Using Rational SoDA® for Word

Version 2001A.04.00

Rational®
the e-development company™

support@rational.com
http://www.rational.com

IMPORTANT NOTICE

Contents

3 Generating Reports and Documents

4 Customizing a Template

1 Installing Rational SoDA for Word

Installation Overview

The Rational Software Setup program lets you perform standard and custom installations of Rational software products.

This document provides you with:

- An overview of the installation procedures for Rational software products, included in this chapter.
- Software licensing description and procedures. The online help for Rational License Key Administrator contains detailed information about licensing activities.
- Information needed to perform a typical installation of Rational SoDA for Word.. Release notes are available in your SoDA installation “docs” directory in Microsoft Word format. Online documentation is available in PDF. Adobe Acrobat Reader is required to view PDF files. A copy of the Adobe Acrobat Reader installation kit is available on the Rational Solutions for Windows CD in the [extras] directory. Rational Suite and other Rational products are on a separate Rational Solutions for Windows – Online Documentation CD.
- Support information, including references to additional sources of information for Rational software and licensing. See “Technical Support Information” on page 9.

Installing Rational Software Products and License Keys

This section provides a summary of the steps for installing Rational software products and the FLEXlm licensing software.

Note This guide assumes drive C as your default installation drive. Substitute your actual installation drive name, as needed.

Installation Quick Start

Table 1, Installation Quick Start Guide, summarizes the steps for installing Rational software and license keys:

Table 1: Installation Quick Start Guide

Step	For More Information
Install Rational software from the Rational Solutions for Windows CD. Make certain that you are installing the product you have purchased.	See the "Installing SoDA for Word with Rational Software Setup" for installation options and configuration procedures.
Use the Rational License Key Administrator to install the startup license key.	See the Rational License Key Administrator online help for detailed instructions. The information you need to install the startup license key is included on your Startup License Key Certificate in your media kit. See "SoDA License Management" for more information.
Request permanent license keys from Rational using the License Key Administrator. Make certain that you are requesting keys for the product you have purchased and installed.	See the Rational License Key Administrator online help.
If you are using node-locked licenses: Install the new permanent license key on your client system. **If you are using floating licenses:** Install the new permanent license key on your license server system.	See the Rational License Key Administrator online help.
If you are using floating licenses: Set up your client systems to use the licenses from the license server system.	See the Rational License Key Administrator online help.
Use the Start menu to select and start the program.	See the program's online help.

Note Startup license key information is included with your Rational Suite software media kit. The startup license expiration date is noted on your startup license key certificate. For additional licensing information, see the Rational License Key Administrator online help.

Caution **Make certain that you select the product you purchased when you use the Rational Software Setup program. Review the License Key Certificate that you received with your purchase. If you install a program other than the one you purchased and for which you do not have a license key, you will not be able to use that program.**

Before You Start the SoDA Installation

The following sections proved the steps you must take and information you must review prior to installing SoDA.

Note For the most current information related to SoDA for Word features and known issues, refer to the Release Notes document that is, by default, at the following location on your system after installation:

`c:\Program Files\Rational\SoDAWord\docs\relnote.doc`

Installation Requirements

The following table describes the system and software requirements for installing Rational SoDA for Word:

Table 2: SoDA for Word Requirements

Item	Requirement
Operating Systems	Windows 2000 Windows NT 4.0, Service Pack 3 or greater Windows 98
Processor	166 MHz or greater
Memory	32 MB
Disk Space	50 MB
Word Processor	Microsoft Word 97, or 2000

Table 2: SoDA for Word Requirements

Monitor	800 X 600 X 256-color video resolution, or greater
Mouse/pointing device	Microsoft Mouse or compatible pointing device
Automated License Key Requests	Internet connection required for automated license requests

Caution Installation of Rational SoDA for Word on dual-boot systems is not supported.

Make certain that you have a current backup of your Registry and system directories prior to running the Rational Software Installation procedure.

You must install either a startup or permanent license key to use this software. The Rational License Key Administrator online help provides detailed instructions for installing startup and permanent license keys.

The installation program requires specific versions of Microsoft files. The installation program will install them or you may choose to install them yourself from other sources.

To use the Rational Software Setup program on a Windows 2000 or NT system, you must have administrator privileges on the local machine.

Installation Types

The Rational Software Setup program provides you with several installation types, letting you install the configuration most appropriate for your system. Table 3, Installation Types, describes the installation types supported with the Rational Software Setup program.

Table 3: Installation Types

Type	**Description**
Typical	Installs the most commonly used features for a product. Use this option for standard installations.
Custom/Full	Allows you to add or remove features.

Table 3: Installation Types

Type	Description
Compact/Laptop	Installs a subset of the standard configuration. May omit optional files, including online documentation or online help. Use this option on systems with limited disk space.
Minimal	Installs the files needed to run the program from a CD or network location. Use this option to run the program from a centrally managed location.

Rational Software Setup lets you choose the Custom Installation option; you can set or clear the check box for products or product features on the Choose Features page. Setting or clearing installation options lets you install selected components of Rational software.

Installing Shared Components

The Rational Software Setup program needs to install shared components. A shared component is software provided by a company other than Rational. It is potentially available to other applications on your system.

If the setup program needs to update shared components, the setup program displays a list of the required files. The files listed must be installed on your system before the installation can proceed.

Setup installs U.S. English versions of the files. It does not overwrite newer file versions.

The check box, **Replace files with newer versions in English**, is enabled when you have installed earlier versions of the files that are localized to a non-U.S. English language. If you select this check box, the U.S. English versions of the files will replace your versions. If you clear this check box, the files will not be updated and you will need to update them yourself.

When to Install Files Yourself

In general, we recommend that you allow the installation procedure to install shared files for you. Under some circumstances, you may want to install the files yourself:

- You are using a U.S. English system, but installing new files may invalidate your current environment. In this case, you need to determine how to correct your environment so that you can run existing tools and the Rational products you want to install.
- Your site may mandate that you obtain shared files directly from the source, for example, from Microsoft, rather than using files supplied by a third party. Or your site may prohibit end-users from installing shared components.
- Rational supplies U.S. English versions of shared files. You may want to install equivalent files that are localized to your language.
- There may be a later version of the files available. Rational products should work with the supplied version of shared files or any later versions.

Installing SoDA for Word with Rational Software Setup

The Rational Software Installation procedure uses `c:\Program Files\Rational` as the default installation path. You may specify another drive during the installation procedure.

Note If you have installed another Rational Suite product, you cannot select an alternate location for your SoDA for Word installation.

Caution Canceling an installation that is in progress may leave your system in an indeterminate state. If you click **Cancel** while the installation is in progress, you are asked to confirm that you want to exit from the incomplete installation.

Typical Installation

This section describes a typical installation of Rational SoDA for Word.

1. Insert the Rational Solutions for Windows CD into your system's CD drive. The setup program starts automatically.

   If autorun is disabled on your system, click **Start > Run**. Using the drive letter of your CD-ROM drive, enter `drive:\SETUP.EXE`.

2. The Rational Software Setup wizard guides you through the software installation. On each page, click **Next** to proceed to the next page.

   The Rational Software Setup program writes a log of the installation activities. The log file is located in `<Install Path>\Rational\RSSetup\RSSetup.log`.

3. At the Choose Products page, select **Rational SoDA for Word**.

4. At the Setup Configuration page, select the **Custom/Full** installation if you want to choose integrations for specific SoDA for Word domains. If not, select the **Typical** installation option and complete the installation.

Possible Reboot Required

If files that are required for the installation are in use during the installation procedure, the Rational Windows Setup program may need to reboot your system to complete the installation.

1. After rebooting, log on as the same user to complete the installation procedure.

   Part 2 of the installation automatically starts on your system.

2. Click **Finish** to exit from the Rational Software Setup program.

   If the **Launch License Key Administrator** check box is set, the Rational License Key Administrator will start after you click **Finish**.

Setting the Template Path

SoDA automatically configures access to SoDA wizards and templates during installation. This default configuration requires no intervention from you, as an installer.

An advanced feature in Microsoft Word allows you to configure access to user templates (custom .dot files) using the File Locations tab on the Tools > Options dialog box. The custom templates are then available when you create a new office document (File > New) in Word. During installation, SoDA automatically sets this User Templates path on the File Locations tab to the `Program Files\Rational\SoDAWord\wizards` directory.

If the User Templates path is changed, the SoDA wizards and soda.dot file are no longer visible in the File > New dialog box. To reset the access to these SoDA wizards and templates, run the following utility from your Windows Start menu:

Start > **Programs** > **Rational**... >

Rational SoDA for Word > **Set User Template Path**

Removing Rational SoDA for Word

This section describes how to remove SoDA from your system.

Preparing to Remove SoDA

Make sure that no one is using SoDA and any associated files. You will not be able to remove files that are in use.

To remove SoDA from a Windows 2000 or NT system, you must have administrator privileges on the local machine.

To Remove SoDA

Use the **Add/Remove Programs** control panel to select and remove SoDA. The Rational Software Setup removes SoDA from your system.

Note Removing SoDA does not remove directories that contain files that you have created using any Rational Suite products.

Installation Messages

Contact Rational Customer Support for information and assistance regarding any error messages you encounter while installing Rational software. Table 4 below, provides contact information.

Technical Support Information

At Rational Software Corporation, we welcome your comments on our products and services. If you have product suggestions or recommendations, please contact our Technical Support staff or your Rational Account Manager.

Rational Software Corporation offers telephone, fax, and e-mail support to customers with an active maintenance contract. If your maintenance contract has expired, please contact your Rational Account Manager. Potential customers using evaluation copies of SoDA for Word should contact their Rational Account Manager instead of Technical Support for assistance.

Before calling for assistance, please consult the online help or this SoDA for Word User's Guide. For late-breaking updates, see the Release Notes. Technical notes for SoDA for Word are available on the Rational Software Web site at:
http://www.rational.com/products/soda/support/index.jtmpl

If you still cannot find the information you need, contact Rational Customer Support:

Table 4: Rational Technical Support Contact

Customer Area	Telephone	Fax	E-mail
North America	800-433-5444	303-544-7333	support@rational.com
Europe	+31-(0)20-4546-200	+31-(0)20-4546-201	support@europe.rational.com
Asia Pacific	+61-2-9419-0111	+61-2-9419-0123	support@apac.rational.com

Required Information for Technical Support

When contacting Rational for support, please provide the following information:

- Your SoDA for Word serial number and version number
- The version number of Microsoft Word and related service packs
- The version number of all related Rational domains: Rational Rose, RequisitePro, etc.
- Your operating system and its version number, and related service packs
- A description of your computer hardware:
  - The processor (CPU) speed
  - Amount of memory (RAM) installed
  - Amount of free space available on your hard drive
- A description of the problem, the steps that led to the problem, any error messages displayed, and the soda.log file from the Logs directory

Problems with Templates in Dynamic Domains

If you need help with a customized SoDA template that references specific objects within a Rose model or RequisitePro project, please provide a copy of the template and the model or project (or at least a sample subset of the model or project that includes the referenced objects). These are very useful in debugging problems.

Licensing Support

Table 5 provides Rational License Support contact information. If you have questions about acquiring license keys for your Rational Software products, contact Rational License Support. See the online help in the Rational License Key Administrator for additional licensing support information.

Table 5: Rational Licensing Support

Region	Telephone	E-mail
North and South America	1-800-728-1212 1-781-676-2510 FAX: 781-676-2510	lic_americas@rational.com
Europe (includes Israel and Africa)	Phone: +31 23 554 10 62 FAX: +31 23 554 10 69	lic_europe@rational.com
Japan	Phone: +81-3-5423-3611 FAX: +81-3-5423-3622	lic_japan@rational.com
North Asia Pacific (China, Hong Kong, Taiwan)	Phone: +852 2143 6382 FAX: +852 2143 6018	lic_apac@rational.com
Korea	Phone: +82 2 556 9420 FAX: +82 2 556 9426	lic_apac@rational.com
South Asia Pacific (includes Australia, New Zealand, Malaysia, Singapore, Indonesia, Thailand, Vietnam, the Philippines, India, and Guam)	Phone: +612-9419-0100 FAX: +612-9419-0160	lic_apac@rational.com

2 SoDA License Management

This chapter provides an overview of the Rational software licensing, including descriptions of the types of licenses and license keys used with Rational software products.

Table 1, Installation Quick Start Guide, on page 2 provides a summary of the steps associated with installing and setting up license keys with Rational software products.

The online help in the Rational License Key Administrator describes how to use the Rational License Key Administrator to review and modify your license configuration. The online help also provides information about configuring the FLEXlm License Server software.

The Rational License Key Administrator online help is available by clicking **Help** in the License Key Administrator program or by opening `<Install Path>\Rational\Common\licadmin.hlp`.

The Rational Software Licensing Model

Rational Software uses FLEXlm, a software-based license management tool from GLOBEtrotter, Inc. FLEXlm provides users with a powerful and flexible mechanism for managing licensing resources.

For more information about FLEXlm licensing, see the FLEXlm for Windows FAQ file on www.globetrotter.com/lmwinfaq.htm.

The Rational Software installation procedure automatically installs the FLEXlm licensing software on client systems, allowing client systems to use either node-locked or floating licenses. (Table 7, License Types, on page 16 describes node-locked and floating licenses.)

Most end users configure their own systems for licensing using software provided by Rational. In cases where customers choose to use floating licenses, a system administrator typically configures a

license server system for licensing, using software provided by Rational and GLOBEtrotter.

For additional information about Rational software licensing, see the Rational Suite Installation Guide on your Documentation CD or the online help for the Rational License Key Administrator.

License Types and License Key Types

Table 6 describes the types of license keys used by Rational licensing. Table 7, License Types, on page 16 describes the types of licenses supported by Rational licensing.

Table 6: License Key Types

License Key Type	Description	Notes
Startup	A time-limited license.	The expiration date for the startup license keys is noted on the startup license key certificate included with your software media kit. You can use a startup license key on any system.

Table 6: License Key Types

License Key Type	Description	Notes
Permanent	A license issued to a customer for running Rational products. Permanent licenses are keyed to a product and machine. Permanent Keys can be node-locked or floating. Node-locked Permanent Keys are installed on a client machine, and floating Permanent Keys are installed on a License Server machine.	The Rational issues Permanent Keys upon request. Use the Rational License Key Administrator to prepare and send your license requests to Rational.
TLA (Term License Agreement)	Variations of a Permanent Key. TLAs are issued to a site to allow their employees to use Rational software for a negotiated period of time.	TLAs are issued by the Rational Sales Team. If you are interested in obtaining TLAs for your organization, contact your local Rational Sales Team

Table 7: License Types

License Type	Description	Notes
Node-locked	A license that permits a user to use the licensed software on a specified system. A node-locked license is configured for a specific system. To move a node-locked license to another system, you must uninstall the license key from the old system and request a new license key for the new system.	Use the Rational License Key Administrator to add or modify a node-locked license. Contact Rational Support for help with node-locked licenses.
Floating	A floating license is installed on a license server system and permits a specified number of users to use the licensed software from client systems. Floating licenses are shared among all users of the licensed software.	A system administrator must install the FLEXlm License Server software on a server to set up floating licenses. Use Rational License Key Administrator to set up floating licenses for your system.

Installing a Startup License on a Client System

After you install Rational software, you may install a startup license, allowing you to use Rational software until you obtain your permanent license key. The startup license key information is included with your software kit. The license key expiration date is noted on the startup license key certificate.

You can request permanent licenses keys, if available. The date that your permanent license key is available is noted on your startup license key certificate. You can request the permanent key as soon as it is available, whether you have installed a startup key or not.

In order to maintain uninterrupted use of your software, make sure you obtain and install your permanent license key before your startup license key expires.

To Install a Startup License Key on a Client System

1. Using your product's Windows Start menu, find and run the License Key Administrator. The Rational License Key Administrator is located in the program group for the program you have installed (for example, Rational SoDA for Word or Rational Suite Enterprise).
2. On the **License Key(s)** tab, click on **Enter a License Key**.
3. On the first wizard page, select **Startup License Key**.
4. On the second wizard page, select **Node-Locked License Key**.
5. On the wizard screen, provide the information in the fields based on the columns on the Startup License Key Certificate.

   You must enter the information exactly as presented or the key will not work. If you enter incorrect or incomplete information, the License Key Administrator reports the following message:

   ```
   There is an error in the license key as it was entered.
   Please check your entries for a possible typo.
   ```

   Review and correct the information in each of the fields.
6. Click **Finish**.

   After you complete this step, the License Key Administrator displays the startup license key on the **License Key(s)** tab.

Configuring Your Client System to Use a Node-Locked License

If you are using a node-locked license, you do not need to set up or connect to a license server system; you simply install your license keys on your client system. The Rational License Key Administrator online help describes the license installation process.

Configuring Your Client System to Use a Floating License

Before configuring your system to use a floating license, you must obtain the name of your license server system from your system administrator. (If you are the system administrator, see the

Rational Suite Installation Guide on your Documentation CD or the online help for the Rational License Key Administrator for information about setting up server-based floating licenses.)

1. Ensure that the FLEXlm license server software is running on the license server system. Contact your system administrator or see Rational License Key Administrator online help.
2. Start the Rational License Key Administrator on the client system.
3. Click the **Settings** tab.
4. Select the **Search Server** check box and specify the name of the FLEXlm license server system.
5. Click **Exit** to exit from the Rational License Key Administrator.

Acquiring a Node-Locked Permanent Key for Your Client System

This section summarizes the steps you follow to submit your request.

The Rational License Key Administrator online help provides instructions for preparing, sending, and receiving license key requests, and installing license key files.

You must have an Internet connection to request license keys electronically with the Rational License Key Administrator.

1. Use the License Key Administrator to prepare the license request.
2. Send the request to Rational. You may send the request to Rational electronically using the Rational License Key Administrator, by printing and faxing the request, or by printing the request and making your request by telephone.

3 Generating Reports and Documents

Starting SoDA

Start SoDA by selecting Rational SoDA from the Windows Start menu. This command opens Word, enables the SoDA menu, and opens a blank document. Use one of the following procedures (New User or Experienced User) for working with a SoDA template.

New user

1 Do one of the following:

- In Microsoft Word, click **File** > **New**.
- In Microsoft Office, select **New Office Document**.

2 In the dialog box, do one of the following:

- Select SoDA Getting Started.wiz and click **OK**.
  The SoDA Getting Started Wizard guides you through selecting a SoDA template, saving it for your own use, and identifying the source of information, and generating a report.
- Select **___portal create template.wiz___** and click **OK**.
  The Template View enables you to build a ___Portal Designer___ report template by selecting an information source and specifying the artifacts to be documented from that source.

Experienced user

The SoDA code is automatically loaded into Microsoft Word whenever an existing SoDA document is opened. A new SoDA template can be created as follows.

1 Do one of the following:

- In Microsoft Word, click **File** > **New**.

- In Microsoft Office, select **New Office Document**.

2 In the dialog box, do one of the following:

- Select soda create template.wiz and click **OK**.
  The Template View enables you to build a SoDA report template by selecting an information source and specifying artifacts to be documented from that source.
- Select **soda.dot** and click **OK**.

Understanding SoDA

SoDA is an acronym for Software Documentation Automation. SoDA is a report generation tool that supports day-to-day reporting as well as formal documentation requirements with an easy-to-use interface for defining custom reports and documents. SoDA is tightly integrated with Rational's market leading development tools, giving you a single interface for reporting on requirements, design, test, and defect status.

SoDA automates the production of software documentation, substantially reducing the effort required to produce software documentation. The primary function of SoDA is to retrieve information from various sources and use it to generate a document or report based on a SoDAtemplate. SoDA comes with several predefined templates, which can be used as-is, or can be modified, using its WYSIWYG template-building component to build templates to meet your specific needs.

SoDA for Word:

- Adds document generation to the capabilities of Microsoft Word.
- Performs incremental document regeneration to reduce turnaround time
- Preserves data entered directly into the document from generation to generation.
- Enables extraction of data from multiple information sources, such as Rational Rose and Rational RequisitePro, to create a single document
- Maintains consistency between documents and information sources

The following sections briefly discuss each of the following aspects of documentation automation using SoDA:

- SoDA Templates vs. Word Templates
- Information Retrieval
- Document Generation
- Report Generation
- Template Customization

SoDA Templates vs. Word Templates

The term *template* has a slightly different meaning in SoDA than it does in Word.

A **Word template** is a .DOT file, which, when attached to a new or existing document, provides (amongst other things) styles and macros to a document. The Word template soda.dot enables the SoDA environment in Word, including the SoDA menu and its commands.

A SoDA template is a Word document It is a .DOC file that is based on the Word template: soda.dot. As such, it contains SoDA commands and text specific to the template.

A SoDA template can be copied and used for document generation with no changes. Or, a SoDA template can act as a starting point for new or revised project-specific templates.

Information Retrieval

SoDA extracts information using special programs called ***domains***. Each domain understands a single source of information, such as Rational Rose and the File System. Embedded in each domain is the knowledge of how information is modeled by that source, in terms of classes, attributes and relationships. For instance, the Rose domain understands that a class category has attributes of name and documentation, contains relationships to classes, and has import relationships to other categories. This domain-specific knowledge enables you to retrieve exactly the information you want to document.

One of the benefits of SoDA is that it can support multiple domains, even within the same document. This means that project team members need to use (and learn) just one documentation tool throughout the software life cycle, rather than using a different tool unique to design, coding, or testing.

Document Generation

There are several products that generate documents in an automated fashion. What makes SoDA unique is how well it can regenerate a document. SoDA maintains consistency between the document and its source(s) through a process called Intelligent Document Merging™. If you delete an object in the source domain, such as a class in Rose, SoDA will remove the section(s) of the document generated from that object. If you add to the source, SoDA will create a new section of the document in the proper location.

Consistency is only half the story of Intelligent Document Merging. The other half is how SoDA handles information that is not extracted from an outside source. SoDA lets you add descriptive text, including special formatting such as bulleted lists, equations, and even drawings. When a document is regenerated, SoDA updates the information taken from the domain(s) without affecting the user-added portions.

Another key feature of SoDA is its ability to regenerate a portion of a document. If you know you made a change to the source that affects just one paragraph in the document, you can choose to regenerate that one paragraph, which is much faster than regenerating your entire document every time.

Report Generation

Intelligent Document Merging has a cost, and that cost is performance. If you don't need to maintain supplemental text, you will be better off generating reports instead of documents. Reports use the exact same templates that documents do, except they are created from scratch every time. Since SoDA does not have to keep track of what information came from the source and what information was entered manually, it can generate the report much faster than a document.

SoDA enables you to generate reports as Word documents or as HTML documents. In Microsoft Word 2000, the generated HTML document can be saved as a Word document (with a .doc extension).

Template Customization

SoDA templates are completely customizable; you can add, change, or delete portions as needed. You can even start from scratch and create your own template.

The customization process itself is completely interactive. You don't need to be a programmer or learn a macro language to make template changes. You simply use the Microsoft Word's What-You-See-Is-What-You-Get (WYSIWYG) interface to specify structure and style. Through the SoDA menu, dialog boxes are displayed to aid in defining SoDA commands. For more on customizing templates, see "Template Customization Concepts" on page 27.

Generating Web Pages, Reports, and Documents

Choosing a Template

SoDA provides templates that support report or document generation in conjunction with selected Rational products. SoDA templates specify information types to be extracted from the source product domains. Many of these templates are compliant with industry specifications, such as MIL-498, IEEE, CMM, and the Rational Unified Process (RUP).

These templates are located in the templates directory in your SoDA installation. They are organized in subdirectories by domain.

To Select a SoDA template, use any of the following:

- from RequisitePro, click **Project** > **Generate SoDA Report**
- from Rose, click **Report** > **SoDA Report**
- or refer to: "Starting SoDA" on page 19.

Maintaining Generated Documents

After your document is generated, you can change SoDA commands within the document, and you can also change the source objects in the information source domain. When you make source changes, however, you must regenerate the corresponding portions of your document to keep the document and source information consistent.

Modifying the Link to a Source Object

If you change the names of source objects, and you have added additional information to the generated section, you must change the links that refer specifically to those objects. If you move objects, you must change all links that refer to those objects.

To edit a single link:

1. Place the insertion point anywhere in the section containing the link and click **SoDA** > **Edit Link**.
2. Examine the Edit Link dialog box listing the selected link and its possible sources. The list of sources is the result of re-evaluating the REPEAT command from which the selected link was created.
3. Choose the link you want to change and click **OK**.

   The selected link is updated to reflect its new source.

Changing the Path of Many Source Objects

If you have a document that uses files as sources, and you move those sources to another directory, SoDA could lose the connection between the sources and their generated sections. Thus, the next time you generate the document SoDA could delete all of those sections and you would lose any supplemental information that you added.

The Adjust Links dialog box lets you make global path name changes to your links so you do not have to change each link individually.

To change the path in many SoDA links:

1. Position your cursor anywhere in your document, and click **SoDA** > **Adjust Links**.
   The Adjust Links dialog box will appear.
2. Enter the old path in the From field, enter the new path in the To field, and click **OK**.

   When SoDA completes the task, you will see a report indicating which links were updated successfully, and which failed.

4 Customizing a Template

Making Templates Available for Other Users

SoDA provides “tight integration” with Rose and RequisitePro, which enables you to generate SoDA reports directly from those products. Within Rose or RequisitePro, SoDA displays a list of available templates, with a description of each template.

You can add your own templates to these lists by following these steps:

1. Click **File** > **Properties**.
2. In the Title field of the Summary tab, enter a brief description of the template. Click **OK**.
3. In the Keyword field, enter the SoDA class name of an available OPEN command. Sensible entries are second level commands, such as "Model", or "Package", or "Class". The word must be spelled as it appears in the SoDA command (no spaces, punctuation, etc., allowed), and it is case-sensitive. Only one (1) keyword can be used per template.
4. Open the template in Word.
5. Click **SoDA** > **Modify Command**. The OPEN Command dialog box appears.
6. Ensure that the value in the Argument field is blank. (To clear this field, select the argument value, press **Delete**, then press **Enter**.) Click **OK** to close the dialog box.
7. Use the **File** > **Save As** command to save the template within the TEMPLATE subdirectory where SoDA is installed, in the subdirectory of the tool referenced in the OPEN command, such as TEMPLATE\Rose or TEMPLATE\ReqPro.

   If you prefer to define the directory that SoDA searches in order to determine which templates are available to display for the tight

integration, then you need to ensure that the User Template Path reflects the desired target directory. Select SoDA>Options to display the Options dialog box, where the User Template Path is defined.

Template Customization Concepts

Each information source used by SoDA is defined by a source domain *schema*. The schema contains information about all of the objects in the domain and the relationships of those objects to one another.

Schemas are defined using object-oriented methods. Therefore, the concepts and terminology used to describe schemas are also object-oriented.

The following sections provide an introduction to basic SoDA concepts and terminology:

- Object-Oriented Concepts
- Introducing SoDA Commands
- What Happens During Report and Document Generation

Object-Oriented Concepts

The following concepts and terms are used to define a source domain schema. Understanding them will help you understand how SoDA determines what to make available for commands.

Domains are the available sources from which SoDA can extract information. The standard information source domains delivered with SoDA include: Rational ClearCase, Rational ClearQuest, Rational Rose, Rational Rose RealTime, Rational RequisitePro, Rational Administrator, Rational TeamTest, Microsoft Project, Microsoft Word, and the File System.

Classes are a collection of objects that share a common structure and behavior. Classes are one item found within domains. Other items are relationships and attributes. The File System domain, for example, is made up of these classes: directories, files, directory objects, and file records.

An ***object*** is an instance of a class. An object found in the directory class of the File System domain might be C:\WINDOWS.

Relationships are associations between classes within a domain. Relationships may be “N-ary” or “Unary.”
N-ary relationships are one-to-many relationships. For example, a directory and the files contained in it have an N-ary relationship: there are many files in one directory. Unary relationships are one-to-one. For example, a file and its parent directory have a Unary relationship: there is only one parent directory for every file.

An ***attribute*** is a characteristic of a particular class. Attributes result in either text or graphics. DISPLAY commands are created using attributes. A common attribute, for example, is the name of an object.

A ***subclass*** is a special case of a class, one that inherits attributes and relationships from its parent class.

Introducing SoDA Commands

Each SoDA template contains one or more of these SoDA commands: OPEN, REPEAT, DISPLAY, and LIMIT. Here is a brief description of each command:

- The OPEN Command identifies a particular object in a source domain. It normally provides the highest abstraction of a domain object from which otherSoDA commands can be defined. For instance, you might OPEN a directory, or perhaps a Rose model.
- The REPEAT Command identifies sections within a document that are repeated for each object found based on some relationship. This command is useful when there is a set of objects in the source domain that needs to be uniformly documented. The template defines the format and generic content of the section. SoDA builds a section for each object found in the source. SoDA maintains the consistency between the document and the objects in the source.
- The DISPLAY Command inserts attributes of a source object into a generated document. You can display both text and graphics. Any of the following might be displayed: the name of a Rose class category, the cardinality of a “has” relationship, the name of a directory, or the contents of a text file.

- The LIMIT Command determines whether an object exists with certain characteristics. Normally a LIMIT command defines an expression that evaluates to True or False. If the object fails the test defined by the expression, the corresponding section is left out of the document.

What Happens During Report and Document Generation

When you generate report or document, SoDA searches through the information sources.

A **report** is a "snapshot in time," which presents a single view of the current state of the source information. A report is always generated from a template. Additional text can only be added outside of the SoDA commands (the hidden annotations) in the template. See also: "Report Generation" on page 22.

A **document** is a "living" representation of the source information. You can regenerate the whole document or sections of the document. You do not need to use the template. Additional text can be added in generated sections of the document. Document generation is slower than report generation due to validation and merging of information. See also: Document Generation

When generating a report, SoDA references your template and adds text and graphics to the report depending upon the type of command being executed.

The first time you generate a document, SoDA references the template to create the document. During subsequent generations, SoDA references the previously generated document. SoDA adds, deletes, or ignores objects based on changes within the source.

SoDA uses the following commands when generating:

- SoDA uses the arguments of each OPEN command to create a pointer to a particular object within the referenced source domain.
- For each object returned by a REPEAT command, a section is created. (The term section can mean a set of paragraphs, a list item, or even a set of names followed by commas.)

  In report generation, the section that includes the original

REPEAT command is not copied to the report.
In document generation, the link refers only to the one object that corresponds to the new section, and is used for Visit Source and document regeneration. The section that includes the original REPEAT command remains in the document, but is hidden.

- When a DISPLAY command is encountered the value specified in the command is returned and inserted into the report.
- A LIMIT command results in a new section or nothing. If the object exists and its specified characteristics are satisfied, the section will be created.
  In report generation, the section is not copied to the report.
  In document generation, the command remains in the document, but it is hidden. Whenever the document is regenerated, the LIMIT commands are checked again to see if the characteristics of the object have changed.

Customizing a SoDA Template

When you need to customize a SoDA template, follow these steps:

1. Identify the domain that contains the information you need to document.
2. Do one of the following:
   - Choose a Template as a starting point, normally one of the supplied templates for that domain.
   - Create a new template based on **soda.dot**.
3. Use the Template View (SoDA>Template View) to add, modify, and/or delete SoDA commands as needed:
   - OPEN Commands
   - REPEAT Commands
   - DISPLAY Commands
   - LIMIT Commands
   - Special LIMIT Commands

Note Be careful when editing text within annotations in a template. Do not modify or remove the hidden annotation text associated with SoDA commands.

4. Save the template.
5. Test the template.
6. Repeat steps 4-6 until complete.

   A good understanding of your information as it exists within each domain and a detailed document plan will ensure that the placement of SoDA commands within a template will yield the desired results.

SoDAProjectConsole Template Builder's use of Annotations

All SoDA information is stored in Word documents as *annotations*. Annotations are Word comments, in hidden text, within a SoDA template. Annotations do not affect the viewing or printing of the document. They are most commonly used for readers to review and comment on a Word document. SoDA uses annotations to

- store the strings that hold the values of theSoDA commands
- identify the beginning and ending points of SoDA commands
- traverse quickly through a document to locate SoDA commands

Although you could use annotations to review SoDA templates, it is recommended that you make a copy of a SoDA template and place your comments in the copy. This is to ensure that SoDA annotations are not accidentally deleted or modified.

Annotation Names

Annotations are marked by a hidden name and number enclosed in brackets, for example, [JSMITH1]. The name of the annotation reflects the initials of the user, as found in the User Info folder in Word's Customize dialog box. The number is sequential in the document, from 1 to the total number of annotations. SoDA uses

the annotation names to help identify commands. Here are the names used by SoDA:

Annotation Name	Description
OPEN	An OPEN Command (a single command)
REPEAT	The start of a REPEAT command
ENDREP	The end of a REPEAT command
DISPLAY	The start of a DISPLAY command
ENDDISP	The end of a DISPLAY command
LIMIT	The start of a LIMIT command
ENDLIM	The end of a LIMIT command
LINK	The start of a link section
ENDLINK	The end of a link section
MASTER	The start of a master section
ENDMAST	The end of a master section

While most of these annotations are self explanatory, the last two pairs warrant further explanation. A link section is the text and graphics generated for one object retrieved by its parent REPEAT command. If the object is deleted from the source the location of the [LINK] and [ENDLINK] annotations defines the section that will be deleted from the document. The [LINK] annotation also contains the unique identifier for the corresponding object in the source, which assists in updating and Visit Source.

A MASTER section is the text and graphics for a REPEAT command and all its associated generated sections. The [MASTER] and [ENDMAST] annotations are used internally by SoDA to identify which link sections are associated with a particular REPEAT command.

The Annotation Hierarchy

Although annotations are by nature sequential, SoDA uses the annotation names not only to identify the scope and values of each command, but to define a hierarchy as well.

You can view the hierarchy of SoDA commands using the Template View, which is displayed when you choose Template View from the SoDA menu.

Choosing a Domain

When you create a SoDA template you must determine what information will act as sources to the document. The following domains are available:

- Rose Domain
- Rose RealTime Domain
- RequisitePro Domain
- TeamTest Domain
- ClearQuest Domain
- ClearCase Domain
- Microsoft Project Domain
- Microsoft Word Domain
- File System Domain

There are two ways to customize domains:

- Domain Aliases
- Domain Extensions

Testing SoDA Templates

The following suggestions can be helpful as you design your templates:

- Maintain backups of all templates.
- When creating or modifying templates, save them in a working directory. Do not modify the original SoDA-supplied templates in theSoDA templates domain subdirectories. These are used in the tight integration within each Rational product. These also provide a clean starting point for new templates.
- Establish a numbering scheme for the different revisions. Use the **File** > **Save As** command in Word to name each revision.

- Create a minimal sample domain that includes all the elements in your actual domain. This sample set allows you to generate quick tests of a template on a representative subset of your actual source information.
- Edit each template, as needed, directly in the template or in the Template View.

SoDA Commands

Viewing the SoDA Commands

The Template View is a convenient way to examine all of the SoDA commands in a template. The view is an indented list of the commands and their arguments. When you select a line in the template view, the corresponding SoDA command is selected in your template.

To create a Template View from Microsoft Word, click **SoDA>Template View**.

Note: The Template View is used to build or edit a SoDA template. You cannot generate a document or report directly from the Template View.

Viewing a list of the template commands

The Template View report function creates a Word document that lists all SoDA commands and their arguments in hierarchical order. This report does not include any of the template content – only the SoDA commands in the template.

To create a template command report, do one of the following:

- Click **Generate Report** button in the Template View.
- Click **Tree** > **Report**.

Note: This function is not the same as the Generate Report command on the SoDA menu in Word.

Modifying Existing Commands

In most cases one of the standard templates can act as a reasonable starting point for template customization. There are

two ways to modify commands: through the Template View or directly in the SoDA template.

- In the Template View:
  - double-click the command you want to modify
  - highlight the command, right-click and select the **Modify** from the shortcut menu
  - highlight the command and select **Edit** > **Modify**
- In the SoDA template itself, place the cursor (insertion point) inside the command you want to modify, but outside any child commands:
  - For OPEN Commands, place the cursor to the right of the [OPEN] annotation, but before any other annotations.
  - For REPEAT Commands, place the cursor to the right of the [REPEAT] annotation, but before any other annotations in the section
  - For DISPLAY Commands, place the cursor between the [DISPLAY] and [ENDDISP] annotations. (DISPLAY commands have no child commands.
  - For LIMIT Commands, place the cursor to the right of the [LIMIT] annotation, but before any other annotations in the section.

When the cursor is in the proper location, choose the **SoDA>Modify Commands** command. The corresponding dialog box appears:

- OPEN Command dialog box
- REPEAT Command dialog box
- DISPLAY Command dialog box
- LIMIT Command dialog box

Make the desired changes and click **OK**.

Adding SoDA Commands

The easiest way to add SoDA commands is to use the Template View. The following steps describe how to add SoDA commands without using Template View.

To add a single command without Template View:

1. Place the insertion point where you want to insert the command. For REPEAT Commands and LIMIT Commands, you may want to first select a section. Be sure your selection does not split the annotations for another SoDA command.
2. Click **SoDA>Add Command**. The Select Command to Add dialog box appears.
3. Select a command to insert.

   After you select a command, the corresponding dialog box appears:

   - OPEN Command dialog box
   - REPEAT Command dialog box
   - DISPLAY Command dialog box
   - LIMIT Command dialog box

Deleting SoDA Commands

There are two ways to delete commands in a SoDA template: through the Template View or directly into the template itself. Both methods give the same results. Deleting commands in the SoDA template varies slightly based on the type of command:

- To delete an OPEN Command, locate the [OPEN] annotation, select it, and delete it.
- To delete a DISPLAY Command, place the insertion point somewhere between the [DISPLAY] annotation to the corresponding [ENDDISP] annotation, and click **SoDA>Delete Command.**
- To delete a REPEAT Command or a LIMIT Command, you must decide whether you want to delete the entire section including the command, or just the command. Place the insertion point to the right of the starting annotation, but before any other annotations in the section. Then click **SoDA>Delete Command**. In the dialog box that appears, choose one of the following:

Note REPEAT commands can be defined as recursive. If you simply wish to delete the recursive behavior and restore the REPEAT command to standard behavior, then modify the command, as discussed above, and be sure to click the Modify button on the REPEAT dialog box followed by clicking the Delete button on the Recursion dialog box.

- **Delete Command –** Removes the annotations for the specific command only, leaving all other internal commands and text in tact.
- **Delete Command and Text** – Removes all annotations and all other commands and text within the deleted command.

You can also delete commands from the Template View, by selecting the command you want to delete, and choosing the Delete button.

Warning Always use the **Delete Command** menu option to ensure that you delete all annotations corresponding to the command you are deleting. Failure to do so may result in the template being in an inconsistent state.

Creating Hyperlinks

SoDA gives you the ability to create documents that include hyperlinks from one section of the document to another.

In order to use this feature, you must have a template where an item is listed in at least two places: The first place will act as the "anchor" or "address" of the link. The second place will be underlined, so that when the reader clicks on the item the document jumps to the address location.

Here are some common examples:

REPEAT Classes < - - acts as the address for the link

    REPEAT Relationships

        DISPLAY Relationship.ToClass.Name < - - acts as the hyperlink back to the address

REPEAT Requirements < - - the address

REPEAT TracesTo

DISPLAY TracesTo.Requirement.Prefix < - - the hyperlink

As you can see, the REPEAT command acts as the the address, and the DISPLAY command acts as the hyperlink. You cannot arbitrarily select any old REPEAT command and any DISPLAY command and have them link to each other—they must both refer to the same class of object.

OPEN Command

OPEN commands are used to "open" an object within an information source domain which results in one and only one source object. For example, in the Rose domain, the OPEN command connects to a specific Rose model and its associated components.

The OPEN command is used in one of two ways:

- To specify an initial starting point in a source domain from which further SoDA commands can be defined.
- To create a direct reference to a particular piece of data.

The OPEN command establishes the context or reference point for other SoDA commands (such as REPEAT commands and LIMIT commands) in your template. An OPEN command can be placed anywhere in a SoDA template, but can influence only those commands that are below it in the command hierarchy. Therefore, most OPEN commands are placed at the beginning of the tempalte.

Creating a new OPEN Command

To add an OPEN command:

1. Position the cursor where you wish to insert the command, normally at the top of the template.
2. Click **SoDA>Add Command**.
3. Select the OPEN Command.

4 In the OPEN Command dialog box, choose a source class, fill in the required arguments (unless finalizing for distribution), and click OK.

The name of your OPEN command will default to the name of the class being opened. If you want to change this name, be sure to choose something meaningful. Since SoDA commands are relative to other commands, make their names relative, also. The names will be visible to you in the Template View of your template. When you have many OPEN commands stored in many places within one template, meaningful names will make it easier to define other elements.

You can add multiple OPEN commands to your template. The additional commands can point to information either in the same source domain or in a different domain. The Template Wizard supports a single OPEN command only.

Example:

You are creating a template that will extract objects from the File System domain. You would like SoDAProjectConsole to create a section for each directory in your projects directory.

First, you must "open" your home directory within your template, so you name the OPEN command project_directory. Here is the constructed command:

Name	project_directory
Class	File System -> Directory
Arguments	Filename: C:\PROJECTS

In the example above, SoDA produced the "Filename:" prompt in the Argument area when the directory class was chosen. In the text-entry box provided, specify the object's filename. You can use an absolute or relative path name.

To modify the OPEN command, refer to: Modifying Existing Commands

REPEAT Command

REPEAT commands are used to create sections in your document for each object found in the source. For example, a REPEAT

command can be used to generate a section for each file in a directory.

Each REPEAT command is based on the context of one of the OPEN, REPEAT, or LIMIT commands defined above it in the command hierarchy. (At least one OPEN command must be specified in a template before a REPEAT command can be created.)

REPEAT commands can be defined to be recursive. Recursive REPEAT commands are used to drill down through heirarchial structures to collect objects.

To add an REPEAT command:

1. Place the insertion point at where you wish to insert the command. If a section placeholder already exists, you can select the section.
2. Click **SoDA>Add Command**.
3. Select the REPEAT Command.
4. In the REPEAT Command dialog box, select the relationship to be used, and modify any options.

   If the REPEAT is not to be recursive, then click OK and skip the next step.

