteration by the elements of the metatype, selected in the **MetaType** tab. There are five different options to find out which elements of the subtree should be included in the generated documentation: **– 'Recurse subpackages'**: traverses the packages tree searching for elements of the current metatype. **– 'Recurse Subnodes':** searches inside elements. **– 'Include Parent Element'**: Visits the parent element (the current element, from which the iteration was initiated). Normally, an iterator goes through the contents of an element, not the element itself. This option is for documenting the containing element as well. **– 'Visit Diagrams':** Searches for element's references on diagrams as well as for elements themselves. Used in conjunction with 'Include Shortcuts'. **– 'Include Shortcuts':** Searches shortcuts contained in elements.
	Customized	If this option is selected, the **Customized Iteration Scope** section is displayed. Enables you to specify the expressions that return the first and the subsequent elements of the iteration: the first expression defines an element to begin with; the other describes how to get the subsequent element from the current one. Expressions can be defined in the Expression Editor using OCL or legacy notation.
	Programmed	If this option is selected, the **Programmed Iteration Scope** section is displayed. You can specify either class and method that returns Collection, or create an expression in the Expression Editor using the OCL or legacy notation.
Filter expression	Use this field to restrict the search scope to satisfy the filter condition. The filter expression can contain properties of the element by which the iteration is currently performed, as well as calls to DG functions returning DG options and template parameters (see the list of DG functions and variables in the “Documentation Template Designer” section).	

Related Concepts

Documentation Template Sections

Related Procedures

Setting Section Properties

Related Reference

Documentation Template Designer
Property Iterator Properties
Folder Section Properties

Export Diagram to Image Wizard

File ▸ Export ▸ Modeling ▸ Image (GIF, JPEG, Bitmap, EMF, SVG)

This wizard lets you save a diagram or selected elements in the specified format.

Option	Description
Destination file	Use this field to specify the fully qualified name of the resulting image file. You can enter the file name manually, confirm default setting, or click the **Browse** button and navigate to a specific location.
Diagram scope	Choose one of the following options: Current, All opened, Selected elements.
Format	Select the format of the resulting image from the list of supported formats.
Scale	Specify zoom factor. You can select one from the drop-down list, or enter a specific value.
Export heading	If this option is checked, the image will be saved together with the diagram title.
Open in viewer	If this option is checked, the image will open in the default image viewer.

Related Concepts

Model Import and Export Overview

Related Procedures

Exporting a Diagram to an Image

Export Pattern Conversion Profiles

Window ▸ Preferences ▸ Patterns ▸ Export...

Use the **Export Pattern Conversion Profiles** dialog box to specify which profiles you want to export to a file and the file location.

Option	Description
Profiles to export	Displays the list of available pattern conversion profiles. Check the check boxes against profiles that you want to export.
Select All	Checks the check boxes against all profiles in the list.
Deselect All	Unchecks the check boxes against all profiles in the list.
Target directory	Specifies the full path to the folder where you want to store the exported file. Click the adjacent **Browse** button to open the dialog box, which lets you browse for the folder.

Related Concepts

Patterns and Templates

Related Procedures

Patterns and Templates

Export QA Results To A File

Use the **Export QA Results To A File** dialog box to save the Audit or Metric results to a file.

Option	Description
Save as	The location path and name for the new file. Or you can accept the default: <project file>/out/qa/audit.*. Directories and files that do not exist will be created in the process.
Type	Format of the exported file. Choices include: **Text file, comma separated/tab separated**: preferred for use by spreadsheet programs. **HTML file**: single html file that contains a simple table listing information about the audits. **Summary HTML report**: a series of html files that include a graphical overview and audit statistics grouped by package, class, and so on. **Save in loadable format**: *.atbl file, a form that can be imported for later viewing in the audit results table using the **Load Audit Results** command.
Selected rows only	If checked, report will include only selected results rows.
Expand nodes (Text and HTML files)	Expands nodes, listing the information as separate rows.
Open in browser (HTML files)	If checked, file opens automatically in browser when created.
Include documentation (HTML files)	If checked, copies of the audit description files from Together are included in a separate directory within the destination directory.
Select encoding (Summary HTML report)	Options include: UTF-8, Cp1251, KOI8-R

Related Procedures

Saving and Loading Metric Results
Saving and Loading Audit Results

Export Wizard: SQL/DDL Script from DB Schema

File ▸ Export ▸ Modeling ▸ DDL/SQL Script

Use this wizard to create an SQL/DDL script from a DB Schema.

Related Reference

- Select Generation Options page
- Select Generation Objects page
- Save to File page

Select Generation Objects page

File ▸ Export ▸ Modeling ▸ DDL/SQL Script ▸ Select Generation Objects page

Use this page to specify the source objects and schema file.

Option/Button	Sub Options
Project	Selects source Data Modeling project.
Schema	Select the schema from the list of schemata available in the project.
Objects	Selects the schema objects.
Select All	Selects all schema objects.
Clear All	Clears the selection.

Related Reference

Select Generation Objects page
Save to File page

Select Generation Options page

File ▸ Export ▸ Modeling ▸ DDL/SQL Script ▸ Select Generation Options

DBMExportWizard

Use this page to specify conversion options for specific Data Modeling project elements. The following options alter the presence of some SQL constructs in a generated DDL script.

Option/Button	Sub Options	
Schema options	Pre SQL	Generate schema pre SQL
	Post SQL	Generate schema post SQL
Table options	CREATE statement	Generate CREATE statements
	DROP statement	Generate DROP statements
	Check constraint	Generate table check constraints
	Pre SQL	Generate schema pre SQL
	Post SQL	Generate schema post SQL
	Storage options	Generate RDBMS-specific table storage options
View options	CREATE statement	Generate CREATE statements
	DROP statement	Generate DROP statements
	Check constraint	Generate table check constraints
	Pre SQL	Generate schema pre SQL
	Post SQL	Generate schema post SQL
	Column list	Do not generate unique column names for propagated view columns (CREATE VIEW "myView" AS SELECT "myTable.column1" [AS "column1"])
Primary Key options	As ALTER TABLE	Generate primary key definition in ALTER TABLE statement
	As CREATE TABLE	Generate primary key definition in CREATE TABLE statement
	As column definition	Generate primary key definition in column definition
	Constraint name	Include constraint name in primary key definition
Column options	Default	Include DEFAULT in column definition
	Check	Include CHECK in column definition
Index options	CREATE statement	Generate CREATE statements
	DROP statement	Generate DROP statements
	Storage Options	Generate RDBMS-specific index storage options
Unique constraint options	As ALTER TABLE	Generate primary key definition in ALTER TABLE statement
	As CREATE TABLE	Generate primary key definition in CREATE TABLE statement
	As column definition	Generate primary key definition in column definition
	Constraint name	Include constraint name in primary key definition
Foreign key options	As ALTER TABLE	Generate primary key definition in ALTER TABLE statement
	As CREATE TABLE	Generate primary key definition in CREATE TABLE statement
	Constraint name	Include constraint name in primary key definition
UDT options	CREATE statement	Generate CREATE statements
	DROP statement	Generate DROP statements
Miscellaneous	Statement delimiter	
	Quote names	
	Generate owner	
	Generate comments	

Preview	Opens the **DDL Preview** dialog that displays a preview of the resulting script.

Related Reference

Select Generation Options page
Save to File page

Save to File page

File ▸ Export ▸ Modeling ▸ DDL/SQL Script ▸ Save to File

Use this page to specify the name and location of the target script file.

Option/Button	Description
File	Specifies the name and location of the target script file.
Open script in editor	If selected, the generated script will be open in the SQL editor.

Related Reference

Select Generation Options page
Select Generation Objects page

Find Analyzer Dialog

Window ▸ Preferences ▸ Modeling ▸ QA Source ▸ Java ▸ QA Builder

Use this dialog box to find an analyzer by the specified string.

Item	Description
Choose an analyzer (* = any string)	Enter the search string in this text field, using wildcards if necessary.
Matching analyzers	This area displays the list of analyzers that match the specified search string. Note that the matching analyzers are selected by their full names rather than abbreviations.

Related Reference

QA Source

Frameset Template Properties Dialog Box

Template Designer toolbar ▸ Show Template Properties

Use this dialog to view and modify properties of a documentation template.