   If the REPEAT command is to be recursive, continue with the next step.
5. To convert the REPEAT command just created into a recursive REPEAT, click the Advanced checkbox, followed by clicking the Add button at the bottom right of the expanded dialog box, select the proper recursive relationship, click Accept, followed by clicking OK.

   To modify an existing REPEAT command, refer to: Modifying Existing Commands

Using REPEAT Commands for Table Rows

You can use a REPEAT command to produce one table row for each object you specify. Follow these steps:

1. Click **Table** > **Insert Table**.
2. Select the number of columns you need and the desired format. If your table will have a heading row then create a 2-row table; otherwise create a 1-row table.
3. Make any required format changes to the borders and shading. It is important to determine this information before document generation, as once the document is generated table styles cannot be changed. This is because information concerning the table is embedded into the SoDA commands of generated documents. (This does not apply to reports.)
4. Enter the headings in the heading row, if desired.
5. Select the second row of the table (or the only row of a 1-row table). Be sure to select only the row and not any additional text beyond the table.
6. Click **SoDA>Add Command**, and follow the steps for inserting a standard REPEAT command.
7. Add DISPLAY commands in the table cells as needed.

Using REPEAT Commands within Table Cells

You can also nest a REPEAT command within a single table cell. To do so, follow the steps for inserting a standard REPEAT command.

REPEAT Commands Refined with And Where

The And Where expression specifies criteria which must be met by items in the set of objects returned by a REPEAT command. If the criteria are not met by a particular object, that object will not become part of the set.

Expressions consist of operands and operators. Operands are the attributes or literals on either side of an operator. The operands that are available at a given time in the REPEAT command dialog box depend upon the information source domain specified in the current context. Operators specify the test that will be applied to the operands to determine their relationship, for example, the test of equality or the test of inequality.

Here are the operators available for And Where expressions (note that all of the following operators are case sensitive):

=	exactly equal to
!=	not equal to
>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to
LIKE	matches the regular expression
NOT LIKE	does not match the regular expression
IS	the class of the object matches

Note that when a DISPLAY value returns a Boolean value, “True” or “False”, you must enter these values exactly.

Example:

You are creating a document that will extract objects from the File System domain. You would like to create a section for each directory in your project directory that starts with “ROSE”.

After creating an OPEN command called project_directory, create a REPEAT command that will result in a section for each directory in project_directory, and specify an And Where expression which limits the set of directories to those that start with “ROSE”.

Select Relationship	project_directory -> Contents
Where Is A	Directory
And Where	SimpleName LIKE ^ROSE

Metacharacters for LIKE

When you use the LIKE operator in an And Where expression, you can use any of the following metacharacters:

. The dot metachacter matches a single instance of any single character.

For example: “bat.” is a pattern that matches any four character string that begins with “bat”. It would retrieve “bata”, ‘batb”, but not “batab” (because it is longer than

four characters, the length of “bat.”).

As another example: “basebal.” would match any eight character string that begins with “basebal”.

”basebal.xyz” would match any character string that begins with “basebal”, then has any single character, then ends with “xyz”.

The dot metacharacter can also return a space: so “basebal.xyz” can also return “basebal xyz (where a space is between the “l” and the “x".

* The star metacharacter matches zero or more of the preceding character in the expression. An expression followed by an asterisk “*” can be repeated any number of times (including zero, (i.e., 0-n times)).

For example: “ba*” matches all of “b”, “ba”, “baaa”, “be”, “beee”, etc., but not “aaaa” because it doesn’t start with a “b”

The star metacharacter can also return spaces: so “b *” (a space between the “b” and the * metacharacter) can return “b “, and “b “.

^ The NOT metacharacter matches patterns by using the characters to the left of the ^ metacharacter and excluding those to the right of the ^ metacharacter.

For example: “bat^abc” is a pattern that matches any string (regardless of size) that begins with “bat” (the characters to the left of the ^ metacharacter) and isn’t followed by abc (the characters to the right of the ^ metacharacter).

Using the pattern “bat^abc”, “batabc” would not be returned while “batabc123” would be returned. The first would not be returned because it ends with “abc” (the pattern to the right of the ^ metacharacter, while the second would be returned because begins with “bat” (the characters defined to the left of the ^ metacharacter) and it ends with “abc123” which is NOT the pattern to the right of the ^ metacharacter.

”^bat” means means match anything except “bat”, while “^xyz” means match anything except “xyz”.

The ^ metachacter can block out spaces as well: so “^this is a test” will return “thisisatest”, but will not return “this is a test”.

$ The dollar metacharacter matches expressions only at the end of the string.

For example: “bat..$abc” will return any string that begins with “bat”, then contains any two characters (note the double dot metacharacters), then ends with the exact string “abc”. So, “bat12abc” would be returned, “batxyabc123” would not be returned (as it terminates with “abc123” and not just “abc”), and “batabc” would not be returned as it terminates with a “c” and not “abc”. (Wait you say, is this a trick!, it does terminate with “abc”. Well, in the scheme of the definition “bat..$abc”, the “abc” in “batabc” is not parsed by the $ metacharacter, but rather the “ab” in “abc” is parsed by the double dot metacharacters, leaving only the terminating “c” to be worked on by the $ metacharacter).

The $ metacharacter can also match spaces: so “bat..$a b c” would match “bat12a b c” but not “bat12abc”.

\ The backslash metacharacter is used as an ESCAPE code so that SoDA metacharacters (those listed here) can be used as literals.

For example, “\\” means that you want a literal backslash, while \$ means that you want a literal dollar-sign.

As a further example, if you are specifying a path that includes backslashes (“\”), you must use double backslashes as follows: Path LIKE ‘C:\xyz’ returns nothing, while Path LIKE ‘C:\\xyz’ returns all targets items in the directory C:\xyz

[ ] The bracket metacharacters match any one of the enclosed characters. Characters between the brackets are to be matched.

For example: “bat[123]xyz” would match “bat1xyz”, and “bat2xyz”, and “bat3xyz”, but NOT “bat123xyz” (because the bracket metacharacter applies to single character matches only).

To match “bat123xyz” and “bat321xyz” and “bat312xyz” would require the following pattern to be used: “bat[123][123][123]xyz”.

A dash may be used with the bracket metacharacter. For example: “Name LIKE bat[a?d] would return “bata”, “batb”, “batc”, and “batd”.

An interesting pattern for the bracket metacharacter is the inclusion of the * metacharacter. For example, “bat[xyz*] means that “batx” is returned, “baty” is returned, “batz” is returned, and “batzzzzzz” is returned, but NOT “batzzzz” or “batyyyy”.

Another interesting pattern for the bracket metacharacter is the inclusion of the NOT metacharacter (^). For example, you can use the pattern “bat[^123]xyz” to obtain any string that begins with “bat”, then has a single character that is not “1” or “2” or “3”, and ends with “xyz”. Such a pattern would return “batHxyz”, but would not return “bat1xyz”.

The backslash metacharacter (\) can be used with the backslash metacharacter. For example: “bat[\\abc]xyz” returns any character string that begins with “bat”, then has a single backslash (remember, the double backslash means that you are ESCAPing a single backslash), or an “a”, or a “b”, or a “c”, finally ending with “xyz”.

The bracket metacharacter pair can also contain spaces, meaning that a returned string can have an embedded space.

Final Exam on Metacharacters:

What values might the following metacharacter combination return: “base*..[\\x][^7X]*$ball” ?

Answer:

Well, let's look at it. First of all, note that there are a number of metacharacters being used: "*", ".", "[ ]", "\", and "$". Any returned string will begin with "bas", the characters to the left of the first metacharacter (not including the character to the "immediate" left of the first metacharacter). Then the first * metacharacter will return any number of "e"s (zero to infinity, well not quite infinite, more like a very large number of them). So up to this point, the following would qualify: "bas", "base", and "baseeeeeeee". Next we have a pair of dot metacharacters. These characters return any single character, and being that there are two of this type of metacharacter, we get two single characters of any kind. So now we can get "base56", "baseeeeHX", or even "bas \" (note: the backslash is a returned value and has nothing to do with the "\" metacharacter) (also note the space between the "s" and the "\", as dot metacharacters can return spaces).

Continuing, we come upon the the first pair of bracket metacharacters "[\\x]". This pair returns a match if a backslash or the letter "x" exist at this position. So now we can get "base56\", "baseeeeeeeeeZHx", or even "baseee \\" (Here the double backslash is not to be confused with the ESCAPE metachacter. The first of the pair comes form the second dot metacharacter (as discussed above, and the second backslash comes from the ESCAPEd pair defined by the bracket metachacter.

Following the first bracket metacharacter combination is a second pair of bracket metacharacters: "[^7]". This pair states anything can be included at this point in the string except an uppercase X. Now our string can consist of such things as: "baseeee56xY", or even "baseeeeee23xH, but NOT "baseee56x7".

Are you catching on? Next we find another * metacharacter. What is interesting about this particular instance of the * metacharacter is that it is acting against a bracket metacharacter that contains a ^ metacharacter. So what does it mean? It means that we are multiplying (repeating, if you will), the ^ metachacter. So, you can get such things as "baseee56xY", or even "baseeee56xYYYYYY", but not "baseee56x7", or "baseeeee56x77777" (as the * metacharacter is acting on the "[^7]" combination.

Next we come to the final metacharacter combination: “$ball”. This one means that any string that complies with the discussion above can be used, AND those strings can end with anything except the string “ball”. Thus “baseeeee56xyballlll” would be returned by SoDA, but “baseeeee56xyball” would not.

Ordering

Ordering specifies one or more attributes by which the objects resulting from the REPEAT command will be sorted. Sorting can take place alphanumerically or numerically, in forward or reverse order, and case can be ignored.

Prompt

The Prompt (for Filter) field enables SoDA users to perform realtime filtering, during the SoDA generation process, for REPEAT and LIMIT commands.

This flexibility means that you can generate a template multiple times, each with a different set of filtering criteria, without having to explicitly open the REPEAT / LIMIT commands involved.

To define a Prompt for Filter, click the Advanced checkbox and then click the Prompt column (the right-most column). When Prompt for Filter is enabled, the field is populated with an asterisk (“*”), and the Right Operand field is disabled.

Each REPEAT command and each LIMIT command can get its own individual set of filter prompts.

When a template that contains prompt commands is generated, a dialog box displays requesting information for the Right Operand. A prompt will be displayed for each command defined for to prompt.

DISPLAY Command

DISPLAY commands insert text or graphic values from the source domain. You can display names of files and Rose objects, bitmaps, class diagrams, and more.

Each DISPLAY command is based on the context of one of the OPEN, REPEAT, or LIMIT commands defined above it in the

command hierarchy. At least one OPEN command must be specified in a template before a DISPLAY command can be created.

Adding a New DISPLAY Command

To add an DISPLAY command:

1. Position your cursor precisely where you want the value displayed.
2. Click **SoDA>Add Command**.
3. Select the DISPLAY Command.
4. In the DISPLAY Command dialog box, choose an attribute and your desired modifiers and click OK.

   To modify an existing DISPLAY command, refer to: Modifying Existing Commands

Example:

You are creating a template that will extract objects from the File System domain. You would like SoDA to create a section for each directory in your project directory. You would also like the name of each directory to appear in the corresponding section heading.

After creating an OPEN command called project_directory, and creating a REPEAT command which will result in a section for each directory in project_directory, insert a DISPLAY command in the section heading to return the name of each directory.

Select	all_directories > Simple Name
Modifiers	
Display Options	Capitalize
Remove Punctuation?	no (default)
Single Paragraph?	yes (default)

LIMIT Command

A LIMIT command is used to conditionally include or exclude the section to which it is attached. LIMIT commands resolve to “true” or “false.” If an object is found that meets the specified conditions, the command is “true,” and the section is included in the

document. If an object is not found, the command is “false,” and the section is not included in the document. For example, suppose you have a repeated section for each file in a directory. You may want to include a special section if a file has a “.DAT” extension.

Each LIMIT command is based on the context of one of the OPEN, REPEAT, or LIMIT commands defined above it in the command hierarchy. At least one OPEN command must be specified in a template before a LIMIT command can be created.

See also the section on Special LIMIT Commands.

Adding a New LIMIT Command

To add an LIMIT command:

1. Place the insertion point where you want to create the LIMIT command. If you already have placeholder text, you can select the text.
2. Click **SoDA>Add Command**.
3. Select the LIMIT Command. If the command is not available, click Cancel and be sure text is selected in your document.
4. In the LIMIT Command dialog box, choose an object. Then limit the object either by class or by And Where expression.

   To modify an existing LIMIT command, refer to: Modifying Existing Commands

   Example:

   You are creating a template that will extract objects from the File System domain. You would like SoDA to create a section in the template for each directory object (directory, file) in your project directory. However, you would like one particular subsection to be created only if the directory object is a file.

   After creating an OPEN command called project_directory, and a REPEAT command called all_dir_objects for each directory object in project_directory, create a LIMIT command that will only create a section if a file is found.

   Select Objectall_dir_objects -> <Self>

   Where Is AFile

LIMIT Commands Refined with And Where

The And Where expression specifies criteria which must be met by the object examined by a LIMIT command. If the criteria are not met by the object, that LIMIT command will be “false”.

Expressions consist of operands and operators. Operands are the attributes or literals on either side of an operator. The operands that are available at a given time in the LIMIT command dialog box depend upon the information source domain specified in the current context. Operators specify the test that will be applied to the operands to determine their relationship, for example, the test of equality or the test of inequality.

Example:

You are creating a template that will extract objects from the File System domain. You would like SoDA to create a section for each directory object (directory, file) in your project directory. You would like one particular subsection to be created, however, only if the directory object is a file that starts with “ROSE”.

After creating an OPEN command called project_directory, and a REPEAT command called all_dir_objects for each directory object in project_directory, create the following LIMIT command:

Select Objectall_dir_objects -> Self

Where Is AFile

And WhereSimpleName LIKE ROSE

Special LIMIT Commands

There are two kinds of special LIMIT commands -- OMIT and OTHERWISE -- both of which are associated with REPEAT commands. These commands, unlike regular LIMIT commands, are not defined through the LIMIT Command dialog box and cannot be edited. Also, descendent SoDAProjectConsole commands cannot use special LIMIT commands for context.

OMIT Command

An OMIT command is used to omit a section from a document when a REPEAT command returns no objects. For example, you could use a REPEAT command to create a numbered list item for

each object, and in case there are no objects, you could use an OMIT command to entirely omit the list and any introduction to it.

To insert an OMIT command:

1. Select the section that you want to omit, including the entire REPEAT command which the OMIT command is evaluating. (If the command's descendants include multiple REPEAT commands, the OMIT command is associated with the "first" or "closest" one.)
2. Click **SoDA>Add Command**.
3. Select the Special LIMIT Command: OMIT radio button. SoDA automatically fills in all required values for the command; you do not need to complete a dialog box.

Example:

The following example is from the Greenhse demo template in the demos directory.

Consider this section:

2.1.1Class Diagrams

[MASTER16][REPEAT17][DISPLAY18]{INCLUDEPICTURE.....}[ENDDISP19]

[DISPLAY20]<ClassDiagrams.Name>[ENDDISP21] Class Diagram

[Describe the interactions between the classes.]

[ENDREP22][ENDMAST23][ENDREP24][ENDMAST25]

Suppose you want to suppress the 2.1.1 heading if there are no class diagrams. Select the heading through and including [ENDMAST23] and do **SoDA>Add Command** to add the Omit command. ([MASTER16] starts the REPEAT command for the class diagrams.)

The resulting structure looks like this:

2.1.1[LIMIT16]Class Diagrams

[MASTER17][REPEAT18][DISPLAY19]{INCLUDEPICTURE.....}[ENDDISP20]

[DISPLAY21]<ClassDiagrams.Name>[ENDDISP22] Class Diagram

[Describe the interactions between the classes.]

[ENDREP23][ENDMAST24][ENDLIM25][ENDREP26][ENDMAST27]

OTHERWISE Command

An OTHERWISE command is used to include a section of a document only when a REPEAT command returns no objects. For example, you could use an OTHERWISE command to include a paragraph that said: “No objects were found.”

To insert an OTHERWISE command:

1. Create the section that may or may not be included. You may find it helpful to type attribute names where you will eventually create DISPLAY commands, if any. The limited section must immediately follow its associated REPEAT command.
2. Select the text. In most cases you will select entire paragraphs, from the beginning of one paragraph to the end of another.
3. Click **SoDA>Add Command**.
4. Select the Special LIMIT Command: OTHERWISE radio button. SoDA automatically fills in all required values for the command; you do not need to complete a dialog box.

Examples:

Using the same example as above, we start with this:

2.1.1 Class Diagrams

[MASTER16][REPEAT17][DISPLAY18]{INCLUDEPICTURE.....}[ENDDISP19]

[DISPLAY20]<ClassDiagrams.Name>[ENDDISP21] Class Diagram

[Describe the interactions between the classes.]

[ENDREP22][ENDMAST23][ENDREP24][ENDMAST25]

This time rather than omitting the heading, we want to include a message if there are no diagrams. To do this we:

1. Put your cursor after the [ENDMAST23] annotation and insert a carriage return.
2. Type whatever message you want included into the document if there are no diagrams found. The message can be one or more paragraphs.
3. Select the message.
4. Select **SoDA>Add Command** and choose Special Limit Command: Otherwise

   The resulting structure will look like this:

   2.1.1 Class Diagrams

   [MASTER16][REPEAT17][DISPLAY18]{INCLUDEPICTURE.....}[ENDDISP19]

   [DISPLAY20]<ClassDiagrams.Name>[ENDDISP21] Class Diagram

   [Describe the interactions between the classes.]

   [ENDREP22][ENDMAST23]

   [LIMIT24]There are no diagrams.[ENDLIM25][ENDREP26][ENDMAST27]

Using Both OMIT and OTHERWISE

You can associate both an OMIT and OTHERWISE command with a given REPEAT command. Here's how:

1. Create the repeated section and the REPEAT command first.
2. Add the OMIT command enclosing the REPEAT command.
3. Add the OTHERWISE following, and outside of, the REPEAT and OMIT commands.

Simply be aware that the OTHERWISE command must be inserted outside the scope of the OMIT command so that the OTHERWISE command is not hidden by the OMIT command when the document is generated.

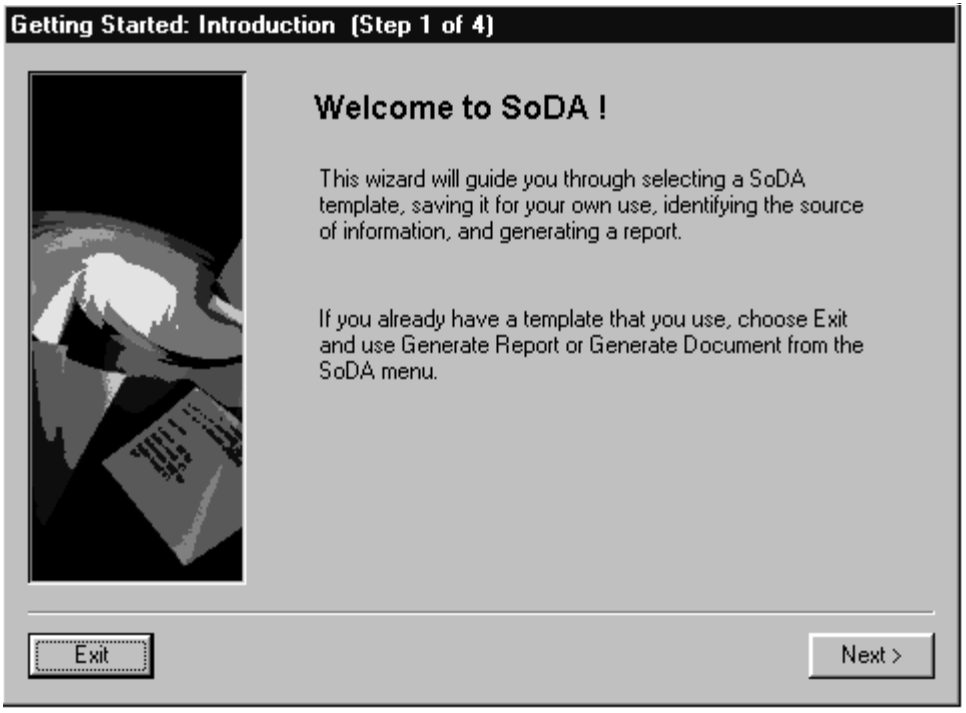
5 Wizards and Dialog Boxes

SoDA adds the following wizards and dialog boxes to Microsoft Word. Each is described in this chapter.

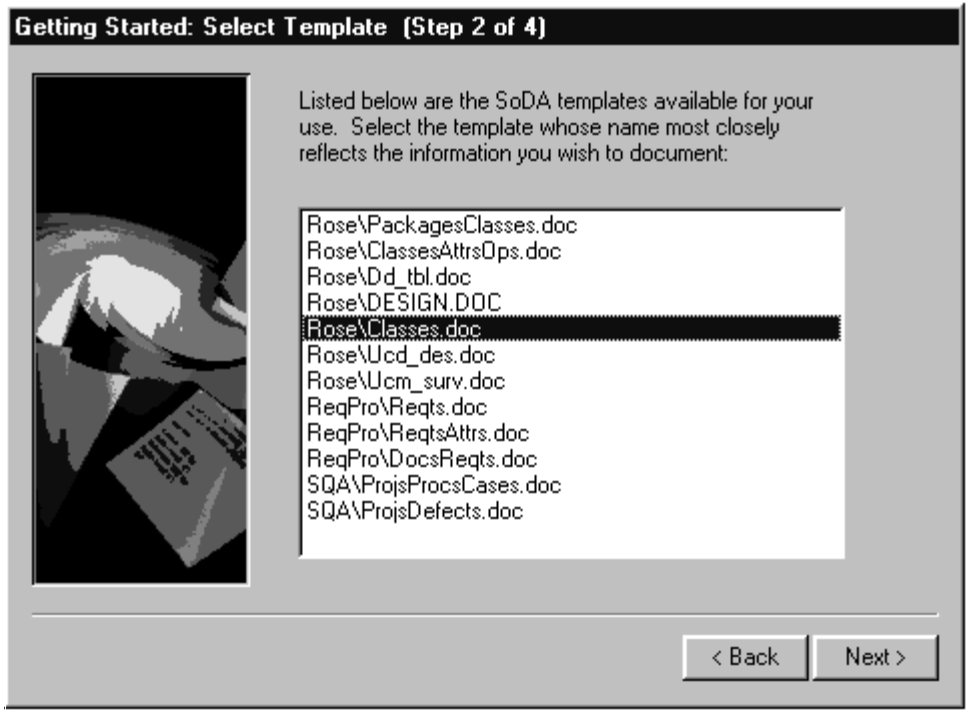
- Getting Started Wizard
- Template View
- SoDA Generator Dialog Box
- Identify the <Class> Dialog Box
- Select Command to Add Dialog Box
- OPEN Command Dialog Box
- REPEAT Command Dialog Box
- DISPLAY Command Dialog Box
- LIMIT Command Dialog Box
- Edit Link Dialog Box
- Adjust Links Dialog Box
- SoDA Options

Getting Started Wizard

The Getting Started Wizard guides you through selecting a SoDA template, saving it for future use, associating it wth a source, and generating a report. The first panel looks like this:

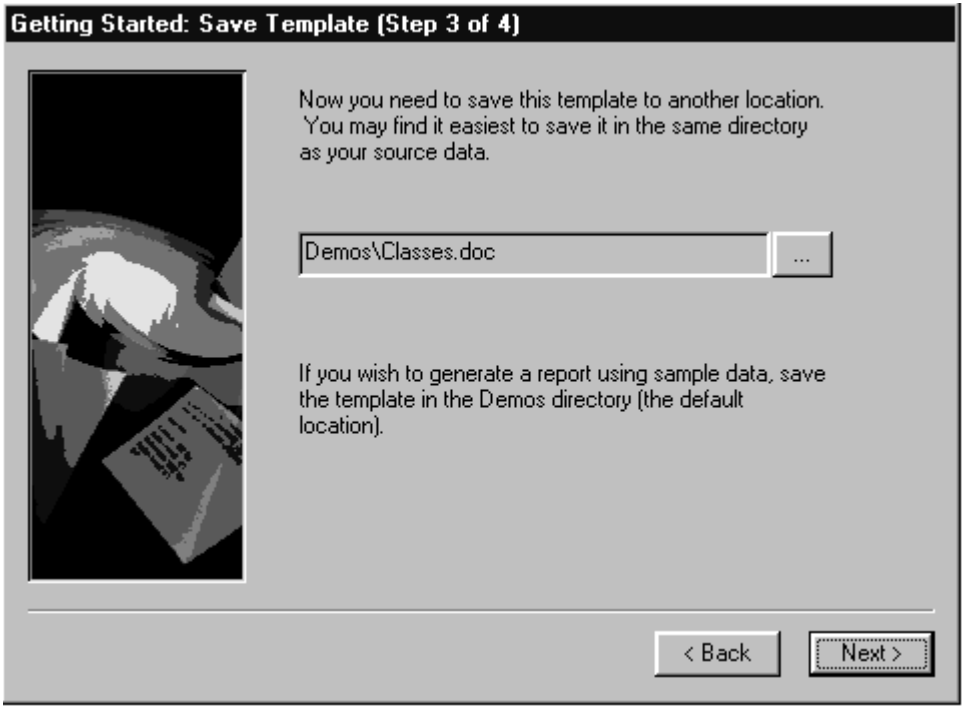


Once you have read the introduction, choose Next to display the list of available templates:



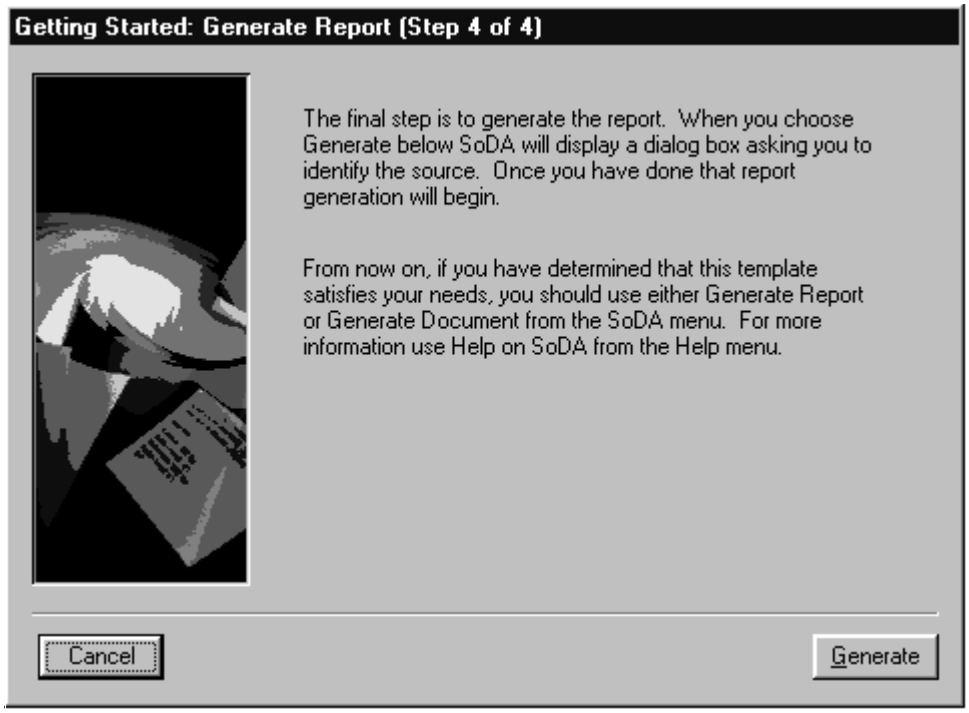
The name of the template should give you an idea about what the template documents. Select the one that most closely fits the type of report you wish to produce.

After selecting a template, you will be given the opportunity to save the template in another directory. If you have a source such as a Rose model or RequisitePro project, you may find it convenient to save the template in the same directory as the source.



If you do not have a model, project, or repository, SoDA includes some samples in the Demos subdirectory, which is the default save location.

The final step is outline in the fourth panel:



When you select Generate from this panel you will see the Identify the <Class> Dialog Box where you will choose the source you wish to document.

Once you have used the Getting Started Wizard, and have a template you wish to use, you no longer need to use the wizard. Rather, use **File->Open** to display the template then choose the Generate Report command from the SoDA menu.

Template View

The Template View guides you through creating a new SoDA template, or through adding, modifying, and deleting commands in an existing SoDA template.

The Template View is the easiest way to create or enhance a SoDA template.

Once you have used the Template View to add and modify the commands, you can then return to the template and make formatting changes, such as adding headings, lists, bullets, and so on.

The template view includes the following buttons across the top of the view:

Open Adds the OPEN command to access a domain.

Display Adds the DISPLAY command.

Repeat Adds the REPEAT command.

Limit Adds the LIMIT command.

Omit Adds the Limit-Omit command.

Otherwise Adds the Limit-Otherwise command.

Modify Enables you to Modify the Selected Command

Delete Enables you to Delete the Selected Command

Go Back Moves the template view up one level in the domain hierarchy

Generate Report
Creates a printable Word document with the same text that is in the template view

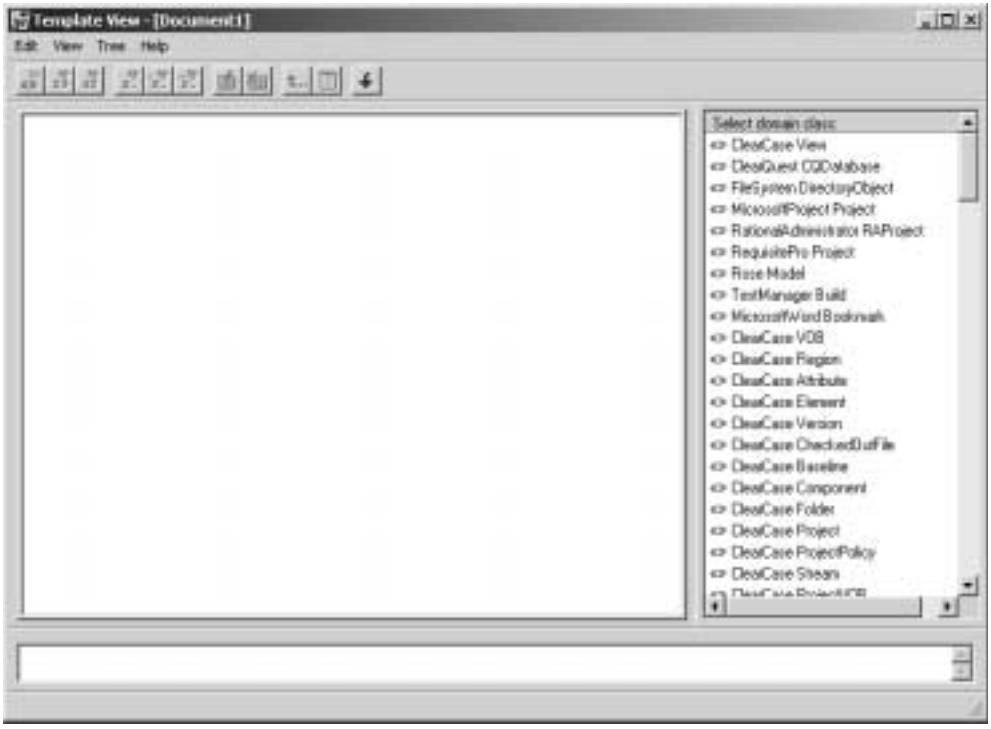
Expand Adds values to the Select value window on the right of the view

Help Opens the online help

Template View: Establishing the Source Kind

If you are starting with an empty template based on the soda.dot Word template, the template view will look like this:

Here you select the starting point for the template. If you are unsure where you want to begin, find the first line that contains your source (Rose Model, ReqPro Project, ClearQuest database or TeamTest Repository).

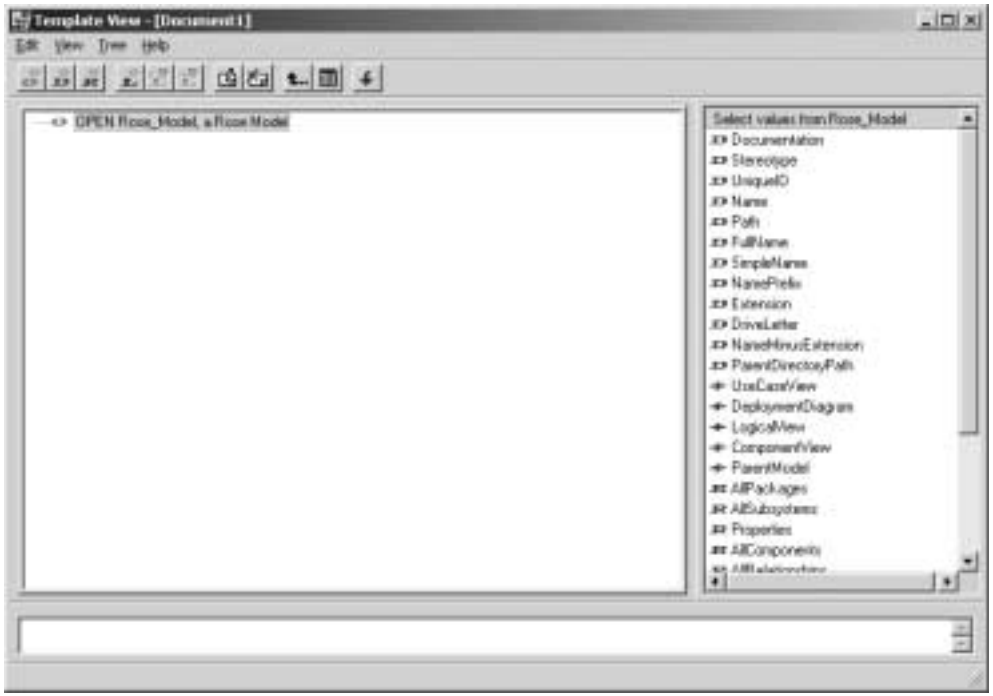


The Template View will guide you through the OPEN command for the first source; for additional OPEN Commands, use the OPEN command button.

Template View: Adding Values

Once you have selected a source (by double-clicking it), or if you are adding a section to an existing template, you will see a panel that looks like this:

The panel on the right contains a list of values that are available based on the current class you are documenting. Some of these values are single values, and some are repeated values.



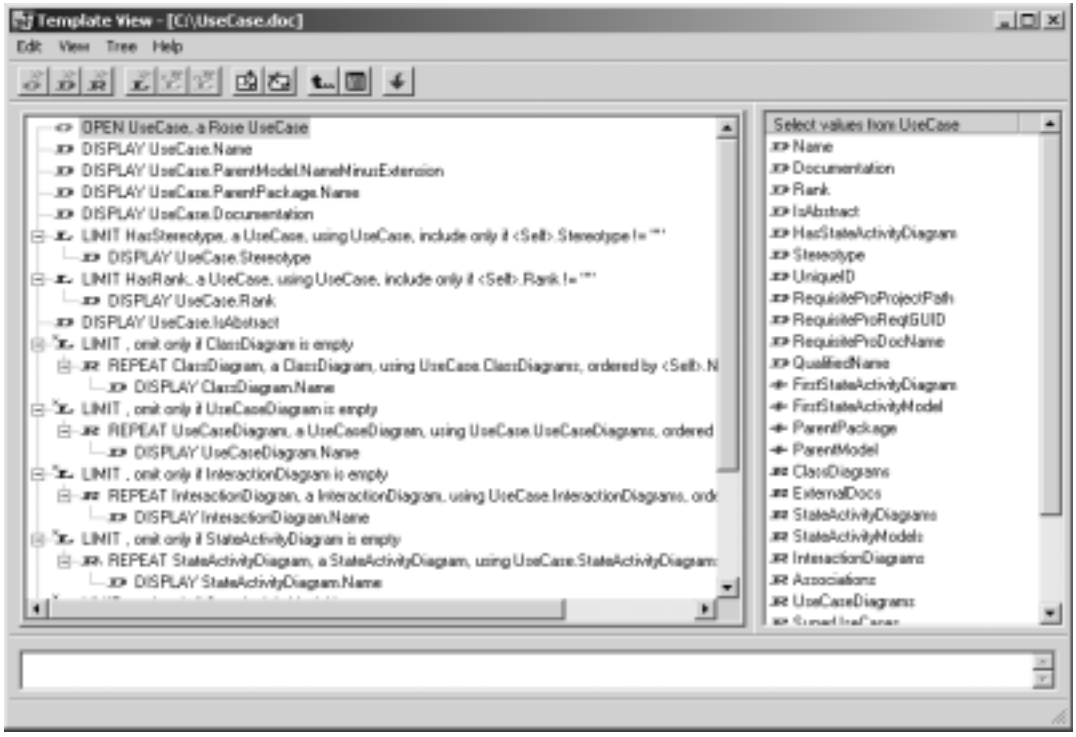
Select the values you wish to document. As you select them you will see them appear in the panel on the left, and in the document.

As you continue to select options, the tree on the left side of the panel will grow, reflecting the choices you have made. Here is a completed template:

The Template View creates REPEAT commands with no sorting or filtering. If you need to include one of these advanced options, double-click the command to change the REPEAT Command.

The DISPLAY commands created by the Template View use default options. If you need to change one of these options, double-click on the command.

Template View: Other Template View Commands



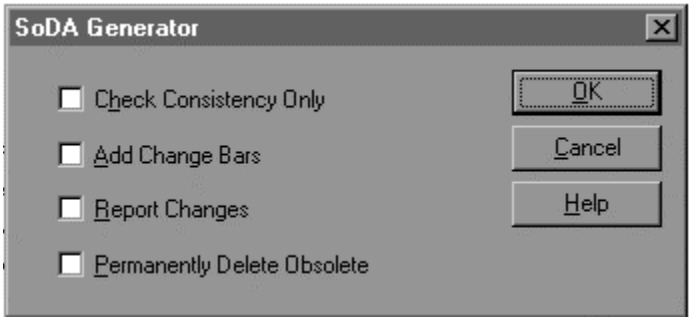
The toolbar in the Template View provides additional features:

From left to right, the toolbar buttons provide the following functions:

OPEN	Create an OPEN command
DISPLAY	Create a DISPLAY command
REPEAT	Create a REPEAT command
LIMIT	Create a LIMIT command
OMIT	Create a special LIMIT: OMIT command (greyed out unless a REPEAT is selected)
OTHERWISE	Create a special LIMIT: OTHERWISE command (greyed out unless a REPEAT is selected)
Modify	Modify the current selected command
Delete	Delete the current selected command
Up	Back up one level in the tree
Report	Create a command report, a printable version of the Template View (not the same as Generate Report from the SoDA menu).

SoDA Generator Dialog Box

This section describes the options available in the SoDA Generator Dialog box. For more on document generation, see “Generating the Document” on page 37.



Check Consistency Only

When you mark this check box, SoDA will examine the document to see what needs to be added, changed, or deleted, and will generate a report, but will not change the existing document.

Add Change Bars

Mark the Add Change Bars check box to highlight changed text in the generated document. Change bars are similar to the notation used in the Microsoft Word "Track Changes/Changed lines" option. The change bars appear in the left margin where changes have been made as a result of the generation process. The change bars can be turned off using Word's Revisions command.

Note: If you turn change bars "on" during the first generation of a document, the entire document will be generated with change bars.

Report Changes

If you mark the Report Changes check box, SoDA will create a Microsoft Word document showing the changes that were made to the document during generation. The Report Changes document opens in Microsoft Word. It has a default "Document #" name. To retain the document, save it using an appropriate name and location.

Permanently Delete Obsolete Sections

This check box specifies how obsolete text resulting from regenerating your document is to be presented in the document. Obsolete text generally results from a deleted object in the information source.

SoDA may also find an object "deleted" if that object has been moved. For example, if your document extracts all of the files in a particular directory, and one of those files is moved to another directory, SoDA will consider that object deleted.

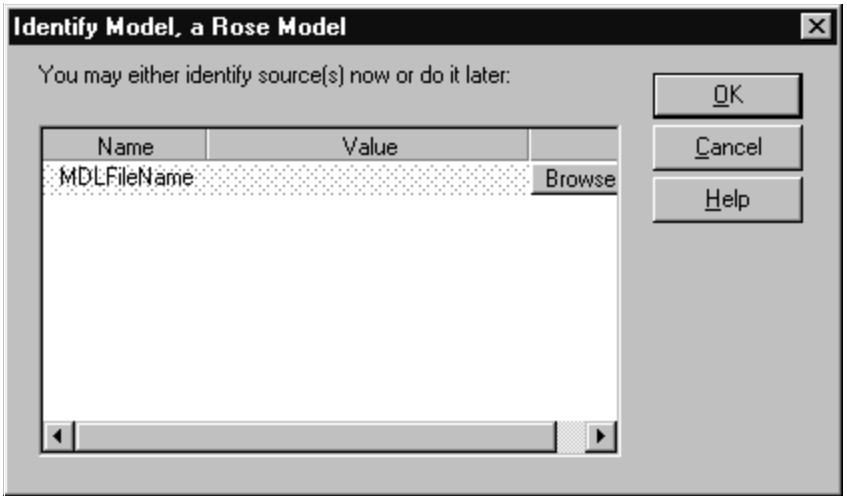
Because an object may be moved by accident (and not deleted), SoDA allows you to leave "deleted" text in your document. This allows you to edit a link to point to a moved object's new location or modify a query to compensate for some other change in the information source.

SoDA gives you two choices for handling obsolete sections. The default method is to "hide" the section by applying the Hidden font attribute to the text. If you check Permanently Delete Obsolete Section, SoDA deletes any sections, text, and graphics associated with a deleted, missing, or moved object in the information source. This includes all children of a deleted section and their children.

Warning This is not reversible. Removed text is permanently deleted from your document and must be recreated manually.