General tab	Model	UML Metamodel.
	Template type	Frameset template.
	Template Description	Enter description of the template.
	Report Title Expression	Click the **Editor** button to open the **Edit Expression** dialog.
	Root Object Metatype	Select metatype from the drop-down list.
Frameset Structure	Layout	The topmost property of the frameset is its Layout, with radio buttons for selecting Columns or Rows. A frameset with a row layout divides its window (HTML frame) into rows. A frameset with a column layout divides its window into columns.
	Frame Name	Enter frame name here. The DocGen engine translates the Frame Name into the name parameter of the corresponding <frame> tag. You can use that name in a hyperlink to load the referenced document into the frame window. The tree in the left pane of the Frameset Structure tab shows the Frame names.
	Percent size	Specify the percentage of the frameset's total space to be allocated to the child. The total of the sizes of a frameset's children should be 100%.
	Scrolling	Choose scrolling type from the drop-down list.
	Source File Name Expression	Click the Editor button to open the Edit Expression dialog and enter the expression. The DocGen engine evaluates the Source File Name Expression expression to determine the name of the HTML file that will be initially loaded into the frame.
	Enabling Condition	Click the Editor button to open the Edit Expression dialog and enter the expression, which defines the enabling contition. The DocGen engine evaluates the enabling condition, which determines if the frameset is to be skipped or included in the frameset file.
	Delete	Press this button to remove the selected frame from the treeview.
Template Parameters	These parameters include:	
	Parameter	Enter the parameter name.
	Description	Enter optional description.
	Default Value	Enter optional default value.
Buttons	Set	Click this button to add a parameter to the list.
	Delete	Click this button to remove the selected parameter from the list.

Related Concepts

Multi-frame Documentation Templates

Related Procedures

A Typical Scenario of Creating a Template for Multi-Frame Documentation

Generate HTML Documentation dialog box

Project ▸ Documentation ▸ Generate HTML

Together features a UML documentation wizard that you can use to generate HTML documentation for your projects.

Output path	Use this text field to enter the target location of the generated report, or click the chooser button and specify the target folder in the **Choose output folder** dialog.		
Scope options	You can limit the scope of the documentation to a smaller set by choosing a different Scope option. The Scope section at the top of the dialog has a drop-down list from which you can choose the parts of the project to be parsed and included in the generated documentation:		
		All projects in the workspace	Generated output includes all projects in the workspace. It is important to note that documentation generation is not supported for the workspaces that contain data or business process projects, and UML projects.
		<project>	Generated output includes the selected project.
Options settings	The Options section of the dialog has options to specify the destination and other optional actions:		
		Include diagrams	Check to include diagram images in the output.
		Include navigation tree	Check to include a navigation tree in the output.
		Audits	Check to include audits into the generated report.
Open in Viewer	Check to load the documentation into your external web browser.		
Buttons		Finish	Accepts the input and starts the generate documentation process.
		Cancel	Cancels your input and closes the dialog box without generating documentation.

Related Concepts

Documentation Generation Overview

Related Procedures

Generating HTML Documentation

Generate Documentation Using Template dialog box

Project ▸ Documentation ▸ Generate Using Template

Together features a UML documentation wizard that you can use to generate documentation for your projects.

Output path	Use this text field to enter the target location of the generated report, or click the chooser button and specify the target folder in the **Choose output folder** dialog.	
Format	Choose an output format from the drop-down list of supported formats.	
Template	Use this section to choose between the default and custom documentation template.	
	Default	Click this radio button to enable the list of available predefined documentation templates delivered with Together.
	Custom	Click this radio button to enable the list of available custom templates and the chooser button to select the desired user-defined template in the **Choose Template File** dialog.
Scope	You can limit the scope of the documentation to a smaller set by choosing a different Scope option. The Scope section at the top of the dialog has radio buttons to indicate which parts of the project should be parsed and included in the generated documentation:	
	Project	Generated output covers all projects in the workspace, or the selected project. It is important to note that documentation generation is not supported for the workspaces that contain data or business process projects, and UML projects.
	Current package	Generated output includes only the current package selected in the Model Navigator or in the Diagram Editor .
	Current diagram	Generated output for the current diagram that is in focus in the Diagram Editor .
	Current package with descendent packages	Generated output includes the current package selected in the Model Navigator and any descendent packages.
Include	This section of the dialog has options to specify the composition of generated output:	
	Include diagrams	Check to include diagram images in the output.
	Include navigation tree	Check to include a navigation tree in the output.
Open in Viewer	Check to load the documentation into your external web browser.	
Buttons	Finish	Accepts the input and starts the generate documentation process.
	Cancel	Cancels your input and closes the dialog box without generating documentation.

Related Concepts

Documentation Generation Overview

Related Procedures

Generating HTML Documentation

Generate Sequence Diagram dialog box

Context menu ▸ Generate Sequence Diagram

To open this dialog box, right-click a method (or function) and choose Generate Sequence Diagram from the context menu. The Generate Sequence Diagram dialog box lists the classes and namespaces involved in the method (function) and lets you choose which classes/namespaces to display on the generated sequence diagram.

Fields	Name	Lists the names of namespaces/classes involved in the method (function).
	Show On Diagram	Check the namespaces/classes that you want to show on the generated sequence diagram. All namespaces and classes are selected by default. However, some classes may not be relevant. To increase the meaningfulness of the generated diagram, clear the checkboxes that are not helpful in explaining the sequence of operations.
	Show Implementation	For the elements that you decide to show in the diagram, check whether to show the implementation details in the generated sequence diagram.
Buttons	OK	Generates the new sequence diagram and opens the diagram in a new tab in the **Diagram View**.
	Cancel	Closes the dialog box without generating a sequence diagram.
	Help	Displays this help topic.

Related Reference

UML 2.0 Interaction Diagrams

Import Wizard: DB Schema from ER Logical Diagram Profile UML 2.0 Project

File ▸ Import ▸ Modeling ▸ DB Schema from ER Logical Diagram Profile UML 2.0 Project

Use this wizard to create a DB Schema from a UML 2.0 project with ER Logical Diagram Profile enabled.

Related Reference

Import Wizard: DB Schema from JDBC

Select Source and Target Objects page

File ▸ Import ▸ Modeling ▸ DB Schema from ER Logical Diagram Profile UML 2.0 Project ▸ Select Source and Target Objects

Use this wizard page to define source classes, target project and target schema file.

Option/Button	Description
Source	Lets you select logical data model elements found in UML 2.0 projects located in your current workspace. The selected elements are used as import source.
Target project	Lets you select a Data Modeling project where the resulting schema will be created.
Target schema file	Specifies the target schema name.

Related Concepts

Data Modeling

Related Procedures

Data Modeling Procedures

Related Reference

Data Modeling Reference

Select Options page

Use this wizard page to define how Together should process specific elements.

Option	Description
Create cross table for many-to-many relationship	If selected, Together creates a cross table for each many-to-many relationship found in the source ER Diagram.
Create unique constraints for alternate key group	If selected, Together creates unique constraints for each alternate key group found in the source ER Diagram.

Related Concepts

Data Modeling

Related Procedures

Data Modeling Procedures

Related Reference

Select Source and Target Objects page

Import Wizard: DB Schema from JDBC

File ▸ Import ▸ Modeling ▸ DB Schema from JDBC

Use this wizard to create a DB Schema from a JDBC connection.

Related Concepts

Data Modeling

Related Procedures

Data Modeling Procedures

Related Reference

Data Modeling Reference
DB Schema from JDBC Import Wizard: Select Objects to Import page
Connect to Database Dialog

DB Schema from JDBC Import Wizard: Select Objects to Import page

Menu ▸ Sub Menu ▸ Command

Use this page to create a JDBC connection and specify which schemata, tables and views located on the remote database should be used as import source.

Option/Button	Description
Connect	Opens the **Connect to Database** dialog.
Source	Lets you select objects from the connected database.
Select All	Selects all objects from the connected database.
Clear All	Clears the selection.
Target project	Selects a target Data Modeling project where the source object will be imported.

Related Concepts

Data Modeling

Related Procedures

Data Modeling Procedures

Related Reference

Data Modeling Reference
Connect to Database Dialog

Connect to Database Dialog

File ▸ Import ▸ Modeling ▸ DB Schema from JDBC ▸ Connect

Use this dialog to specify JDBC database connection options and connect to the database.

Option/Button	Description
Source	Selects the data source.
New	Creates a new connection profile for the selected data source.
Delete	Deletes the selected connection profile.
Name	Specifies the name for the connection profile.
JDBC driver	Specifies the name of the JDBC driver associated with the selected data source.
Lib(s) location	Defines location of the JDBC libraries.
URL	Defines the JDBC URL supported by the selected data source.
URL pattern	Defines the pattern for the JDBC URL.
Prefix	Specifies the prefix of the JDBC URL.
Database name	Specifies the database name.
Host	Specifies the database server host.
Port	Specifies the database server port.
User name	Specifies the name of the user authorized to access the database.
Password	Specifies the user's password.
Apply	Saves the selected connection profile.
Test	Attempts to connect to the database specified in the selected connection profile.

Related Reference

Import Wizard: DB Schema from JDBC

Import Wizard: DB schema from SQL script

File ▸ Import ▸ Modeling ▸ DB schema from SQL script

Use this wizard to create a DB Schema from an SQL script.

Related Concepts

Data Modeling Overview

Related Reference

Select Objects to Import page
Connect to Database Dialog

Select Objects to Import page

File ▸ Import ▸ Modeling ▸ DB Schema from SQL script

Use this page to specify the source SQL script, database server, target project and the target schema name.

Option/Button	Description
File	Specifies the path to the file containing source SQL script.
Open script in editor	If selected, the imported script will be open in the SQL editor.
Server	Selects the server type from the list of available database servers.
Target project	Selects a target Data Modeling project where the target schema file will be stored.
Target schema name	Specifies the target schema name.

Import Pattern Conversion Profiles

Window ▸ Preferences ▸ Patterns ▸ Import...

Use the **Import Pattern Conversion Profiles** dialog box to specify which profiles you want to import to a file, and the file location.

Option	Description
File containing profiles	Selects the file with the profiles that you want to import. Click the adjacent **Browse** button to open the dialog box, which lets you specify the folder containing the file.
Select profiles to import	Displays the list of pattern conversion profiles stored in the specified file. Check the check boxes against profiles that you want to import.
Select All	Checks the check boxes against all profiles in the list.
Deselect All	Unchecks the check boxes against all profiles in the list.

Related Concepts

Patterns and Templates

Related Procedures

Patterns and Templates

Import Together Project Wizard

File ▸ Import ▸ Modeling ▸ Together Project

Use this dialog box to migrate a legacy Together project to the current version of Together.

Migrate legacy Together project to Together <version>

Specify the Together project file and select the migrations type.

Item	Description	
Project Path	Click the Browse button to navigate to a specific source project.	
Diagram folders	This read-only area displays the folders of the legacy project that contain diagrams.	
Design elements storage policy	Use the radio-buttons in this section to define how to handle the design elements (as standalone or as file mates).	
	The same as in the original project	If this option is selected, the settings of the original project are preserved. The existing standalone design elements remain standalone. The new design elements are created according to the project settings.
	Force creating design elements in separate files	If this option is selected, all existing design elements are converted to standalone. All new design elements are created as standalone.
Migration type	Choose one of the possible ways to process the project roots.	
	Merge all roots contents into the new project	Click this radio-button to create a single project from a multi-rooted source project.
	Create a separate project for each root	Click this radio-button to create a Together project for each root.

Merged project name

This page will be displayed if the **Merge all roots contents into the new project** option is selected.

Item	Description
Project name	Enter the name of the resulting project. The default project name is constructed from the names of the last two folders of the source project file location.

Create a set of Together <version> projects

This page will be displayed if the **Create a separate project for each root** option is selected.

Item	Description
Root location	Displays the list of roots of the source project.
Together <version> project name	Displays the default name of the resulting project for the selected root. The default name is constructed from the package prefix, if any. If there is no

	package prefix, the project name is created from the names of the last two folders of the root location. Edit the project name as required.
Content type	Displays information about the type of contents in the selected root (design files or source code).
Diagram format	Displays information about the diagram format in the selected root, if any.
Decision	Select the way to handle information of the selected root. If the root contains design files, you can either copy them to the target location or skip the root. If the root contains source code files, you have the choice to copy it as is, copy and convert it to the design language, or skip the root.

Master project

This page is displayed when multiple projects are created.

Item	Description
Master Project Name	Specify the name of the master project that contains references to all projects created in the course of migration. The default name of the master project is based on the source project name. The master project is created to demonstrate the contents and structure of the source project. It is read-only and not intended for editing. Use the real projects to create or edit contents, and establish dependencies.

Related Concepts

Together Interoperability and Migration

Related Procedures

Importing Legacy Projects

Manage Traces Dialog

Use this dialog to define traces from a model element selected in the Diagram editor or Model Navigator to requirements.

Option/Button	Description
CaliberRM	Selects one or more requirements in the CaliberRM projects tree.
RequisitePro	Selects one or more requirements in the RequisitePro projects tree.
Selected	Displays the requirements currently traced from the model element, as well as any requirements you added to the tab using the **Add** button.
Add	Adds the requirements selected on the **CaliberRM** or **RequisitePro** tabs to the **Selected** tab.
Remove	Removes the selected requirements from the **Selected** tab.
Remove All	Removes all requirements from the **Selected** tab.

Related Concepts

Requirements Management

Related Procedures

Creating Traces from Requirements to Model Elements
Deleting Traces

Modeling Preferences

Window ▸ Preferences ▸ Modeling

Use these preferences to change startup, deletion, error reporting, ignored folders, and team sharing options.

Copy/Paste Tab

Option	Description
Show warning about relationships when elements copied	Before elements are pasted into another package, prompts for a confirmation that relationships between elements will be mapped to the target package. This option is On by default.

Deletion Tab

Option	Description
Show confirmation when element is about to be deleted	Prompts for a confirmation before an element is deleted.
On pressing 'Delete' key always delete from:	
Model	Element is deleted from both model and view.
View only	Element is deleted from view, but remains in model.

Ignored Folders Tab

Use this tab to specify the folders you want Together to ignore. Usually this list contains **CVS, bin, lib** and **doc** directories. Ignored folders are not parsed, so no diagram will be generated for them.

Button	Description
Add	Opens the name field so you can enter a new folder.
Remove	Removes the selected folder.

Referenced projects Tab

Option	Description
Don't show referenced projects content under referring project node.	When this option is On, the content of the referenced project is not shown in the model tree of referring project. Note that in this mode it is impossible to copy content form the referenced project to referring one. This option is Off by default.

Team/Compare Tab

Use this tab to specify how you want to work with Team/Compare menus and version control in the Model Navigator.

Option	Description
Include diagram folders in Team/Compare	This option has an impact when the design and Java roots differ. When it is checked, Team/Compare (context menu) actions respect both folders (merged, seen as a single model node known as package in the Model Navigator). When it is unchecked, only folders from Java root are considered. If you do not store diagram elements in CVS, you should leave this option unchecked. When your diagram folders are in CVS alongside the folders from the Java root and you want to synchronize both of them with one action, you should check this option. A package in the Model Navigator is a logical view of two physical locations. One is the real directory in the project and the other is the directory under the model directory (named Together Model by default) where the Together diagrams are stored (these are updated automatically by Together and probably do not need to be shared). Default state is Off

Related Procedures

Diagrams

Related Reference

Preferences

New MDA Ant Task

Use the **New MDA Ant Task** wizard to add a new Ant task to your Composite transformation.

In This Section

Choose Data Source Type
Use the **Choose Data Source Type** page of the **New MDA Ant Task** wizard to choose how you want to provide parameters required to generate the Ant script for your task.

Select Launch Configuration
Use the **Select Launch Configuration** page of the **New MDA Ant Task** wizard to select which parameters from the existing launch configuration you want to use for your task.

Select Launch Configuration Type
Use the **Select Launch Configuration Type** page of the **New MDA Ant Task** wizard to select the launch configuration type matching the transformation that you want to execute using the Ant task.

Preview
Use the **Preview** page of the **New MDA Ant Task** wizard to display the resulting Ant script fragment generated for your task.

Choose Data Source Type

Use the **Choose Data Source Type** page of the **New MDA Ant Task** wizard to choose how you want to provide parameters required to generate the Ant script for your task.