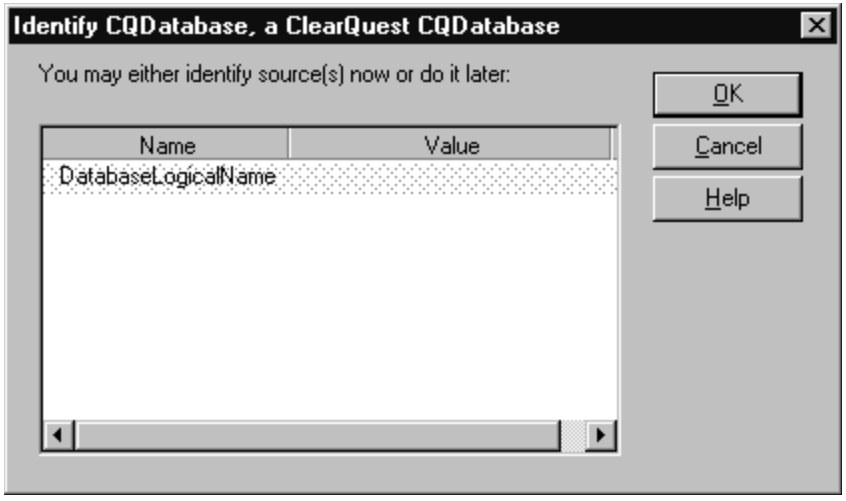
Identify the <Class> Dialog Box

SoDA templates are stored without references to specific files, directories, models, projects, and databases. This dialog box appears when you initially generate a report or document, or when you select a domain in the Template View. The title of the dialog box specifies the class of source required by the template, such as a Rose model.

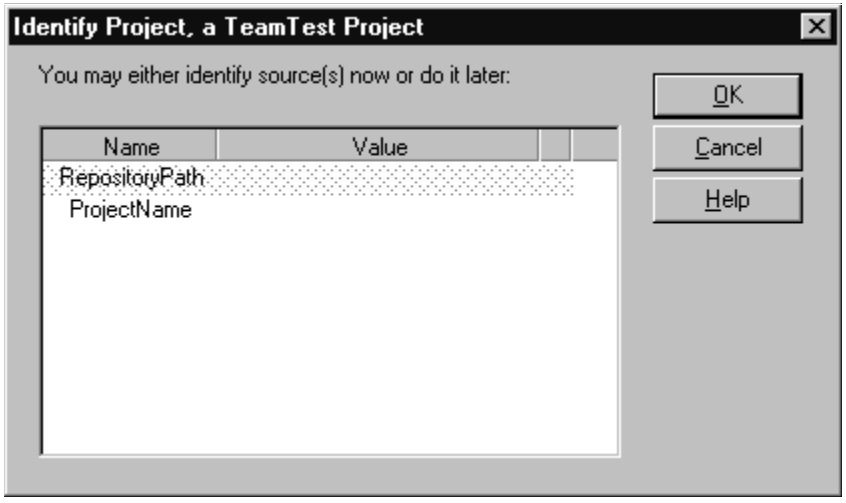


The grid area contains the specific details that identify a particular object of the specified class. In most cases, only one argument is required; however, sometimes there are two or more, depending on the class. If there is a Browse button, you can browse for a specific model, project, or file. Otherwise, click in the Value field and type the required information.

For ClearQuest, enter the database logical name, which appears in the database list that is displayed when you log into ClearQuest.

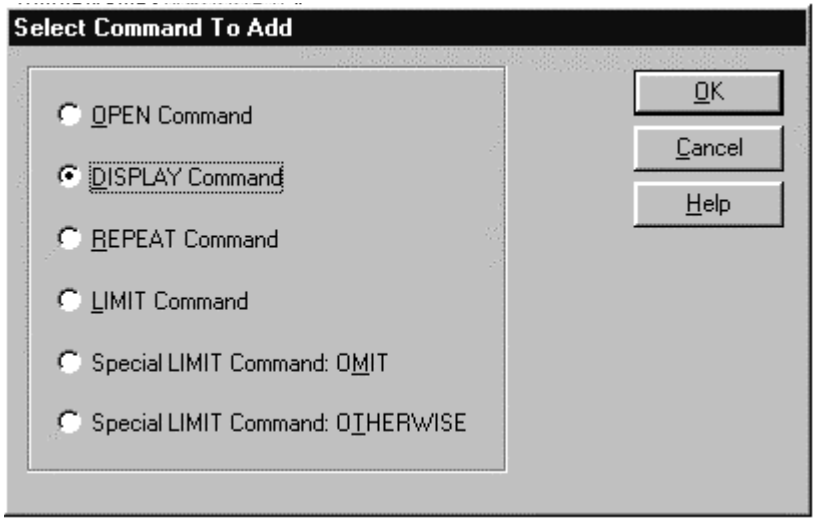


For TeamTest, enter the path for the repository. The repository must exist in the Rational Adminstrator application on your system. The ProjectName is defined in the repository. Type the project name in the applicable Value field. Be sure that spaces within the path and project name match the source exactly.



Select Command to Add Dialog Box

This section describes the Select Command to Add dialog box. For more information on adding commands to your SoDA template, see Adding SoDA Commands.



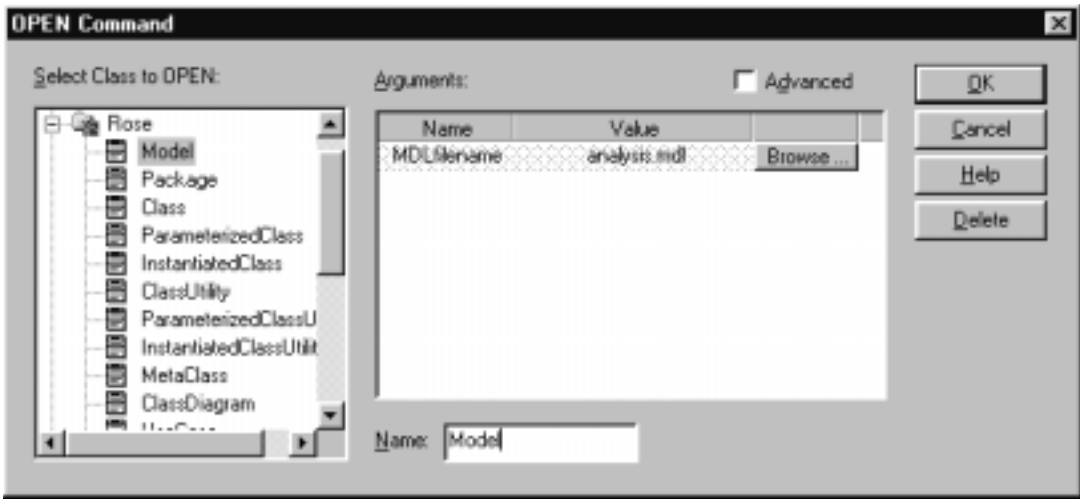
The Select Command to Add dialog box will display a list of the commands that can be added:

- OPEN Command
- DISPLAY Command
- REPEAT Command
- LIMIT Command
- Special LIMIT Command: OMIT
- Special LIMIT Command: OTHERWISE

Choose the command you wish to add, and click OK.

OPEN Command Dialog Box

This section describes the fields in the OPEN command dialog box. For more information on using OPEN commands, see OPEN Command.



Select Class to OPEN:

Depending on the domains available, SoDA provides a list of valid Classes. Choose the one you want to access. The Class list is a general list, based on the domain definition; it is not a list of values from the actual source, such as your Rose model.

Name

Every OPEN command must have a unique name. The name can consist of letters, numbers, and underscores. The OPEN Command Dialog Box automatically sets the name of the command to be the same as the name of the selected Class. If there is a name collision, you must change the name after selecting the class.

When generating a report in the ClearCase domain, the Name field contains the full path and file name (view/VOB/file) for the target file. When the VOB contains more than one branch, it may be necessary to indicate the specific branch that contains the file.

To do so, after browsing for a file, add the following branch notation to the path and file name returned by the browse feature: @@\. For example, c:\my_view \ my_VOB \ my_file.txt@@ \ main.

Arguments

The Arguments area identifies the actual source of the specified class. One, two, or more entries may be required, depending on the class. The Browse button allows you to navigate to a file. If the argument is a filename, either absolute or relative pathnames can be used. If more than one argument is required, it/they must be typed with exact capitalization, spacing, etc.

Advanced

The Advanced check box allows for relative (or Calculated) arguments. Relative arguments are used to open a source, based on the value of another object. To create a relative OPEN:

- Check the Advanced key; a new column in the Arguments area is listed (Kind).

- Click on "Literal" to toggle the Kind to "Calculated."

- Click in the Value column; a tree control lists available options.

- Choose the attribute you need.

Example:

External Word documents, containing the UseCase documentation, can be attached to a Rose UseCase (or other Rose objects). SoDA can include these Word documents into the SoDA document. Follow these steps to create the commands to do this:

1 In the Template (not the Template View), put the cursor within the REPEAT for UseCases, at the point where you want the Word document to appear (if desired, press Enter for better positioning):

- Use **SoDA**>**Add Command** to add a **REPEAT** command.

- Select the **ExternalDocs** relationship.

- Set the **Name** to **ExternalDoc**.

2 Without moving the cursor (i.e., just inside the REPEAT command for ExternalDocs), use the **SoDA> Add Command** to add an OPEN command. An OPEN Command dialog box is displayed.

3 In the Select Class area (left-hand box), click **Word** > **Document**.

4 Click the **Advanced** check box. A new field called **Kind**, is added to the Arguments area.

5 Click on the word **Literal** to toggle the value to **Calculated**.

6 Move the cursor to the green area under the title **Value**, click once to open a tree control.

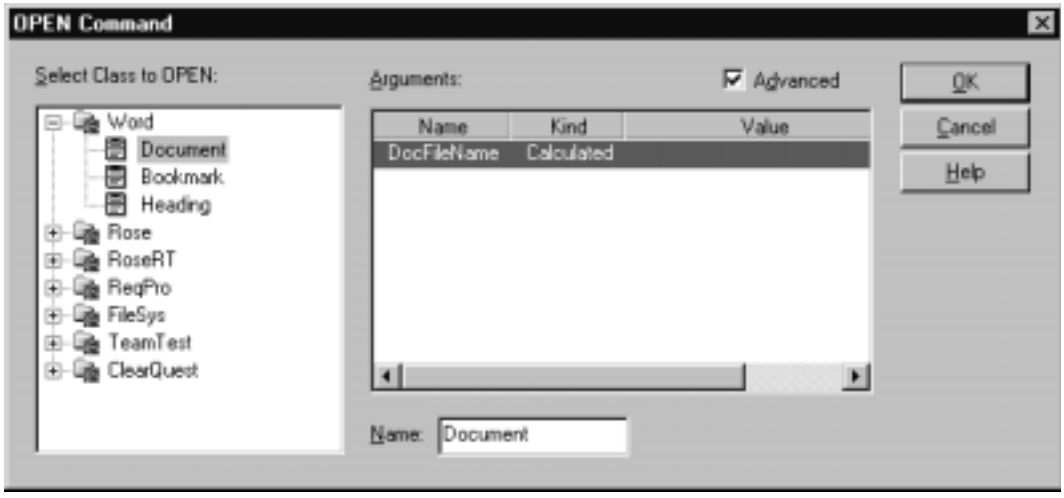
- In the tree control, select **ExternalDoc** > **Value**.

- Click **OK** to create the OPEN command.

7 Without moving the cursor (i.e., just to the right of the OPEN command), use **SoDA**>**Add Command**, to add a **DISPLAY** command. A DISPLAY Command dialog box is displayed.

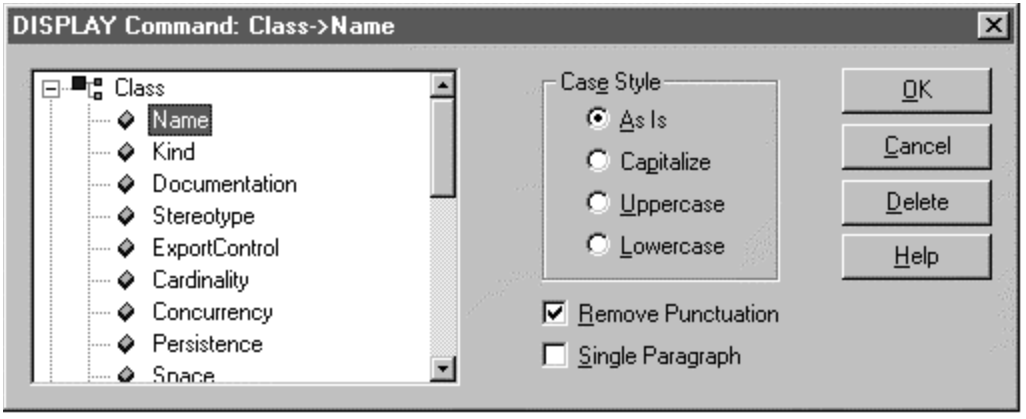
8 In the Select Attribute area, choose **WordFile** > **FormattedText**.

During generation, any external document attached to the UseCase will be inserted into the SoDA document at this point.



DISPLAY Command Dialog Box

This section describes the fields in the DISPLAY command dialog box. For more information on using DISPLAY commands, see DISPLAY Command.



Select Attribute to DISPLAY:

The Select tree control is used to specify the attribute that you wish to select from the source command. To set or change the attribute, simply click on the attribute you need.

Text Value Modifiers

If you choose a text attribute in the Select area, you will see the following modifiers:

Case Style

The Case Style list box lets you specify the case of the text being displayed. Options include: As Is, Capitalize, Upper Case, and Lower Case. For example, “file” would appear in the document as “File” if Capitalize was selected.

Remove Punctuaion

To remove punctuation from generated text, click the Remove Punctuation check box. For example, “Test_Project” will be presented as “Test Project” in the document.

The following punctuation characters are removed: periods, commas, question marks, semi-colons, colons, exclamation marks, underscores, and dollar signs.

The following punctuation characters are NOT removed: single quote, double quote, and dash.

Single Paragraph

To import generated text as a single paragraph, click the Single Paragraph check box. This causes any embedded carriage returns to be ignored. If this option is not selected, all carriage returns in the source text are maintained.

Create Hyperlink

When you select this check box, SoDA will create a “gotolink” hypertext command as part of each DISPLAY command. For more information, see Creating Hyperlink Documents. Use this option in conjunction with the Create Hyperlink Address option in the REPEAT Command Dialog Box.

Graphic Value Modifiers

If you choose a graphic attribute in the Select field, you will see the following modifiers:

Scaling

The Scaling list box lets you decide whether to display the graphic As Is or Scale To Fit.

If you specify As Is, SoDA will display the graphic exactly as it appears in the source. If the graphic is wider or taller than the page margins allow, SoDA will shrink the graphic, maintaining aspect ratio.

If you specify Scale To Fit, SoDA will stretch or shrink the graphic so that it fits the dimensions you specify exactly. This option does not maintain aspect ratio.

Width and Height

To specify the horizontal size at which you want your graphic imported, click in the Width text-entry box and type a number in inches.

To specify the vertical size at which you want your graphic imported, click in the Height text-entry box and type a number in inches.

REPEAT Command Dialog Box

This section describes the fields in the REPEAT command dialog box. For more information on using REPEAT commands, see REPEAT Command.

The dialog box includes the following fields:

Select Objects to Repeat

The Select area is used to identify the n-ary relationship that you wish to use to repeat the section. At the first level of the tree control are the commands that are in the context of the REPEAT command. When you expand one of these you will see N-ary relationships for that class of objects. To select one of these, simply click on the name.

The remaining options are unary relationships. When you expand any of these lines, you will be given additional n-ary (and possibly unary) relationships as choices.

Where Is A

The Where Is A list box lets you limit the results of the REPEAT command to only those objects in a particular class. To choose a class from the Where Is A list box, click the left mouse button in the box, and choose the desired class.

Name

The Name text-entry box is used to specify a name for your repeated section. Every REPEAT command must have a name, and the name must be unique within the current scope. The name can consist of letters, numbers, and underscores. By default, the name of the REPEAT command will match the name of the class listed in the Where Is A list box.

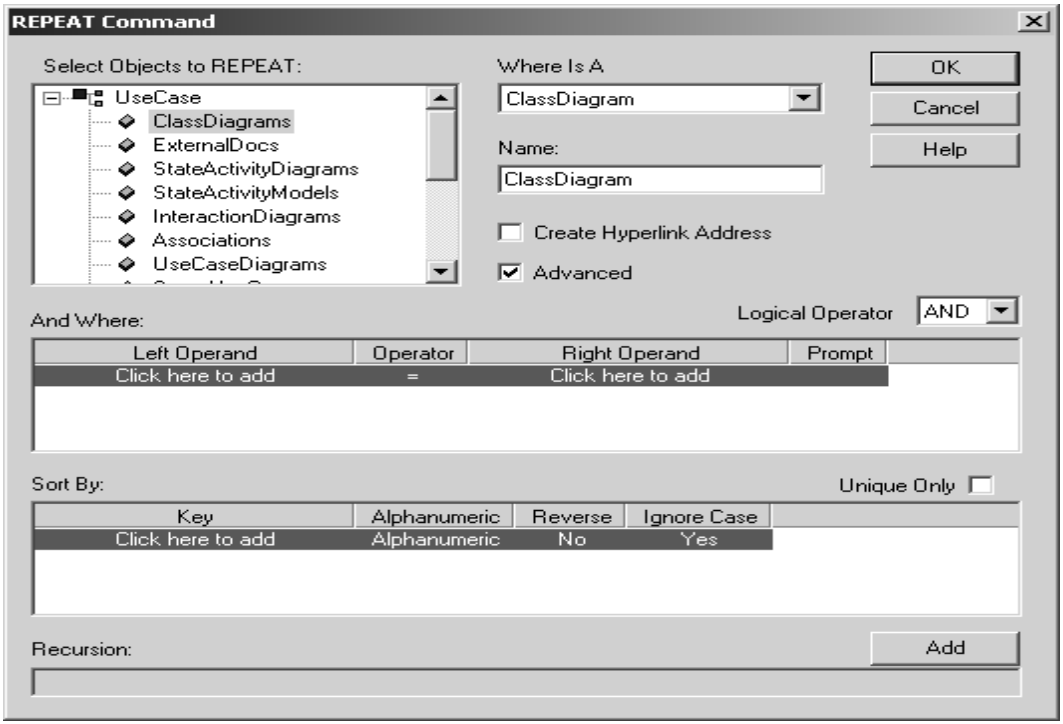
Create Hyperlink Address

When you select this check box, SoDA will create a unique Word bookmark as part of every linked section. Use this option in

conjunction with the Create Hyperlink option in the DISPLAY Command Dialog Box to create hypertext cross-references. When you save the document as HTML these cross-references will become hypertext references and anchors.

And Where (Advanced)

The And Where area (visible only when Advanced is checked) lets



you limit the results of the REPEAT command to only those objects that satisfy a given expression. To create an expression, choose “Click here to add” below the Left Operand. In the tree control, select the attribute that will serve as the left half of the expression. Use the Operator column to choose an operator. Finally, choose “Click here to add” below the Right operand. The right operand can be a literal or another attribute. (Note: Operands that are literals always evaluate to a string value. Therefore, numeric values are note considered to be numbers, but

rather strings. What is the effect?: If you are filtering based on a numberic value, you may get results that you don't expect. For example: fi you are filtering for an operand greater than "42", you may get 43 through 99, but you won't get 100 or beyond. this is because the "1" in "100" is considered to be less than the "4" in "42". Again, operands are evaluated based on string representations, not numeric representations.

When adding a second And Where expression, be sure the Logical Operator is set to the desired value ("And" or "Or").

To remove an And Where expression, right-click the row you wish to remove and choose Delete.

Order By (Advanced)

The Order By area is used to specify how resulting document sections are to be ordered. By default, no sorting is done, i.e., sections are created based on the order the objects are returned by the domain. To insert a sort key, choose "Click here to add" in the Key column. In the tree control, select the attribute that will serve as the sort key.

Choose "Alphanumeric" for alphabetically ordered sections; choose "Numeric" for numerically ordered sections.

Choose "Reverse Order" for reverse alphabetical or numeric order.

Choose "Ignore Case" and SoDA will not consider case when ordering.

If you check the "Unique Only" box, only one of the objects for which all the keys compare equally will be produced.

Prompt

The Prompt (for Filter) field enables SoDA users to perform realtime filtering, during the SoDA generation process, for REPEAT and LIMIT commands.

This new flexibility means that you can generate a template multiple times, each with a different set of filtering criteria, without having to explicitly open the REPEAT / LIMIT commands involved.

To define a Prompt for Filter, click the Advanced checkbox and then click the Prompt column (the right-most column). When

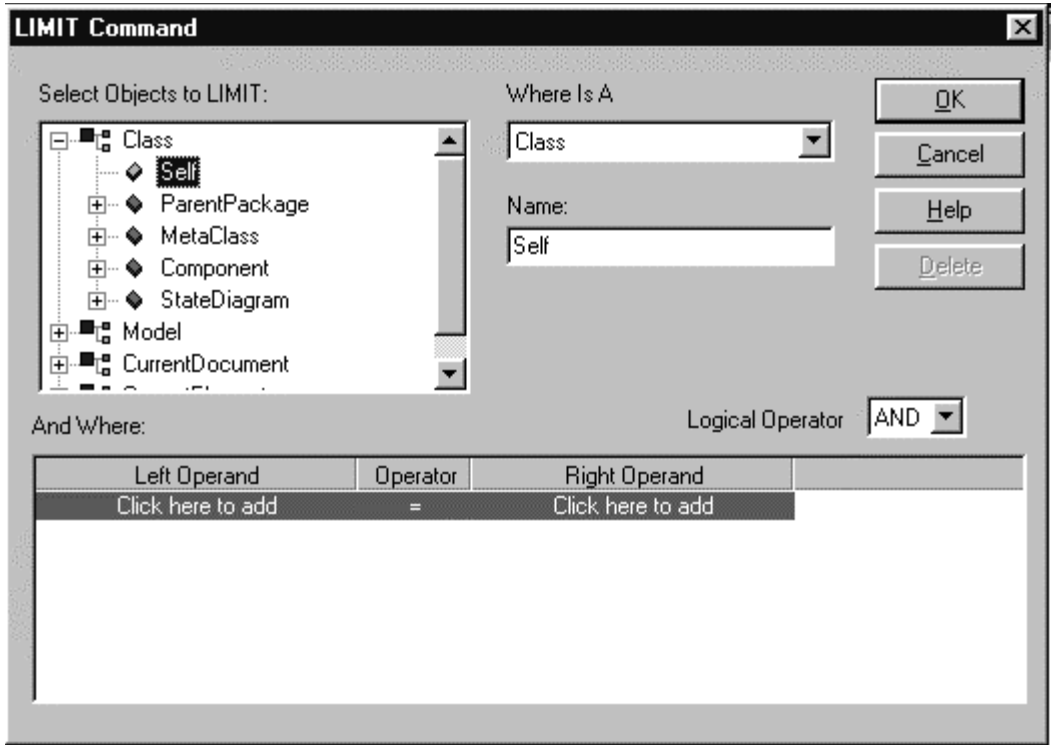
Prompt for Filter is enabled, the field is populated with an asterisk (“*”), and the Right Operand field is disabled.

Each REPEAT command and each LIMIT command can get its own individual set of filter prompts.

When a template that contains prompt commands is generated, a dialog box displays requesting information for the Right Operand. A prompt will be displayed for each command defined for to prompt.

LIMIT Command Dialog Box

This section describes the fields in the LIMIT command dialog box. For more information on using LIMIT commands, see LIMIT Command.



Select Object to Limit

The Select area is used to select the class you wish to examine, based on the current context. The tree control contains all the possibilities for the classes to be limited by the LIMIT command. The name "<Self>" refers to the context object itself. To select a class, use the tree control to highlight the desired class.

Where Is A

The Where Is A list box lets you include a section only if the object is in a particular subclass.

Name

The Name text-entry box is used to specify a name for your LIMIT command.

Every LIMIT command must have a name, and the name must be unique within the current scope. The name can consist of letters, numbers, and underscores.

And Where

The And Where area lets you filter the results of the LIMIT command to only those objects that satisfy a given expression. To create an expression, choose "Click here to add" below the Left Operand. In the tree control, select the attribute that will serve as the left half of the expression. Use the Operator column to choose an operator. Finally, choose "Click here to add" below the Right operand. The right operand can be a literal or another attribute.

When adding a second And Where expression, be sure the Logical Operator is set to the desired value ("And" or "Or").

To remove an And Where expression, right-click the row you wish to remove and choose Delete.

Edit Link Dialog Box

The Edit Link dialog box allows you to change the path of one link to the path of another link in your SoDA document. This is useful when the name of an object in the information source domain has changed. (If you need to change many objects' pathnames because a large number of source objects have moved, see Adjust Links.)

The Edit Link dialog box contains a list of all the possible links in that context. From this list of links, you can choose a path to replace the path specified in the selected link.

Scope of Displayed Links

The scope of the list of links shown in the Edit Link dialog box is determined by the parent REPEAT command. SoDA re-evaluates the command to create a list of all links that could be generated based on the source.

Result of Editing a Link

When you edit a link, you will not notice an immediate change in your document. When you regenerate your document, however, the change may become apparent. As the document regenerates, the new path is followed instead of the old one. Every SoDA link from that point in the hierarchy down will be updated to reflect the new source path.

To edit a link created by SoDA during document generation:

1. Select the section containing the link you wish to edit.
2. Choose **Edit Link** from the **SoDA** menu.
3. In the Edit Link dialog box, choose the path from the scroll list that you want the link selected in the document to have.
4. Click the OK button.

   The path specified in the document's selected link is changed to the path chosen in the Edit Link dialog. When you regenerate your document, information dependent upon that link will change to reflect the new path.

Adjust Links Dialog Box

The Adjust Links dialog box allows you to change the path of a set of links. This is useful if you move the sources for a document without moving the document. (If you need to change only one objects' pathname, or only a few unrelated pathnames, see Edit Link.)

Pathnames are stored by SoDA in a document-relative format. If the sources move and the document does not, the internal

pathnames within links in the document must be adjusted. If they are not, the sections generated from the sources in their original locations may be deleted from the document when SoDA cannot find the moved sources during regeneration.

For example, a REPEAT command is defined to create a section for every file in C:\PROJECTS\JIM. After the document is generated and text has been added, the project is transferred from Jim to Susan. Now, the files are located in C:\PROJECTS\SUSAN. Adjust Links allows you to make this change to your SoDA document simply.

Without Adjust Links, you would have to either

- change each link manually, or
- edit all OPEN commands to point to the correct directory, and regenerate, losing all added text in the process

Adjust Links first converts relative pathnames to absolute pathnames. SoDA then checks the absolute pathname to determine if the link should be updated. For this reason, full pathnames must be specified in the Adjust Links dialog box.

When you adjust links, you will not notice an immediate change in your document. When you regenerate your document, however, the change may become apparent. As the document regenerates, the new path in each link is followed instead of the old one. Every SoDA link from the changed link in the hierarchy down will be updated to reflect the new source path in each link. Of course, if you have adjusted the links properly, and the sources haven't changed other then their location, your document should stay the same.

To modify links:

1. Choose **Adjust Links** from the **SoDA** menu.
2. From the Adjust Links dialog box, enter the invalid path and the path to which you want to change it:
   - Enter the full path to modify in the From field.
   - Enter the new full path in the To field.
3. Click the **OK** button.

SoDA compares the pathname you entered in the From field to each OPEN command (only the absolute paths) and link. It then changes all the paths matching the pathname in the From field to the path you entered in the To field. When you regenerate your document, the new paths will be used and information in the generated document will be updated to reflect the change.

Get From File

The Adjust Links dialog box contains a Get From File check box. When the option is off, SoDA treats the pathnames in the From and To fields as pathnames found in links, OPEN commands, etc. When the toggle is on, SoDA uses the pathname in the From field to find a text file. This "From" text file contains a list of pathnames to be changed. SoDA uses the pathname in the To field to find another text file. This "To" text file contains a list of pathnames to which you want to change.

For example, Jim has the following files:

```
c:\people\jim\projects
    documentation
    code
    misc
```

A REPEAT command has created links to all the files in Jim's documentation, code, and misc directories.

Because Jim has left the company, all of his project files must be transferred. His documentation files will go to Susan. His code files will go to Cynthia. His miscellaneous files will go to Dave. Now, the files have the following paths:

```
c:\people\susan\projects\documentation

c:\people\cynthia\projects\code

c:\people\dave\projects\misc
```

To update the links in the SoDA document, a "From" text file called

c:\people\susan\from_list contains the following lines:

```
c:\people\jim\projects\documentation

c:\people\jim\projects\code
```

c:\people\jim\projects\misc

A “To” text file called \people\susan\to_list contains the following lines:

c:\people\susan\projects\documentation

c:\people\cynthia\projects\code

c:\people\dave\projects\misc

When the Adjust Links command is issued, Susan enters

" c:\people\susan\from_list” in the From field, and she enters

" c:\people\susan\to_list” in the To field. She then toggles the Get From File option on.

When Susan clicks OK, SoDA compares the SoDA document with the “From” file. SoDA then changes paths containing the first line of the “From” file to pathnames containing the first line of the “To” file.

Each path in the “From” file is replaced by the path on the corresponding line of the “To” file, thus order within the files is very important. If the third line of the “To” file in the example above was

" c:\people\cynthia\projects\code” then all references to Jim’s miscellaneous files would be changed to references to Cynthia’s code.

SoDA Options Dialog Box

Thisection describes the fields on the SoDA Options dialog box. SoDA>Options displays a dialog box with two tabs: File Locations and Generation. Fields on the tabs are used to set file locations and properties for generating documents/reports.

File Locations tab

Generated report path: The purpose of this field is to define a location where a SoDA generated report can be saved, if the report

is being generated through a tight integration mechanism (such as from within Rose or RequisitePro) and the directory wherein the SoDA template resides is not a read/write directory (e.g., the template is on a CD due to a minimal install).

Tight integration report directory: This field allows you to assign multiple paths that SoDA can search when serving up a list of templates to be used by the getting started wizard and through tight integration mechanisms (such as from within Rose or RequisitePro).

In the case of the getting started wizard, every Word document based on the SoDA template soda.dot, located in the paths assigned through the SoDA Options dialog box, will be displayed by the wizard as well as the standard templates located in the default SoDA templates directory.

In the case of tight integrations, every Word document based on the SoDA template soda.dot, that contains a document title (a standard Word property), and a domain name as a keyword will be displayed.

For example, a template with the filename “AnotherTemplate.doc” with the title property set to “My Best Project” and the keyword property set to the three words “Rose,Simple,ReqPro” will display on the list of templates available to the Rose integration, as “My Best Project” because “Rose” is a part of the keyword property list.

The file will also display on the list of templates available to the RequisitePro integration as “My Best Project” because “ReqPro” is a part of the keyword property list.

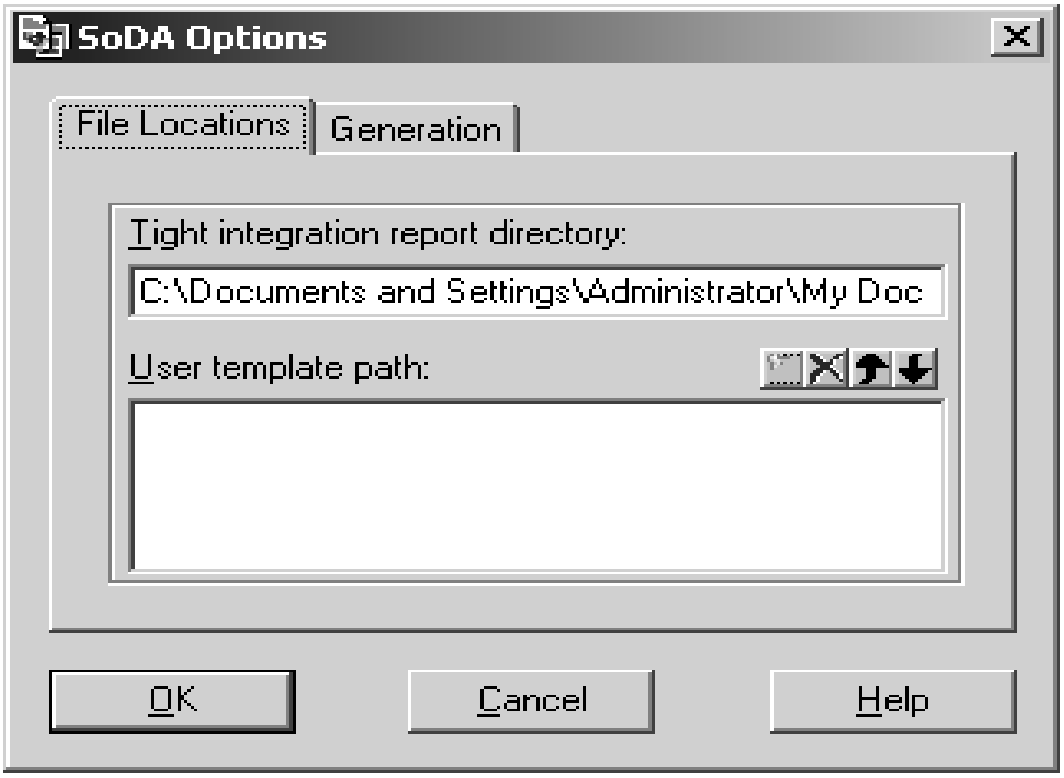
However, given the same template with the keyword property set to “ReqPro” and the file will only be listed by the RequisitePro integration and not by the Rose integration. The reason is won’t be listed for Rose is because the domain name “Rose” does not appear in the keyword property for the template.

Generation tab:

Enable logging: Check to enable logging. Log files may be useful if technical support issues arise.

Save graphics: Check to embed graphics into the generated document/report. By default, the documents are linked to graphic

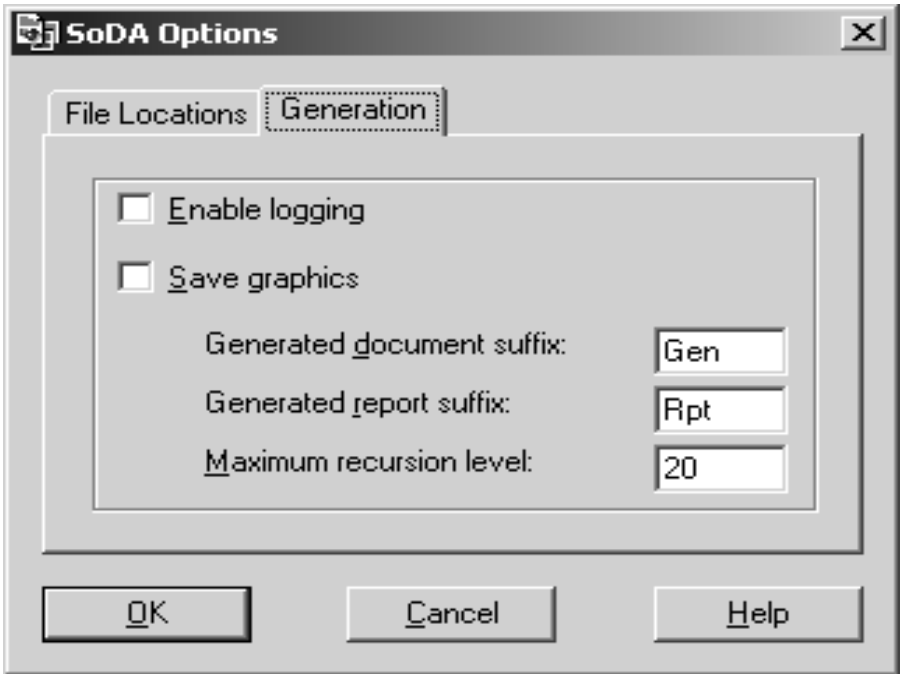
files. Checking this option breaks the link and embeds the graphic into the document/report.



Generated document suffix: The suffix to the appended to the name of all generated documents. The default value is Gen.

Generated report suffix: The suffix to be appended to the name of all generated reports. The default value is Rpt.

Maximum recursion level: The maximum number of levels that a recursive repeat will recurse. The default value is blank (empty). Leaving the field blank denotes that all levels are to be recursed.


SoDA Options
File Locations
Generation
Enable logging
Save graphics
Generated document suffix:
Gen
Generated report suffix:
Rpt
Maximum recursion level:
20
OK
Cancel
Help

A Rational SoDA for Word Domains

Overview

This chapter provides detailed information about SoDA domains. The class-by-class specifics for each RSE adapter are located in this chapter (as well as in the SoDA online help).

It is worthwhile noting that virtually everything in this chapter was automatically generated using a SoDA template and a Rose model.

This appendix also includes a description of:

- Domain aliases
- Domain extensions
- Domain Extension Syntax
- Parsed Attributes
- Script Attributes
- Unary Relationships
- N-ary Relationships

Domain Aliases

Domain aliases let you customize the names of domain classes, attributes, and relationships. SoDA will use the alias names in the Template View and dialog boxes, but will use the reserved name internally.

To create an alias you must modify the description file in the SODA_HOME\domains\rdsi directory. Place the name of the alias as the last word in lines that begin with **class** or **selector**.

For example, if instead of StateTransition you want to use the term Activity. To do so, change the line in Rose.als from:

```
class CStateTransition StateTransition
```

to:

```
class CStateTransition Activity
```

You can also use aliases to translate domain terms to other languages. You cannot use aliases for the File System or Word domains.

Domain Extensions

SoDA allows you to extend the schema of its source domains with a domain extension file. This file is named DOMAIN.EXT and must be located in the SODA_HOME\domains directory. Domain extensions allow you to define new:

- Attributes for a source class that are:
  - Parsed from an existing attribute
  - Derived by a batch script that is passed existing attributes as arguments
- Unary relationships for a class to files and/or directories
- N-ary relationships for a class to files and/or directories

Domain Extension Syntax

The syntax for describing the various domain extensions is described in the following sections. These syntax descriptions use pointy brackets (<>) to indicate the description of an item and ellipsis (...) to indicate that an item may be repeated. All other characters are literal.

The domain extension file contains one or more extension specifications for existing classes. The syntax for these extensions is:

```
EXISTING_CLASS <Domain>.<Class>
   <Parsed Attribute>
   ...
   <Script Attribute>
   ...
   <Unary Relationship>
   ...
   <N-ary Relationship>
   ...
END_CLASS
...
```

where <Domain> is the name of an information source domain—for example: FileSys, Frame, or Rose—and <Class> is the name of any class defined in that domain—for example: Directory or File in the File System domain.

Parsed attributes, script attributes, Unary relationships, and N-ary relationships are described in the following pages.

Parsed Attributes

Parsed attributes provide a convenient way to define additional structure within an existing attribute. Parsed attributes are evaluated by searching the source attribute for the start delimiter, then searching for the end delimiter, and returning the text in between but not including the two delimiters. The syntax for specifying a set of parsed attributes to be generated from a source attribute is:

```
EXISTING_CLASS <Domain>.<Class>
    SOURCE %<Attribute Name>%
        ATTRIBUTE <Name> "<Start Delimiter>" "<End Delimiter>"
        ...
    END_SOURCE
END_CLASS
```

Note that the ATTRIBUTE keyword, the name, and the start and end delimiters must all appear on the same line in the domain extension file. If you omit the end delimiter, it defaults to the newline character.

The most common use of parsed attributes is to define keywords to be specified in program comments. For example, if you wanted to define ID, author, and purpose keywords for Rose classes, you would add the following lines to your domain extension file:

```
EXISTING_CLASS Rose.CClass
    SOURCE %Documentation%
        ATTRIBUTE @Ident "@ID:" "@"
        ATTRIBUTE @Author "@AUTHOR:" "@"
        ATTRIBUTE @Purpose "@PURPOSE:" "@"
    END_SOURCE
END_CLASS
```

These lines will cause @Ident, @Author, and @Purpose attributes to appear in the SoDA Field dialog for Rose classes. The use of the at-sign (@) character in the attribute name is not required, but rather is a convention followed to distinguish schema extensions from the base attributes of a class.

Script Attributes

Script attributes allow you to provide shell scripts that derive new attributes for a class from existing attributes of the class. The syntax for adding script attributes to a class is:

```
EXISTING_CLASS <Domain>.<Class>
     ATTRIBUTE <Name> <Script Name> <Argument List>
     ...
END_CLASS
```

Note that the ATTRIBUTE keyword, the script attribute's name, the script name, and the script's arguments must all appear on the same line in the domain extension file. The <Script Name> must be a simple command name, and the <Argument List> must be a list of one or more attributes of the class enclosed in percent signs (%).

Beware of blanks or newlines in arguments. The script name and its arguments are simply passed to the shell—which will behave as usual—for interpretation.

For example, if you wanted to add attributes for the number of lines and number of words in a file, you could add the following lines to your domain extension file:

```
EXISTING_CLASS File.File
     ATTRIBUTE @LineCount linecount %FullName%
     ATTRIBUTE @WordCount wordcount %FullName%
END_CLASS
```

These lines will cause @LineCount and @WordCount attributes to appear in the SoDA Field dialog for File System files. You will also need to write two executables, named linecount and wordcount, which given the full pathname to a file, calculate and return the number of lines and words in it, respectively.

Unary Relationships

It is possible to define additional Unary relationships for a class. These relationships can only be to the following File System domain classes: DirectoryObject, File, or Directory.

The syntax for defining Unary relationships from one class to another is:

```
EXISTING_CLASS <From Domain.Class>
     UNARY_RELATIONSHIP <Name> <FileSys.Class> <Naming Exp>
     ...
END_CLASS
```

where <Naming Exp> is any valid file naming expression. You can reference other attributes of the (from) class within the naming expression by enclosing them in percent signs (%).

Note that the UNARY_RELATIONSHIP keyword, the name, the domain, the class, and the naming expression must all appear on the same line in the domain extension file.

For example, adding the following lines to your domain extension file defines a new relationship named @Design from an Apex view to a subdirectory within that view called design:

```
EXISTING_CLASS Apex.View
   UNARY_RELATIONSHIP @Design File.Directory %FullName%”/design”
END_CLASS
```

N-ary Relationships

It is also possible to define N-ary relationships to File System domain classes in the same manner as Unary relationships. The syntax for defining N-ary relationships from one class to another is:

```
EXISTING_CLASS <From Domain.Class>
    NARY_RELATIONSHIP <Name> <To Domain.Class> <Naming Exp>
    ...
END_CLASS
```

<Naming Exp> is the same as for Unary relationships with the exception that wildcard characters can be used.

Note that the NARY_RELATIONSHIP keyword, the name, the domain, the class, and the naming expression must all appear on the same line in the domain extension file.