Item	Description
Enter data manually	Indicates that you want to provide the required parameters manually.
Select existing launch configuration	Indicates that you want to copy the required parameters from an existing launch configuration.

Related Concepts

Model Transformation Support

Related Procedures

Creating a Composite Transformation

Related Reference

QVT Ant Tasks
Model-To-Text Ant Tasks

Select Launch Configuration

Use the **Select Launch Configuration** page of the **New MDA Ant Task** wizard to select which parameters from the existing launch configuration you want to use for your task.

Note: This wizard page is displayed if you choose the **Select existing launch configuration** option on the **Choose data source type** wizard page.

Item	Description
Configurations	Displays the list of MDA launch configurations available in your workspace.

Related Concepts

Model Transformation Support

Related Procedures

Creating a Composite Transformation

Related Reference

QVT Ant Tasks
Model-To-Text Ant Tasks

Select Launch Configuration Type

Use the **Select Launch Configuration** page of the **New MDA Ant Task** wizard to select the launch configuration type that you want to use for your Ant task.

Note: This wizard page is displayed if you choose the **Enter data manually** option on the **Choose data source type** wizard page.

Item	Description
Configurations	Displays the list of available MDA launch configuration types.
Enter launch configuration data...	Displays the **Edit launch configuration properties** dialog box for the selected transformation type.

Note: The **Edit launch configuration properties** dialog box is used here only to collect required task parameters; actually, no real launch configuration is created.

Related Concepts

Model Transformation Support

Related Procedures

Creating a Composite Transformation

Related Reference

QVT Ant Tasks
Model-To-Text Ant Tasks

Preview

Use the **Preview** page of the **New MDA Ant Task** wizard to display the resulting Ant script fragment generated for your task.

Related Concepts

Model Transformation Support

Related Procedures

Creating a Composite Transformation

Related Reference

QVT Ant Tasks
Model-To-Text Ant Tasks

MDL Import Wizard

File ▸ Import ▸ Modeling ▸ Project from MDL file

The MDL Import Wizard is used to import MDL projects created in another application for use in Together.

Option/Button	Description
Add/Add Folder	Specifies the name (or names) of the Rational Rose project file (or files) to be imported (several model files can be imported at once).
Remove	Deletes the selected file or files from the Paths list.
Remove all	Deletes all files from the Paths list.
Scale factor	Specifies the element dimensions coefficient. Default value is 0.3.
Convert Rose default colors	If this option is selected, the default Rational Rose colors will be replaced with the default Together colors. Deselected by default.
Preserve diagram nodes and bounds	If this option is selected, user-defined bounds are preserved in the resulting diagrams. Otherwise, the default values are applied. Deselected by default.
Convert Rose actors	If the option is selected, the Rose actors are mapped to Together actors. Deselected by default.
Generate source code	If this option is selected, a new Java Modeling project is created; otherwise, a Modeling project is created from imported MDL.

Related Concepts

Model Import and Export Overview

Related Reference

MDL Projects Import Options

MDX Import Wizard

File ▸ Import ▸ Modeling ▸ Project from MDX file

The MDX Import Wizard is used to import MDX projects created in another application for use in Together.

MDX Import Options page

Option	Description
Path to the MDX file	Specifies the name of the IBM® Rational® XDE .mdx file.
Scale factor	Specifies the element dimensions coefficient. The default value is 0.03.
Preserve diagram nodes and bounds	If this option is selected, user-defined bounds are preserved in the resulting diagrams. Otherwise, the default values are applied. Deselected by default.
Convert Rose XDE colors	If this option is selected, the Rational Rose XDE colors will be replaced with the default Together colors. Deselected by default.

Model Search and OCL Model Search

Search ▸ Model

Model Search Tab

This tab enables you to search the current diagram or all opened diagrams for the specified string in a certain scope. You can create search strings using wildcards and regular expressions.

OCL Model Search Tab

Together allows you to compose model queries using OCL syntax and use them in model search functionality. For example, to search for all UML 2.0 classes that have the stereotype `MyStereotype`, in the Context field, enter:

```
uml20::classes::Class
```

And in the Invariant field, enter:

```
self.stereotypes->includes('MyStereotype')
```

Related Procedures

- Searching Model Elements
- Searching Model with OCL queries

New Together Project Wizards

This section describes the common pages of the Wizards used to create new Together modeling projects, and language-specific pages for C++ and IDL projects.

In This Section

New Project Wizard Common Pages

File ▸ New ▸ Project ▸ Modeling ▸ < Project type>

These pages are common for the majority of project types provided by Together. For more information, refer to the topics that describe project settings for the specific project types.

Modeling Project page	
Project name	Use this text field to enter the project name.
Use default location	If this option is checked, the new project is created in the current workspace.
Location	Use this field to define the project location. This field is only available when the **Use default location** option is not checked.

Modeling Settings page	
Metamodel	These controls are only available for Java projects. You can choose the UML version to comply with. The default option is UML 2.0.
Start with Diagram	If this option is checked, the new project starts with the default package diagram. If this option is not checked, you can select the type of starting diagram from the drop-down list and specify its name.
Store package properties in package diagram files	If this option is checked, all properties of the package diagram, both visual and semantical, are preserved in the `default.txvpck` diagram file. If this option is not checked, only diagram-specific information (visual information, such as layout) is retained in the `default.txvpck` diagram file, while settings that you treat as package properties (semantical information, such as descriptions and custom properties) are moved from the `default.txvpck` file into the `default.txaPackage` file. Turning this option off allows you to track your package changes using version control. This option is on by default.
Create design elements in separate files	If this option is checked, the design elements are created as standalone. If this option is not checked, all design elements are stored in one file as file mates.

Profiles page	
Available Profiles	Displays the list of available profiles with checkboxes. The **Profile Description** field displays a brief description of the selected profile.
Select all	Checks all available profiles.
Deselect all	Unchecks all profiles.
Set Defaults	Resets profiles to the default settings

Related Procedures

Creating a Project

Related Reference

New project Wizard C++ Language-Specific Options
New project Wizard IDL Language-Specific Options

New project Wizard C++ Language-Specific Options

File ▸ New ▸ Project ▸ Modeling ▸ C++ Modeling Project

Access the properties for your existing project via **Project ▸ Properties** or **Project context menu ▸ Properties**.

Tab	Description	
Project Source Path	Use this tab to define projects paths.	
	Use as a source folder	Use this button to add the selected package to build path. Add the folder corresponding to the package to the build path if the package is the root of packages and source files. Entries on the build path are visible to the compiler and used for building.
	Remove from build path	Children of the folder will not be seen by the compiler anymore and will not be included when building the project.
	Toggle Read-Only Status	Use this button to make selected roots read-only or to clear the read-only attribute.
	Make Default Root	Use this button to choose the selected folder as the default root. This root is used as a target container when automatically creating new files.
	Exclude/Include	Use these alternative buttons to make the folder contents invisible or visible to the compiler.
	Configure inclusion and exclusion filters	Use this button to create the inclusion and exclusion filters instead of including and excluding each folder or file manually.
	Configure entry point	Use this button to add selected files from a package to the project in the **Configure Entry Points** dialog. The dialog displays a model tree with the check boxes for each file or folder. If a node is checked, it is considered an entry point. If a root is added to the project, all `*.cpp` files are automatically included in the project, but the header files should be added individually.
	Link additional source to project	Use this button to open the **Link Additional Source** dialog and add the sources that reside outside of the project.
Include Paths	Use this tab to include search paths to the project.	
	Include search paths	Folders in this area are included in the search path.
	Add	Click this button to add a folder to the project search path. Enter the path to the text field, or use the **Browse** button to locate the specific folder.
	Edit	Click this button to modify the include folder.
	Remove	Click this button to delete the selected folder from the path.
C++ Processing Settings	Use this tab to define C++ specific settings.	