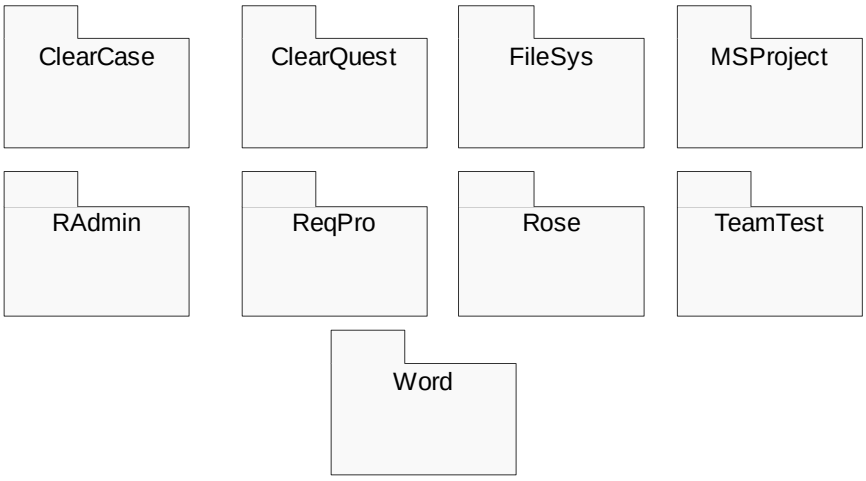
For example, adding the following lines to your domain extension file defines a new N-ary relationship named @Pics from a directory to all PostScript files within that directory:

```
EXISTING_CLASS FileSys.Directory
    NARY_RELATIONSHIP @Pics FileSys.File %FullName%"/*.ps"
END_CLASS
```

Domains (RSE Adapter Overview)

This chapter was automatically generated and formatted using SoDA. All adapter specific information (diagrams, names, attributes, descriptions, etc., (excepting the Rose Realtime domain)) was extracted by SoDA from a Rose model.

Being that this document is demonstration of what SoDA can produce, only minimal changes have been made.



The collection of RSE adapters.

RSE Adapter	Documentation
ClearCase	The RSE adapter for Rational ClearCase.
ClearQuest	The RSE adapter for Rational ClearQuest.
FileSys	The RSE adapter for Microsoft File System.
MSProject	The RSE adapter for Microsoft Project.
RAdmin	The RSE adapter for Rational Administrator.

RSE Adapter	Documentation
ReqPro	The RSE adapter for Rational RequisitePro.
Rose	The RSE adapter for Rational Rose.
TeamTest	The RSE adapter for Rational Test Manager.
Word	The RSE adapter for Microsoft Word.

RSE Adapter: ClearCase

The RSE adapter for ClearCase enables extraction of version control, activity management, workspace management, and VOB meta-data information from ClearCase for inclusion in your SoDA documents.

The structure of the SoDA ClearCase domain reflects the public external interface of ClearCase, also known as CAL (ClearCase Automation Library). There is a direct correspondence between SoDA Classes and ClearCase interfaces. Therefore, the ClearCase documentation for CAL may be helpful in understanding and using the SoDAA ClearCase domain.

Accessing Objects with Pathnames

An object with a pathname (e.g., Element, Version, Branch) cannot be accessed by SoDA without a ClearCase view. When a VOB is identified using only a VOB tag, SoDA doesn't know the view it needs to use to access objects for that VOB. To provide a view, use the pathname of the VOB as seen through a specific view in the VOBIdentifier parameter of the Open command in place of just using the VOB tag. This view will then be used to resolve the names of objects accessed via the VOB.

The following classes are available through the ClearCase adapter:

Activity

Attribute

AttributeType

Baseline

Branch

BranchType

CheckedOutFile

Component

Element

File

Folder

HistoryRecord

Hyperlink

HyperlinkType

Label

LabelType

Lock

Name

Project

ProjectPolicy

ProjectVOB

Region

Stream

Trigger

TriggerType

UCMObject
Value
Version
View
VOB
VOBObject

Class: Activity (ClearCase Adapter)

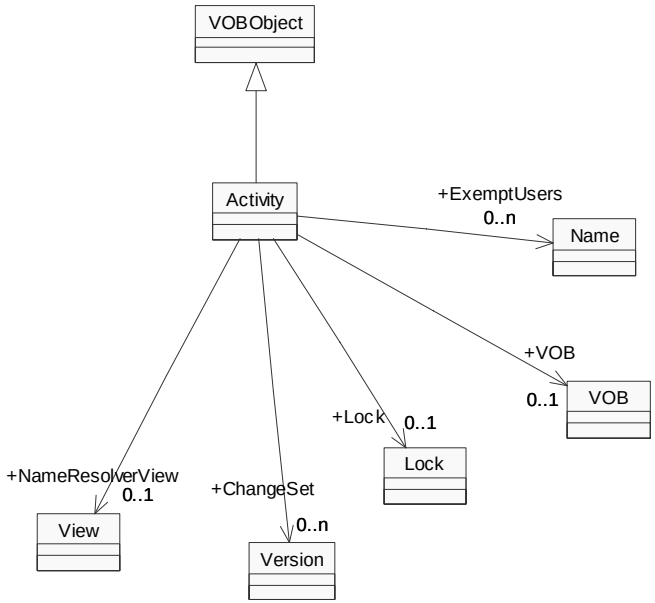
In the UCM model, an activity is a ClearCase object that you use to track the work required to complete a development task. An activity includes a text headline, which describes the task, and a change set, which identifies the versions that you create or modify while working on the activity.

Class Hierarchy: VOBObject>Activity

SubClasses of Activity

Activity has no subclasses.

Class Diagram



Attributes Specific to Activity

Attributes	Inherited From	Description
Comment	VOBObject	The comment associated with the VOB object.
CreatedBy	VOBObject	The user who created the object.
CreatedOn	VOBObject	The date the object was created.
Headline		The title of this activity.
LockDescription		The description of the lock on this activity.
LockedBy		The name of the user who locked the activity.
LockedOn		The date the activity was locked.
Master		The master replica for this activity.
Name	VOBObject	The name of the versioned object.
OID	VOBObject	The object identifier for the VOB object.
Selector	VOBObject	An expression used by ClearCase's file typing mechanism to match a file system object (or just the name of one).
State		The state of the lock on this activity.
TypeName	VOBObject	The VOBObject type name.
VOBFamilyUUID	VOBObject	The VOB family UUID for the VOB of this VOB object.

Relationships Specific to Activity (see also class diagram above)

Name	Kind	Class	Documentation
Lock	0..1	Lock	The lock on this activity.

Name	Kind	Class	Documentation
ExemptUsers	0..n	Name	The list of users exempted from the lock.
VOB	0..1	VOB	The VOB containing the activity.
ChangeSet	0..n	Version	The versions in this activity s change set.
NameResolverView	0..1	View	A best guess view for resolving the names of versions in a change set.

Class: Attribute (ClearCase Adapter)

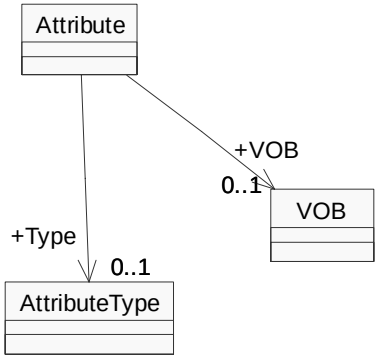
An attribute is a meta-data annotation attached to a VOB object, in the form of a name/value pair. The names of attributes are specified by user-defined attribute types; values of these attributes can be set by users. For example, a project administrator may create an attribute type whose name is QAed. A user may then attach the attribute QAed with the value Yes to a version. An attribute is a VOB object.

Class Hierarchy: Artifact>Attribute

SubClasses of Attribute

Attribute has no subclasses.

Class Diagram



Attributes Specific to Attribute

Attributes	Inherited From	Description
Name		The attribue name.
TypeName		The attribute type name.
Value		The attribute value.

Relationships Specific to Attribute (see also class diagram above)

Name	Kind	Class	Documentation
VOB	0..1	VOB	The VOB containing the object having this attribute.
Type	0..1	AttributeType	The attribute type of this attribute.

Class: AttributeType (ClearCase Adapter)

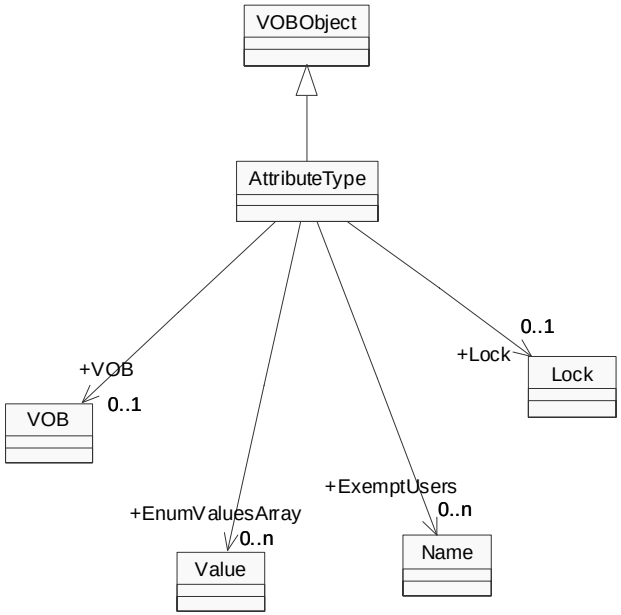
An attribute type is a VOB object that defines an attribute name for use within a VOB. It constrains the attribute values that can to paired with the attribute name (for example, an integer in the range 1-10).

Class Hierarchy: VOBObject>AttributeType

SubClasses of AttributeType

AttributeType has no subclasses.

Class Diagram



Attributes Specific to AttributeType

Attributes	Inherited From	Description
Comment	VOBObject	The comment associated with the VOB object.
Constraint		The constraint for this attribute type.
CreatedBy	VOBObject	The user who created the object.
CreatedOn	VOBObject	The date the object was created.
DefaultValue		The default value for this attribute type.
Group		The group to which this attribute type belongs.
HasSharedMas-tership		Whether this attribute type is shared or can be mastered.
LockDescription		The user comment for the lock
LockedBy		The name of the user who locked this attribute type.
LockedOn		The date this attribute type was locked.
LowerIsInRange		Whether or not the lower value is in the range of legal values for this attribute type.
LowerValue		The lower value for this attribute type.
Master		The master replica for this attribute type.
Name	VOBObject	The name of the versioned object.
NumberOfEnum-Values		The number of enumerated values for this attribute type.
OID	VOBObject	The object identifier for the VOB object.
Owner		The owner of this attribute type.

Attributes	Inherited From	Description
Scope		The scope of this attribute type (for example, local to this VOB).
Selector	VOBObject	An expression used by ClearCase's file typing mechanism to match a file system object (or just the name of one).
State		The state of the lock on this attribute type.
TypeName	VOBObject	The VOBObject type name.
UpperIsInRange		Whether or not the upper value is in the range of legal values for this attribute type.
UpperValue		The upper value for this attribute type.
ValueType		The value type for this attribute.
VOBFamilyUUID	VOBObject	The VOB family UUID for the VOB of this VOB object.

Relationships Specific to AttributeType (see also class diagram above)

Name	Kind	Class	Documentation
Lock	0..1	Lock	The lock on this attribute type.
ExemptUsers	0..n	Name	The list of users who are exempt from the lock.
EnumValue-sArray	0..n	Value	The enumerated values for this attribute type.
VOB	0..1	VOB	The VOB containing this attribute type.

Class: Baseline (ClearCase Adapter)

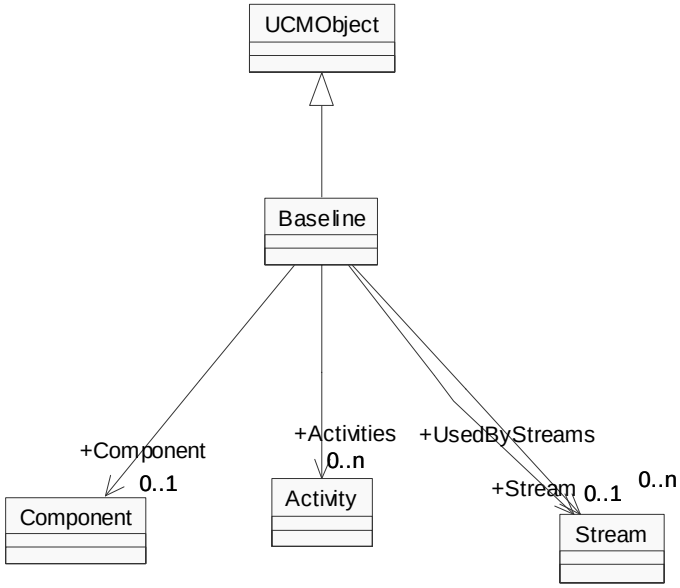
A ClearCase UCM object that typically represents a stable configuration for one or more components. A baseline identifies activities and one version of every element visible in one or more components.

Class Hierarchy: UCMObject>Baseline

SubClasses of Baseline

Baseline has no subclasses.

Class Diagram



Attributes Specific to Baseline

Attributes	Inherited From	Description
Comment	VOBObject	The comment associated with the VOB object.
CreatedBy	VOBObject	The user who created the object.
CreatedOn	VOBObject	The date the object was created.
Group	UCMObject	The group to which the UCM object belongs.
LabelStatus		The label status for the baseline UCM object.
LockDescription	UCMObject	The comment of the user who locked this UCMObject.
LockedBy	UCMObject	The user who locked this UCMObject.
LockedOn	UCMObject	The date on which this UCMObject was locked.
Master	UCMObject	The master replica for the UCM object.
Name	VOBObject	The name of the versioned object.
OID	VOBObject	The object identifier for the VOB object.
Owner	UCMObject	The owner of the UCM object.
PromotionLevel		The promotion level for the baseline UCM object.
Selector	VOBObject	An expression used by ClearCase's file typing mechanism to match a file system object (or just the name of one).
State	UCMObject	The state of the lock on this UCMObject.
Title	UCMObject	The title of the UCM object.

Attributes	Inherited From	Description
TypeName	VOBObject	The VOBObject type name.
VOBFamilyUUID	VOBObject	The VOB family UUID for the VOB of this VOB object.

Relationships Specific to Baseline (see also class diagram above)

Name	Kind	Class	Documentation
Activities	0..n	Activity	The activities included in the baseline UCM object.
Component	0..1	Component	The component containing the baseline UCM object.
Stream	0..1	Stream	The stream in which the baseline UCM object was created.
Used-ByStreams	0..n	Stream	All of the streams for which the baseline UCM object serves as a foundation.

Class: Branch (ClearCase Adapter)

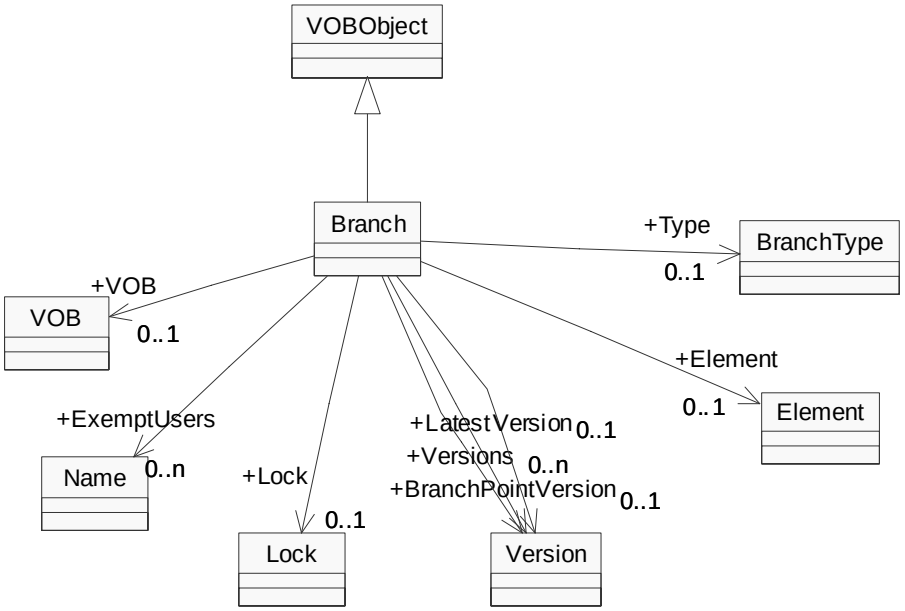
A branch is an object that specifies a linear sequence of versions of an element. The entire set of versions of an element is called a version tree; it always has a single main branch, and may also have subbranches. Each branch is an instance of a branch type object. A branch is a VOBObject, and thus may have a lock preventing modification.

Class Hierarchy: VOBObject>Branch

SubClasses of Branch

Branch has no subclasses.

Class Diagram



Attributes Specific to Branch

Attributes	Inherited From	Description
BranchPath		The pathname of this branch.
Comment	VOBObject	The comment associated with the VOB object.
CreatedBy	VOBObject	The user who created the object.
CreatedOn	VOBObject	The date the object was created.
ExtendedPath		Extended pathname of the branch.
LockDescription		The description of the current lock on the branch.
LockedBy		The name of the user who locked the branch.
LockedOn		The date the branch was locked.
Master		The master replica for this branch.
Name	VOBObject	The name of the versioned object.
OID	VOBObject	The object identifier for the VOB object.
Selector	VOBObject	An expression used by ClearCase's file typing mechanism to match a file system object (or just the name of one).
State		The state of the lock on this branch.
TypeName	VOBObject	The VOBObject type name.
VOBFamilyUUID	VOBObject	The VOB family UUID for the VOB of this VOB object.

Relationships Specific to Branch (see also class diagram above)

Name	Kind	Class	Documentation
Lock	0..1	Lock	The lock on this branch.
ExemptUsers	0..n	Name	A list of users exempt from the lock.
Type	0..1	BranchType	The branch type of this branch.
BranchPoint-Version	0..1	Version	The version from which this branch sprouts.
Element	0..1	Element	The element to which this branch belongs.
Versions	0..n	Version	An enumeration of all versions along this branch.
LatestVersion	0..1	Version	The latest version of this branch
VOB	0..1	VOB	The VOB containing this branch.

Class: BranchType (ClearCase Adapter)

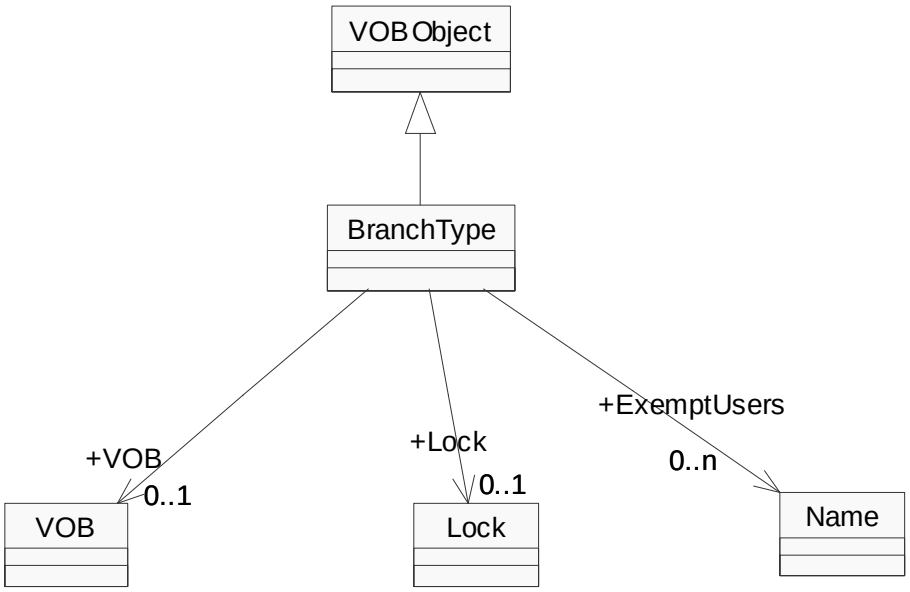
A branch type defines a branch name for use within a VOB.

Class Hierarchy: VOBObject>BranchType

SubClasses of BranchType

BranchType has no subclasses.

Class Diagram



Attributes Specific to BranchType

Attributes	Inherited From	Description
Comment	VOBObject	The comment associated with the VOB object.

Attributes	Inherited From	Description
Constraint		The constraint for this branch type.
CreatedBy	VOBObject	The user who created the object.
CreatedOn	VOBObject	The date the object was created.
Group		The group to which this branch type belongs.
LockDescription		The user comment for the lock on this branch type.
LockedBy		The name of the user who locked this branch type.
LockedOn		The date on which this branch type was locked.
Master		The master replica for this branch type.
Name	VOBObject	The name of the versioned object.
OID	VOBObject	The object identifier for the VOB object.
Owner		The owner of this branch type.
Scope		The scope of this branch type (for example, local to this VOB)
Selector	VOBObject	An expression used by ClearCase's file typing mechanism to match a file system object (or just the name of one).
State		The state of the lock on this branch type.
TypeName	VOBObject	The VOBObject type name.
VOBFamilyUUID	VOBObject	The VOB family UUID for the VOB of this VOB object.

Relationships Specific to BranchType (see also class diagram above)

Name	Kind	Class	Documentation
Lock	0..1	Lock	The lock on this branch type.
ExemptUsers	0..n	Name	The list of users who are exempt from the lock on this branch type.
VOB	0..1	VOB	The VOB containing this branch type.

Class: CheckedOutFile (ClearCase Adapter)

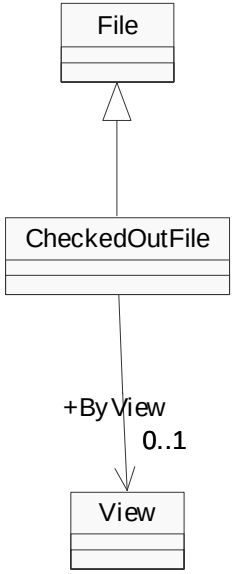
A checked out file is a placeholder in the VOB database created by the checkout command. This object corresponds to the view-private object (file or directory) that you work with after checking out an element. A checkout will be marked reserved if reserved checkout has been performed (meaning the file is exclusively locked for one user).

Class Hierarchy: VOBObject>File>Version>CheckedOutFile

SubClasses of CheckedOutFile

CheckedOutFile has no subclasses.

Class Diagram



Attributes Specific to CheckedOutFile

Attributes	Inherited From	Description
Comment	VOBObject	The comment associated with the VOB object.
CreatedBy	VOBObject	The user who created the object.
CreatedOn	VOBObject	The date the object was created.
ExtendedPath	File	The VOB-extended pathname of this file system object.
Extension	File	The file extension (the portion after the final dot).
Identifier	Version	The version s identifier string.
IsCheckedOut	Version	Whether or not this object represents a checked-out file.
IsDifferent	Version	Whether or not this version is different from its predecessor.
IsDirectory	File	Whether or not the file is a directory.
IsHijacked	Version	Whether or not this version is hijacked.
IsLatest	Version	Whether or not this version is the latest on its branch.
IsReserved		Whether or not this checkout is reserved.
Name	VOBObject	The name of the versioned object.
NameMinusEx-tension	File	The simple name of the file without the extension and final.dot.
OID	VOBObject	The object identifier for the VOB object.
Path	File	The pathname to this file system object.

Attributes	Inherited From	Description
Selector	VOBObject	An expression used by ClearCase's file typing mechanism to match a file system object (or just the name of one).
SimpleName	File	The simple name of the file, i.e., the name of the file without the path.
TypeName	VOBObject	The VOBObject type name.
VersionNumber	Version	This version s version number.
VOBFamilyUUID	VOBObject	The VOB family UUID for the VOB of this VOB object.

Relationships Specific to CheckedOutFile (see also class diagram above)

Name	Kind	Class	Documentation
ByView	0..1	View	The view to which this file is checked out.

Class: Component (ClearCase Adapter)

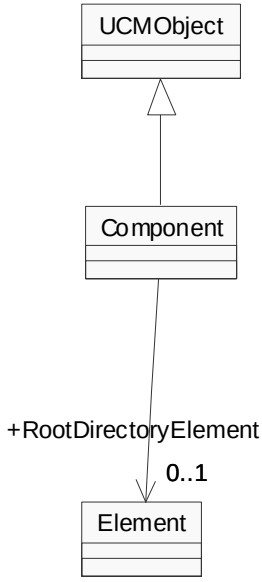
A ClearCase object that you use to group a set of related directory and file elements within a UCM project. Typically, you develop, integrate, and release the elements that make up a component together. A project must contain at least one component, and it can contain multiple components. Projects can share components.

Class Hierarchy: UCMObject>Component

SubClasses of Component

Component has no subclasses.

Class Diagram



Attributes Specific to Component

Attributes	Inherited From	Description
Comment	VOBObject	The comment associated with the VOB object.
CreatedBy	VOBObject	The user who created the object.
CreatedOn	VOBObject	The date the object was created.
Group	UCMObject	The group to which the UCM object belongs.
LockDescription	UCMObject	The comment of the user who locked this UCMObject.
LockedBy	UCMObject	The user who locked this UCMObject.
LockedOn	UCMObject	The date on which this UCMObject was locked.
Master	UCMObject	The master replica for the UCM object.
Name	VOBObject	The name of the versioned object.
OID	VOBObject	The object identifier for the VOB object.
Owner	UCMObject	The owner of the UCM object.
Selector	VOBObject	An expression used by ClearCase's file typing mechanism to match a file system object (or just the name of one).
State	UCMObject	The state of the lock on this UCMObject.
Title	UCMObject	The title of the UCM object.
TypeName	VOBObject	The VOBObject type name.
VOBFamilyUUID	VOBObject	The VOB family UUID for the VOB of this VOB object.

Relationships Specific to Component (see also class diagram above)

Name	Kind	Class	Documentation
RootDirecto-ryElement	0..1	Element	The root directory for the com-ponent.

Class: Element (ClearCase Adapter)

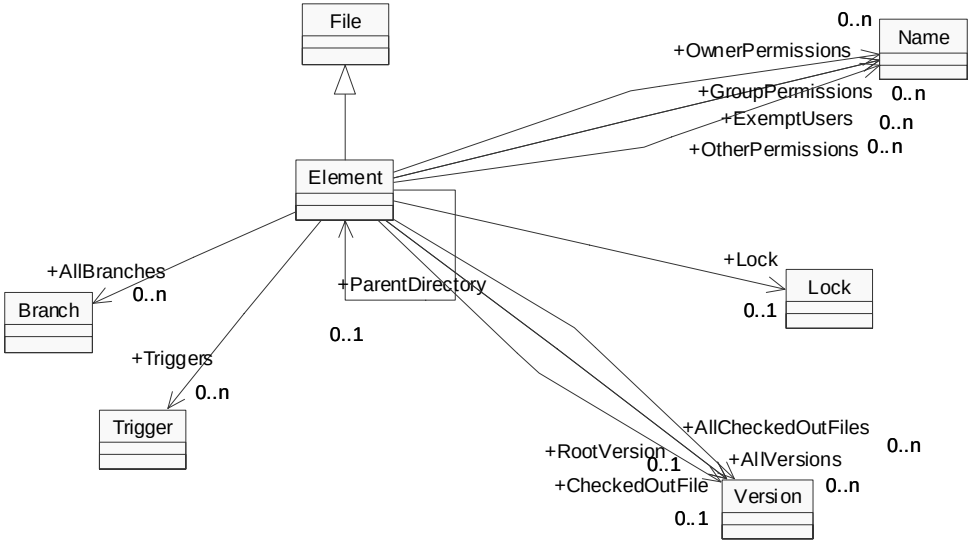
An element is an object that encompasses a set of versions, organized into a version tree. An element may have a lock if a version of the element is checked out in a view.

Class Hierarchy: VOBObject>File>Element

SubClasses of Element

Element has no subclasses.

Class Diagram



Attributes Specific to Element

Attributes	Inherited From	Description
Comment	VOBObject	The comment associated with the VOB object.

Attributes	Inherited From	Description
CreatedBy	VOBObject	The user who created the object.
CreatedOn	VOBObject	The date the object was created.
ElementType		The element type of this element.
ExtendedPath	File	The VOB-extended pathname of this file system object.
Extension	File	The file extension (the portion after the final dot).
Group		The group to which this element belongs.
IsDirectory	File	Whether or not the file is a directory.
LockDescription		A comment associated with the history record for the lock.
LockedBy		The use who locked this element.
LockedOn		The date the element was locked.
Master		The master replica for this element.
Name	VOBObject	The name of the versioned object.
NameMinusEx-tension	File	The simple name of the file without the extension and final.dot.
OID	VOBObject	The object identifier for the VOB object.
Owner		The owner of the element.
Path	File	The pathname to this file system object.
Selector	VOBObject	An expression used by ClearCase's file typing mechanism to match a file system object (or just the name of one).
SimpleName	File	The simple name of the file, i.e., the name of the file without the path.

Attributes	Inherited From	Description
State		The current state of the lock on this element.
TypeName	VOBObject	The VOBObject type name.
VOBFamilyUUID	VOBObject	The VOB family UUID for the VOB of this VOB object.

Relationships Specific to Element (see also class diagram above)

Name	Kind	Class	Documentation
Lock	0..1	Lock	The lock on this element.
ExemptUsers	0..n	Name	Array of string values containing the names of users exempted from the lock being created.
OwnerPermissions	0..n	Name	The owner permissions of the element (the owner has these permissions).
GroupPermissions	0..n	Name	The group permissions of the element (users within the same group have these permissions).
OtherPermissions	0..n	Name	The other permissions of the element (all users).
RootVersion	0..1	Version	The particular version of this element specified by the version selector.
CheckedOutFile	0..1	Version	The version of the element checked out to the associated view.

Name	Kind	Class	Documentation
AllChecked-OutFiles	0..n	Version	The versions of the element checked out to any view.
AllVersions	0..n	Version	Versions in the version tree for this element.
AllBranches	0..n	Branch	All branches in the version tree for this element.
ParentDirec-tory		Element	This element s parent directory element.
Triggers	0..n	Trigger	The collection of triggers attached to this file of directory element.

Class: File (ClearCase Adapter)

The File class represents all VOB objects, which are physical files such as elements and versions. A File object does not include view-private objects.

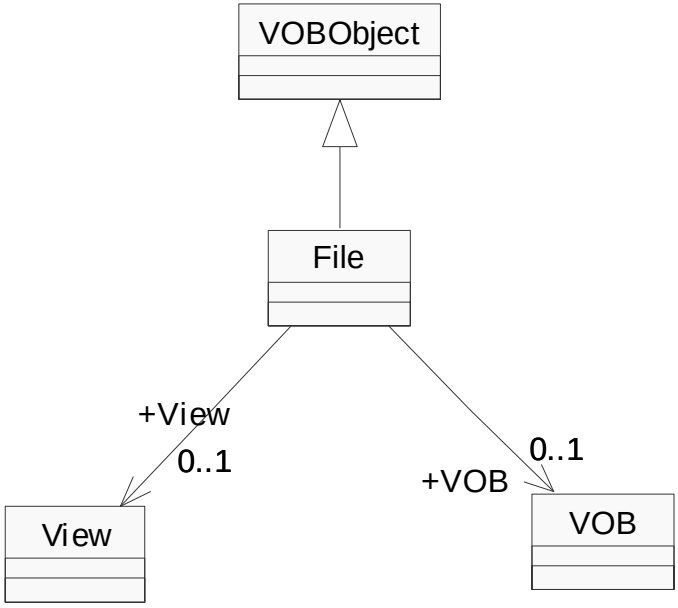
Subclasses of File Class: Element, Version.

Class Hierarchy: VOBObject>File

SubClasses of File

CheckedOutFile Element Version

Class Diagram



Attributes Specific to File

Attributes	Inherited From	Description
Comment	VOBObject	The comment associated with the VOB object.
CreatedBy	VOBObject	The user who created the object.
CreatedOn	VOBObject	The date the object was created.
ExtendedPath		The VOB-extended pathname of this file system object.
Extension		The file extension (the portion after the final dot).
IsDirectory		Whether or not the file is a directory.
Name	VOBObject	The name of the versioned object.
NameMinusExtension		The simple name of the file without the extension and final.dot.
OID	VOBObject	The object identifier for the VOB object.
Path		The pathname to this file system object.
Selector	VOBObject	An expression used by ClearCase's file typing mechanism to match a file system object (or just the name of one).
SimpleName		The simple name of the file, i.e., the name of the file without the path.
TypeName	VOBObject	The VOBObject type name.
VOBFamilyUUID	VOBObject	The VOB family UUID for the VOB of this VOB object.

Relationships Specific to File (see also class diagram above)

Name	Kind	Class	Documentation
VOB	0..1	VOB	The VOB associated with this file.
View	0..1	View	The view associated with this file.

Class: Folder (ClearCase Adapter)

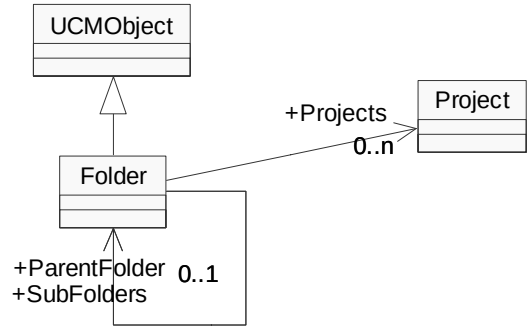
Folder is a ClearCase UCM object that contains one or more projects.

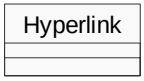
Class Hierarchy: UCMObject>Folder

SubClasses of Folder

Folder has no subclasses.

Class Diagram





Attributes Specific to Folder

Attributes	Inherited From	Description
Comment	VOBObject	The comment associated with the VOB object.
CreatedBy	VOBObject	The user who created the object.
CreatedOn	VOBObject	The date the object was created.

Attributes	Inherited From	Description
Group	UCMObject	The group to which the UCM object belongs.
IsRootFolder		TRUE if the folder is the root of the project hierarchy in its project VOB.
LockDescription	UCMObject	The comment of the user who locked this UCMObject.
LockedBy	UCMObject	The user who locked this UCMObject.
LockedOn	UCMObject	The date on which this UCMObject was locked.
Master	UCMObject	The master replica for the UCM object.
Name	VOBObject	The name of the versioned object.
OID	VOBObject	The object identifier for the VOB object.
Owner	UCMObject	The owner of the UCM object.
Selector	VOBObject	An expression used by ClearCase's file typing mechanism to match a file system object (or just the name of one).
State	UCMObject	The state of the lock on this UCMObject.
Title	UCMObject	The title of the UCM object.
TypeName	VOBObject	The VOBObject type name.
VOBFamilyUUID	VOBObject	The VOB family UUID for the VOB of this VOB object.

Relationships Specific to Folder (see also class diagram above)

Name	Kind	Class	Documentation
ParentFolder		Folder	The name of the parent folder.

Name	Kind	Class	Documentation
Projects	0..n	Project	The projects contained in the folder.
SubFolders		Folder	The folders contained within the folder.

Class: HistoryRecord (ClearCase Adapter)

A history record is meta-data in a VOB, representing an event record involving a VOB object. The history of a file element includes history records for creation of the element, creation of each version of the file, creation of each branch, assignment of attribute to the element and/or its versions, attaching of hyperlinks to the element and/or its versions, and so on.

Class Hierarchy: Artifact>HistoryRecord

SubClasses of HistoryRecord

HistoryRecord has no subclasses.

Class Diagram

Attributes Specific to HistoryRecord

Attributes	Inherited From	Description
Comment		The comment associated with the operation indicated by this history record.
Date		The date and time the operation was executed.

Attributes	Inherited From	Description
EventKind		Indicates the type of operation that was executed.
Group		The name of the login group that performed the operation indicated by this history record.
Host		The name of the host machine from which the operation indicated by this history record was executed.
Selector		An expression used by ClearCase's file typing mechanism to match a file system object (or just the name of one).
UserFullName		The full name of the user who performed the operation indicated by this history record.
UserLoginName		The login name of the user who performed the operation indicated by this history record.

Relationships Specific to HistoryRecord (see also class diagram above)

Name	Kind	Class	Documentation
VOB	0..1	VOB	The VOB containing the object to which the operation was applied.

Class: Hyperlink (ClearCase Adapter)

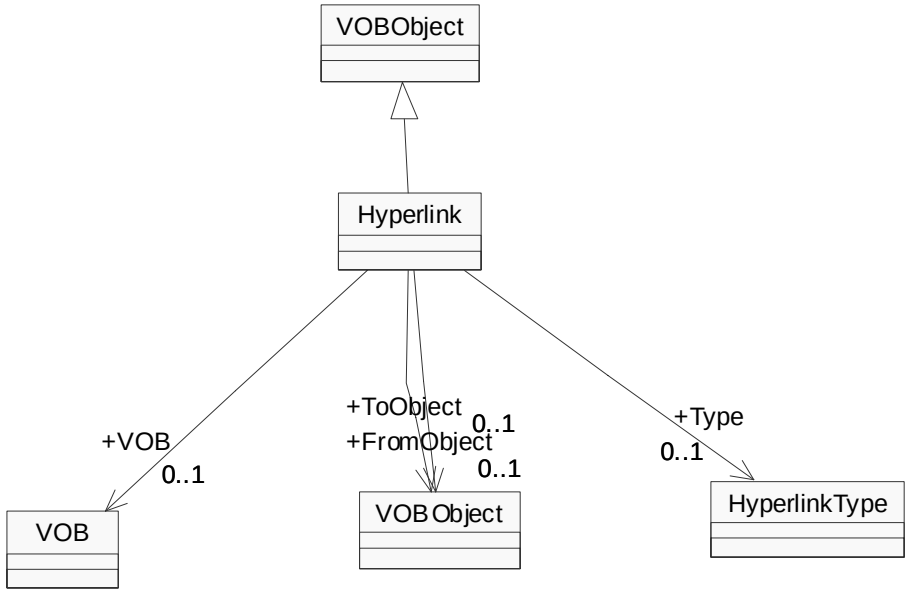
A hyperlink is a logical pointer between two objects. A hyperlink is a VOB object, it derives its name by referencing another VOB object, a hyperlink type. A hyperlink can have from text and to text, which are technically string-valued attributes on the hyperlink object. A hyperlink has a from-object and to-object, which are VOB objects. A hyperlink may be bi-directional, indicating that it can be traversed both from to-object to from-object and from-object to to-object. The IsUnidirectional selector will be False if a hyperlink is bi-directional.

Class Hierarchy: VOBObject>Hyperlink

SubClasses of Hyperlink

Hyperlink has no subclasses.

Class Diagram



Attributes Specific to Hyperlink

Attributes	Inherited From	Description
Comment	VOBObject	The comment associated with the VOB object.
CreatedBy	VOBObject	The user who created the object.
CreatedOn	VOBObject	The date the object was created.
FromText		The from-text on the from-object of the hyperlink.
Group		The group to which this hyperlink belongs.
IDString		The string identifying the hyperlink (type-name@id@vob-selector).
Master		The master replica for this hyperlink.
Name	VOBObject	The name of the versioned object.
OID	VOBObject	The object identifier for the VOB object.
Owner		The owner of this hyperlink.
Selector	VOBObject	An expression used by ClearCase's file typing mechanism to match a file system object (or just the name of one).
ToText		The to-text on the to-object of the hyperlink.
TypeName	VOBObject	The VOBObject type name.
Unidirectional		Whether of not the hyperlink object can be navigated only in one direction (from-object -> to-object).
VOBFamilyUUID	VOBObject	The VOB family UUID for the VOB of this VOB object.

Relationships Specific to Hyperlink (see also class diagram above)

Name	Kind	Class	Documentation
VOB	0..1	VOB	The VOB containing this hyperlink.
FromObject	0..1	VOBObject	The from-object of the hyperlink.
ToObject	0..1	VOBObject	The to-object of the hyperlink.
Type	0..1	Hyperlink-Type	The hyperlink type of this hyperlink.

Class: HyperlinkType (ClearCase Adapter)

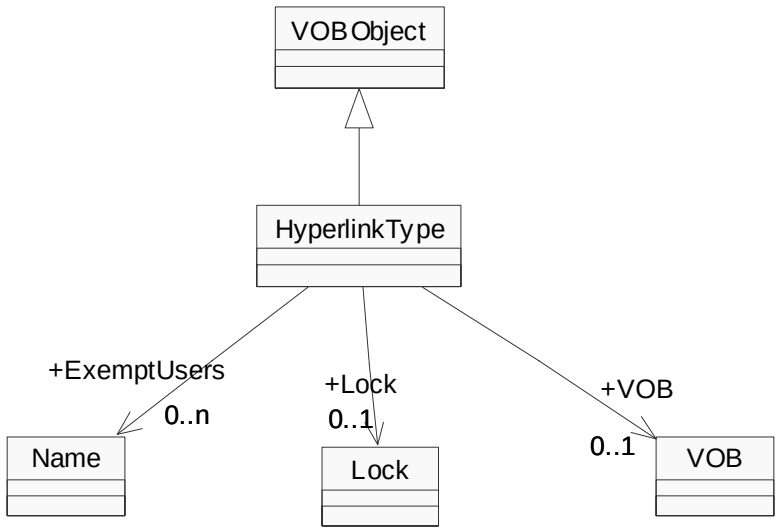
A HyperlinkType is an object that defines a hyperlink name for use within a VOB. A HyperlinkType may be shared or local.

Class Hierarchy: VOBObject>HyperlinkType

SubClasses of HyperlinkType

HyperlinkType has no subclasses.

Class Diagram



Attributes Specific to HyperlinkType

Attributes	Inherited From	Description
Comment	VOBObject	The comment associated with the VOB object.
CreatedBy	VOBObject	The user who created the object.

Attributes	Inherited From	Description
CreatedOn	VOBObject	The date the object was created.
Group		The group to which this hyperlink type belongs.
HasSharedMastership		Whether this hyperlink type is shared or can be mastered.
LockDescription		The comment of the user who locked this hyperlink type.
LockedBy		The name of the user who locked this hyperlink type.
LockedOn		The date on which this hyperlink type was locked.
Master		The master replica for this hyperlink type.
Name	VOBObject	The name of the versioned object.
OID	VOBObject	The object identifier for the VOB object.
Owner		The owner of this hyperlink type.
Scope		The scope of this hyperlink type (for example, local to this VOB).
Selector	VOBObject	An expression used by ClearCase's file typing mechanism to match a file system object (or just the name of one).
State		The state of the lock on this hyperlink type.
TypeName	VOBObject	The VOBObject type name.
VOBFamilyUUID	VOBObject	The VOB family UUID for the VOB of this VOB object.