	C++ generating class name prefix	Each new class name starts with the specified prefix.
	C++ generating definition file extension	Each new class has the specified extension.
	C++ generating file name prefix	Name of the file that contains C++ classes starts with the specified prefix.
	C++ generating file extension	File name has the specified extension.
	Support **wchar_t** as keyword	If this option is checked, the compiler recognizes the keyword **wchar_t** as a data type.
	Enable messenger	If this option is checked, the decorators describing compilation errors will display in the **Problems View** and **Resource Navigator**.
	Package filter	Enter the names of the packages that you would like to filter out.
	Skip standard includes	If this option is checked, standard includes are ignored.
	Predefined macros	Specify the list of predefined macros, which will be available for the whole project.
	Preinclude file name	Specify the name of the preinclude file (if any) to make its contents available for all participants of the project.
	New file default head comment	Specify the text that will be displayed in the generated C++ files.
	Recognize free comments as doc	If this option is checked, the Javadoc comments will be recognized.
	C++ dialect support	Select a C++ dialect from the list. The possible options are: GNU, MS, or pure C++.
Add CDT features to project	If this option is checked, CDT features become available in the project. You can access these features in the project properties dialog.	

Related Procedures

Creating a Project

Related Reference

C++ Projects

New project Wizard IDL Language-Specific Options

File ▸ New ▸ Project ▸ Modeling ▸ IDL Modeling Project

Access the properties for your existing project via **Project ▸ Properties** or **Project context menu ▸ Properties**.

Tab	Description	
Project Source Path	Use this tab to define projects paths.	
	Use as a source folder	Use this button to add the selected package to the build path. Add the folder corresponding to the package to the build path if the package is the root of packages and source files. Entries on the build path are visible to the compiler and used for building.
	Remove from build path	Children of the folder will not be seen by the compiler anymore and will not be included when building the project.
	Toggle Read-Only Status	Use this button to make selected roots read-only or to clear the read-only attribute.
	Make Default Root	Use this button to choose the selected folder as the default root. This root is used as a target container when automatically creating new files.
	Exclude/Include	Use these alternative buttons to make the folder contents invisible or visible to the compiler.
	Configure inclusion and exclusion filters	Use this button to create the inclusion and exclusion filters instead of including and excluding each folder or file manually.
	Link additional source to project	Use this button to open the **Link Additional Source** dialog and add the sources that reside outside of the project.
Include Paths	Use this tab to include search paths to the project.	
	Include search paths	Folders in this area are included in the search path.
	Add	Click this button to add a folder to the project search path. Enter the path to the text field, or use the **Browse** button to locate a specific folder.
	Edit	Click this button to modify the include folder.
	Remove	Click this button to delete the selected folder from the path.
IDL Processing Settings	Use this tab to define IDL-specific settings.	
	Preinclude file name	Specify the name of the preinclude file (if any) to make its contents available for all participants of the project.
	Show typedefs as classes	If this option is checked, typedefs display as classes in diagrams.
	Skip standard includes	If this option is checked, standard includes are ignored.
	Show natives as classes	If this option is checked, all types marked as natives display as classes in diagrams.
	Rename file when renaming class	If this option is checked, the container file is renamed together with its class.

Warn about not found include files	If this option is checked, a warning is displayed for the missing files.
Use preprocessor	If this option is checked, the existing macros are opened and includes are attached.
Predefined macros	Specify the list of predefined macros that will be available for the whole project.
Copy non-doc comments	If this option is checked, free comments are copied or moved together with the elements located next to them.
Recognize free comments as doc	If this option is checked, the Javadoc comments will be recognized.

Related Procedures

Creating a Project

New project Wizard Data Modeling Specific Options

File ▸ New ▸ Project ▸ Modeling ▸ Data Modeling Project

Access the properties for your existing project via **Project ▸ Properties** or **Project context menu ▸ Properties**.

Option	Description
Server	Choose the target database server to which the physical data model is bound.
Default schema	If this option is checked, the default schema with the specified name will be created during project creation. If this option is not checked, the project will be created without a schema. You can add a schema later using the **New** command of the project context menu.
Schema name	This field is only available when the **Default schema** option is checked. Use this text field to specify the name of the default schema.

Related Procedures

Creating a Data Modeling Project

Convert MDL Wizard

Use this wizard to create a design project around an existing IBM Rational Rose (MDL) model. The wizard is invoked by the **Design Projects** ▸ **Convert from MDL** template of the **New Project** dialog box.

Paths section

Button	Description
Add	Adds one model file to the Paths section. Press this button to open the **Select Model File** dialog box, navigate to the desired model file and click Open.
Add Folder	Adds all model files in the selected folder. Press this button to open the **Browse for Folder** dialog box, navigate to the desired folder that contains the model files and click OK.
Remove	Press this button to delete the selected entry from the Paths section.
Remove all	Press this button to delete all model files from the Paths section.

Options section

Option	Description
Scale factor	Specify the element dimensions coefficient. By default, the scale factor is 0.3.
Convert Rose default colors	If this option is checked, the default Rational Rose will be replaced with the default Together colors.
Preserve diagram nodes bounds	If this option is checked, user-defined bounds are preserved in the resulting diagrams. Otherwise, the default values are applied.
Convert Rose actors	This options enables you to choose mapping for the Rose classes with actor-like stereotypes (Actor, Business Actor, Business Worker, Physical Worker). If the option is checked, the Rose actors are mapped to Together actors. If the option is not checked, the Rose actors are mapped to the classes with the Actor stereotype.

Related Procedures

Importing a Project in IBM Rational Rose (MDL) Format

Related Reference

Together Projects

Print Audit dialog box

Audit results pane ▸ Print button

This dialog box enables you to print selected sets of audit report results to the specified printer. The dialog box is invoked from the audit results report view.

Select View	Choose the scope of the results to print using the Select View list box. Audit results display in tabbed-pages in the audit results report view. You can group and ungroup the results using the **Group by** command on the report view context menu. Unless the results have been grouped using the **Group by** command, the **Active Group** option is not enabled in the dialog. The possible view options are: **All Results**: If the results are grouped, choosing **All Results** prints a report for all groups in the current tabbed-page. If the results are not grouped, all results print for the current tabbed-page. **Active Group**: If the results are grouped, you can select a group in the current tabbed page to print a report for the selected group. **Selected Rows**: You can select single or multiple rows in the audit results report view. Choosing **Selected Rows** prints a report for such selections.
Print zoom	Type in a zoom factor for the printout. By default, the zoom factor is set to 1.
Fit to page	Check this option if you want to print the results on a single page. If checked, the Print zoom field is disabled.
Preview	Click the down arrow to show the print preview page.
Preview zoom	Use the Preview zoom (auto) slider to set the preview zoom. The current value of the zoom factor is displayed to the left of the slider.
Auto preview zoom	Check this option to fit the image to the preview window.
Buttons	
Print	Click Print to send the selected audits report to the default printer. Use the down arrow to choose the Print dialog command, which enables you to configure the printer options.
Cancel	Click to close the dialog box without printing the audits report.
Help	Clicking Help opens this page.

Related Procedures

Viewing Audit Results
Printing Audit Results

Print Diagram Dialog Box

File ▸ Print

This dialog box enables you to print selected diagrams to the specified printer. The dialog box is invoked by choosing **File ▸ Print** from the main menu with a diagram open in the **Diagram View**.

Print diagrams	From this list box, choose the diagrams to be printed. The possible options are: Active diagram Active with neighbors (all diagrams within the same namespace) All opened diagrams All diagrams in the model
Print zoom	Enter a zoom factor for the printout. By default, the zoom factor is set to 1.
Fit to page	Check this option if you want to print the diagram on a single page. If checked, the Print zoom field is disabled.
Preview	Click the down arrow to show the print preview page.
Preview zoom	Use the slider to set up the preview zoom. The current value of the zoom factor is displayed to the left of the slider.
Auto preview zoom	Check this option to fit the image to the preview window.
Print	Press this button to send the selected diagrams to the default printer. Use the down arrow to choose the Print dialog box command, which enables you to configure the printer options.

Related Procedures

Printing Diagrams

Print Dialog

File ▸ Print

Option/Button	Description
Current	Prints only the current diagram.
Current with subdiagrams	Prints the current diagram and, recursively, any other diagram it contains.
All opened	Prints all diagrams currently open in the Diagram editor.
All in Model	Prints all diagrams open in the current Model.
Print	Opens the standard Print dialog from which you can select and setup your printer.
Preview	Expands the dialog to display the diagram as it will be displayed when printed (see below).
Print Options...	Displays the Print Preferences dialog. For more information, see “Print Preferences.”

Preview Dialog

Option/Button	Description
Scale	Use this option to obtain the required magnification in the Preview pane of the Print dialog box.
Print whole diagram as an image	Select this option to print the diagram as an image.
Print diagram as black and white image	Select this option to print the diagram in black and white.

Related Reference

Print Preferences
Printing Diagram Elements

Project Properties

Project ▸ Properties

You can modify the Together Project properties using the **Properties** dialog box. In the Model Navigator or in the Navigator, right-click a project and choose **Properties** on the context menu.

Among the general properties for a project, there are some properties specific to Together projects:

Open this property page...	to...
Model path	— Define the name of the model folder used to store diagrams and design elements. — Enable cross-project references in the list of imported projects.
QVT Settings	— Specify the Java source folder that is used for storing Java code generated by QVT Builder. This page is available for transformation projects only.
Profiles	— Select profiles to be used in the project.
QA Model	— View or modify the sets of audits and metrics to be used in the project. Note that you can only change sets of audits and metrics if you check the **Override workspace settings** check box.
QA Builder Properties	— View or modify the set of audits that is used during your source project build. Note that you can only change the set of audits if you check the **Enable project specific settings** check box.
Store package properties in package diagram files	— Preserves all properties of the package diagram, both visual and semantical, in the `default.txvpck` diagram file. If this option is not checked, only diagram-specific information (visual information, such as layout) is retained in the `default.txvpck` diagram file, while settings that you treat as package properties (semantical information, such as descriptions and custom properties) are moved from the `default.txvpck` file into the `default.txaPackage` file. Turning this option off allows you to track your package changes using version control. This option is on by default. Changing this option from this dialog converts the project files.
Create elements in separate files	— Create elements as standalone files. If this option is not checked, all design elements are stored in one file as file mates.
Sort elements in design files	— Enforces sorting of elements in the design and diagram files. Originally information saved without predefined order. Thus it was quite possible that minor change caused subsequent revisions of a file to look very different if compared as plain text or XML. If this option is checked information is ordered on save. Note: normally this is not needed and also it may take extra CPU time. Please don't change unless you clearly understand the implications.

Related Concepts

UML Profiles

Related Reference

QVT Settings
QA Builder Properties

Project Specific Configuration

Use the **Project Specific Configuration** dialog box to select a C++ or Java project for which you want to configure QA Builder properties.

Item	Description
Select the project to configure	Lists projects in your workspace that contain code supported by the respective QA Builder.
Filter project with no project specific settings	Check this check box if you want to hide projects that contain project-specific settings.

Related Concepts

Quality Assurance

Related Reference

QA Source

Property Iterator Properties

Template Designer ▸ Property Iterator section context menu ▸ Properties

Use this dialog box to access properties of the property iterators. The dialog displays the following tabs: **Iteration Scope**, **Sorting**, **Output Style** and **Other**. This topic describes the **Iteration Scope** and **Sorting** tabs. For the description of **Output Style** and **Other** tabs, refer to the “Folder Section Properties” topic.

Iteration Scope tab

Choose an iteration scope from the list. Depending on your choice, the dialog displays different controls.

Iteration scope	Description	
All User-Defined Properties	Iterates over the properties that are not described in the metamodel.	
	Exclude already iterated properties	If this option is checked, all properties that were already iterated for the current element are skipped.
	Iterate only unknown properties	If this option is checked, only those properties that were not included in the metamodel are included.

	Filter expression	Use this field to restrict the search scope to satisfy the filter condition. Expressions can use DG Variables available inside Property Iterators. Refer to “DG Variables” reference for details.
		
		Example of a filter expression for Property Iterator:
		
		`context: OclAny`
		
		`body: (getDGVariable ('curPropertyFullName') <> 'Stereotype') and (getDGVariable ('curPropertyFullName') <> 'Visibility')`
		
		Click the editor button to open the Expression editor.

Set of properties	These are properties that belong to the metatype of the parent element iterator. A property iterator can iterate over multiple properties. Use CTRL + CLICK to select multiple properties from the Available properties list, and then use the double-arrow button to move these properties to the Selected properties list. You can change the order in which the properties are documented by arranging the properties in the Selected properties list.	
Instances of a single property	These are properties that can have multiple values, for example, `@see` or `@author`. Use the Filter expression to restrict the search elements that satisfy the filter properties.	
All properties	Iterates by all properties defined for the current element metatype (only properties described in the metamodel are iterated). Use the Filter expression to restrict the search elements that satisfy the filter properties.	

Sorting tab

Use this tab to specify the order in which the elements are to be searched and thus documented.

Option	Description
Sorting mode	The following options are available: none, by name, by value, by key expression.
Reverse order	Elements are always documented in ascending order. Check the Reverse scope order box to list elements in descending order.

Related Concepts

Documentation Template Sections

Related Procedures

Setting Section Properties

Related Reference

Folder Section Properties
DG functions in Formulae Expressions
DG Variables

QA Builder Properties

Project ▸ Properties ▸ QA Builder Properties

Use the **QA Builder Properties** dialog box if you want to customize the workspace-level set of audits for the QA Builder to suit your C++ or Java project needs.

Item	Description
Run incremental QA builder	Check this option if you want to enable the incremental QA builder. Using the incremental project builder allows you to define project-specific QA settings.
Enable project-specific settings	Check this option if you want to override QA builder settings defined at the workspace level.
Configure Workspace Settings	Displays the **QA Builder Preferences** dialog box, which lets you change QA builder settings defined at the workspace level.
Current set	Displays the name of the current QA Builder set.
Load set of options from the file	Opens the **Choose configuration file** dialog box, which lets you load a file containing a QA set (*.qa).
Save set of options to a file	Opens the **Choose configuration file** dialog box, which lets you save current QA set to a file (*.qa).
Expand all nodes	Expands all categories.
Collapse all nodes	Collapses all categories.
Select all	Selects all categories and all elements within categories. Selected categories will be included in the current QA set.
Clear all	Deselects all categories and all elements. Deselected categories will not be included in the current QA set.
Find an analyzer	Opens the **Find Analyzer** dialog box, which lets you quickly find a necessary analyzer in any category.
Property	Displays the properties of the selected element used in the calculation of the audit.
Value	Click a row under **Value** to edit the property's value.
Restore Defaults	Restores the selection to default settings.

Related Concepts

Quality Assurance

Related Reference

QA Model
Find Analyzer Dialog
Project Specific Configuration

QA Search

Right-click in the Audit or Metric view ▸ Search In...

Use the **QA Search** dialog box to search within Audits View or Metrics View.

Audit Search Options

Option	Description
Search in	Narrow your search to one of the following: Severity of audit result, General description of audits, Name of resource that caused the violation, Path location of the above resource, Specific line the problem occurred on.
Direction	Starting from the currently selected line, search moves in the direction chosen.
Search	The text you want to locate. This drop-down list includes previously searched for strings.

Metric Search Options

Option	Description
Search in	Together searches according to the name of the resource.
Direction	Starting from the currently selected line, search moves in the direction chosen.
Search	The text you want to locate. This drop-down list includes previously searched for strings.

Related Reference

Audit View
Metric View

QVT Settings

Use the **QVT Settings** page of the **Project Properties** dialog box to specify if you want to generate Java code for your MDA transformation project, and choose a Java container where you want to store the code.

Item	Description
Generate Java code	Specifies if you want to generate Java code for your MDA transformation project.
Java source container	Specifies a Java container to store your code.

Related Concepts

Model Transformation Support

Related Procedures

Applying Model-To-Text Transformations

Related Reference

QVT Builder

Run

Use the **Run** dialog to create, manage, and run configurations.

In This Section

Model-To-Text Application

Run ▸ Run... or Debug... ▸ Model-To-Text Application

Use the **Model-To-Text Application** page of the **Run** or **Debug** dialog box to create, manage, and run (debug) configuration for your compiled Model-To-Text transformation.

Transformation tab

Item	Description
Transformation file	Specifies the transformation file. The adjacent **Browse** button opens the **Select Transformation** dialog box which, which lets you choose the file in your workspace.
Source model URI	Specifies the URI of the source model.
Target folder	Specifies the folder where you want to store the transformation results. The adjacent **Browse** button opens the **Select Folder** dialog box, which lets you choose the folder in your workspace.
Refresh target folder on debug events	Specifies if you want to monitor changes in the target folder during the debugging process.