Relationships Specific to HyperlinkType (see also class diagram above)

Name	Kind	Class	Documentation
Lock	0..1	Lock	The lock on this hyperlink type
ExemptUsers	0..n	Name	The list of users who are exempt from the lock.
VOB	0..1	VOB	The VOB containing this hyperlink type.

Class: Label (ClearCase Adapter)

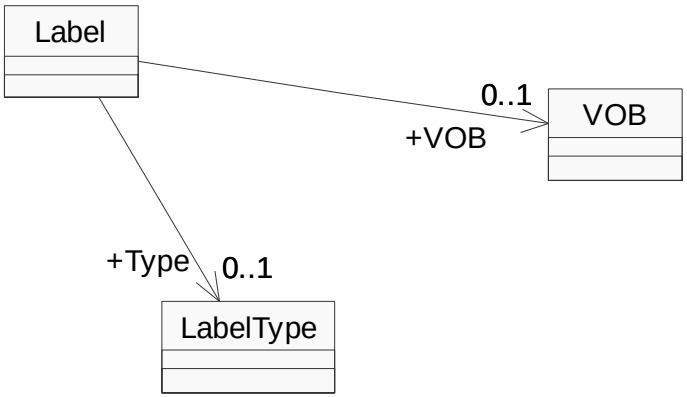
A label is an instance of a LabelType object, supplying a user-defined name for a version. One or more labels may be assigned to a given version.

Class Hierarchy: Artifact>Label

SubClasses of Label

Label has no subclasses.

Class Diagram



Attributes Specific to Label

Attributes	Inherited From	Description
TypeName		The label type name.

Relationships Specific to Label (see also class diagram above)

Name	Kind	Class	Documentation
VOB	0..1	VOB	The VOB containing the labeled version.
Type	0..1	LabelType	The label type of this label.

Class: LabelType (ClearCase Adapter)

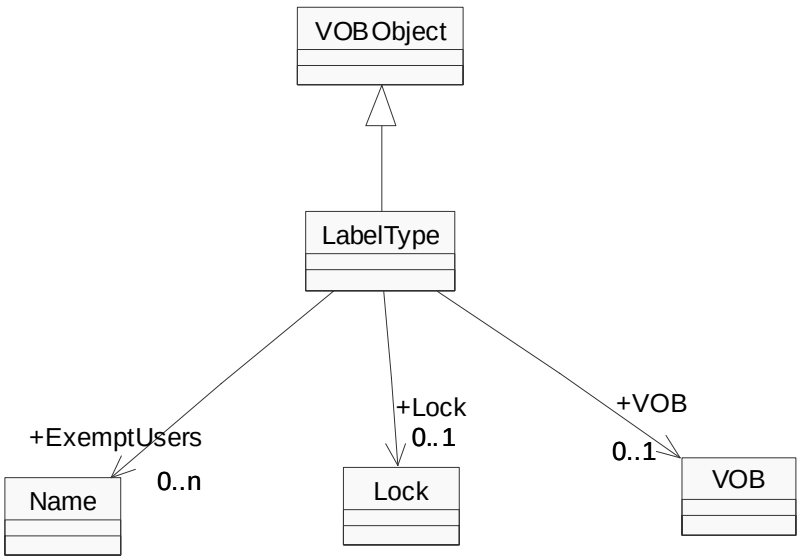
A label type is a type object that defines a version label for use within a VOB.

Class Hierarchy: VOBObject>LabelType

SubClasses of LabelType

LabelType has no subclasses.

Class Diagram



Attributes Specific to LabelType

Attributes	Inherited From	Description
Comment	VOBObject	The comment associated with the VOB object.
Constraint		The constraint for this label type.

Attributes	Inherited From	Description
CreatedBy	VOBObject	The user who created the object.
CreatedOn	VOBObject	The date the object was created.
Group		The group to which this label type belongs.
HasSharedMastership		Whether this label type is shared or can be mastered.
LockDescription		The comment of the user who locked this label type.
LockedBy		The name of the user who locked this label type.
LockedOn		The date on which this label type was locked.
Master		The master replica for this label type.
Name	VOBObject	The name of the versioned object.
OID	VOBObject	The object identifier for the VOB object.
Owner		The owner of this label type.
Scope		Whether this label type is global for VOBs using this as an admin VOB or local to this VOB.
Selector	VOBObject	An expression used by ClearCase's file typing mechanism to match a file system object (or just the name of one).
State		The state of the lock on this label type.
TypeName	VOBObject	The VOBObject type name.
VOBFamilyUUID	VOBObject	The VOB family UUID for the VOB of this VOB object.

Relationships Specific to LabelType (see also class diagram above)

Name	Kind	Class	Documentation
Lock	0..1	Lock	The lock on this label type.
ExemptUsers	0..n	Name	The list of users who are exempt from the lock on this label type.
VOB	0..1	VOB	The VOB containing this label type.

Class: Lock (ClearCase Adapter)

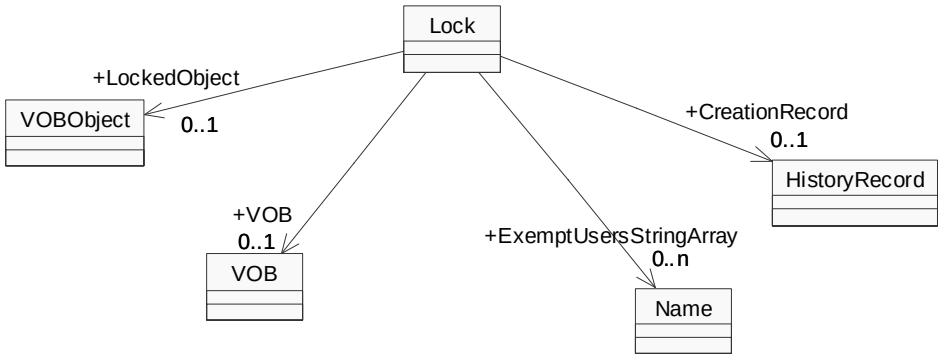
A lock is a mechanism that prevents a VOB object from being modified (for file system objects) or from being instanced (for type objects).

Class Hierarchy: Artifact>Lock

SubClasses of Lock

Lock has no subclasses.

Class Diagram



Attributes Specific to Lock

Attributes	Inherited From	Description
Comment		The user s comment for the lock.
CreatedBy		The name of the user who created the lock.
CreatedOn		The date the lock was created.
Index		An index to the lock.

Attributes	Inherited From	Description
IsObsolete		Whether the locked object is marked as obsolete.
NumberOfExempt-Users		The number of users who are exempt from this lock.
Selector		An expression used by ClearCase's file typing mechanism to match a file system object (or just the name of one).

Relationships Specific to Lock (see also class diagram above)

Name	Kind	Class	Documentation
LockedObject	0..1	VOBObject	The object held by this lock.
VOB	0..1	VOB	The VOB in which this lock resides.
Creation-Record	0..1	HistoryRecord	The creation record for this lock.
ExemptUs-ersStringAr-ray	0..n	Name	The users who are exempt from this lock.

Class: Name (ClearCase Adapter)

The Name class represents a string corresponding to the name or pathname of a ClearCase object.

Class Hierarchy: Artifact>Name

SubClasses of Name

Name has no subclasses.

Class Diagram

Attributes Specific to Name

Attributes	Inherited From	Description
Name		Simple name string.

Relationships Specific to Name (see also class diagram above)

This class has no relationships.

Class: Project (ClearCase Adapter)

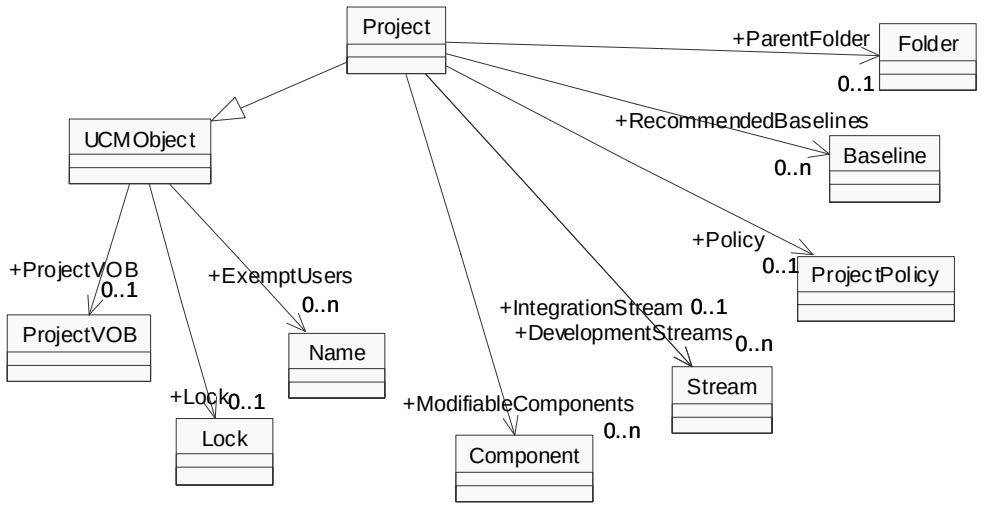
A Project defines a set of development policies and a set of configurations used in a development effort.

Class Hierarchy: UCMObject>Project

SubClasses of Project

Project has no subclasses.

Class Diagram



Attributes Specific to Project

Attributes	Inherited From	Description
ClearQuestData-baseName		The name of the ClearQuest database linked to the CRM-enabled project.

Attributes	Inherited From	Description
Comment	VOBObject	The comment associated with the VOB object.
CreatedBy	VOBObject	The user who created the object.
CreatedOn	VOBObject	The date the object was created.
Group	UCMObject	The group to which the UCM object belongs.
HasStreams		TRUE if there are any streams associated with the project.
IsCRMEnabled		TRUE if the project is CRM enabled (i.e., it it linked to a ClearQuest database.
LockDescription	UCMObject	The comment of the user who locked this UCMObject.
LockedBy	UCMObject	The user who locked this UCMObject.
LockedOn	UCMObject	The date on which this UCMObject was locked.
Master	UCMObject	The master replica for the UCM object.
Name	VOBObject	The name of the versioned object.
OID	VOBObject	The object identifier for the VOB object.
Owner	UCMObject	The owner of the UCM object.
RequiredPromotionLevel		The minimum promotion level a baseline must have in order to be a recommended baseline in a rebase operation.
Selector	VOBObject	An expression used by ClearCase's file typing mechanism to match a file system object (or just the name of one).

Attributes	Inherited From	Description
State	UCMObject	The state of the lock on this UCMObject.
Title	UCMObject	The title of the UCM object.
TypeName	VOBObject	The VOBObject type name.
VOBFamilyUUID	VOBObject	The VOB family UUID for the VOB of this VOB object.

Relationships Specific to Project (see also class diagram above)

Name	Kind	Class	Documentation
Development-Streams	0..n	Stream	The development streams of the project.
Integration-Stream	0..1	Stream	The integration stream for the project.
Modifiable-Components	0..n	Component	The set of components that can be modified by the project.
ParentFolder	0..1	Folder	The folder containing the project.
Policy	0..1	ProjectPolicy	The policy settings associated with the project.
Recommend-edBaselines	0..n	Baseline	The project s list of recommended baselines.
Streams	0..n	Stream	The streams for the project.

Class: ProjectPolicy (ClearCase Adapter)

A project's policies specifies how developers access and modify sets of source files and directories (called components). To record and configure the development work that proceeds on components, projects use the following objects:

Baseline

Stream

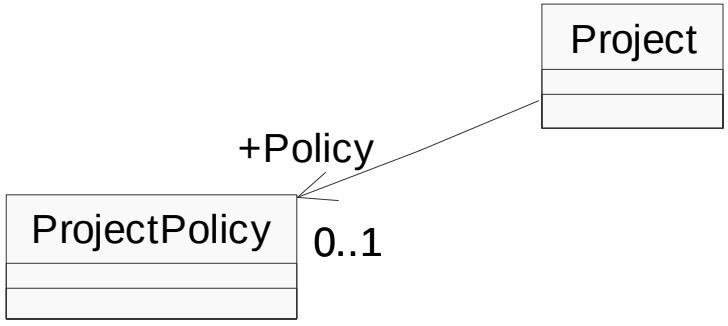
Activity

Class Hierarchy: Artifact>ProjectPolicy

SubClasses of ProjectPolicy

ProjectPolicy has no subclasses.

Class Diagram



Attributes Specific to ProjectPolicy

Attributes	Inherited From	Description
DeliverRequire-Checkin		TRUE if delivery is denied from a development stream that has checkouts.

Attributes	Inherited From	Description
DeliverRequireRebase		TRUE if development stream must be based on the current recommended baselines before it can be used to deliver changes to the integration stream.
UNIXDevelopmentSnapshot		Recommended snapshot views for development work on UNIX platforms.
UNIXIntegrationSnapshot		Recommended snapshot views for integration work on UNIX platforms.
WinDevelopmentSnapshot		Recommended snapshot views for development work on Window platforms.
WinIntegrationSnapshot		Recommended snapshot views for integration work on Window platforms.

Relationships Specific to ProjectPolicy (see also class diagram above)

This class has no relationships.

Class: ProjectVOB (ClearCase Adapter)

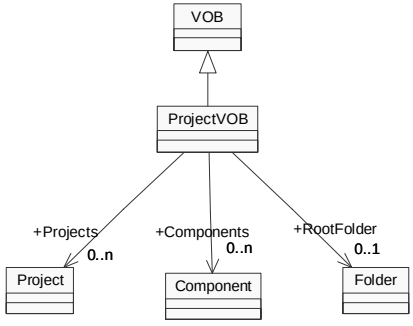
A special type of VOB used in the Unified Configuration Management (UCM) facilities of ClearCase. Contains some additional properties that a VOB does not.

Class Hierarchy: VOB>ProjectVOB

SubClasses of ProjectVOB

ProjectVOB has no subclasses.

Class Diagram



Attributes Specific to ProjectVOB

Attributes	Inherited From	Description
Comment	VOBObject	The comment associated with the VOB object.
CreatedBy	VOBObject	The user who created the object.
CreatedOn	VOBObject	The date the object was created.
DefaultPromotion-Level		The default promotion level in the project VOB.
Group	VOB	The group to which this VOB belongs.
HasMSDOSText-Mode	VOB	Whether or not this VOB has MSDOS text mode enabled.
Host	VOB	The host on which the storage area for this VOB resides.
IsMounted	VOB	Whether or not the VOB is mounted.
IsReplicated	VOB	Whether or not this VOB is replicated.
LockDescription	VOB	The description of the lock for the VOB.
LockedBy	VOB	The name of the user who locked the VOB.
LockedOn	VOB	The date the VOB was locked
Name	VOBObject	The name of the versioned object.
NumberOfAddi-tionalGroups	VOB	The number of additional groups to which this VOB belongs.
NumberOfPromo-tionLevels		The number of promotion levels in the project VOB.
NumberOfRepli-cas	VOB	The number of replica names for the VOB family of this VOB, if this VOB is replicated.

Attributes	Inherited From	Description
OID	VOBObject	The object identifier for the VOB object.
Owner	VOB	The owner of the VOB.
Selector	VOBObject	An expression used by ClearCase's file typing mechanism to match a file system object (or just the name of one).
State	VOB	The state of the lock on the VOB.
TagName	VOB	The VOB-tag name.
ThisReplica	VOB	The replica name for this VOB, if the VOB is replicated.
TypeName	VOBObject	The VOBObject type name.
VOBFamilyUUID	VOBObject	The VOB family UUID for the VOB of this VOB object.

Relationships Specific to ProjectVOB (see also class diagram above)

Name	Kind	Class	Documentation
Components	0..n	Component	The components in the project VOB.
Projects	0..n	Project	The projects in the project VOB.
RootFolder	0..1	Folder	The root folder in the project VOB.

Class: Region (ClearCase Adapter)

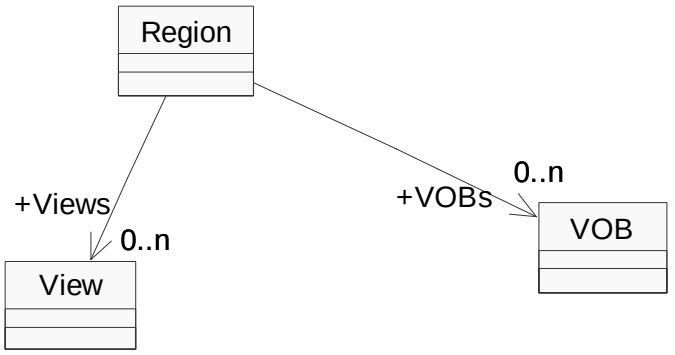
Region is a ClearCase file. A network region is a logical subset of a local area network, within which all hosts refer to VOB storage directories and view storage directories with the same network pathnames. The ClearCase domain supports retrieval of VOB's and Views within a region.

Class Hierarchy: Artifact>Region

SubClasses of Region

Region has no subclasses.

Class Diagram



Attributes Specific to Region

Attributes	Inherited From	Description
Region		The name of the region.

Relationships Specific to Region (see also class diagram above)

Name	Kind	Class	Documentation
VOBs	0..n	VOB	VOBs contained within the region.
Views	0..n	View	Views contained within the region.

Class: Stream (ClearCase Adapter)

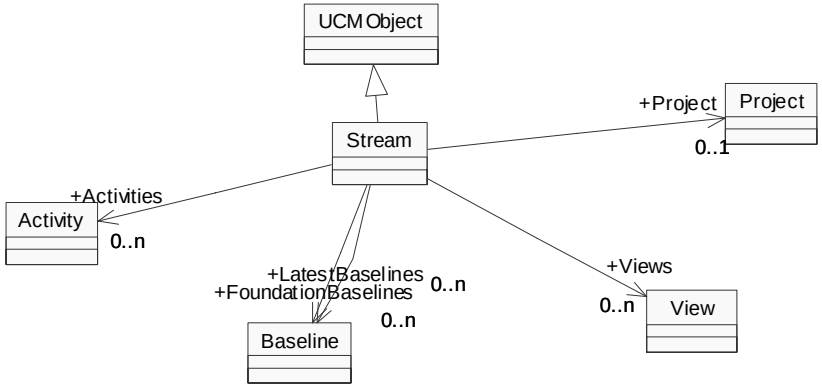
Stream is a mechanism for creating and recording configurations. A stream identifies the exact set of versions currently available for you to view, modify, or build.

Class Hierarchy: UCMObject>Stream

SubClasses of Stream

Stream has no subclasses.

Class Diagram



Attributes Specific to Stream

Attributes	Inherited From	Description
Comment	VOBObject	The comment associated with the VOB object.
CreatedBy	VOBObject	The user who created the object.
CreatedOn	VOBObject	The date the object was created.
Group	UCMObject	The group to which the UCM object belongs.

Attributes	Inherited From	Description
HasActivities		TRUE if there are any activities associated with the stream.
IsIntegration-Stream		TRUE if the stream is an integration stream in the project.
LockDescription	UCMObject	The comment of the user who locked this UCMObject.
LockedBy	UCMObject	The user who locked this UCMObject.
LockedOn	UCMObject	The date on which this UCMObject was locked.
Master	UCMObject	The master replica for the UCM object.
Name	VOBObject	The name of the versioned object.
OID	VOBObject	The object identifier for the VOB object.
Owner	UCMObject	The owner of the UCM object.
Selector	VOBObject	An expression used by ClearCase's file typing mechanism to match a file system object (or just the name of one).
State	UCMObject	The state of the lock on this UCMObject.
Title	UCMObject	The title of the UCM object.
TypeName	VOBObject	The VOBObject type name.
VOBFamilyUUID	VOBObject	The VOB family UUID for the VOB of this VOB object.

Relationships Specific to Stream (see also class diagram above)

Name	Kind	Class	Documentation
Activities	0..n	Activity	The activities associated with the Stream.
Foundation-Baselines	0..n	Baseline	The foundation baselines for the stream for all components.
LatestBase-lines	0..n	Baseline	The latestBaseline in the stream for all components.
Project	0..1	Project	The project for the stream.
Views	0..n	View	The set of views associated with the stream.

Class: Trigger (ClearCase Adapter)

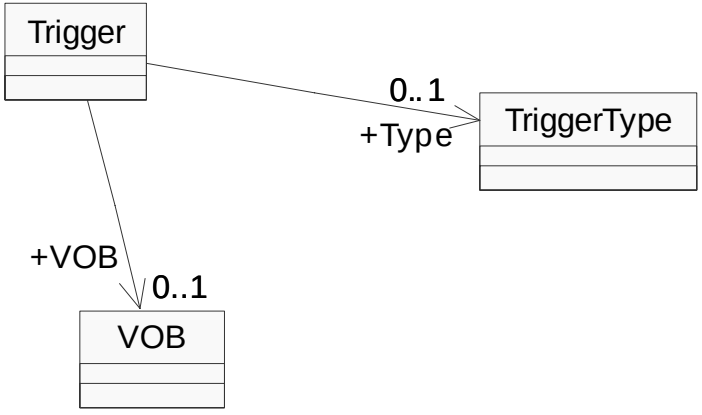
A trigger is a monitor that specifies one or more standard programs or built-in actions to be executed automatically whenever a certain ClearCase operation is performed. A trigger is associated with a TriggerType object, which groups triggers of similar properties.

Class Hierarchy: Artifact>Trigger

SubClasses of Trigger

Trigger has no subclasses.

Class Diagram



Attributes Specific to Trigger

Attributes	Inherited From	Description
IsOnAttachedList		Whether this trigger is on the attached list of the element.

Attributes	Inherited From	Description
IsOnInher-itenceList		Whether this trigger is on the inherit-ance list of an element, if the element is a directory element.
TypeName		The trigger type name.

Relationships Specific to Trigger (see also class diagram above)

Name	Kind	Class	Documentation
VOB	0..1	VOB	The VOB containing this ele-ment trigger.
Type	0..1	TriggerType	The trigger type of this element trigger.

Class: TriggerType (ClearCase Adapter)

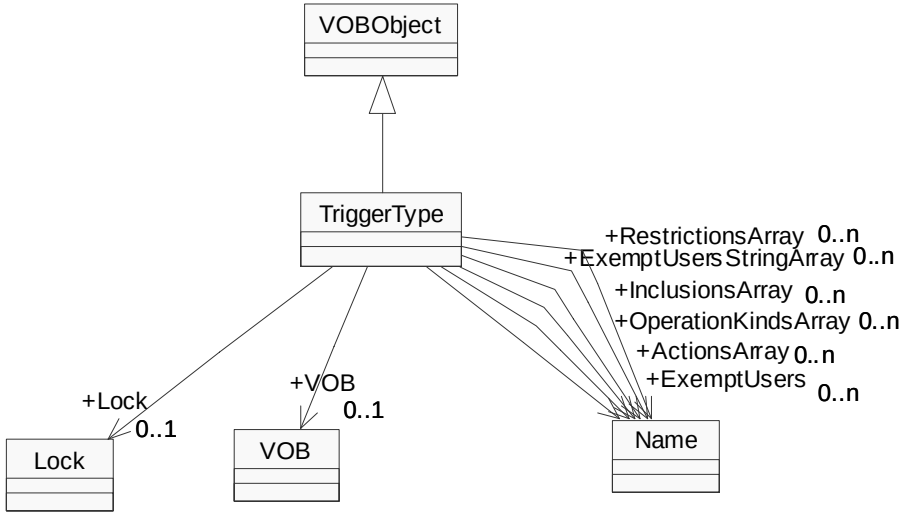
A trigger type is an object through which triggers are defined. The trigger kind for a trigger type includes element, all-element, and type. Instances of an element trigger type can be attached to one or more individual elements. An all-element trigger type is implicitly attached to all elements in a VOB. A type trigger type is attached to a specified collection of type object.

Class Hierarchy: VOBObject>TriggerType

SubClasses of TriggerType

TriggerType has no subclasses.

Class Diagram



Attributes Specific to TriggerType

Attributes	Inherited From	Description
Comment	VOBObject	The comment associated with the VOB object.
CreatedBy	VOBObject	The user who created the object.
CreatedOn	VOBObject	The date the object was created.
DebugPrinting		Whether or not debug printing happens when the trigger fires.
Firing		The trigger type firing order, before or after the operation (pre-op or post-op)
Group		The group to which this trigger type belongs.
KindOfTrigger		The kind of trigger for this trigger type.
LockDescription		The comment of the user who locked this trigger type.
LockedBy		The user who locked this trigger type.
LockedOn		The date on which this trigger type was locked.
Name	VOBObject	The name of the versioned object.
NumberOfActions		The number of actions for this trigger type.
NumberOfExemptUsers		The number of users for whom this trigger type does not fire.
NumberOfInclusions		The number of inclusions for this element trigger type.
NumberOfOperationKinds		The number of operation kinds which fire this trigger type.

Attributes	Inherited From	Description
NumberOfRestrictions		The number of restrictions for this trigger type
OID	VOBObject	The object identifier for the VOB object.
Owner		The owner of this trigger type.
Selector	VOBObject	An expression used by ClearCase's file typing mechanism to match a file system object (or just the name of one).
State		The state of the lock on this trigger type.
TypeName	VOBObject	The VOBObject type name.
VOBFamilyUUID	VOBObject	The VOB family UUID for the VOB of this VOB object.

Relationships Specific to TriggerType (see also class diagram above)

Name	Kind	Class	Documentation
Lock	0..1	Lock	The lock on this trigger type.
ExemptUsers	0..n	Name	The users exempted from the firing of triggers for this trigger type.
VOB	0..1	VOB	The VOB containing this trigger type.
ExemptUsersStringArray	0..n	Name	An array of users who are exempt from this trigger type.
RestrictionsArray	0..n	Name	The restriction list for this element trigger type.

Name	Kind	Class	Documentation
ActionsArray	0..n	Name	An array of action/value pairs for this trigger type (that is, a type followed by one or two values).
OperationKindsArray	0..n	Name	An array of kinds of operations which fire this trigger type.
InclusionsArray	0..n	Name	The inclusion list for this trigger type.

Class: UCMObject (ClearCase Adapter)

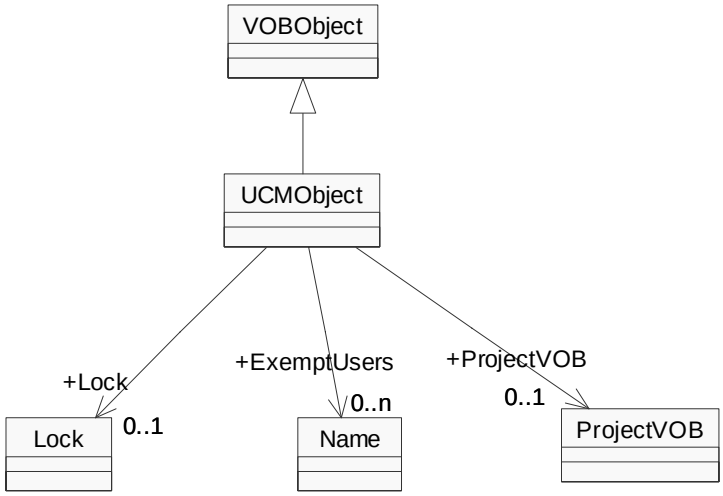
The UCMObject class is the class from which all UCM objects are based. For historical reasons, the Activity class is based on VOBObject instead.

Class Hierarchy: VOBObject>UCMObject

SubClasses of UCMObject

Baseline Component Folder Project Stream

Class Diagram



Attributes Specific to UCMObject

Attributes	Inherited From	Description
Comment	VOBObject	The comment associated with the VOB object.
CreatedBy	VOBObject	The user who created the object.

Attributes	Inherited From	Description
CreatedOn	VOBObject	The date the object was created.
Group		The group to which the UCM object belongs.
LockDescription		The comment of the user who locked this UCMObject.
LockedBy		The user who locked this UCMObject.
LockedOn		The date on which this UCMObject was locked.
Master		The master replica for the UCM object.
Name	VOBObject	The name of the versioned object.
OID	VOBObject	The object identifier for the VOB object.
Owner		The owner of the UCM object.
Selector	VOBObject	An expression used by ClearCase's file typing mechanism to match a file system object (or just the name of one).
State		The state of the lock on this UCMObject.
Title		The title of the UCM object.
TypeName	VOBObject	The VOBObject type name.
VOBFamilyUUID	VOBObject	The VOB family UUID for the VOB of this VOB object.

Relationships Specific to UCMObject (see also class diagram above)

Name	Kind	Class	Documentation
Lock	0..1	Lock	The lock for the UCM object.

Name	Kind	Class	Documentation
ExemptUsers	0..n	Name	The users exempted from this UCMObject.
ProjectVOB	0..1	ProjectVOB	The project VOB for the UCM object.

Class: Value (ClearCase Adapter)

The value class represents a string value occurring within a collection of values.

Class Hierarchy: Artifact>Value

SubClasses of Value

Value has no subclasses.

Class Diagram

Attributes Specific to Value

Attributes	Inherited From	Description
Value		Simple value string.

Relationships Specific to Value (see also class diagram above)

This class has no relationships.

Class: Version (ClearCase Adapter)

A version is an object that implements a particular revision of an element. The versions of an element are organized into a version tree structure. Also, a checked-out version can refer to the view-private file that corresponds to the object created in a VOB database by the checkout command. If a version is a directory, it may contain subversions corresponding to those versions within the directory.

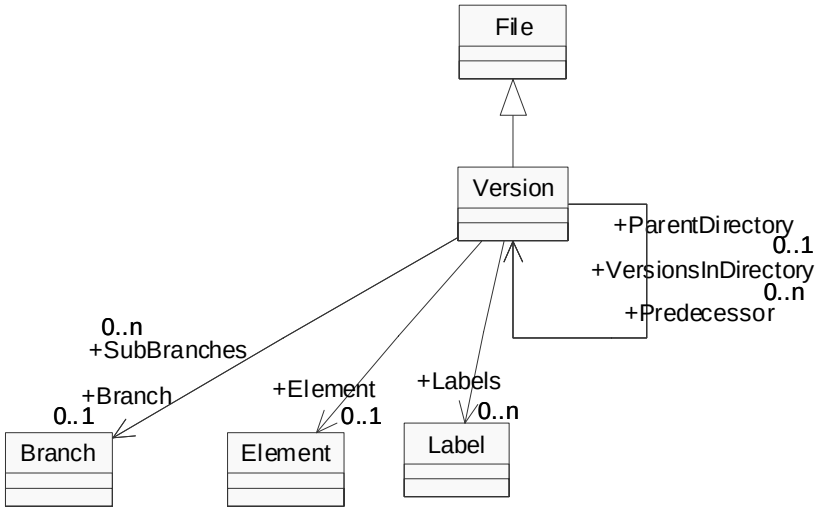
Class Hierarchy: VOBObject>File>Version

Subclasses of Version Class: CheckedOutFile.

SubClasses of Version

CheckedOutFile

Class Diagram



Attributes Specific to Version

Attributes	Inherited From	Description
Comment	VOBObject	The comment associated with the VOB object.
CreatedBy	VOBObject	The user who created the object.
CreatedOn	VOBObject	The date the object was created.
ExtendedPath	File	The VOB-extended pathname of this file system object.
Extension	File	The file extension (the portion after the final dot).
Identifier		The version s identifier string.
IsCheckedOut		Whether or not this object represents a checked-out file.
IsDifferent		Whether or not this version is different from its predecessor.
IsDirectory	File	Whether or not the file is a directory.
IsHijacked		Whether or not this version is hijacked.
IsLatest		Whether or not this version is the latest on its branch.
Name	VOBObject	The name of the versioned object.
NameMinusExtension	File	The simple name of the file without the extension and final.dot.
OID	VOBObject	The object identifier for the VOB object.
Path	File	The pathname to this file system object.
Selector	VOBObject	An expression used by ClearCase's file typing mechanism to match a file system object (or just the name of one).

Attributes	Inherited From	Description
SimpleName	File	The simple name of the file, i.e., the name of the file without the path.
TypeName	VOBObject	The VOBObject type name.
VersionNumber		This version s version number.
VOBFamilyUUID	VOBObject	The VOB family UUID for the VOB of this VOB object.

Relationships Specific to Version (see also class diagram above)

Name	Kind	Class	Documentation
Predecessor		Version	This version s predecessor version.
Element	0..1	Element	This version s element.
Branch	0..1	Branch	The branch for this version
SubBranches	0..n	Branch	Any branches sprouting from this version.
Labels	0..n	Label	The collection of labels associated with this version.
ParentDirectory		Version	The current view s version of this version s parent directory.
VersionsInDirectory		Version	Represents the file and directory versions contained in this (directory) version.

Class: View (ClearCase Adapter)

A View is a ClearCase object that provides a work area for one or more users. Users in different views can work with the same files without interfering with each other. For each element in a VOB, a view's configspec selects one version from the element's version tree, which is visible within the view. Each view can also store view-private files and view-private directories, which do not appear in other views. View-private objects and directories are not represented by any class within the ClearCase domain, however they may be documented through the File System domain. The ClearCase domain enables you to identify snapshot and dynamic views, as well as views that build non-shareable derived objects

Class Hierarchy: Artifact>View

SubClasses of View

View has no subclasses.

Class Diagram

Attributes Specific to View

Attributes	Inherited From	Description
BuildsShareable-DOs		Whether or not this view builds non-shareable derived objects.
ConfigSpec		The configuration spec for this view.
DisplayableCon-figSpec		A displayable form of the config spec for this view.
Host		The host on which the storage area for this view resides.
IsActive		Whether or not the view is started on the local machine.
IsSnapShot		Whether or not this view is a snapshot view.
TagName		The view-tag name.

Relationships Specific to View (see also class diagram above)

This class has no relationships.

Class: VOB (ClearCase Adapter)

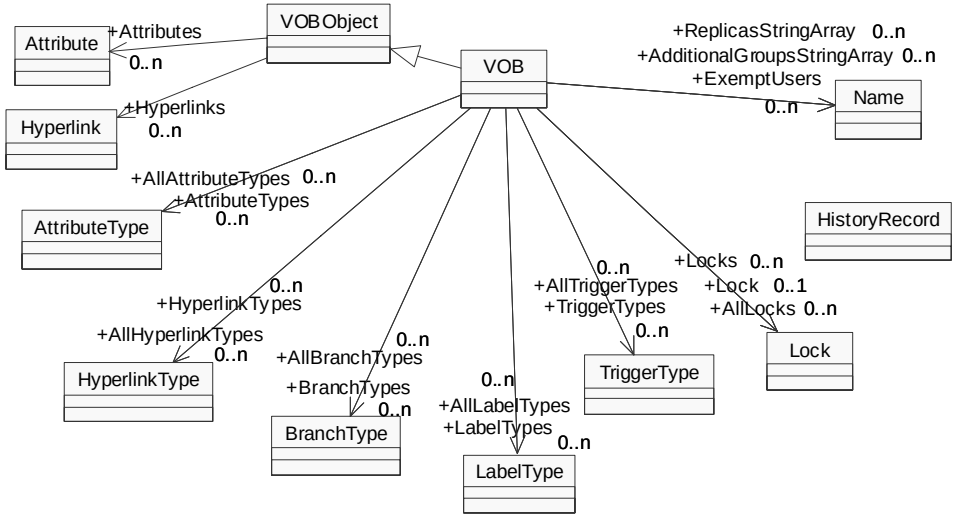
A VOB is the database that stores your project's files. A VOB, or versioned object base, is a repository that stores versions for file elements, directory elements, derived objects, and meta-data associated with these objects. SoDA supports MultiSite by enabling retrieval of a list of replicas (by name) for a given VOB. A SoDA template can include an OPEN command for a VOB, which must identify the VOB by full pathname, VOB-tag, or VOB family UUID.

Class Hierarchy: VOBObject>VOB

SubClasses of VOB

ProjectVOB

Class Diagram



Attributes Specific to VOB

Attributes	Inherited From	Description
Comment	VOBObject	The comment associated with the VOB object.
CreatedBy	VOBObject	The user who created the object.
CreatedOn	VOBObject	The date the object was created.
Group		The group to which this VOB belongs.
HasMSDOSText-Mode		Whether or not this VOB has MSDOS text mode enabled.
Host		The host on which the storage area for this VOB resides.
IsMounted		Whether or not the VOB is mounted.
IsReplicated		Whether or not this VOB is replicated.
LockDescription		The description of the lock for the VOB.
LockedBy		The name of the user who locked the VOB.
LockedOn		The date the VOB was locked
Name	VOBObject	The name of the versioned object.
NumberOfAddi-tionalGroups		The number of additional groups to which this VOB belongs.
NumberOfRepli-cas		The number of replica names for the VOB family of this VOB, if this VOB is replicated.
OID	VOBObject	The object identifier for the VOB object.
Owner		The owner of the VOB.

Attributes	Inherited From	Description
Selector	VOBObject	An expression used by ClearCase's file typing mechanism to match a file system object (or just the name of one).
State		The state of the lock on the VOB.
TagName		The VOB-tag name.
ThisReplica		The replica name for this VOB, if the VOB is replicated.
TypeName	VOBObject	The VOBObject type name.
VOBFamilyUUID	VOBObject	The VOB family UUID for the VOB of this VOB object.

Relationships Specific to VOB (see also class diagram above)

Name	Kind	Class	Documentation
Lock	0..1	Lock	The lock on this VOB, if there is one.
ExemptUsers	0..n	Name	The list of users exempted from the lock on the VOB.
Hyperlink-Types	0..n	Hyperlink-Type	All existing hyperlink types in the VOB.
AllHyperlink-Types	0..n	Hyperlink-Type	All existing hyperlink types in the VOB, including obsolete types.
TriggerTypes	0..n	TriggerType	All existing trigger types in the VOB.
AllTrigger-Types	0..n	TriggerType	All existing trigger types in the VOB, including obsolete types.

Name	Kind	Class	Documentation
AttributeTypes	0..n	AttributeType	All existing attribute types in the VOB.
AllAttributeTypes	0..n	AttributeType	All existing attribute types in the VOB, including obsolete types.
BranchTypes	0..n	BranchType	All existing branch types in the VOB.
AllBranchTypes	0..n	BranchType	All existing branch types in the VOB, including obsolete types.
LabelTypes	0..n	LabelType	All existing label types in the VOB.
AllLabelTypes	0..n	LabelType	All existing label types in the VOB, including obsolete types.
Locks	0..n	Lock	An enumeration of all the locks in this VOB.
AllLocks	0..n	Lock	An enumeration of all the locks in this VOB, including obsolete locks.
ReplicasStringArray	0..n	Name	The array of replica names for the VOB family of this VOB, if this VOB is replicated.
AdditionalGroupsStringArray	0..n	Name	Additional groups to which this VOB belongs.

Class: VOBObject (ClearCase Adapter)

A VOB Object represents an object stored in a VOB, including elements, versions, types, hyperlinks, branches, activities, etc. VOBObject is the base class from which all other VOB object classes derive.

Class Hierarchy: Artifact>VOBObject

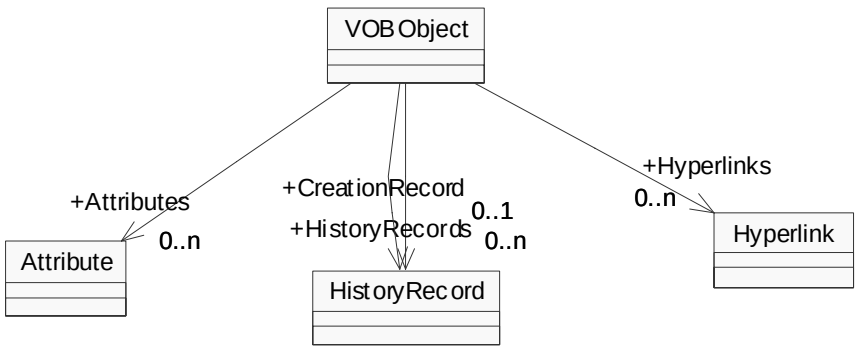
Subclasses of VOBObject Class:

Activity, AttributeType, Branch, BranchType, File, Hyperlink, HyperlinkType, LabelType, TriggerType, VOB, UCMObject.

SubClasses of VOBObject

Activity AttributeType Branch BranchType File Hyperlink HyperlinkType LabelType TriggerType UCMObject VOB

Class Diagram



Attributes Specific to VOBObject

Attributes	Inherited From	Description
Comment		The comment associated with the VOB object.
CreatedBy		The user who created the object.
CreatedOn		The date the object was created.
Name		The name of the versioned object.
OID		The object identifier for the VOB object.
Selector		An expression used by ClearCase's file typing mechanism to match a file system object (or just the name of one).
TypeName		The VOBObject type name.
VOBFamilyUUID		The VOB family UUID for the VOB of this VOB object.

Relationships Specific to VOBObject (see also class diagram above)

Name	Kind	Class	Documentation
Attributes	0..n	Attribute	The collection of attributes associated with this VOB object
Hyperlinks	0..n	Hyperlink	The collection of hyperlinks associated with this VOB object
Creation-Record	0..1	HistoryRecord	The creation record for the VOB object
Histo-ryRecords	0..n	HistoryRecord	The collection of history records for this object

RSE Adapter: ClearQuest

The ClearQuest domain lets you incorporate information from your ClearQuest database into your SoDA documents. This information can come from defects, histories, attachments, and so on.

The ClearQuest integration is very database-specific. It is critical that you supply the database name in your OPEN commands immediately. This will trigger SoDA to retrieve the database-specific classes, attributes, and relationships from ClearQuest.

Regarding Queries

The ClearQuest domain has the ability to use all public queries created in ClearQuest, as well as any personal queries to which the current user has access. The queries appear as Repeat selectors in Template View or Add Command. They are created with the dynamic domain.

ClearQuest allows you to create a query that launches a dialog box to prompt the user for input. It then performs the query using the input as a filter. For example, in a query for a weekly report of new defects, the user could be prompted for a Submit_Date in order to specify all defects submitted since the previous Saturday. When this type of query is used from SoDA, the input dialog is not launched and SoDA ignores that filter. Any other non-input filters defined by the query are still used.

Filtering Query Results

SoDA enables you to specify filter criteria for the results of a REPEAT command by clicking on the "Advanced" check box in the REPEAT Command dialog box. For a ClearQuest template, this criteria is passed to the ClearQuest domain during generation where an SQL query is built corresponding to the specified criteria.

Using the ClearQuest query engine greatly improves performance during document or report generation. There are situations where ClearQuest cannot build a query corresponding to a filter from SoDA. In these cases, filtering of the results of the REPEAT command takes place inside SoDA. Since this has a negative effect on performance, it is important to note when these scenarios might occur:

- If you are REPEATing over the results of a ClearQuest query. If a query returns too many rows (for example, more than a few hundred), performance degrades.
- If your Where expression contains the "IS" operator (which is not supported by ClearQuest).
- If you have an expression where both the left- and right-hand sides reference a unary relationship; for example, owner.login_name = submitter.login_name. ClearQuest only supports literals in the right-hand side of an expression.

For maximum performance, you should design your ClearQuest template to avoid these situations.

The following classes are available through the ClearQuest adapter:

Attachments

CQDatabase

Groups

History

Query

Record

Users

Class: Attachments (ClearQuest Adapter)

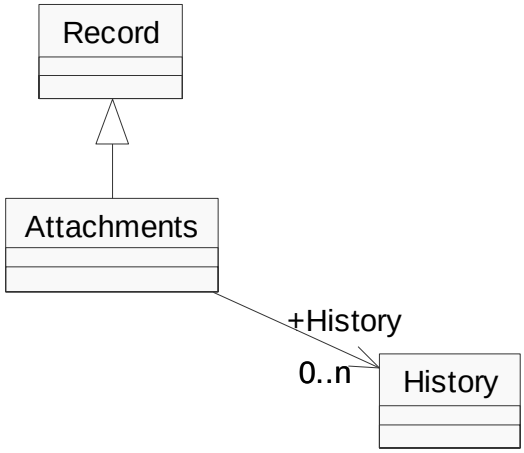
An attachment is a file associated with a particular record in the database.

Class Hierarchy: Artifact>Record>Attachments

SubClasses of Attachments

Attachments has no subclasses.

Class Diagram



Attributes Specific to Attachments

Attributes	Inherited From	Description
Dbid	Record	The internal database ID for the record.
Description		The description of the attachment

Attributes	Inherited From	Description
Entity_dbid		The database ID for the record that owns (has) the attachment. This identifies the database record (entity) that the attachment record belongs to. An Entity object represents a record in the database.
Entity_fielddef_id		The ID of the field definition (fieldInfo) for a field in the record (entity) that references the attachment. This ID identifies the attachment.
Filename		The name of the attachment
Filesize		The size of the attachment
FullPath		The location of the attachment
Id	Record	The record ID
Is_active	Record	True if the record is active
Lock_version	Record	The version of the locking mechanism
Locked_by	Record	The user who locked the record
Record_type	Record	The type of record.
Version	Record	The version of the record.

Relationships Specific to Attachments (see also class diagram above)

Name	Kind	Class	Documentation
History	0..n	History	The history records for this attachment

Class: CQDatabase (ClearQuest Adapter)

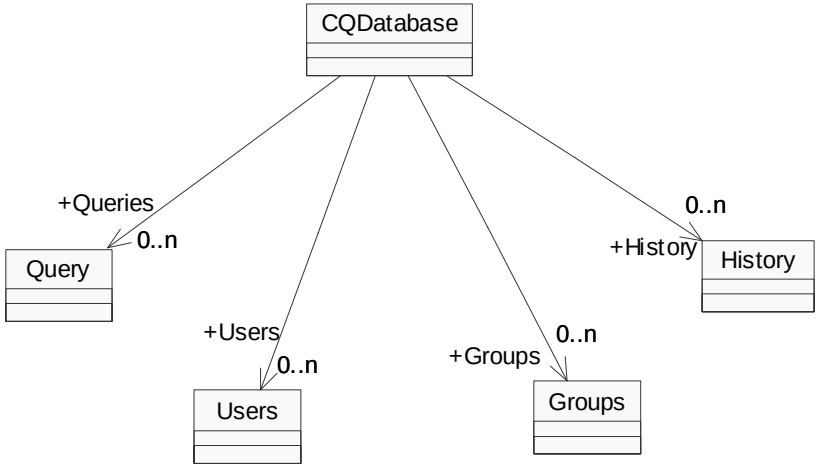
A CQDatabase contains all user data and a copy of the associated schema.

Class Hierarchy: Artifact>CQDatabase

SubClasses of CQDatabase

CQDatabase has no subclasses.

Class Diagram



Attributes Specific to CQDatabase

Attributes	Inherited From	Description
DatabaseSet		DatabaseSet is the name given to a master database or schema. The DatabaseSet name is chosen by the individual user when he or she defines the DatabaseSet.

Attributes	Inherited From	Description
Description		The description of the database

Attributes	Inherited From	Description
MinimizeSpace		MinimizeSpace effects the performance of the adapter by changing how queries are invoked. By default, MinimizeSpace is FALSE. When it is FALSE, the adapter requests that queries return all simple fields of each result record (this excludes multi-line fields such as Description and fields that contain lists). The advantage of using false is that it may be possible to get all the required fields from the query and thereby avoid expensive calls to ClearQuest that retrieve each record in its entirety in order to get simple fields that are not included in the query. However, more space is required for each result record to store these additional fields. If you use an artifact collection iterator to process the result record collection and you discard each record as processed, the extra space required for these fields is of no consequence. If MinimizeSpace is TRUE, the adapter only requests the result fields that are already defined by queries. This may save considerable memory space when it is known that the queries retrieve all of the required fields and that the result records will not be discarded as they are retrieved. In this case, substantial time may be saved because less data is transferred. If you do not know whether all required fields are returned by queries, it is usually best to use the default value of false for MinimizeSpace, otherwise TRUE may be used to get better performance. Retrieval of multi-line fields is expensive in either case.

Attributes	Inherited From	Description
Name		The logical name of the database. The Database Name is the name of a collection of CQ records that conform to the master database or schema.
QualifiedName		The qualified name of the database. The qualified name consists of DatabaseSet and the database Name. <DatabaseSet>:<DatabaseName> For example, CMBU:LABST
SessionType		The database session type.

Relationships Specific to CQDatabase (see also class diagram above)

Name	Kind	Class	Documentation
Queries	0..n	Query	All queries stored in the database
Users	0..n	Users	All users stored in the database
Groups	0..n	Groups	All groups stored in the database
History	0..n	History	The history records for this database

Class: Groups (ClearQuest Adapter)

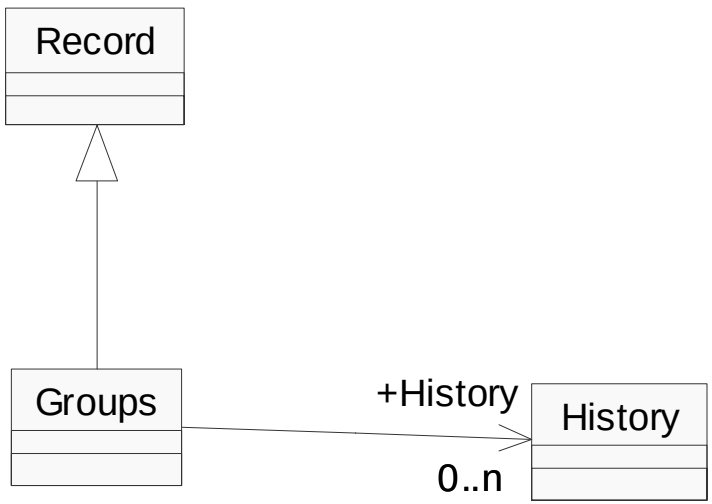
A group is a list of users with similar privileges.

Class Hierarchy: Record>Groups

SubClasses of Groups

Groups has no subclasses.

Class Diagram



Attributes Specific to Groups

Attributes	Inherited From	Description
Dbid	Record	The internal database ID for the record.

Attributes	Inherited From	Description
Id	Record	The record ID
Is_active	Record	True if the record is active
Lock_version	Record	The version of the locking mechanism
Locked_by	Record	The user who locked the record
Master_dbid		The master database ID for the Groups object. A master database is a schema repository for one or more user databases.
Name		The name of the group collection.
Record_type	Record	The type of record.
Version	Record	The version of the record.

Relationships Specific to Groups (see also class diagram above)

Name	Kind	Class	Documentation
History	0..n	History	The history records of this group

Class: History (ClearQuest Adapter)

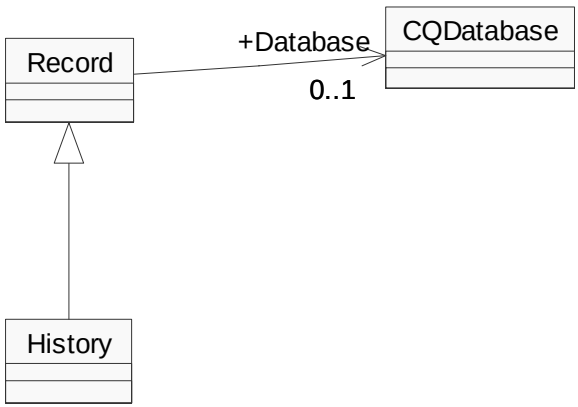
History records all changes made to the records in the database.

Class Hierarchy: Record>History

SubClasses of History

History has no subclasses.

Class Diagram



Attributes Specific to History

Attributes	Inherited From	Description
Action_name		The action that was entered
Action_timestamp		The time the action was entered

Attributes	Inherited From	Description
Comments		Any comments associated with the event
Dbid	Record	The internal database ID for the record.
Entity_dbid		The database ID for the record that owns (has) the history. This identifies the database record (entity) that the History record belongs to. An Entity object represents a record in the database.
Entitydef_id		The database ID of the record type.
Entitydef_name		The name of the record type.
Expired_timestamp		The time the history expires
Id	Record	The record ID
Is_active	Record	True if the record is active
Lock_version	Record	The version of the locking mechanism
Locked_by	Record	The user who locked the record
New_state		The state of the record following the action
Old_state		The state of the record prior to the action
Record_type	Record	The type of record.
User_name		The user who triggered the action
Version	Record	The version of the record.

Relationships Specific to History (see also class diagram above)

This class has no relationships.

Class: Query (ClearQuest Adapter)

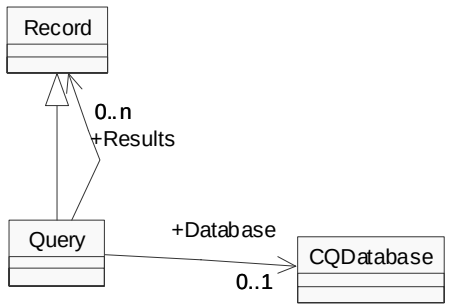
A query is used to retrieve specific records from a database.

Class Hierarchy: Artifact>Record>Query

SubClasses of Query

Query has no subclasses.

Class Diagram



Attributes Specific to Query

Attributes	Inherited From	Description
Dbid	Record	The internal database ID for the record.
Id	Record	The record ID
Is_active	Record	True if the record is active
IsAggregated		Returns a Boolean indicating whether any fields of the query are aggregated.
IsDirty		Returns a Boolean indicating whether the query has changed.

Attributes	Inherited From	Description
IsFamilyQuery		Returns true if the Query defines a family.
Lock_version	Record	The version of the locking mechanism
Locked_by	Record	The user who locked the record
Name		The name of the Query.
QueryType		The type of query. An integer indicating list, report, or chart.
Record_type	Record	The type of record.
ResultType		The result type of the query
SQL		The SQL string associated with the query.
Version	Record	The version of the record.

Relationships Specific to Query (see also class diagram above)

Name	Kind	Class	Documentation
Database	0..1	CQDatabase	The CQDatabase that the Query is associated with.
Results	0..n	Record	The records associated with a Query

Class: Record (ClearQuest Adapter)

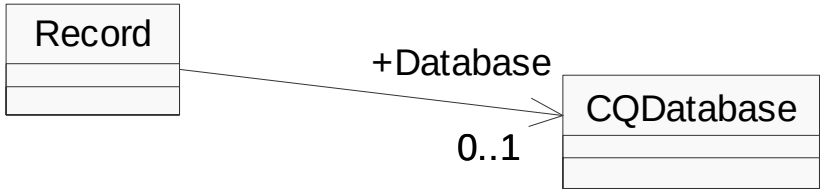
Records represent the data records the user creates, modifies, and views.

Class Hierarchy: Artifact>Record

SubClasses of Record

Attachments Groups History Query Users

Class Diagram



Attributes Specific to Record

Attributes	Inherited From	Description
Dbid		The internal database ID for the record.
Id		The record ID
Is_active		True if the record is active
Lock_version		The version of the locking mechanism

Attributes	Inherited From	Description
Locked_by		The user who locked the record
Record_type		The type of record.
Version		The version of the record.

Relationships Specific to Record (see also class diagram above)

Name	Kind	Class	Documentation
Database	0..1	CQDatabase	The CQDatabase which this record is associated with.

Class: Users (ClearQuest Adapter)

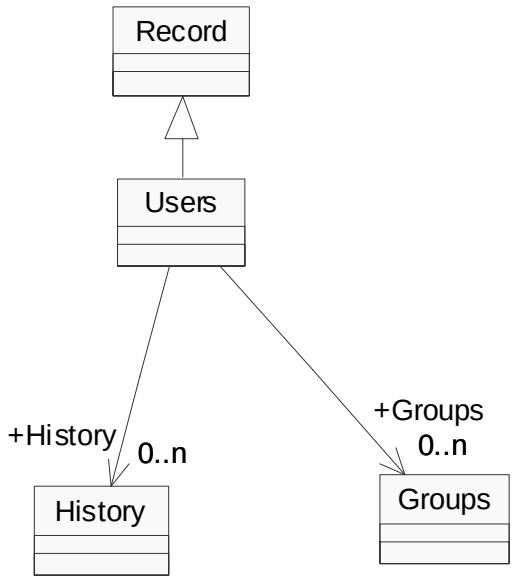
A user is someone who can log in to the ClearQuest database.

Class Hierarchy: Record>Users

SubClasses of Users

Users has no subclasses.

Class Diagram



Attributes Specific to Users

Attributes	Inherited From	Description
Dbid	Record	The internal database ID for the record.

Attributes	Inherited From	Description
Email		The email address of the user
Encrypted_password		The password of the user
Fullname		The full name of the User.
Id	Record	The record ID
Is_active	Record	True if the record is active
Is_appbuilder		True if the user has AppBuilder privileges
Is_superuser		True if the user is a superuser
Is_user_maint		True if the user has maintenance privileges
Lock_version	Record	The version of the locking mechanism
Locked_by	Record	The user who locked the record
Login_name		The login name for the User.
Master_dbid		The master database ID. A master database is a schema repository for one or more user databases.
Misc_info		Miscellaneous information
Phone		The phone number of the user
Record_type	Record	The type of record.
Version	Record	The version of the record.

Relationships Specific to Users (see also class diagram above)

Name	Kind	Class	Documentation
History	0..n	History	The history records for this user record
Groups	0..n	Groups	The groups which this user is a member

RSE Adapter: FileSys

The File System (FileSys) domain allows you to incorporate information from your file system into your SoDA documents. This information can come from directories, files, or records within files.

Following is some special information concerning the FileSystem FileRecord Class.

Directives

You can add directives at the beginning of the file to change the defaults. The directives are:

#RECORD_DELIMITER, which specifies a character other than a newline to delimit records within the file;
#FIELD_DELIMITER, which specifies a character other than a space to delimit fields within records;
#QUOTE_DELIMITER, which specifies a character other than a double quote to delimit a single field that may include the field delimiter character; and
#KEY_FIELDS, which specifies a list of field numbers, separated by spaces, used to uniquely identify each record.

You can use the following special characters in these directives:

\n, for newline
\t, for horizontal tab
\b, for backspace
\r, for carriage return
\f, for formfeed
\\, for backslash

For example:

```
#FIELD_DELIMITER /
#KEY_FIELDS 2
William(Bill)/Clinton/Democrat/1993/1996
George/Bush/Republican/1989/1992
Ronald/Reagan/Republican/1981/1988
James(Jimmy)/Carter/Democrat/1977/1980
```

Now, with the directives added, fields are separated by a slash instead of a space, and the second field (containing the last name) is used as the key instead of the first field (containing the first name).

The following classes are available through the FileSys adapter:

Directory
DirectoryObject
File
FileRecord

Class: Directory (FileSys Adapter)

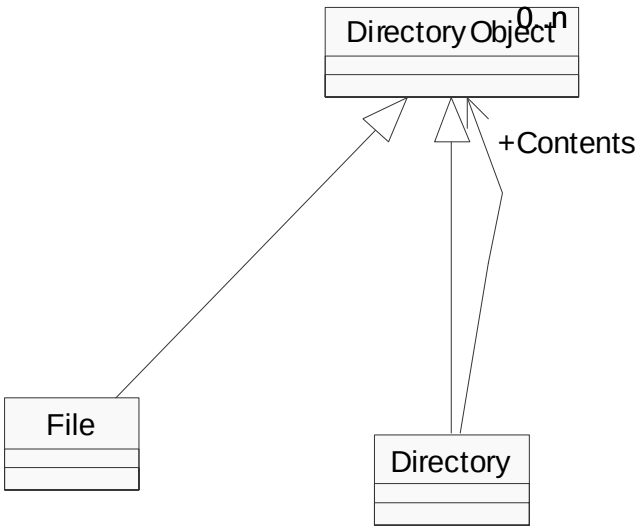
A directory, sometimes called a folder, contains other files or directories.

Class Hierarchy: DirectoryObject>Directory

SubClasses of Directory

Directory has no subclasses.

Class Diagram



Attributes Specific to Directory

Attributes	Inherited From	Description
DirectoryPath		The path of the directory.

Attributes	Inherited From	Description
DriveLetter	DirectoryObject	The drive letter of the location of the directory.
Extension	DirectoryObject	The segment of a SimpleName following the last period. For example, the Extension of c:\bill\file.txt is txt. If the SimpleName contains no period, then Extension returns a null string.
NameMinusExtension	DirectoryObject	The segment of a SimpleName preceding the last period. For example, the NameMinusExtension of c:\bill\file.txt is file. If the SimpleName contains no period, then NameMinusExtension returns the SimpleName.
NamePrefix	DirectoryObject	
Path	DirectoryObject	The complete pathname of an object. For example, c:\bill\file.txt
SimpleName	DirectoryObject	The context-independent portion of an object s name. For example, the SimpleName of c:\bill\file.txt is file.txt.

Relationships Specific to Directory (see also class diagram above)

Name	Kind	Class	Documentation
Contents	0..n	DirectoryObject	The DirectoryObjects that reside within the directory (subdirectories are included but not their contents).

Class: DirectoryObject (FileSys Adapter)

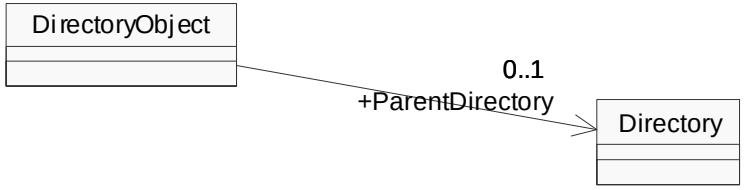
Anything that can be found in a Directory, including files and (sub)directories.

Class Hierarchy: Artifact>DirectoryObject
Subclasses of DirectoryObject:
Directory
File

SubClasses of DirectoryObject

Directory File

Class Diagram



Attributes Specific to DirectoryObject

Attributes	Inherited From	Description
DriveLetter		The drive letter of the location of the directory.
Extension		The segment of a SimpleName following the last period. For example, the Extension of c:\bill\file.txt is txt. If the SimpleName contains no period, then Extension returns a null string.

Attributes	Inherited From	Description
NameMinusExtension		The segment of a SimpleName preceding the last period. For example, the NameMinusExtension of c:\bill\file.txt is file. If the SimpleName contains no period, then NameMinusExtension returns the SimpleName.
NamePrefix		
Path		The complete pathname of an object. For example, c:\bill\file.txt
SimpleName		The context-independent portion of an object s name. For example, the SimpleName of c:\bill\file.txt is file.txt.

Relationships Specific to DirectoryObject (see also class diagram above)

Name	Kind	Class	Documentation
ParentDirectory	0..1	Directory	The directory containing the object. If you try to object ParentDirectory from the root directory, SoDA will generate an error.

Class: File (FileSys Adapter)

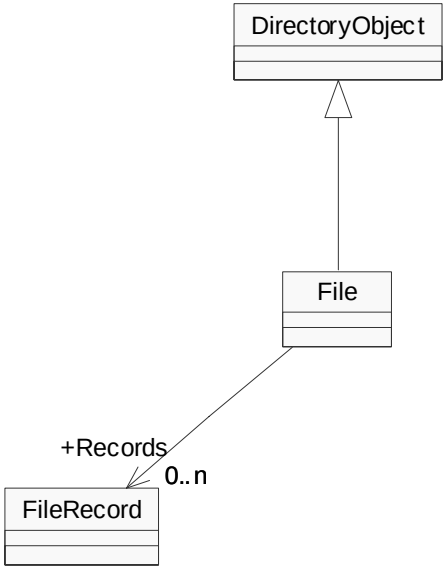
A subclass of DirectoryObject that does not contain other files or directories. Files can be ASCII or binary. They can contain text, bitmaps, program source, object code, executable code, or any other form of information that can be stored in a file. Note that the Graphic and Text attributes may not be defined for certain types of files.

Class Hierarchy: DirectoryObject>File

SubClasses of File

File has no subclasses.

Class Diagram



Attributes Specific to File

Attributes	Inherited From	Description
DriveLetter	DirectoryObject	The drive letter of the location of the directory.
Extension	DirectoryObject	The segment of a SimpleName following the last period. For example, the Extension of c:\bill\file.txt is txt. If the SimpleName contains no period, then Extension returns a null string.
FilePath		The path of the file
NameMinusExtension	DirectoryObject	The segment of a SimpleName preceding the last period. For example, the NameMinusExtension of c:\bill\file.txt is file. If the SimpleName contains no period, then NameMinusExtension returns the SimpleName.
NamePrefix	DirectoryObject	
Path	DirectoryObject	The complete pathname of an object. For example, c:\bill\file.txt
SimpleName	DirectoryObject	The context-independent portion of an object s name. For example, the SimpleName of c:\bill\file.txt is file.txt.
Text		The complete contents of an ASCII text file. Undefined for other file types.

Relationships Specific to File (see also class diagram above)

Name	Kind	Class	Documentation
Records	0..n	FileRecord	The records contained in a text file. By default, SoDA uses newlines to distinguish separate records within a file. It is possible to override this default by including a RECORD_DELIMITER directive in the file.

Class: FileRecord (FileSys Adapter)

ASCII text files can be decomposed into file records. File records are especially useful for parsing flat database files.

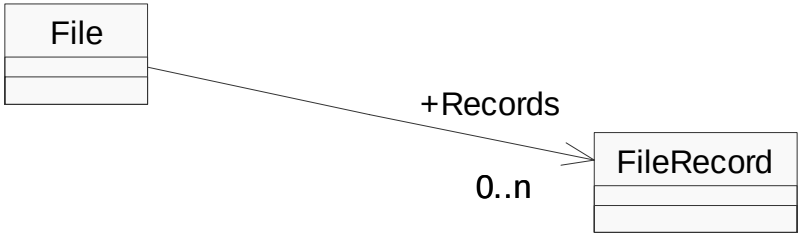
By default, SoDA uses newlines to delimit records within a file, spaces to delimit fields within a record, and double quotes (“) to surround a single field that includes spaces. Records must also contain key fields that uniquely identify each record. By default the first field is the key.

Class Hierarchy: Artifact>FileRecord

SubClasses of FileRecord

FileRecord has no subclasses.

Class Diagram



Attributes Specific to FileRecord

Attributes	Inherited From	Description
Field01		Text of the specified field, numbered from left to right. The extent of each field is determined by the field delimiter character, which defaults to a space. You can change the default by including a FIELD_DELIMITER directive in your file. Double quotes ( ) can be used to designate a single field that includes field delimiters. You can change the quote character by including a QUOTE_DELIMITER directive in your file.
Field02		
Field03		
Field04		
Field05		
Field06		
Field07		
Field08		
Field09		
Field10		
Field11		
Field12		
Field13		
Field14		
Field15		

Attributes	Inherited From	Description
Field16		
Field17		
Field18		
Field19		
Field20		
Field21		
Field22		
Field23		
Field24		
Field25		
Field26		
Field27		
Field28		
Field29		
Field30		
Filename		The full pathname of the file that contains this record.
Text		The complete contents of an ASCII text file. Undefined for other file types.

Attributes	Inherited From	Description
UniqueKey		The field or combination of fields used to uniquely identify the record. The default is to use the first field as the unique key. You can change the default by including a KEY_FIELDS directive in your file. If you have used the KEY_FIELDS directive to specify a multiple-field key, you enter a key by supplying each field, in order, separated by the field delimiter.

Relationships Specific to FileRecord (see also class diagram above)

This class has no relationships.

RSE Adapter: MSProject

The RSE adapter for Microsoft Project.

The following classes are available through the MSProject adapter:

Assignment

Project

Resource

Task

Class: Assignment (MSProject Adapter)

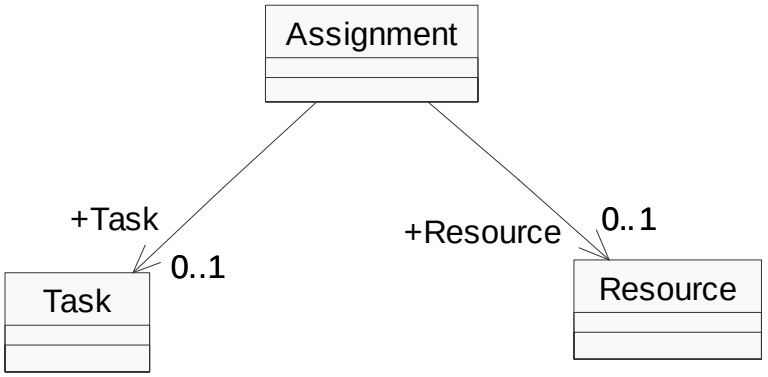
An assignment for a task or a resource.

Class Hierarchy: Artifact>Assignment

SubClasses of Assignment

Assignment has no subclasses.

Class Diagram



Attributes Specific to Assignment

Attributes	Inherited From	Description
ActualCost		The actual cost for this assignment.
ActualWork		The actual work for this assignment.
BaselineFinish		The finish of this assignment

Attributes	Inherited From	Description
BaselineStart		The start of this assignment.
Cost		The estimated cost of this assignment.
RemainingCost		The remaining cost for this assignment.
RemainingWork		The remaining work for this assignment.
ResourceName		The name of the Resource associated with this assignment.
ResourceUniqueID		The ID of the Resource associated with this assignment.
TaskName		The name of the Task associated with this assignment.
TaskUniqueID		The unique ID of the task associated with this assignment
UniqueID		The unique ID of the assignment.
Units		
Work		The work for this assignment

Relationships Specific to Assignment (see also class diagram above)

Name	Kind	Class	Documentation
Task	0..1	Task	The task associated with the assignment
Resource	0..1	Resource	The Resourse assigned to a task.

Class: Project (MSProject Adapter)

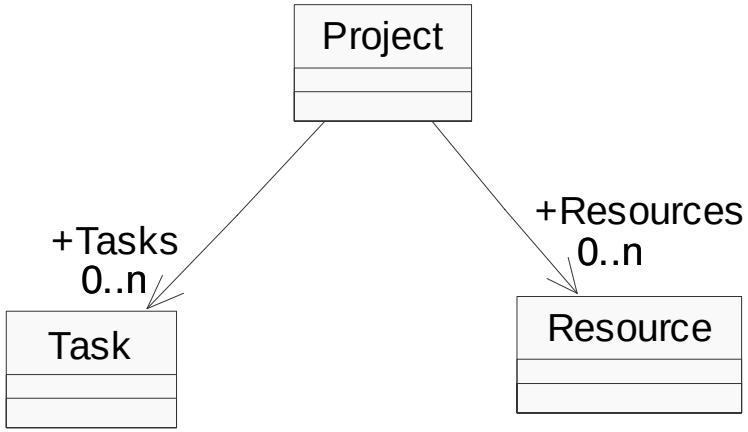
A project is a collection of data, including assignments, resources, and tasks.

Class Hierarchy: Artifact>Project

SubClasses of Project

Project has no subclasses.

Class Diagram



Attributes Specific to Project

Attributes	Inherited From	Description
Directory		The directory for this project
FinishDate		The finish date of the project

Attributes	Inherited From	Description
Name		The name of the project
Path		The file path of the project
StartDate		The start date of the project

Relationships Specific to Project (see also class diagram above)

Name	Kind	Class	Documentation
Tasks	0..n	Task	The Tasks associated with the Project
Resources	0..n	Resource	The Resources of the Project

Class: Resource (MSProject Adapter)

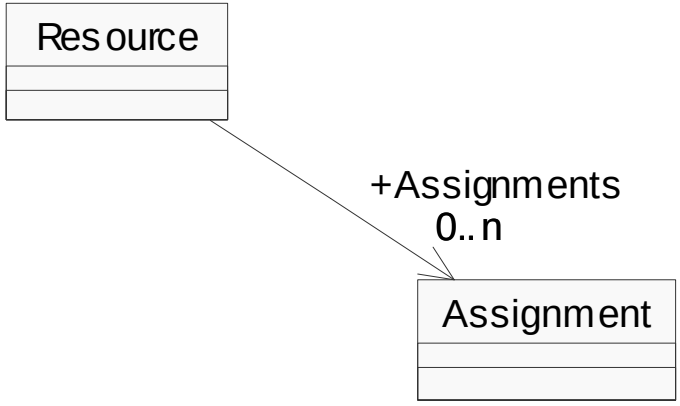
A single resource.

Class Hierarchy: Artifact>Resource

SubClasses of Resource

Resource has no subclasses.

Class Diagram



Attributes Specific to Resource

Attributes	Inherited From	Description
BaselineCost		
BaselineWork		

Attributes	Inherited From	Description
CostPerUse		
Group		
Name		
RemainingCost		
RemainingWork		
UniqueID		

Relationships Specific to Resource (see also class diagram above)

Name	Kind	Class	Documentation
Assignments	0..n	Assignment	

Class: Task (MSProject Adapter)

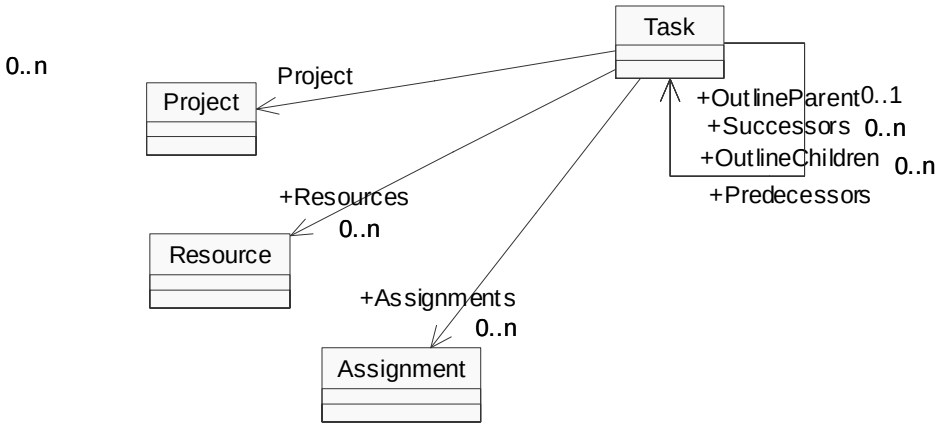
A specific piece of work to be done.

Class Hierarchy: Artifact>Task

SubClasses of Task

Task has no subclasses.

Class Diagram



Attributes Specific to Task

Attributes	Inherited From	Description
ActualCost		
ActualDuration		
ActualFinish		
ActualStart		
ActualWork		

Attributes	Inherited From	Description
ACWP		
BaselineCost		
BaselineDuration		
BaselineFinish		
BaselineStart		
BaselineWork		
BCWP		
BCWS		
ConstraintDate		
ConstraintType		
Cost		
CostVariance		
Critical		
Duration		
DurationVariance		
EffortDriven		
Finish		
FinishVariance		
FixedCost		
FixedCostAccrual		
ID		
Milestone		
Name		

Attributes	Inherited From	Description
Notes		
OutlineLevel		
OutlineNumber		
OvertimeCost		
OvertimeWork		
PercentComplete		
PercentWorkComplete		
Priority		
RegularWork		
RemainingCost		
RemainingDuration		
RemainingOvertimeCost		
RemainingOvertimeWork		
RemainingWork		
Start		
StartVariance		
Type		
UniqueID		
WBS		
Work		

Attributes	Inherited From	Description
WorkVariance		

Relationships Specific to Task (see also class diagram above)

Name	Kind	Class	Documentation
Predecessors		Task	
Successors		Task	
Assignments	0..n	Assignment	
Resources	0..n	Resource	
OutlineChil-dren		Task	
OutlineParent		Task	
Project	0..1	(Project)	
		Project	

RSE Adapter: RAdmin

The RSE adapter for Rational Administrator.

The following classes are available through the RAdmin adapter:

RAProject

RAServer

RoseModel

Class: RAProject (RAdmin Adapter)

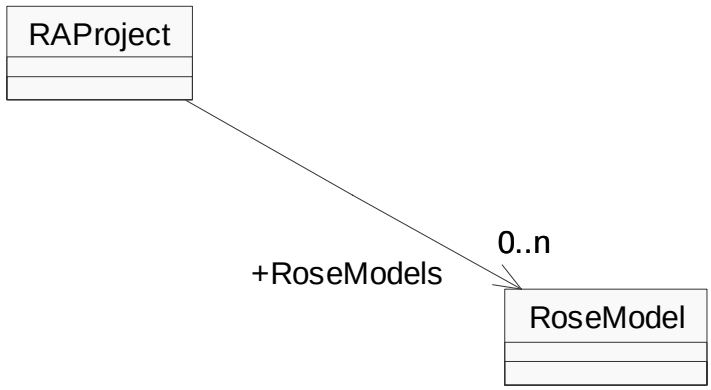
A project stores software testing and development information.

Class Hierarchy: Artifact>RAProject

SubClasses of RAProject

RAProject has no subclasses.

Class Diagram



Attributes Specific to RAProject

Attributes	Inherited From	Description
ClearQuestDatabaseName		The ClearQuestDatabase (CQDatabase object) name associated with the RAProject.
ClearQuestDBSetName		The ClearQuest DatabaseSet (CQDatabase artifact's DatabaseSet property) name associated with the RAProject.
Location		The location of the RAProject.
Name		The name of the RAProject.
Path		The file path of the RAProject.
RequirementsCMManaged		This property returns a boolean indicating whether the Requirements associated with the RA Project may be placed under CM.
RequisiteDatastorePath		The path of RequisitePro datastore that contains the CMManaged Requirements.
SynchronizerRulesPath		This property returns the path of the Synchronizer rules file associated with the RA Project.
TestAssetsCMManaged		This property returns a boolean indicating whether the Test Manager Test Assets associated with the RA Project may be placed under CM.
TestDatastorePath		The path of the Test Manager datastore that contains the CMManaged TestAssets.
UCMEnabled		TRUE if use case management is enabled.

Relationships Specific to RAProject (see also class diagram above)

Name	Kind	Class	Documentation
RoseModels	0..n	RoseModel	List all Rose models used in the project.

Class: RAServer (RAdmin Adapter)

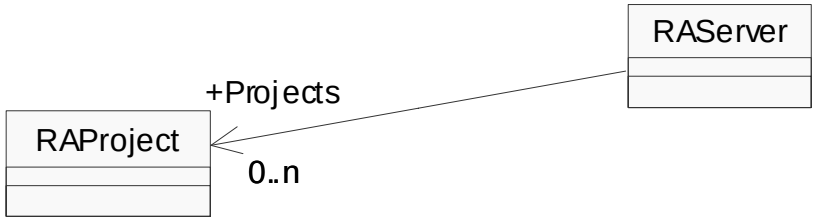
Provides access to all the other interfaces which provide access to the list of registered projects (RAProjects), the list of registered SQLAnywhere servers (RASQLAnywhereServers) and the ability to start various Rational tools (RationalTools).

Class Hierarchy: Artifact>RAServer

SubClasses of RAServer

RAServer has no subclasses.

Class Diagram



Attributes Specific to RAServer

This class has no attributes

Relationships Specific to RAServer (see also class diagram above)

Name	Kind	Class	Documentation
Projects	0..n	RAProject	The RAProjects associated with a RAProject.

Class: RoseModel (RAdmin Adapter)

Rose Models associated with an RAProject.

Class Hierarchy: Artifact>RoseModel

SubClasses of RoseModel

RoseModel has no subclasses.

Class Diagram

Attributes Specific to RoseModel

Attributes	Inherited From	Description
Name		The model s name
Path		The model s path

Relationships Specific to RoseModel (see also class diagram above)

This class has no relationships.

RSE Adapter: ReqPro

The RequisitePro domain allows you to incorporate information from your RequisitePro database into your SoDA documents. This information can come from projects, requirements, attributes, and so on.

If you need to access specific attribute names and values, be sure to see Accessing Project-specific Attributes.

Generating a SoDA Report directly from RequisitePro

If you are using Microsoft Office 97 or Office 2000, you can generate SoDA reports directly from RequisitePro. Follow these steps:

1. Start RequisitePro and open your project using the **Project** > **Open** command.
2. From the **Project** menu, choose **Generate SoDA Report**. (This menu item only appears if the proper versions of each product are installed, and a project has been opened.)
3. Select a template from the list that appears, and click **OK**. SoDA generates a report using the current project.

When generating SoDA reports from within RequisitePro using the Project > Generate SoDA Report command, only templates with the project object are generated with no interruptions. All templates that are provided by default in your SoDA installation offer this uninterrupted generation.

If you create a custom template that documents an individual requirement or document, you are prompted to provide both the project name and the requirement tag or document name. To avoid this, we recommend that you create a template that OPENs a project and uses the selector **Project.SelectedRequirements** or **Project.CurrentDocument**.

Accessing Project-specific Attributes

While SoDA lets you display attribute information using the default templates, it does not have knowledge of the attribute names specific to each project. If you want to be able to filter or sort values based on specific attribute values, you must provide the path of the project file in the OPEN command. SoDA will then add new classes; one for each requirement type, with the names of the attributes defined for each type. Here's an example:

1. Start the **Template View**.
2. Select **ReqPro Project** as your starting point.

3 In the pop-up dialog use the **Browse** button to locate the .rqs file you want to document. Click **OK**.

4 Select **Requirements within the Project**. Another pop-up dialog box appears, showing you the possible requirement types. Choose the one you want to document.

5 Select the attribute values to display.

Improving Performance of RequisitePro Templates

In your RequisitePro template, the REPEAT commands contain information for determining what type of RequisitePro object should be returned by the REPEAT during generation. Additionally, if you have clicked on the “Advanced” check box in the REPEAT command dialog box, the REPEAT command may contain criteria for further filtering the results.

During generation of a RequisitePro template, this criteria is passed to the RequisitePro domain where the domain attempts to build an SQL query corresponding to the specified criteria. By using the RequisitePro query engine, performance during document or report generation is greatly improved. This type of querying is only supported for some of the RequisitePro attributes; for the remainder, filtering of the results of a REPEAT command is done by SoDA.

For optimal performance, it is important to note which types of queries are supported by the RequisitePro query engine. Using these whenever possible results in the best performance:

- Fast filtering is currently only supported for requirements retrieved for a project.
- The best way to improve performance is to narrow your REPEAT to a particular requirement type. This is done by selecting one of the requirement type specific classes in the “Where is A” box in the REPEAT command dialog box. This restricts the results to only requirements of that type.
- If you are specifying additional filter criteria by adding to the “And Where” section in the REPEAT command dialog box, adhere to the following rules for faster queries:
  - Use only the "AND" logical operator in your queries (no "OR" queries).
  - Use user-defined attributes. Since these are specific to each requirement type, you must select a requirement type specific class in the “Where is A” box to make these available.
  - Reference any of the following attributes within your query. These are all attributes for the **requirement** class:

Attribute	Operators supported for "fast" filtering	Description
TagPrefix ReqType->ReqPrefix	=	Search by requirement type
FullTag	=, !=	Search by full tag of a requirement
Text	=, !=, <, <=, >, >=, LIKE[a]	Search by requirement text
Document.Name	=, !=	Search by the name of the containing document.
LatestRevision->Number	=, !=, <, <=, >, >=, LIKE [a]	Search by revision number
HierarchicalLevel	=, !=, <, <=, >, >=	Search by requirement level
LatestRevision->DateTime	=, !=, <, <=, >, >=	Search by the date of the last revision
Bookmark	=, !=, <, <=, >, >=, LIKE [a]	Search by bookmark

a. The following pattern matching characters are supported for fast filtering: ".", "$", "^". See the documentation for the Metacharacters for LIKE commands for more information.

Multiple attributes can be combined in your REPEAT to form a more complex query expression using the AND operator. You can combine the attributes above with others that are not supported for fast filtering. SoDA optimizes the resulting query for best performance.

The following classes are available through the ReqPro adapter:

AttributeValue

Discussion

DocumentType

Group

Permission

Project

Relationship

ReqDocument

Requirement

RequirementType

Response

Revision

User
View

Class: AttributeValue (ReqPro Adapter)

Attributes are descriptive information attached to a requirement that provide important details about that requirement, such as priority, cost, or difficulty.

Class Hierarchy: Artifact>AttributeValue

SubClasses of AttributeValue

AttributeValue has no subclasses.

Class Diagram

Attributes Specific to AttributeValue

Attributes	Inherited From	Description
DataType		The data type of this attribute.
Label		The name of the attribute
Text		The text of the Attribute
ValueID		The value of the attribute

Relationships Specific to AttributeValue (see also class diagram above)

This class has no relationships.