Workspace tab

Item	Description
Projects used	Specifies projects that you want to add to the temporary debugging workspace.

Related Concepts

Model Transformation Support

Related Reference

QVT Language

Model-To-Text Transformation

Run ▶ Run... ▶ Model-To-Text Transformation

Use the **Model-To-Text Transformation** page of the **Run** dialog box to create, manage, and run configuration for your Model-To-Text transformation.

Transformation tab

Item	Description
Transformation file	Specifies the transformation file. The adjacent **Browse** button opens the **Select Transformation** dialog box, which lets you choose the transformation file in your workspace.
Source model URI	Specifies the URI of the source model.
Target folder	Specifies the folder where you want to store the transformation results. The adjacent **Browse** button opens the **Select Folder** dialog box, which lets you choose the target folder in your workspace.

Related Concepts

Model Transformation Support

Related Procedures

Applying Model-To-Text Transformations

Related Reference

QVT Language
QVT Editor

QVT Interpreter

Run ▸ Run... ▸ QVT Interpreter

Use the **QVT Interpreter** page of the **Run** dialog box to create, manage, and run configuration for your transformation, which runs using the QVT Interpreter.

Transformation tab

Item	Description
Transformation module	Specifies the transformation file. The adjacent **Browse** button opens the **Select Transformation** dialog box, which lets you choose the transformation file in your workspace.
Source model URI	Specifies the URI of the source model.
Target type	Selects where you want to store the transformation results:

New model	Saves transformation output to a new EMF resource with URI specified in the **URI** field.
Existing container	Adds transformation output to a containment feature (reference) of an existing model element. Specify the container element in the URI field, and the provide feature name in the Feature field. Clicking the **Select** button displays a dialog presenting all containment references of the selected element. If the **Clear contents** option is checked, all elements contained by the selected reference will be removed before adding the transformation output. This checkbox has no effect for the references with multiplicity 1.
Inplace	Directly modifies the transformation input and returns it as the transformation result (inplace transformation). The URI specified in the **Source model URI** field is used as URI of the transformation output.

Item	Description
URI	Specifies the URI of the target model element when you select **New model** or **Existing container** as the target type.
Browse...	Opens the **Workspace Contents** dialog box which, which lets you choose the target model or model element in your workspace.
Feature	Specifies the name of a container within the selected target model or model element where you want to store the transformation results. Enabled when you select **Existing container** as the target type.
Select...	Opens a dialog box that allows you to choose the feature from the list of all containment references of the selected element.
Clear contents	Specifies if you want to remove elements specified in the chosen reference before you store the transformation results. For the references with multiplicity 1, this option has no effect.
Generate trace file	Specifies if you want to generate the trace file for the transformation. If selected, lets you specify the name and location of the file.

Configuration tab

Item	Description
Property	Specifies the name of the property that is passed to the transformation.
Type	Specifies the property type.

Value	Specifies the property value. You can edit this field.

Related Concepts

Model Transformation Support

Related Procedures

Applying Model-To-Model Transformations
Applying Model-To-Text Transformations

Related Reference

QVT Language
QVT Editor

QVT Transformation

Run ▸ Run... ▸ QVT Transformation

Use the **QVT Transformation** page of the **Run** dialog box to create, manage, and run configuration for your QVT transformation.

Transformation tab

Item	Description	
Transformation module	Specifies the transformation file. The adjacent **Browse** button opens the **Select Transformation** dialog box, which lets you choose the transformation file in your workspace.	
Source model URI	Specifies the URI of the source model.	
Target type	Selects where you want to store the transformation results:	
	New model	Saves transformation output to a new EMF resource with URI specified in the **URI** field.
	Existing container	Adds transformation output to a containment feature (reference) of an existing model element.
	Inplace	Directly modifies the transformation input and returns it as the transformation result (inplace transformation). The URI specified in the **Source model URI** field is used as URI of the transformation output.
URI	Specifies the URI of the target model element when you select **New model** or **Existing container** as the target type.	
Browse...	Opens the **Workspace Contents** dialog box which, which lets you choose the target model or model element in your workspace.	
Feature	Specifies the name of a container within the selected target model or model element where you want to store the transformation results. Enabled when you select **Existing container** as the target type.	
Select...	Opens a dialog box that allows you to choose the feature from the list of all containment references of the selected element.	
Clear contents	Specifies if you want to remove elements specified in the chosen reference before you store the transformation results. For the references with multiplicity 1, this option has no effect.	
Generate trace file	Specifies if you want to generate the trace file for the transformation. If selected, lets you specify the name and location of the file.	

Configuration tab

The **Configuration** tab lists properties defined in your QVT transformation script and allows to specify their values.

Item	Description
Property	Specifies the name of the property which is passed to the transformation.
Type	Specifies the property type.
Value	Specifies the property value. You can edit this field.

Related Concepts

Model Transformation Support

Related Procedures

Applying Model-To-Model Transformations
Applying Model-To-Text Transformations

Related Reference

QVT Language
QVT Editor

XSL Transformation

Run ▶ Run... ▶ XSL Transformation

Use the **XSL Transformation** page of the **Run** dialog box to create, manage, and run configuration for your XSL transformation.

Transformation tab

Item	Description
Transformation file (Workspace)	Specifies if you want to use the transformation file stored in your workspace. The adjacent **Browse** button opens the **Select Transformation** dialog box, which lets you choose the transformation file.
Transformation file (File System)	Specifies if you want to use the transformation file stored in the file system. The adjacent **Browse** button opens the **Open** dialog box, which lets you choose the transformation file.
Source model URI	Specifies the URI of the source model.
Target file	Specifies the URI of the target file. The adjacent **Browse** button opens the **Workspace Contents** dialog box, which lets you choose the file location in your workspace.
Add	Opens the **Add** dialog box, which lets you add a property to the list of parameters of your XSL transformation.
Remove	Removes the selected property from the list of parameters.

Related Concepts

Model Transformation Support

Related Procedures

Applying Model-To-Model Transformations
Applying Model-To-Text Transformations

Related Reference

XSL Editor

Launch BPMN Simulation

Run ▶ Run... ▶ Launch BPMN Simulation

Use the **Simulation** tab of the **Run** dialog box to specify BPMN simulation run parameters.

Item	Description
Name	Specifies the name of the run configuration.
Diagram to simulate	Lists open diagrams that are available for simulation in the project, and diagrams selected earlier using the **Browse** button.
Browse	Browse for available diagrams to simulate.
Time format	Specifies time format for the run. If you select **Natural time**, start and end time should be specified in the format “`M/d/y H:m`”.
Time unit	Specifies time units when **Natural time** is selected in the **Time format** list. You can choose from second, minute, hour, day, week, month, and year. Note that 1 year = 365 days and 1 month = 30 days.
Execution time	Specifies start and end time for the simulation run. Select **Infinite end time** to withdraw end time restriction.
Animation settings	Specifies animation settings including animation speed, animation visualization, and a step-by-step execution (each step is to be confirmed by a user).
Report settings	Specifies the output folder for the report file and whether to open the report in a browser after simulation is complete.

Related Concepts

Business Process Modeling

Related Procedures

Together Business Process Modeling

Run QA

Context menu ▸ QA Source ▸ Audits (Metric)

Use the **Run QA** dialog box to choose resources that you want to process when running quality assurance.

Item	Description
Select resources to process	Displays the resource tree from which you can choose the resources you want to process when running quality assurance.
Remember my decision	Check this check box if you want to disable this dialog box for the selected project in the future.
Preferences...	Opens the **QA Source** page of the **Preferences** dialog box.

Related Concepts

Quality Assurance

Related Procedures

Running Source Code Audits
Viewing Audit Results

Requirement Traces Search Dialog Box

Search ▶ Requirement Traces Search

Use this dialog box to search for specific elements that have traces to requirements, or find all traced elements in the defined search scope.