Class: Discussion (ReqPro Adapter)

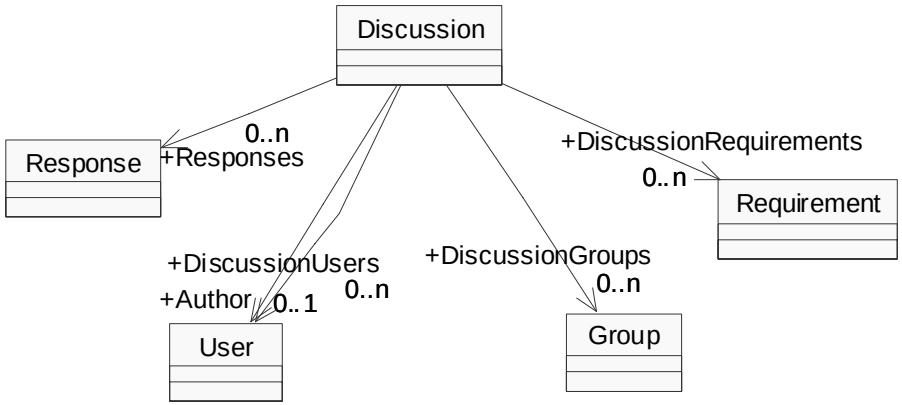
Discussions let RequisitePro users address comments, issues, and questions to a group of discussion participants. Discussions can be associated with one or more specific requirements, or refer to the project in general.

Class Hierarchy: Artifact>Discussion

SubClasses of Discussion

Discussion has no subclasses.

Class Diagram



Attributes Specific to Discussion

Attributes	Inherited From	Description
DateTime		When the discussion was created
DiscussionID		The ID of the discussion.
Message		The text of the discussion

Attributes	Inherited From	Description
Priority		The priority of the discussion: High, Medium, or Low
Restricted		True if the discussion is restricted to the listed participants
Status		The status of the discussion: Open or Closed
Subject		The subject of the discussion

Relationships Specific to Discussion (see also class diagram above)

Name	Kind	Class	Documentation
Responses	0..n	Response	The responses to this discussion
Author	0..1	User	The user who created this reponse
DiscussionRequirements	0..n	Requirement	An associated requirement to the discussion.
DiscussionUsers	0..n	User	Users are participants in the discussion.
DiscussionGroups	0..n	Group	Groups associated with the discussion.

Class: DocumentType (ReqPro Adapter)

A document type is a template that is applied to your documents. The template can include the default font for your document, the available heading and paragraph styles, and the default type of requirements for the document. Or it could encompass both formatting conventions and an outline that helps you organize your requirements information.

Class Hierarchy: Artifact>DocumentType

SubClasses of DocumentType

DocumentType has no subclasses.

Class Diagram

Attributes Specific to DocumentType

Attributes	Inherited From	Description
Description		The purpose and content of the document type
Extension		The file extension applied to all documents associated with this document type
Name		The name of the document type
TemplateDesc		A description of the template, or outline, for this document type
TemplateFilename		The filename of the template, or outline, used when documents of this type are created.
TemplateName		The name of the template, or outline, used when documents of this type are created.
TypeID		The document type ID.

Relationships Specific to DocumentType (see also class diagram above)

Name	Kind	Class	Documentation
Requirement-Type	0..1	Requirement-Type	The default type of requirement stored in this type of document

Class: Group (ReqPro Adapter)

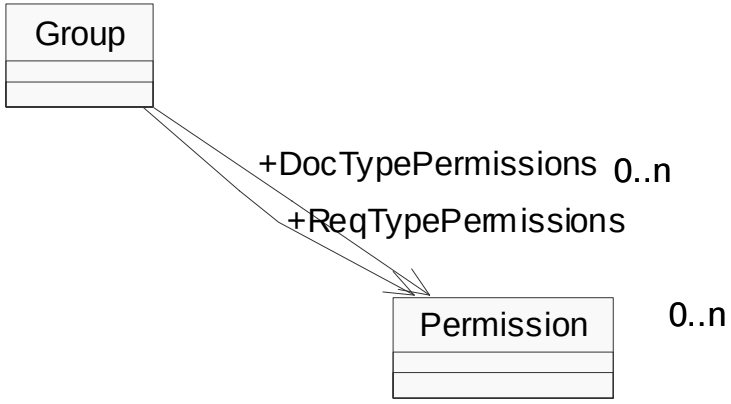
Groups are used for project security. A user group is a list of users, defined in project security and organized by the operations they have privileges to perform. For example, members of the Administrators group can create group accounts and add users to groups.

Class Hierarchy: Artifact>Group

SubClasses of Group

Group has no subclasses.

Class Diagram



Attributes Specific to Group

Attributes	Inherited From	Description
DefAttrPermis-sions		The privileges this group has to create or modify attributes

Attributes	Inherited From	Description
DefDocTypePermissions		The privileges this group has to create or modify document types
DefListItemPermissions		The privileges this group has to create or modify list items
DefProjPermissions		The privileges this group has to create or modify privileges
DefReqTypePermissions		The privileges this group has to create or modify requirements
GroupID		The Group ID
Name		The name of the user group
ProjectPermissions		The Project permissions of the Group

Relationships Specific to Group (see also class diagram above)

Name	Kind	Class	Documentation
DocTypePermissions	0..n	Permission	The document type permissions of the Group
ReqTypePermissions	0..n	Permission	The requirement type permissions of the Group

Class: Permission (ReqPro Adapter)

A privilege granted to a group of RequisitePro users. RequisitePro administrators or members of a group with project security permissions can assign permissions to groups. Permission types include:

Project Permissions

Document Type and Requirement Type Permissions

Traceability Permissions

Class Hierarchy: Artifact>Permission

SubClasses of Permission

Permission has no subclasses.

Class Diagram

Attributes Specific to Permission

Attributes	Inherited From	Description
IsModified		TRUE if the permission has been modified.
PermissionID		The permission ID.
Permissions		
TypeName		The name of the permission type.

Relationships Specific to Permission (see also class diagram above)

This class has no relationships.

Class: Project (ReqPro Adapter)

The concept of a project is used to provide the groundwork for organizing and effectively managing requirements. Each project resides in a separate directory. This storage method simplifies the process of organizing, archiving, and managing project files.

A project includes the following:

- Database
- Documents
- Document types
- Requirements and their attributes
- Requirement types
- Requirement traceability
- Discussions
- User and group security

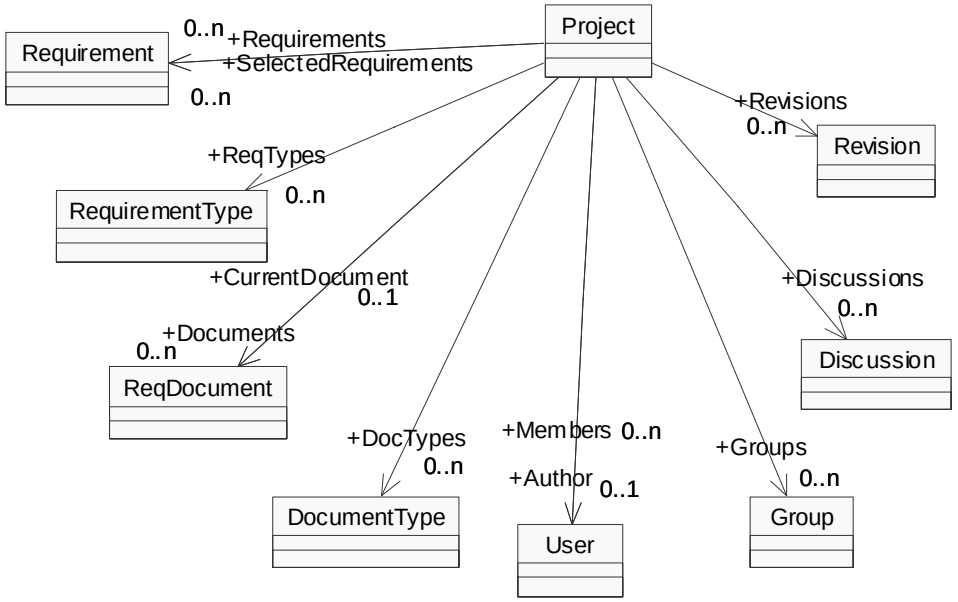
Within RequisitePro, you create a project first. You then create requirement documents and requirements in each document.

Class Hierarchy: Artifact>Project

SubClasses of Project

Project has no subclasses.

Class Diagram



Attributes Specific to Project

Attributes	Inherited From	Description
Description		Optional information describing the purpose and content of the project
FileName		The filename of the Project
Name		The name of the project
Path		The path of the project
Prefix		The prefix that is prepended to requirement tags when using external projects

Relationships Specific to Project (see also class diagram above)

Name	Kind	Class	Documentation
Requirements	0..n	Requirement	All requirements stored in the project database
Documents	0..n	ReqDocument	The set of documents associated with this project
Views	0..n	View	The Views associated with this Project
CurrentDocu-ment	0..1	ReqDocument	The document that is currently open (if any) in RequisitePro. No document is returned if the open document is not in the open project.
ReqTypes	0..n	Requirement-Type	The requirement types defined in this project.
DocTypes	0..n	Document-Type	The document types defined in this project
Revisions	0..n	Revision	The historical data of the project
Author	0..1	User	The creator of the project
External-Projects	0..n	(Project)	The external projects that have been attached to this project
Members	0..n	User	All users who are registered in this project
Groups	0..n	Group	The security groups defined in this project
Discussions	0..n	Discussion	The Discussions associated with the Project

Name	Kind	Class	Documentation
SelectedRequirements	0..n	Requirement	The requirements that are currently selected in the current View in RequisitePro. Only requirements in the open project are included.

Class: Relationship (ReqPro Adapter)

Traceability relationships are established between two or more requirements that exist in the same document, in different documents, or in the database.

Class Hierarchy: Artifact>Relationship

SubClasses of Relationship

Relationship has no subclasses.

Class Diagram

Attributes Specific to Relationship

Attributes	Inherited From	Description
Direction		The direction of the relationship: TraceTo, TraceFrom, Parent, or Child
RelationshipID		The relationship ID
RelationshipType		The type of the relationship, either Hierarchical or Traceability
Suspect		True if the relationship is suspect; otherwise, False

Relationships Specific to Relationship (see also class diagram above)

Name	Kind	Class	Documentation
RelatedReq	0..1	Requirement	The associated requirement

Class: ReqDocument (ReqPro Adapter)

A requirements document created in Microsoft Word or Rational RequisitePro that captures requirements and is used to communicate product development efforts. Each requirements document addresses a particular requirement type, such as product requirements, software specifications, or test plans.

A requirements document differs from a Word document in that you can access requirement attributes and other information directly from within the requirements document.

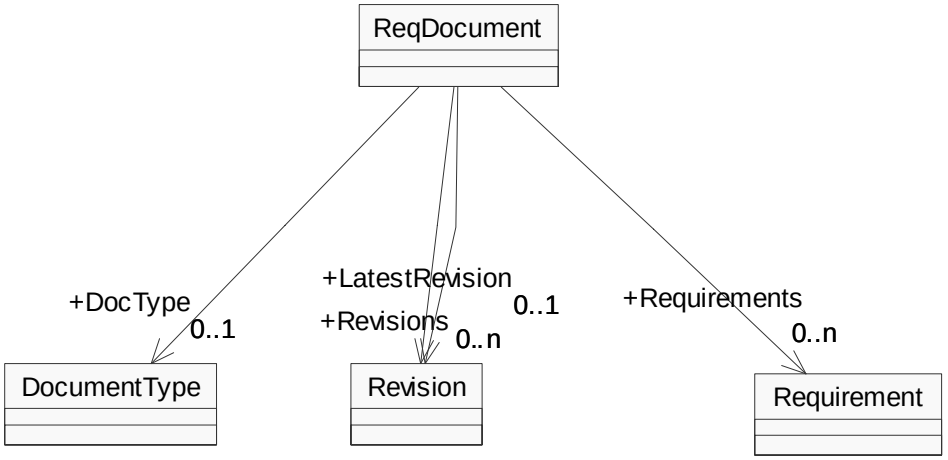
Note: Also referred to as "document," and "RequisitePro document."

Class Hierarchy: Artifact>ReqDocument

SubClasses of ReqDocument

ReqDocument has no subclasses.

Class Diagram



Attributes Specific to ReqDocument

Attributes	Inherited From	Description
Description		A description for the ReqDocument
DocumentID		The ID of the ReqDocument
Extension		The three letter extension of the Req-Document file
FileDateTime		The date of the ReqDocument
FileName		The filename of the ReqDocument
FullPath		The full path of the ReqDocument
Name		The name of the ReqDocument
Path		The path of the ReqDocument

Relationships Specific to ReqDocument (see also class diagram above)

Name	Kind	Class	Documentation
Revisions	0..n	Revision	The revision artifacts associated with a ReqDocument.
LatestRevi-sion	0..1	Revision	The latest Revision artifact associated with a ReqDocument.
DocType	0..1	Document-Type	The Document type associated with a ReqDocument.
Requirements	0..n	Requirement	The Requirements associated with a ReqDocument

Class: Requirement (ReqPro Adapter)

A requirement is the specification for the externally observable behavior of the system (for example, inputs to the system, outputs from the system, functions of the system, attributes of the system, or attributes of the system environment). In RequisitePro, a requirement defines an entity represented by: a piece of text, a set of attributes, and a set of traceability relationships.

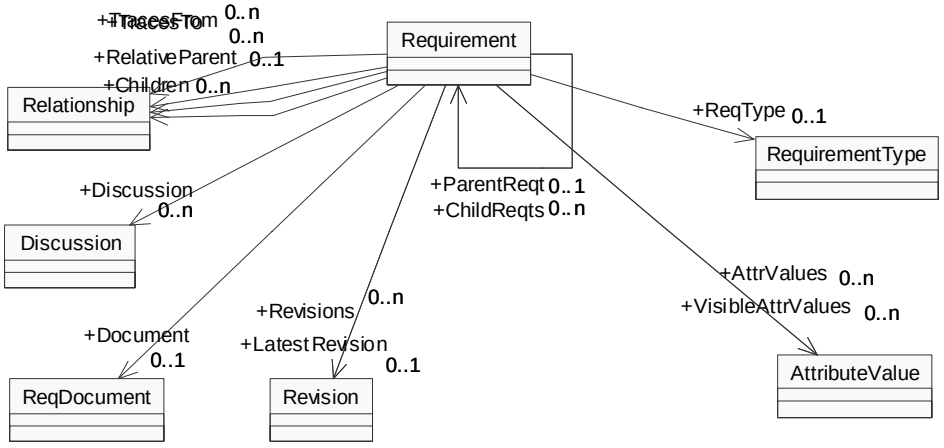
If a template includes the name of a specific project, there will also be subclasses for each <Project-Specific Type>Requirement. RSE automatically creates a series of new classes that are subclasses of the Requirement Class . The name of the class is the concatenation of the requirement type and the word Requirement. For instance, if a project contains requirement types PR, SR, and TST, the new classes will be PRRequirement, SRRequirement, and TSTRequirement.

Class Hierarchy: Artifact>Requirement

SubClasses of Requirement

Requirement has no subclasses.

Class Diagram



Attributes Specific to Requirement

Attributes	Inherited From	Description
Bookmark		The name of the Word bookmark associated with this requirement
DocPosn		The relative position of the requirement in the document. For instance, the second requirement in the document would be position 2. Database-only requirements have position 0.
FullTag		The full tag of the requirement, such as PR1
GUID		The GUI ID for the Requirement
HasChildren		True if this requirement has child requirements; otherwise False.
HasParent		True if this requirement has a parent requirement. This would be the same as Level > 0.
Level		The hierarchical level of the requirement. For instance, if the full tag is PR1.1, the level would be 1; PR1 would be level 0.
Name		The Requirement name
TagNumber		The number of the tag, such as 1
TagPrefix		The prefix of the tag, such PR
Text		The text of the requirement.
UniqueID		The unique ID for the Requirement

Relationships Specific to Requirement (see also class diagram above)

Name	Kind	Class	Documentation
AttrValues	0..n	AttributeValue	The AttributeValue artifacts for a given Requirement. AttrValues includes both the hidden and visible attributes.
VisibleAttrValues	0..n	AttributeValue	The visible AttributeValue artifacts for a given Requirement. VisibleAttrValues does not include hidden attributes. Attributes are hidden or made visible in RequisitePro by selecting the attribute name and editing the "Hidden from display" button.
TracesTo	0..n	Relationship	The relationships traced in to this requirement
TracesFrom	0..n	Relationship	The relationships traced out of this requirement
Revisions	0..n	Revision	The collection of revision artifacts for a given Requirement.
ReqType	0..1	RequirementType	The type of this requirement
Children	0..n	Relationship	The child Relationship artifacts of this Requirement. The Relationship artifacts represent the child Requirement artifacts returned by the ChildReqts relationship. In SoDA, if a template contains an OPEN command to a specific project, you will also see the ChildRequirements option, which lets you go directly to the requirements.

Name	Kind	Class	Documentation
ParentProject	0..1	(Project)	The project that this requirement is contained in. This relationship is especially useful when doing cross-project traceability.
Document	0..1	ReqDocument	The document where the requirement is stored
LatestRevision	0..1	Revision	The latest Revision artifact for a given Requirement.
Discussion	0..n	Discussion	The discussions attached to this requirement
RelativeParent	0..1	Relationship	The parent relationship for a given Requirement. The relationship refers to the parent Requirement artifact returned by the ParentReqt relationship.
ParentReqt		Requirement	The parent Requirement artifact for a given Requirement.
ChildReqts		Requirement	The child (nested) requirements for a given Requirement.

Class: RequirementType (ReqPro Adapter)

A requirement type defines a set of similar requirements. Requirement types are used to classify similar requirements so they can be efficiently managed. When you define a requirement type, you define a common set of attributes, display style, and tag numbering.

Class Hierarchy: Artifact>RequirementType

SubClasses of RequirementType

RequirementType has no subclasses.

Class Diagram

Attributes Specific to RequirementType

Attributes	Inherited From	Description
Description		A general description of the requirement type
Name		The name of the requirement type
ReqPrefix		The prefix of the type, such as SR
TypeID		The RequirementType ID

Relationships Specific to RequirementType (see also class diagram above)

This class has no relationships.

Class: Response (ReqPro Adapter)

An answer to a discussion item. You can respond to the initial discussion topic or to other, previous responses.

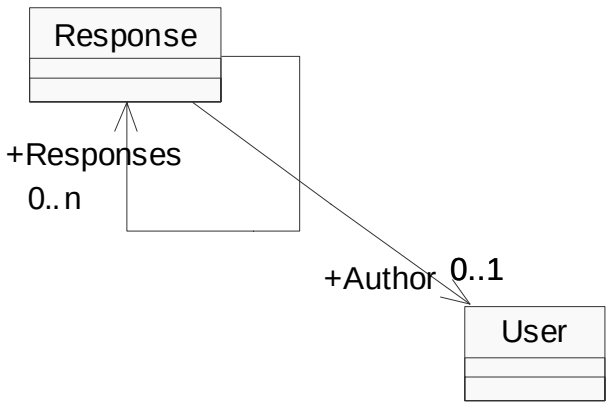
In Rational RequisitePro, the terms "response" and "discussion response" are used interchangeably.

Class Hierarchy: Artifact>Response

SubClasses of Response

Response has no subclasses.

Class Diagram



Attributes Specific to Response

Attributes	Inherited From	Description
DateTime		When the reply was created

Attributes	Inherited From	Description
HasResponses		True if someone has replied to this reply
Message		The text of the reply
ResponseFullKey		
ResponseID		The Response ID
Subject		The subject of the reply

Relationships Specific to Response (see also class diagram above)

Name	Kind	Class	Documentation
Responses		Response	The responses to this reply
Author	0..1	User	The user who created this reply

Class: Revision (ReqPro Adapter)

A distinct version of a project, document, or requirement. A revision is identified by a unique internal revision number, generated by Rational RequisitePro. The Revision object lets you document revision information about a requirement, document or project.

Class Hierarchy: Artifact>Revision

SubClasses of Revision

Revision has no subclasses.

Class Diagram

Attributes Specific to Revision

Attributes	Inherited From	Description
DateTime		The date and time the requirement was created or modified; the correct format for date and time is: yyyy-mm-dd hh:mm:ss. The value "hh" is the two-digit hour in military time. Hyphens and colons must be included as shown. For example, 1999-04-24 20:23:12 means April 24, 1999 at 8:23pm plus 12 seconds. You can drop any trailing part of a date-time, for example >= 1999-04 may be specified in a filter to obtain all Replies marked April 1999 or later.
Description		The change description field
Label		Text associated with a revision number
Number		The revision number, incremented automatically for each revision
ParentID		The revision's parent type ID
ParentType		The Revision parent type.

Attributes	Inherited From	Description
VersionID		The Version ID for the Revision

Relationships Specific to Revision (see also class diagram above)

Name	Kind	Class	Documentation
Author	0..1	User	The user that made the change

Class: User (ReqPro Adapter)

A Rational RequisitePro user. Users are people who have access to project information. Each user that is associated with a license of RequisitePro is a licensed RequisitePro user. RequisitePro tracks which users make changes to project and requirement information.

Note: In RequisitePro, the terms "user" and "licensed user" are used interchangeably.

Class Hierarchy: Artifact>User

SubClasses of User

User has no subclasses.

Class Diagram

Attributes Specific to User

Attributes	Inherited From	Description
FullName		The fullname of the User
Name		The name for this user
UserID		The UserID for this user

Relationships Specific to User (see also class diagram above)

Name	Kind	Class	Documentation
Group	0..1	Group	The group this user belongs to

Class: View (ReqPro Adapter)

A view displays information in spreadsheet-like tables or in outline trees. A view window displays requirements, the attributes assigned to requirements, or the relationships between requirements.

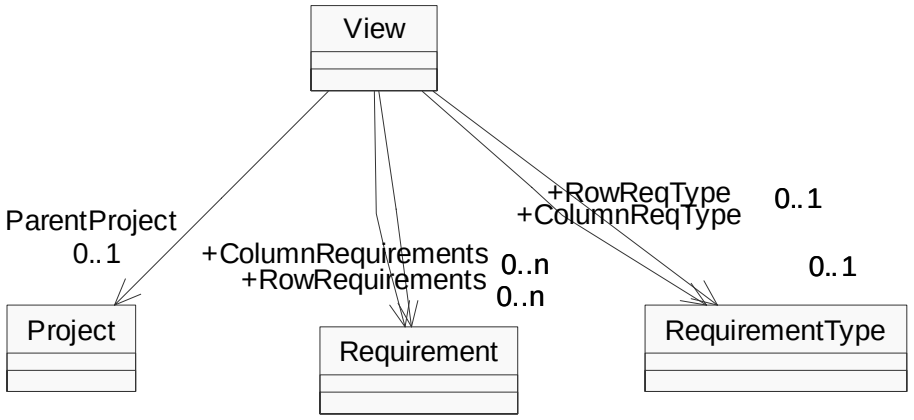
Note: In Rational RequisitePro, the terms "view" and "view window" are used interchangeably.

Class Hierarchy: Artifact>View

SubClasses of View

View has no subclasses.

Class Diagram



Attributes Specific to View

Attributes	Inherited From	Description
Description		The View description

Attributes	Inherited From	Description
FullName		The full name of the view
SimpleName		Th simple name of the View
State		The state of the View
Type		The View type
ViewID		The View ID
Visibility		

Relationships Specific to View (see also class diagram above)

Name	Kind	Class	Documentation
ParentProject	0..1	(Project)	The parent project associated with this view.
RowReqType	0..1	Requirement-Type	The Requirement type of a row in a view.
RowRequire-ments	0..n	Requirement	The Requirements in the rows of a view.
ColumnReq-Type	0..1	Requirement-Type	The Requirement type of a col-umn in a view.
ColumnRe-quirements	0..n	Requirement	The Requirements in the col-umns of a view.
		Project	The

RSE Adapter: Rose

The Rose source domain allows you to incorporate textual and graphical information from Rational Rose models. To extract information from a Rose model, you would typically create an OPEN command to the model specifying its filename. Once this command provides context for the model, you can traverse through the various components.

Before generating a document from a Rose model, you need to save the model. When you generate the document, the Rose domain will create a directory named <document name prefix>.dia and will fill it with .WMF files for each diagram requested from the model.

The Rose domain uses aliases to support multiple notations. SoDA is delivered with UML aliases. To change to use aliases for another notation or another language, simply modify the Rose.dom file.

Generating a SoDA Report directly from Rose

If you have installed Rose 98 or 98i and SoDA, and you are using Microsoft Office 97 or Office 2000, you can generate SoDA reports directly from Rose. (You cannot generate reports from Rose using Microsoft Word 95.) Follow these steps:

1. Start Rose and open your model. (You cannot use a newly created model that has never been saved.)
2. If you want a report for a specific package, class, or use case, select that item in a diagram.
3. From the **Report** menu, choose **SoDA Report**. (You will only see this menu item if the proper versions of each product are installed.)
4. Select a template from the list that appears, and click **OK**. A report will be generated using the current model.

Displaying the Contents of Files Referenced by ExternalDocs

The Files tab in most Rose specifications is for External Documents. In this tab you can identify one or more documents that further describe the model element. Follow these steps to include the contents of these documents in your SoDAdocument or report.

1. Within the context of a model element, create a **REPEAT** command and select the **ExternalDocs** relationship; set the **Name** to **ExternalDoc**.
2. Just inside the **REPEAT** command, create an **OPEN** command.

3 In the **Select Class** area, choose **Word** -> **WordFile**.

4 Click the **Advanced** button.

5 In the **Argument** area, click **Filename** twice to show a tree control next to the argument.

6 In the tree control, select **ExternalDoc** -> **Value**; click **OK** to create the **OPEN** command.

7 Just to the right of the **OPEN** command, create a **DISPLAY** command.

8 In the **Select Attribute** area, choose **WordFile** -> **FormattedText**.

The following classes are available through the Rose adapter:

Action
Activity
Association
Attribute
Class
ClassDiagram
ClassUtility
Decision
DeploymentDiagram
Device
Diagram
ExternalDocument
HasRelationship
InheritRelationship
InstantiatedClass
InstantiatedClassUtility
Item
Link
Message
MetaClass
Model

Module
ModuleDiagram
ModuleVisibilityRelationship
Node
Note
ObjectFlow
ObjectInstance
Operation
Package
PackageDependency
Parameter
ParameterizedClass
ParameterizedClassUtility
Process
Processor
Property
RealizeRelationship
Relationship
Role
Scenario
State
StateDiagram
StateMachine
StateTransition
Subsystem
SyncItem
UseCase
UseCaseDiagram
UsesRelationship

Class: Action (Rose Adapter)

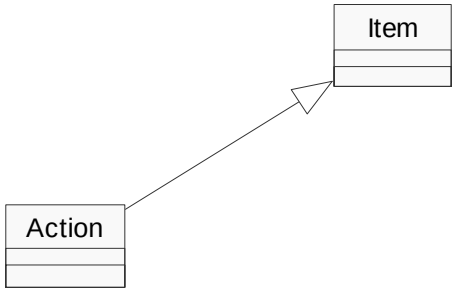
An action is an operation that is associated with a state transition.

Class Hierarchy: Artifact>Item>Action

SubClasses of Action

Action has no subclasses.

Class Diagram



Attributes Specific to Action

Attributes	Inherited From	Description
Arguments		The arguments that accompany the trigger event.
Documentation	Item	The documentation for the item
Name	Item	The name of the item
QualifiedName	Item	The qualified name of the item
Stereotype	Item	The item's stereotype
Target		The name of the event object.

Attributes	Inherited From	Description
UniqueID	Item	The unique id of the item

Relationships Specific to Action (see also class diagram above)

This class has no relationships.

Class: Activity (Rose Adapter)

The Activity class is an abstract class that exposes Rose's activity functionality in the Rose extensibility interface. With the Rose Activity class, you can:

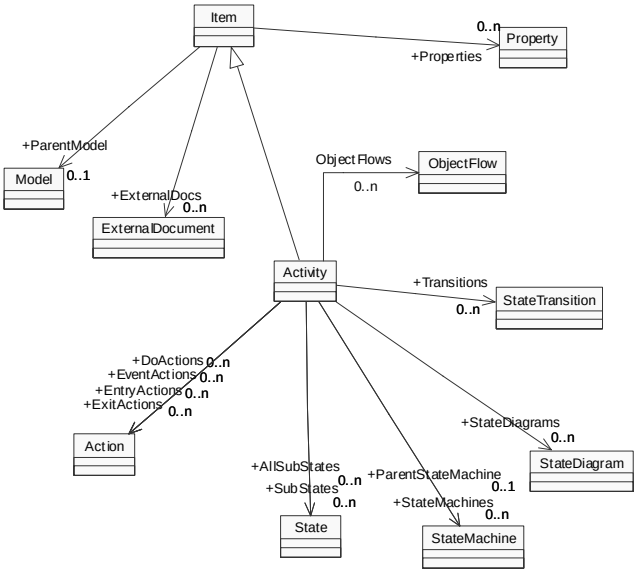
- Retrieve information about activities, such as name, documentation, stereotype
- Retrieve objects associated with activities such as parent activities, parent states, parent state machines, child activities, child decisions, child states, child synchronizations, outgoing transitions, and swimlanes
- Create and retrieve tool and property settings for activities
- Open specification sheets for activities
- Add, delete, and retrieve an activity's actions, state machines, and events
- Add and delete transitions

Class Hierarchy: Artifact>Item>Activity

SubClasses of Activity

Activity has no subclasses.

Class Diagram



Attributes Specific to Activity

Attributes	Inherited From	Description
Documentation	Item	The documentation for the item
Name	Item	The name of the item
QualifiedName	Item	The qualified name of the item
Stereotype	Item	The item's stereotype
UniqueID	Item	The unique id of the item

Relationships Specific to Activity (see also class diagram above)

Name	Kind	Class	Documentation
Transitions	0..n	StateTransition	The transitions that exit from this activity.
SubStates	0..n	State	The states that are part of this activity.
SubActivities	0..n	(Activity)	The activities that are part of this activity.
StateMachines	0..n	StateMachine	The state machines internal to this activity.
StateDiagrams	0..n	StateDiagram	The state or activity diagrams internal to this activity.
EntryActions	0..n	Action	The Entry actions for this activity.
ExitActions	0..n	Action	The Exit actions for this activity.
DoActions	0..n	Action	The Do actions for this activity.
EventActions	0..n	Action	The Event actions for this activity.
AllSubStates	0..n	State	The states associated with this activity
AllSubActivities	0..n	(Activity)	All activities associated with this activity
ParentStateMachine	0..1	StateMachine	The parent state machine associated with this activity.
ObjectFlows	0..n	ObjectFlow	The associated object flows for this activity.
		StateTransition	

Name	Kind	Class	Documentation
		ObjectFlow	

Class: Association (Rose Adapter)

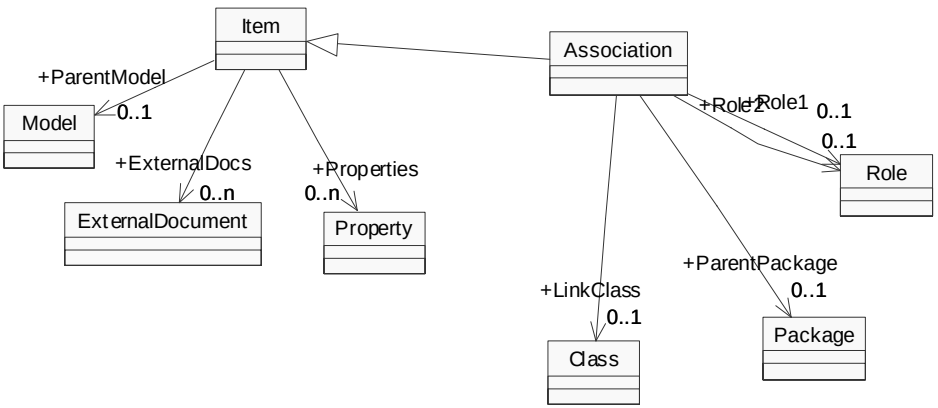
An association provides a pathway for communication. The communication can be between use cases, actors, classes or interfaces. Associations are the most general of all relationships and consequentially the most semantically weak. If two objects are usually considered independently, the relationship is an association.

Class Hierarchy: Artifact>Item>Association

SubClasses of Association

Association has no subclasses.

Class Diagram



Attributes Specific to Association

Attributes	Inherited From	Description
Constraints		The text from the Constraints field in the association specification.
Derived		True if the association is derived; otherwise False.
Documentation	Item	The documentation for the item

Attributes	Inherited From	Description
HasLinkClass		True if the association has an attached association class, otherwise False.
Name	Item	The name of the item
QualifiedName	Item	The qualified name of the item
Stereotype	Item	The item's stereotype
UniqueID	Item	The unique id of the item

Relationships Specific to Association (see also class diagram above)

Name	Kind	Class	Documentation
ParentPack-age	0..1	Package	The parent package attached to the association.
Role1	0..1	Role	The first role defined in the association.
LinkClass	0..1	Class	The linked class attached to the association.
Role2	0..1	Role	The second role defined in the association.

Class: Attribute (Rose Adapter)

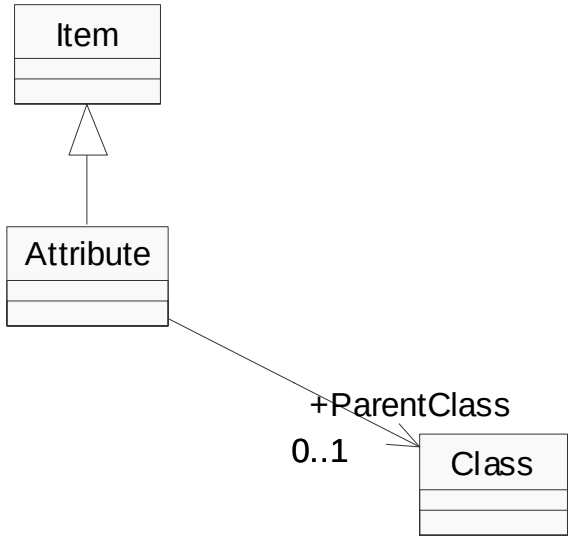
Attributes are data members of a class whose type is not another class. Attributes define the characteristics of a class. Each object in a class has the same attributes, but the values of the attributes may be different.

Class Hierarchy: Artifact>Item>Attribute

SubClasses of Attribute

Attribute has no subclasses.

Class Diagram



Attributes Specific to Attribute

Attributes	Inherited From	Description
Containment		Specifies the physical containment of the attribute. Returns Value, Reference, or Unspecified, depending on the state of the Containment radio control on the attribute specification.
Derived		True if the Derived check box is selected in the attribute specification, otherwise False.
Documentation	Item	The documentation for the item
ExportControl		The export control of the attribute. Returns Public, Protected, Private, or implementation.
InitValue		The initial value of the attribute.
Name	Item	The name of the item
QualifiedName	Item	The qualified name of the item
Static		True if the Static check box is selected in the attribute specification, otherwise False.
Stereotype	Item	The item's stereotype
Type		The type of the attribute.
UniqueID	Item	The unique id of the item

Relationships Specific to Attribute (see also class diagram above)

Name	Kind	Class	Documentation
ParentClass	0..1	Class	The class in which this attribute is defined.

Class: Class (Rose Adapter)

A class captures the common structure and common behavior of a set of objects. A class is an abstraction of real-world items. When these items exist in the real world, they are instances of the class, and referred to as objects. Rational Rose stores class information in a class specification.

Subclasses of Class:

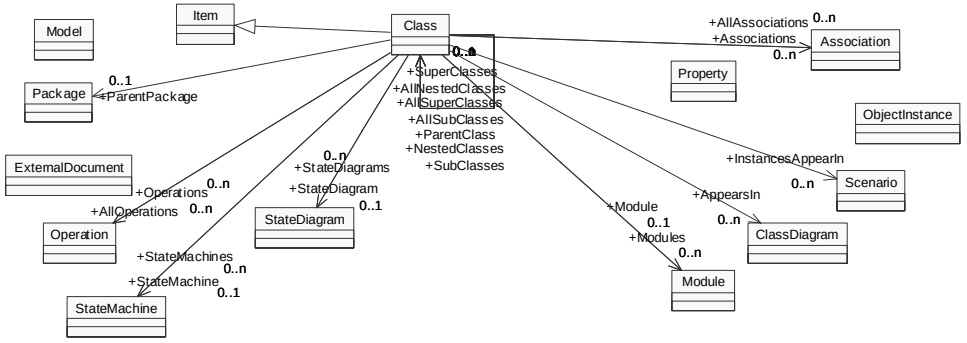
ParameterizedClass, InstantiatedClass, ClassUtility, ParameterizedClassUtility, InstantiatedClassUtility, MetaClass.

Class Hierarchy: Artifact>Item>Class

SubClasses of Class

ClassUtility InstantiatedClass InstantiatedClassUtility MetaClass ParameterizedClass ParameterizedClassUtility

Class Diagram



Attributes Specific to Class

Attributes	Inherited From	Description
Abstract		True if the Abstract check box is selected in the class specification, otherwise False.

Attributes	Inherited From	Description
Cardinality		The string in the Cardinality field of the class specification.
Concurrency		Returns Sequential, Guarded, Active, or Synchronous, depending on the value of the Concurrency radio control in the More dialog of the class specification.
Documentation	Item	The documentation for the item
ExportControl		Returns Public or Implementation, depending on the value of the Export Control radio control in the class specification.
FundamentalType		TRUE if this class is a fundamental type
HasStateDiagram		True if the class has an associated state diagram, otherwise False.
IsNested		True if the class is nested.
Kind		The kind of class
Name	Item	The name of the item
Persistence		This property is Persistent or Transient, depending on the value of the Persistence radio control in the More dialog of the class specification.
QualifiedName	Item	The qualified name of the item
Space		The string in the Space field of the More dialog of the class specification.
Stereotype	Item	The item's stereotype
UniqueID	Item	The unique id of the item

Relationships Specific to Class (see also class diagram above)

Name	Kind	Class	Documentation
AllAttributes	0..n	Attribute	All attributes of this class, including those inherited from other classes.
AllOperations	0..n	Operation	All operations of this class, including those inherited from other classes.
ParentClass		Class	The parent class of this class, if it is nested.
StateDiagram	0..1	StateDiagram	The (first) state/activity diagram associated with this class.
StateMachine	0..1	StateMachine	The (first) state machine associated with this class.
StateDiagrams	0..n	StateDiagram	All state/activity diagrams associated with this class.
StateMachines	0..n	StateMachine	All state machine associated with this class.
Relationships	0..n	Relationship	The relationships that are defined by this class. Does not include inherited relationships.
SubClasses		Class	The classes that directly inherit from this class. Only includes immediate subclasses. For example, if A inherits from B and B inherits from C, then MySubClasses of C would include B but not A.

Name	Kind	Class	Documentation
SuperClasses		Class	The classes that this class directly inherits from. Only includes immediate super-classes. For example, if A inher-its from B and B inherits from C, then MySuperClasses of A would include B but not C.
NestedClasses		Class	The classes that are nested within this class.
AllNested-Classes		Class	All nested classes of this class.
AllSuper-Classes		Class	All classes in the ancestry of this class. For example, if A inherits from B and B inherits from C, then AllSuperClasses of A would include B and C.
AllRelation-ships	0..n	Relationship	All relationships of this class, including those inherited from other classes.
ParentPack-age	0..1	Package	The enclosing package.
Module	0..1	Module	The first module associated with this class.
Associations	0..n	Association	The associations where this class plays a role.
AllAssocia-tions	0..n	Association	All associations where this class plays a role, including those inherited from other classes.

Name	Kind	Class	Documentation
AllSubClasses		Class	All classes in the lineage of this class. For example, if A inherits from B and B inherits from C, then AllSubClasses of C would include B and A
Modules	0..n	Module	All modules associated with this class.
Operations	0..n	Operation	The operations that are defined by this class. Does not include inherited operations.
Attributes	0..n	Attribute	The attributes that are defined by this class. Does not include inherited attributes.
AppearsIn	0..n	ClassDiagram	The class diagrams where this class appears.
InstancesAp-pearIn	0..n	Scenario	The interaction diagrams that include instances of this class.
Instances	0..n	ObjectIn-stance	The object instances associated with this class

Class: ClassDiagram (Rose Adapter)

A class diagram shows the relationships between packages and classes; the essential relationships include association, inherits, has, and uses. Each class diagram provides a logical view of the current model.

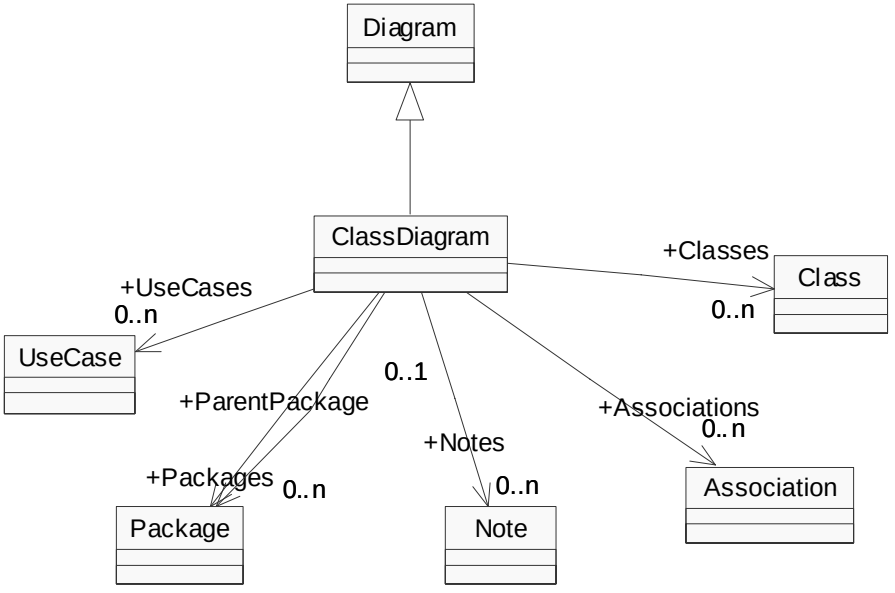
Class diagrams contain icons representing packages and classes. Class diagrams can be considered as filtered views into the model. They do not necessarily depict all the classes or relationships in the model. For example, iterating over all the classes in the main diagram of a package will not necessarily return all the classes defined in that category.