Option	Description
Containing text	Provides a space for typing text to search (consider the note below).
Search all traced elements	Searches for all traced elements within the specified search scope.
Case sensitive	Specifies whether to distinguish between uppercase and lowercase characters when searching for occurrences of the text typed in the **Containing text** field.
Regular expression	Specifies whether regular expressions are used in the **Containing text** field.
Available integrations	Filters the found traces by the selected requirements vendors (CaliberRM and/or RequisitePro).
Available attributes	Selects in which of the trace attributes the search is performed (server name, project name or requirement name). Note: The server field is `localhost` for all requirements for RequisitePro.
Scope	Defines the search scope: **Workspace**, **Selected resources**, **Enclosing projects** and **Working set**.
Search	Click to find traced elements matching your search criteria. The found elements are displayed in the Eclipse Search view.

Note: Together compares the search string with the whole contents of each of the trace attributes defined in the **Available Attributes** field. Use * and ? wildcards if you want to find traces where the search string is displayed as a substring of an attribute value. The * symbol stands for any text, ? - for any symbol. For example, if **Server** is selected in **Available Attributes**, the search string `*localhost*` finds only those traces which are stored locally.

Related Concepts

Requirements Management

Related Reference

Searching for Traced Elements

Select element dialog box

This dialog box displays a tree view of the available contents within your project. Expand the project nodes to reveal the nested classes, select the required element, and click **OK** when ready.

This dialog box belongs to a general group of selection dialogs where you can choose interactions, operations, ancestor classes, instantiated classes for the objects, and so on. This dialog opens when you press the chooser button in a field of the Properties View, or when **More** is selected from the **Choose Class** or **Choose Method** menu nodes.

Related Procedures

Instantiating a Classifier
Working with a Collaboration Use

Selection Manager

This dialog belongs to a general group of selection dialogs where you can select elements from the available contents and add them to a certain destination scope. All Selection Manager dialogs have a similar structure and varying title strings.

Dialog title:	The title of the dialog varies depending on the way it is invoked. It displays the string that corresponds to the invoking object or property.
Model Elements tab or Diagram elements tab:	The pane on the left of the dialog displays the content available in your project. You can use the explorer to navigate to the element and select it for inclusion in the pane of values returned by the dialog to the invoking object.
Existing and/or ready to add:	This pane displays two kinds of data: Values already existing and passed from the invoking object, if any. Values of the selections you have added from the left-hand pane, if any.
Add:	Enabled when an element is selected in the left-hand pane. Adds the selected element to the right-hand pane.
Remove:	Enabled when you select an item in the right-hand pane. Removes the selected item from the pane. All removed values or objects are removed from the invoking property or diagram upon clicking OK.
Remove All:	Enabled when items are present in the right-hand pane. Removes all items from that pane. All removed values or objects are removed from the invoking property or diagram upon clicking OK.

Related Procedures

Creating a Shortcut
Hyperlinking Diagrams
Hiding and Showing Model Elements

Static Section Properties

Template Designer ▸ Static section context menu ▸ Properties

Use this dialog box to access properties of the static sections. The dialog displays the only **Settings** tab.

Item	Description
Enable condition	An enable condition for turning this section on or off. Use the Expression editor to specify an enable condition using the OCL or legacy notation.
Disabled	Check this option to skip the section.

Related Concepts

Documentation Template Sections

Related Procedures

Setting Section Properties

Template Properties Dialog Box

Template Designer toolbar ▸ Show Template Properties

Use this dialog to view and modify properties of a documentation template.

General tab

Option / Button	Description	
Model	UML Metamodel.	
Template type	Documentation template.	
Template Description	Enter description of the template.	
Report Title Expression	Click the **Editor** button to open the **Edit Expression** dialog.	
Root Object Metatype	Select metatype from the drop-down list.	
Formatting Template	Specify the path to the Microsoft Word document whose styles should be used as formatting styles.	
Headers and Footers	Report Header	If the option is checked, the report header is included in the generated output.
	Report Footer	If the option is checked, the report footer is included in the generated output.
	Page Header	If the option is checked, the page header is included in the generated output.
	Page Footer	If the option is checked, the page footer is included in the generated output.

Page Settings tab

Option / Button	Description	
Page Type	Choose page type from the drop-down list. If the value *User* is selected, specify a page width and height.	
Page margins	Left//Right	Specify left and right page margins in millimeters. The default value is 12.7 mm.
	Top/Bottom	Specify top and bottom page margins in millimeters. The default value is 15.24 mm.
Page Orientation	Click either the **Portrait** or **Landscape** radio-button. The default value is *Portrait*.	

Formatting Styles tab

Option / Button	Description
Formatting Styles	Use the **New** button to create a new style in the **Style** dialog, and add the created style to the list of available styles. The **Delete** button removes the selected style from the list of available styles. Use the **Edit** button to modify the selected style in the **Style** dialog. Note that the **Delete** and **Edit** buttons are disabled for the styles from the attached document.

Template Parameters tab

Option / Button	Description	
Parameter	Enter the parameter name.	
Description	Enter optional description.	
Default Value	Enter optional default value.	
Buttons	Set	Click this button to add a parameter to the list.
	Delete	Click this button to remove the selected parameter from the list.

Related Procedures

Creating Custom Documentation Template

Related Reference

Documentation Template Properties
Frameset Template Properties

Trace Synchronizer Dialog Box

Trace Synchronizer View ▸ Trace Synchronizer button on the toolbar

Use this dialog box to search for traced requirements or model elements that become desynchronized in local and server copies. The found desynchronized traces are displayed in the Trace Synchronizer view.

Tab	Description
Requirements	Lets you select requirements, requirement types, baselines, projects and connections to search for desynchronized traced requirements.
Model Elements	Lets you select model elements in your current Eclipse workspace to search for desynchronized traces.

Related Concepts

Requirements Management

Related Procedures

Opening Requirements Views
Synchronizing Traces

Related Reference

Trace Synchronizer View

XMI Export Wizard

File ▸ Export ▸ Modeling ▸ XMI file

The XMI Export Wizard is used to export projects or sections of projects created in Together for use by other applications and languages.

Export Project to XMI File

Open projects list box	Displays currently open Together projects, which you can export as XMI data. Click on the plus sign to expand a project and select only a portion of it for export. You can select only one project at a time.
Select XMI type	Select an XMI type for export. Options: XMI for UML 1.3 (Unisys Extension) XMI for UML 1.3 (Unisys Extension, Recommended for TCC) XMI for UML 1.3 (Unisys Extension, Recommended for Rose) XMI for UML 1.4 (OMG) XMI for UML 2.0 Note: TCC stands for Together ControlCenter. XMI for UML 2.0 compliant with OMG standard XMI for UML 2.1
XMI Version	Specifies the version of XMI to be exported.
XMI Encoding	Specifies the XMI encoding setting.
XMI file	Specifies the path and file name to be used. Together will create these if they do not exist. You may enter a name and path or accept the default
Use prefix of imported root	Enabled for UML 1.4 projects that have imported roots. If this option is checked, a top-level package with the same name as the imported project prefix (specified in **Project ▸ Properties ▸ Model Path**) is created for each imported root.

Run Audits on Exported Project

The **Part-Port Audit** is provided for UML 2.0 projects. This audit provides the possibility to resolve problems that occur in Together 2006 models, where it was possible to add a port to a part. Such ports can be moved to the chosen classifier when one decides to fix the problem found by audit before the XMI export.

The **Required/Provided Interface Audit** is provided for UML 2.0 projects. It searches for Required/Provided Interface links with a null supplier. When the problem found by this audit gets fixed before the XMI export (by clicking **Fix All**), the link target can be changed to the chosen interface.

Related Concepts

Model Import and Export Overview

Related Procedures

XMI Export and Import of the Models with Cross-Project References

XMI Import Wizard

File ▸ Import ▸ Modeling ▸ XMI file

The XMI Import Wizard is used to import XMI projects or sections of projects created in another application for use in Together.

Note: For UML 1.4 and Java Modeling projects, only XMI 1.1/1.2 import is supported. Attempting to import an XMI 1.0 file results in an empty project. After selecting XMI File from the Import wizard, the XMI Import wizard's only dialog opens.

Select the Source File	Specifies the full path to the .xml, .xmi, or .uml2 file you are importing.
Open projects list box	Displays a list of open Together projects into which you can import the XMI data.
Select XMI Type of input file	The radio-buttons in this read-only section turn on corresponding to the type of the chosen project.

Related Concepts

Model Import and Export Overview

Legal Notices for Together

Index