Class Hierarchy: Artifact>Diagram>ClassDiagram

SubClasses of ClassDiagram

ClassDiagram has no subclasses.

Class Diagram



Attributes Specific to ClassDiagram

Attributes	Inherited From	Description
Documentation	Diagram	The documentation text associated with the diagram.
MappedPoints	Diagram	
Name	Diagram	The name of the diagram.
QualifiedName	Diagram	The qualified name of the diagram
UniqueID	Diagram	The unique id for the diagram

Relationships Specific to ClassDiagram (see also class diagram above)

Name	Kind	Class	Documentation
UseCases	0..n	UseCase	All of the use cases that appear on the diagram.
Packages	0..n	Package	All packages associated with this class diagram.
Classes	0..n	Class	All of the classes that appear on the diagram.
Relationships	0..n	(Relationship)	All of the relationships that appear on the diagram.
Associations	0..n	Association	The associations where this class diagram plays a role.
ParentPack-age	0..1	Package	The package that this diagram is contained in, if applicable.
Notes	0..n	Note	The notes that appear in the diagram.

Class: ClassUtility (Rose Adapter)

A class utility is a set of operations that provide additional functions for classes. Class utilities are used to:

- Denote one or more free subprograms
- Name a class that only provides static members and/or static member functions.

Class Hierarchy: Item>Class>ClassUtility

SubClasses of ClassUtility

ClassUtility has no subclasses.

Class Diagram

Attributes Specific to ClassUtility

Attributes	Inherited From	Description
Abstract	Class	True if the Abstract check box is selected in the class specification, otherwise False.
Cardinality	Class	The string in the Cardinality field of the class specification.
Concurrency	Class	Returns Sequential, Guarded, Active, or Synchronous, depending on the value of the Concurrency radio control in the More dialog of the class specification.
Documentation	Item	The documentation for the item
ExportControl	Class	Returns Public or Implementation, depending on the value of the Export Control radio control in the class specification.
FundamentalType	Class	TRUE if this class is a fundamental type

Attributes	Inherited From	Description
HasStateDiagram	Class	True if the class has an associated state diagram, otherwise False.
IsNested	Class	True if the class is nested.
Kind	Class	The kind of class
Name	Item	The name of the item
Persistence	Class	This property is Persistent or Transient, depending on the value of the Persistence radio control in the More dialog of the class specification.
QualifiedName	Item	The qualified name of the item
Space	Class	The string in the Space field of the More dialog of the class specification.
Stereotype	Item	The item's stereotype
UniqueID	Item	The unique id of the item

Relationships Specific to ClassUtility (see also class diagram above)

This class has no relationships.

Class: Decision (Rose Adapter)

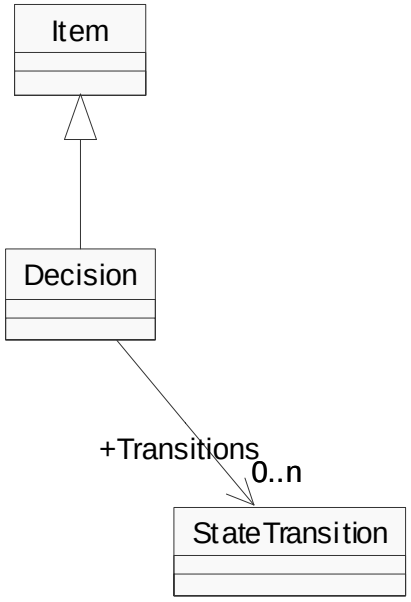
The Decision class is an abstract class that exposes Rose's decision functionality in the extensibility interface.

Class Hierarchy: Artifact>Item>Decision

SubClasses of Decision

Decision has no subclasses.

Class Diagram



Attributes Specific to Decision

Attributes	Inherited From	Description
Documentation	Item	The documentation for the item

Attributes	Inherited From	Description
Name	Item	The name of the item
QualifiedName	Item	The qualified name of the item
Stereotype	Item	The item's stereotype
UniqueID	Item	The unique id of the item

Relationships Specific to Decision (see also class diagram above)

Name	Kind	Class	Documentation
Transitions	0..n	StateTransi-tion	The state transition for this decision.

Class: DeploymentDiagram (Rose Adapter)

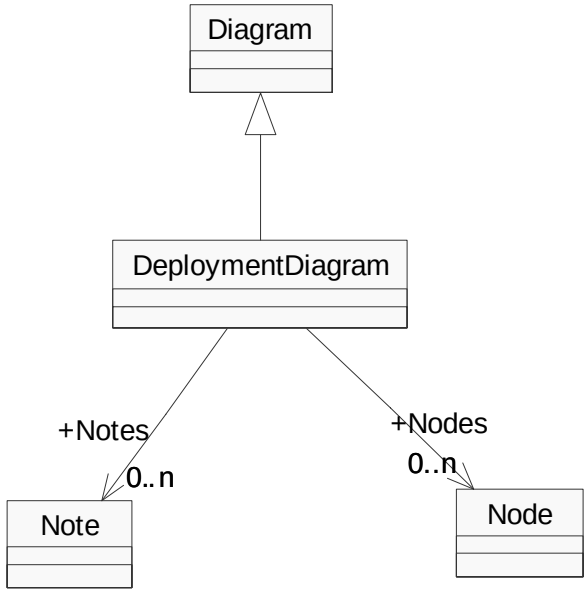
A deployment diagram shows the allocation of processes to processors in the physical design of a system. A deployment diagram may represent all or part of the process architecture of a system.

Class Hierarchy: Artifact>Diagram>DeploymentDiagram

SubClasses of DeploymentDiagram

DeploymentDiagram has no subclasses.

Class Diagram



Attributes Specific to DeploymentDiagram

Attributes	Inherited From	Description
Documentation	Diagram	The documentation text associated with the diagram.
MappedPoints	Diagram	
Name	Diagram	The name of the diagram.
QualifiedName	Diagram	The qualified name of the diagram
UniqueID	Diagram	The unique id for the diagram

Relationships Specific to DeploymentDiagram (see also class diagram above)

Name	Kind	Class	Documentation
Nodes	0..n	Node	The processors and devices contained in the diagram.
Notes	0..n	Note	The notes that appear in the diagram.

Class: Device (Rose Adapter)

A device is a hardware component with no computing power. The Rose device class exposes properties and methods that allow you to define and manipulate the characteristics of devices.

Class Hierarchy: Item>Node>Device

SubClasses of Device

Device has no subclasses.

Class Diagram

Attributes Specific to Device

Attributes	Inherited From	Description
Characteristics	Node	The characteristics of the processor or device.
Documentation	Item	The documentation for the item
Name	Item	The name of the item
QualifiedName	Item	The qualified name of the item
Stereotype	Item	The item's stereotype
UniqueID	Item	The unique id of the item

Relationships Specific to Device (see also class diagram above)

This class has no relationships.

Class: Diagram (Rose Adapter)

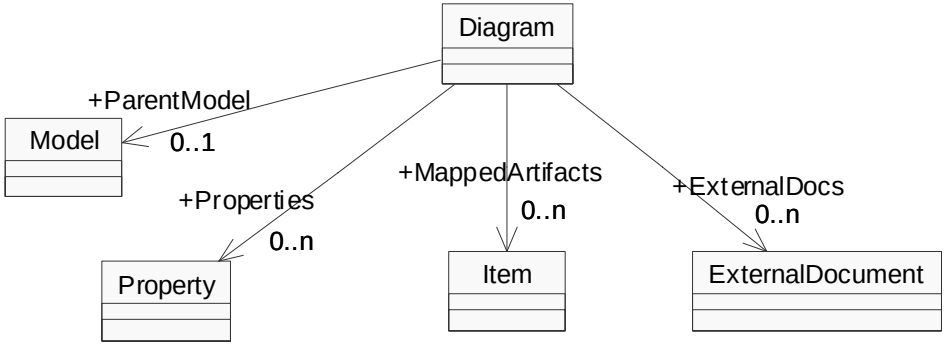
The Rose Diagram class exposes a set of properties and methods, which all other diagram classes (for example, class diagrams, scenario diagrams, etc.) inherit. These properties and methods determine the size and placement of a diagram on the Rose user's computer screen.

Class Hierarchy: Artifact>Diagram

SubClasses of Diagram

ClassDiagram DeploymentDiagram ModuleDiagram Scenario StateDiagram UseCase-Diagram

Class Diagram



Attributes Specific to Diagram

Attributes	Inherited From	Description
Documentation		The documentation text associated with the diagram.
MappedPoints		

Attributes	Inherited From	Description
Name		The name of the diagram.
QualifiedName		The qualified name of the diagram
UniqueID		The unique id for the diagram

Relationships Specific to Diagram (see also class diagram above)

Name	Kind	Class	Documentation
ParentModel	0..1	Model	The model that this diagram is contained in.
ExternalDocs	0..n	ExternalDocument	The external documents attached to this diagram.
Properties	0..n	Property	The property artifact types associated with this diagram
MappedArtifacts	0..n	Item	The items that are associated with this diagram

Class: ExternalDocument (Rose Adapter)

The Rose ExternalDocument class exposes properties and methods that allow you to create external documents (reports) from within the Rose environment. For example, you can start Word for Windows and output information from a Rose model into a Word document.

Class Hierarchy: Artifact>ExternalDocument

SubClasses of ExternalDocument

ExternalDocument has no subclasses.

Class Diagram

Attributes Specific to ExternalDocument

Attributes	Inherited From	Description
CollIndex		
ParentUID		The unique id of the external document's parent class
Value		

Relationships Specific to ExternalDocument (see also class diagram above)

This class has no relationships.

Class: HasRelationship (Rose Adapter)

The Has Relationship indicates a containment or aggregation relationship between classes.

The has relationship, available only with the Booch notation, denotes a whole and part relationship between two classes. This relationship is used to show how instances of the supplier, or aggregate, class are physically constructed from instances of the client class. The FromClass relationship returns the aggregate class. The ToClass relationship returns the client class, whose instances are part of aggregate class instances.

Class Hierarchy: Item>Relationship>HasRelationship

SubClasses of HasRelationship

HasRelationship has no subclasses.

Class Diagram

Attributes Specific to HasRelationship

Attributes	Inherited From	Description
ClientCardinality	Relationship	Indicates the number of possible links from an instance of the client class to an instance of the supplier class. Can be the same values as those listed in CardinalityFrom above.
Containment		Specifies the physical containment of the relationship. Returns Value, Reference, or Unspecified, depending on the state of the Containment radio control on the relationship specification. Containment is also shown by adornments on relationships in diagrams.
Documentation	Item	The documentation for the item

Attributes	Inherited From	Description
ExportControl	Relationship	Specifies the type of access allowed between classes. Returns Public, Protected, Private, or Implementation, depending on the state of the Access radio control on the relationship specification. Access is also shown by adornments on relationships in diagrams.
Kind	Relationship	Kind of the relationship, which will be one of: AggregateRole, AssociationRole, HasRelationship, InheritsRelationship or UsesRelationship.
Name	Item	The name of the item
QualifiedName	Item	The qualified name of the item
Static		Specifies whether the instance of the part class is owned by the class itself and not by its individual instances. Returns True, if the Static check box is checked on the relationship specification. Otherwise, returns False. Static relationships are also designated by special adornments on relationships in diagrams.
Stereotype	Item	The item's stereotype
SupplierCardinality	Relationship	Indicates the number of possible links from an instance of the supplier class to an instance of the client class. Can be one the following values: n, 1, 0..n, 1..n, 0..1, <literal>, <literal>..n, or <literal>..<literal>.
SupplierName	Relationship	The name of the supplier class or use case.
UniqueID	Item	The unique id of the item

Relationships Specific to HasRelationship (see also class diagram above)

This class has no relationships.

Class: InheritRelationship (Rose Adapter)

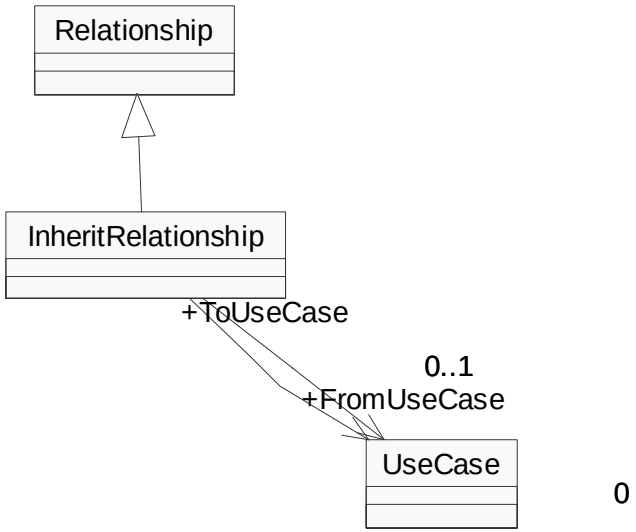
Indicates an inheritance relationship between classes.

Class Hierarchy: Item>Relationship>InheritRelationship

SubClasses of InheritRelationship

InheritRelationship has no subclasses.

Class Diagram



Attributes Specific to InheritRelationship

Attributes	Inherited From	Description
ClientCardinality	Relationship	Indicates the number of possible links from an instance of the client class to an instance of the supplier class. Can be the same values as those listed in CardinalityFrom above.
Documentation	Item	The documentation for the item
ExportControl	Relationship	Specifies the type of access allowed between classes. Returns Public, Protected, Private, or Implementation, depending on the state of the Access radio control on the relationship specification. Access is also shown by adornments on relationships in diagrams.
FriendshipRequired		Indicates whether the supplier class grants rights to the client class to access its non-public parts. Returns True, if the Friendship required check box is checked on the relationship specification. Otherwise, returns False.
Kind	Relationship	Kind of the relationship, which will be one of: AggregateRole, AssociationRole, HasRelationship, InheritsRelationship or UsesRelationship.
Name	Item	The name of the item
QualifiedName	Item	The qualified name of the item
Stereotype	Item	The item's stereotype

Attributes	Inherited From	Description
SupplierCardinal-ity	Relationship	Indicates the number of possible links from an instance of the supplier class to an instance of the client class. Can be one the following values: n, 1, 0..n, 1..n, 0..1, <literal>, <literal>..n, or <lit-eral>..<literal>.
SupplierName	Relationship	The name of the supplier class or use case.
UniqueID	Item	The unique id of the item
Virtual		

Relationships Specific to InheritRelationship (see also class diagram above)

Name	Kind	Class	Documentation
FromUseCase	0..1	UseCase	The supplier use case of the inherits relationship, if it is a use-case.
ToUseCase	0..1	UseCase	The client use case of the inher-its relationship, if it is a use case.

Class: InstantiatedClass (Rose Adapter)

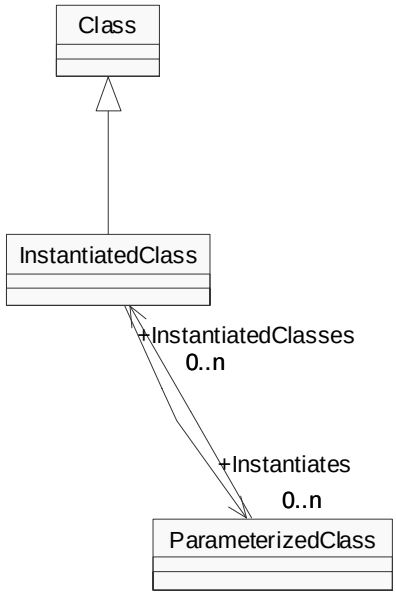
A class which instantiates a parameterized class. Instantiated classes are created by supplying the actual values for the formal parameters of the parameterized class. An instantiated class is concrete, meaning that its implementation is complete, and it may have object instances.

Class Hierarchy: Item>Class>InstantiatedClass

SubClasses of InstantiatedClass

InstantiatedClass has no subclasses.

Class Diagram



Attributes Specific to InstantiatedClass

Attributes	Inherited From	Description
Abstract	Class	True if the Abstract check box is selected in the class specification, otherwise False.
Cardinality	Class	The string in the Cardinality field of the class specification.
Concurrency	Class	Returns Sequential, Guarded, Active, or Synchronous, depending on the value of the Concurrency radio control in the More dialog of the class specification.
Documentation	Item	The documentation for the item
ExportControl	Class	Returns Public or Implementation, depending on the value of the Export Control radio control in the class specification.
FundamentalType	Class	TRUE if this class is a fundamental type
HasStateDiagram	Class	True if the class has an associated state diagram, otherwise False.
IsNested	Class	True if the class is nested.
Kind	Class	The kind of class
Name	Item	The name of the item
Persistence	Class	This property is Persistent or Transient, depending on the value of the Persistence radio control in the More dialog of the class specification.
QualifiedName	Item	The qualified name of the item
Space	Class	The string in the Space field of the More dialog of the class specification.

Attributes	Inherited From	Description
Stereotype	Item	The item's stereotype
UniqueID	Item	The unique id of the item

Relationships Specific to InstantiatedClass (see also class diagram above)

Name	Kind	Class	Documentation
Instantiates	0..n	Parameter-izedClass	The parameterized class that this instantiated class instanti-ates.

Class: InstantiatedClassUtility (Rose Adapter)

A class utility which instantiates a parameterized class utility. Instantiated class utilities are created by supplying the actual values for the formal parameters of the parameterized class utility.

An instantiated class utility is displayed as a 3-part box, with the class name in the top part, a list of attributes (with optional types and values) in the middle part, and a list of operations (with optional argument lists and return types) in the bottom part.

Class Hierarchy: Item>Class>InstantiatedClassUtility

SubClasses of InstantiatedClassUtility

InstantiatedClassUtility has no subclasses.

Class Diagram

Attributes Specific to InstantiatedClassUtility

Attributes	Inherited From	Description
Abstract	Class	True if the Abstract check box is selected in the class specification, otherwise False.
Cardinality	Class	The string in the Cardinality field of the class specification.
Concurrency	Class	Returns Sequential, Guarded, Active, or Synchronous, depending on the value of the Concurrency radio control in the More dialog of the class specification.
Documentation	Item	The documentation for the item
ExportControl	Class	Returns Public or Implementation, depending on the value of the Export Control radio control in the class specification.
FundamentalType	Class	TRUE if this class is a fundamental type

Attributes	Inherited From	Description
HasStateDiagram	Class	True if the class has an associated state diagram, otherwise False.
IsNested	Class	True if the class is nested.
Kind	Class	The kind of class
Name	Item	The name of the item
Persistence	Class	This property is Persistent or Transient, depending on the value of the Persistence radio control in the More dialog of the class specification.
QualifiedName	Item	The qualified name of the item
Space	Class	The string in the Space field of the More dialog of the class specification.
Stereotype	Item	The item's stereotype
UniqueID	Item	The unique id of the item

Relationships Specific to InstantiatedClassUtility (see also class diagram above)

Name	Kind	Class	Documentation
Instantiates	0..n	ParameterizedClass	The parameterized class utility that this instantiated class utility instantiates.

Class: Item (Rose Adapter)

Item maps to RoseItem objects. Every RoseItem is a model element and therefore inherits all Element properties and methods. Item specifies the type of model element that the stereotype settings apply to. Valid items include:

Class

Component

Package (includes logical package, use case package, and component package)

Logical Package

Component Package

Use Case Package

Processor

Device

Use Case

Association

Generalization

Dependency

Connection

Class Attribute

Operation

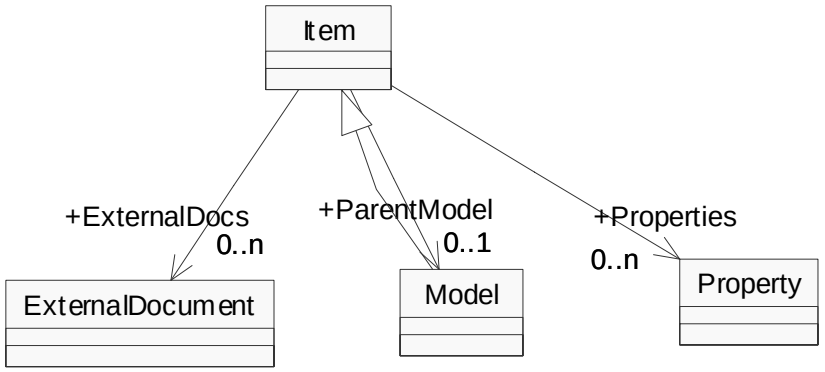
The default setting is Class.

Class Hierarchy: Artifact>Item

SubClasses of Item

Action Activity Association Attribute Class Decision Link Message Model Module ModuleVisibilityRelationship Node ObjectInstance Operation Package PackageDependency Parameter Process Relationship State StateTransition Subsystem SyncItem UseCase

Class Diagram



Attributes Specific to Item

Attributes	Inherited From	Description
Documentation		The documentation for the item
Name		The name of the item
QualifiedName		The qualified name of the item
Stereotype		The item's stereotype
UniqueID		The unique id of the item

Relationships Specific to Item (see also class diagram above)

Name	Kind	Class	Documentation
ParentModel	0..1	Model	The parent Model associated with this item
ExternalDocs	0..n	ExternalDocument	The ExternalDocuments associated with this item

Name	Kind	Class	Documentation
Properties	0..n	Property	The Property artifact types associated with this item

Class: Link (Rose Adapter)

Objects interact through their links to other objects. A Link is an instance of an association, in the same way that an object is an instance of a class.

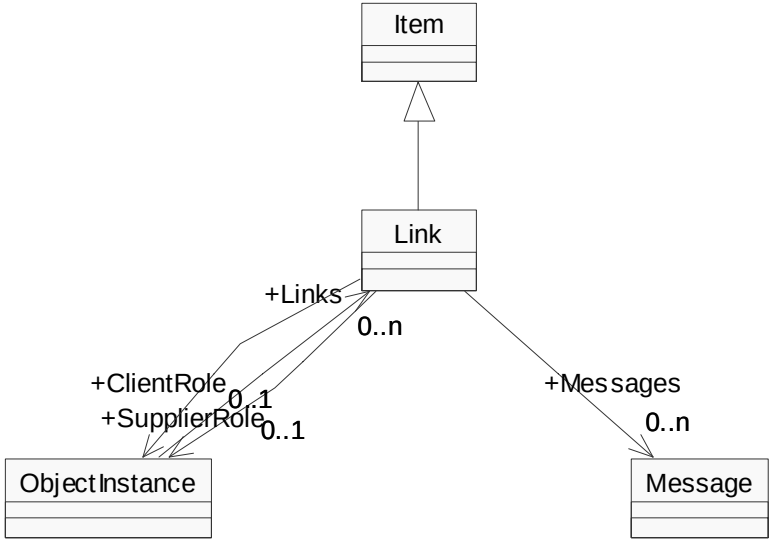
Rose Link properties and methods allow you to define links between objects and determine the nature of the objects' associations.

Class Hierarchy: Artifact>Item>Link

SubClasses of Link

Link has no subclasses.

Class Diagram



Attributes Specific to Link

Attributes	Inherited From	Description
ClientIsShared		True if the Shared box is checked on the client side; otherwise False.
ClientVisibility		One of Unspecified, Field, Parameters, Local, or Global.
Documentation	Item	The documentation for the item
IsLinkToSelf		True if the link goes from an object to itself.
Name	Item	The name of the item
QualifiedName	Item	The qualified name of the item
Stereotype	Item	The item's stereotype
SupplierIsShared		True if the Shared box is checked on the supplier side; otherwise False.
SupplierVisibility		One of Unspecified, Field, Parameters, Local, or Global.
UniqueID	Item	The unique id of the item

Relationships Specific to Link (see also class diagram above)

Name	Kind	Class	Documentation
SupplierRole	0..1	ObjectInstance	The supplier object instance (role) of the link.
ClientRole	0..1	ObjectInstance	The client object instance (role) of the link.
Messages	0..n	Message	The messages associated with the link.

Class: Message (Rose Adapter)

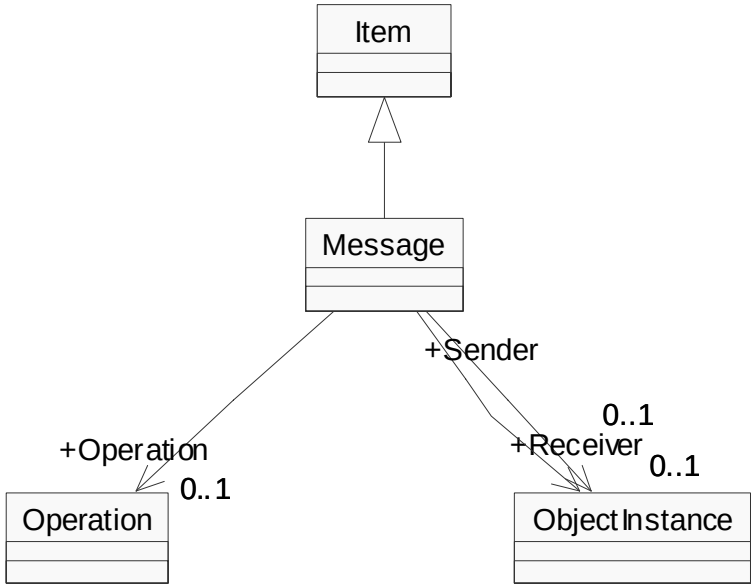
Any message associated with an object. Messages define the interaction between objects. The Rose message class inherits all of the Item (RoseItem) properties and methods. In addition message class methods allow you to retrieve message sender and receiver, along with other message-specific information.

Class Hierarchy: Artifact>Item>Message

SubClasses of Message

Message has no subclasses.

Class Diagram



Attributes Specific to Message

Attributes	Inherited From	Description
Documentation	Item	The documentation for the item
Frequency		The frequency of the message.
HierarchicalSeqNumber		The hierarchical sequence number of the message.
IsOperation		
Name	Item	The name of the item
NameWithoutParentheses		The name of a message without the parenthesized parameters from Rose that are added when a message is associated with a class operation.
QualifiedName	Item	The qualified name of the item
SeqNumber		The sequence number of the message.
Stereotype	Item	The item's stereotype
Synchronization		The concurrency semantics for the operation named in the Operations Field; one of Simple, Synchronous, Balking, Timeout or Asynchronous.
UniqueID	Item	The unique id of the item

Relationships Specific to Message (see also class diagram above)

Name	Kind	Class	Documentation
Operation	0..1	Operation	The associated Operation with this Message
Receiver	0..1	ObjectInstance	The object that receives the message.

Name	Kind	Class	Documentation
Sender	0..1	ObjectInstance	The object that sends the message.

Class: MetaClass (Rose Adapter)

A metaclass is a class whose instances are classes rather than objects. Metaclasses provide operations for initializing class variables and serve as repositories to hold class variables where a single value is required by all objects of a class. Smalltalk and CLOS support the use of metaclasses. C++ does not directly support metaclasses.

A metaclass is displayed as a 3-part box, with the class name in the top part, a list of attributes (with optional types and values) in the middle part, and a list of operations (with optional argument lists and return types) in the bottom part.

Not all languages directly support metaclasses.

Class Hierarchy: Item>Class>MetaClass

SubClasses of MetaClass

MetaClass has no subclasses.

Class Diagram

Attributes Specific to MetaClass

Attributes	Inherited From	Description
Abstract	Class	True if the Abstract check box is selected in the class specification, otherwise False.
Cardinality	Class	The string in the Cardinality field of the class specification.
Concurrency	Class	Returns Sequential, Guarded, Active, or Synchronous, depending on the value of the Concurrency radio control in the More dialog of the class specification.
Documentation	Item	The documentation for the item
ExportControl	Class	Returns Public or Implementation, depending on the value of the Export Control radio control in the class specification.

Attributes	Inherited From	Description
FundamentalType	Class	TRUE if this class is a fundamental type
HasStateDiagram	Class	True if the class has an associated state diagram, otherwise False.
IsNested	Class	True if the class is nested.
Kind	Class	The kind of class
Name	Item	The name of the item
Persistence	Class	This property is Persistent or Transient, depending on the value of the Persistence radio control in the More dialog of the class specification.
QualifiedName	Item	The qualified name of the item
Space	Class	The string in the Space field of the More dialog of the class specification.
Stereotype	Item	The item's stereotype
UniqueID	Item	The unique id of the item

Relationships Specific to MetaClass (see also class diagram above)

This class has no relationships.

Class: Model (Rose Adapter)

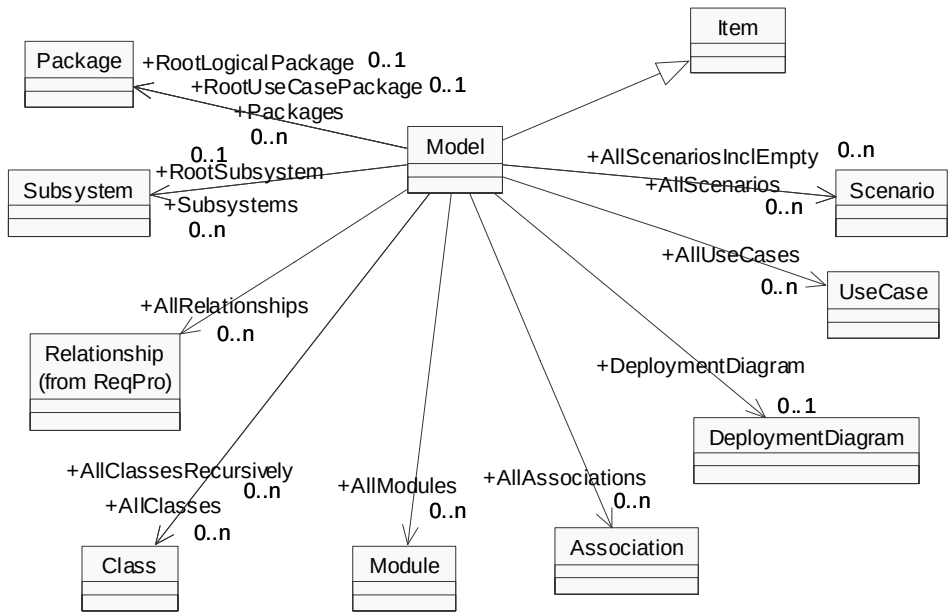
A Rose model file. A model file contains a Rose model, which describes your problem domain and system software. Model files use the default extension .mdl. Models are the highest hierarchical elements of the Rose source domain. Most templates start with connections to a Model.

Class Hierarchy: Artifact>Item>Model

SubClasses of Model

Model has no subclasses.

Class Diagram



Attributes Specific to Model

Attributes	Inherited From	Description
Documentation	Item	The documentation for the item
DriveLetter		The drive letter in the model's path
Extension		The segment of a SimpleName following the last period. For example, the Extension of c:\bill\file.txt is txt. If the SimpleName contains no period, then Extension returns a null string.
FullName		The full name, including the path, of the model.
Name	Item	The name of the item
NameMinusExtension		The segment of a SimpleName preceding the last period. For example, the NameMinusExtension of c:\bill\file.txt is file. If the SimpleName contains no period, then NameMinusExtension returns the SimpleName.
NamePrefix		
ParentDirectoryPath		The directory containing the object.
Path		The complete pathname of an object. For example, c:\bill\file.txt
QualifiedName	Item	The qualified name of the item
SimpleName		The simple name of the model. The context-independent portion of an object s name. For example, the SimpleName of c:\bill\file.txt is file.txt.
Stereotype	Item	The item's stereotype
UniqueID	Item	The unique id of the item

Relationships Specific to Model (see also class diagram above)

Name	Kind	Class	Documentation
Packages	0..n	Package	All packages in the model, including use-case packages (but not including subsystems in the Component View).
RootUse-CasePackage	0..1	Package	The root use-case package in the model; its name is UseCase View. All other use-case packages are nested beneath it.
Subsystems	0..n	Subsystem	All subsystem components in the model.
AllModules	0..n	Module	All modules in the model (including subsystems).
RootLogi-calPackage	0..1	Package	The highest-level package in the model; its name is Logical View. All other packages are nested beneath it.
AllRelation-ships	0..n	Relationship	All relationships in the model.
AllUseCases	0..n	UseCase	All use cases in the model.
AllScenarios	0..n	Scenario	All scenarios in the model.
AllScenarios-InclEmpty	0..n	Scenario	
RootSub-system	0..1	Subsystem	The highest-level subsystem in the model; its name is Component View. All other subsystems are nested beneath it.
AllAssocia-tions	0..n	Association	All associations in the model.

Name	Kind	Class	Documentation
AllClasses	0..n	Class	All classes in the model, including actors.
AllClassesRecursively	0..n	Class	
DeploymentDiagram	0..1	DeploymentDiagram	The deployment diagram (process diagram) for the model.

Class: Module (Rose Adapter)

A module is a unit of code that serves as a building block for the physical structure of a system. The module class exposes properties and methods that allow you to define and manipulate the characteristics of modules.

Class Hierarchy: Artifact>Item>Module

SubClasses of Module

Module has no subclasses.

Class Diagram

Attributes Specific to Module

Attributes	Inherited From	Description
AssignedLan-guage		
Declarations		
Documentation	Item	The documentation for the item
Name	Item	The name of the item
Part		
Path		The path of the module
QualifiedName	Item	The qualified name of the item
Stereotype	Item	The item's stereotype
Type		The type of module
UniqueID	Item	The unique id of the item

Relationships Specific to Module (see also class diagram above)

Name	Kind	Class	Documentation
Assigned-Classes	0..n	Class	The associated classes to this module
ParentSub-system	0..1	Subsystem	The parent subsystem of this module
VisibilityRela-tionships	0..n	ModuleVisibil-ityRelation-ship	The module visibility relation-ships for this module
		Class	
		ModuleVisibil-ityRelation-ship	

Class: ModuleDiagram (Rose Adapter)

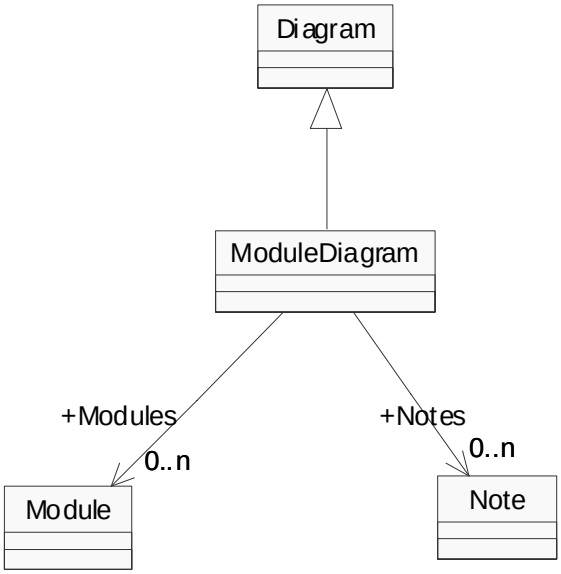
A module diagram maps the allocation classes and objects to modules. The module diagram class exposes properties and methods that allow you to add, retrieve and delete classes and objects in a module diagram.

Class Hierarchy: Artifact>Diagram>ModuleDiagram

SubClasses of ModuleDiagram

ModuleDiagram has no subclasses.

Class Diagram



Attributes Specific to ModuleDiagram

Attributes	Inherited From	Description
Documentation	Diagram	The documentation text associated with the diagram.
MappedPoints	Diagram	
Name	Diagram	The name of the diagram.
QualifiedName	Diagram	The qualified name of the diagram
UniqueID	Diagram	The unique id for the diagram

Relationships Specific to ModuleDiagram (see also class diagram above)

Name	Kind	Class	Documentation
Modules	0..n	Module	The modules in the module diagram
Notes	0..n	Note	The notes that appear in the diagram.

Class: ModuleVisibilityRelationship (Rose Adapter)

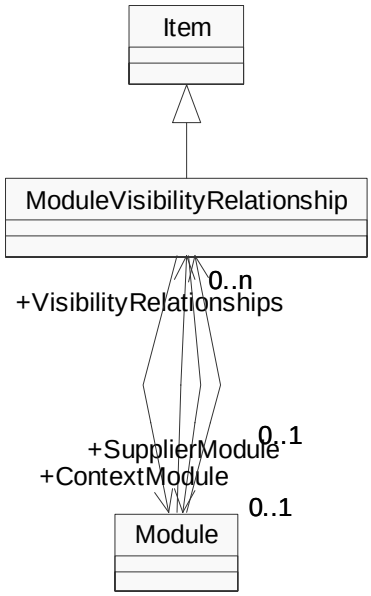
The ModuleVisibilityRelationship class describes the context and supplier relationship between modules.

Class Hierarchy: Artifact>Item>ModuleVisibilityRelationship

SubClasses of ModuleVisibilityRelationship

ModuleVisibilityRelationship has no subclasses.

Class Diagram



Attributes Specific to ModuleVisibilityRelationship

Attributes	Inherited From	Description
Documentation	Item	The documentation for the item

Attributes	Inherited From	Description
Name	Item	The name of the item
QualifiedName	Item	The qualified name of the item
Stereotype	Item	The item's stereotype
UniqueID	Item	The unique id of the item

Relationships Specific to ModuleVisibilityRelationship (see also class diagram above)

Name	Kind	Class	Documentation
SupplierModule	0..1	Module	
ContextModule	0..1	Module	

Class: Node (Rose Adapter)

Node is an abstract class for processors and devices.

Subclasses of Node:

Processor, Device

Class Hierarchy: Artifact>Item>Node

SubClasses of Node

Device Processor

Class Diagram

Attributes Specific to Node

Attributes	Inherited From	Description
Characteristics		The characteristics of the processor or device.
Documentation	Item	The documentation for the item
Name	Item	The name of the item
QualifiedName	Item	The qualified name of the item
Stereotype	Item	The item's stereotype
UniqueID	Item	The unique id of the item

Relationships Specific to Node (see also class diagram above)

This class has no relationships.

Class: Note (Rose Adapter)

A note captures the assumptions and decisions applied during analysis and design. Notes may contain any information, including plain text, fragments of code, or references to other documents. Notes are also used as a means of linking diagrams. A note holds an unlimited amount of text and can be sized accordingly.

Class Hierarchy: Artifact>Note

SubClasses of Note

Note has no subclasses.

Class Diagram

Attributes Specific to Note

Attributes	Inherited From	Description
CollIndex		
Text		The text of the note
Type		The type of note

Relationships Specific to Note (see also class diagram above)

This class has no relationships.

Class: ObjectFlow (Rose Adapter)

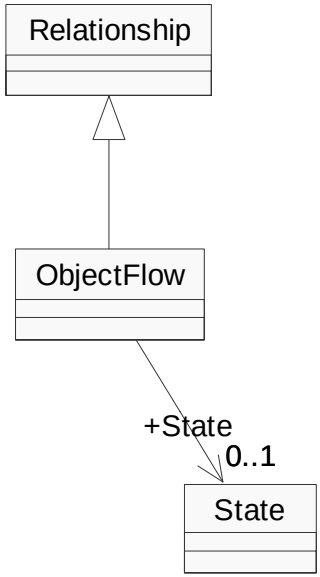
The ObjectFlow class is an abstract class that exposes Rose s object flow functionality in the extensibility interface. An object flow on an activity diagram represents the relationship between an activity and the object that creates it (as an output) or uses it (as an input).

Class Hierarchy: Artifact>Item>Relationship>ObjectFlow

SubClasses of ObjectFlow

ObjectFlow has no subclasses.

Class Diagram



Attributes Specific to ObjectFlow

Attributes	Inherited From	Description
ClientCardinality	Relationship	Indicates the number of possible links from an instance of the client class to an instance of the supplier class. Can be the same values as those listed in CardinalityFrom above.
Documentation	Item	The documentation for the item
ExportControl	Relationship	Specifies the type of access allowed between classes. Returns Public, Protected, Private, or Implementation, depending on the state of the Access radio control on the relationship specification. Access is also shown by adornments on relationships in diagrams.
Kind	Relationship	Kind of the relationship, which will be one of: AggregateRole, AssociationRole, HasRelationship, InheritsRelationship or UsesRelationship.
Name	Item	The name of the item
QualifiedName	Item	The qualified name of the item
Stereotype	Item	The item's stereotype
SupplierCardinality	Relationship	Indicates the number of possible links from an instance of the supplier class to an instance of the client class. Can be one the following values: n, 1, 0..n, 1..n, 0..1, <literal>, <literal>..n, or <literal>..<literal>.
SupplierName	Relationship	The name of the supplier class or use case.
UniqueID	Item	The unique id of the item

Relationships Specific to ObjectFlow (see also class diagram above)

Name	Kind	Class	Documentation
State	0..1	State	The associated State of the ObjectFlow

Class: ObjectInstance (Rose Adapter)

The ObjectInstance class exposes a set of properties and methods that:

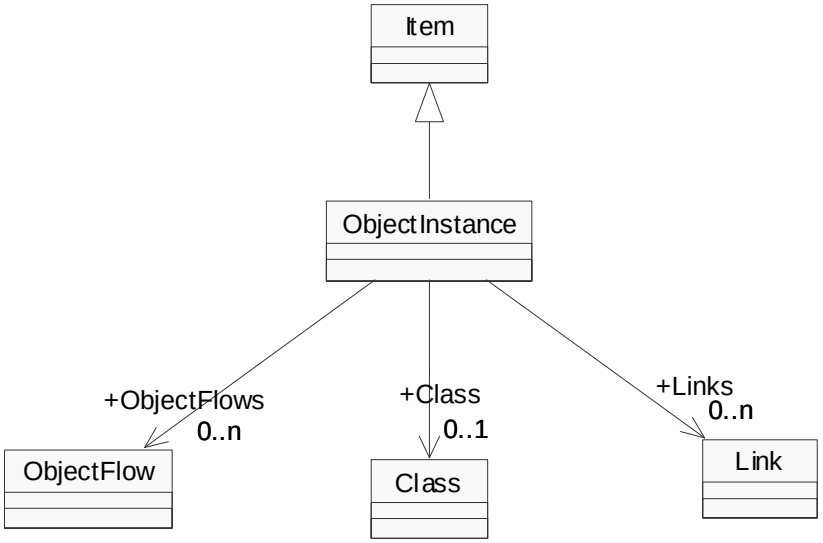
- Determine the characteristics of objects in a model (for example, the class associated with the object and whether multiple instances of the object exist)
- Allow you to retrieve objects from a model

Class Hierarchy: Artifact>Item>ObjectInstance

SubClasses of ObjectInstance

ObjectInstance has no subclasses.

Class Diagram



Attributes Specific to ObjectInstance

Attributes	Inherited From	Description
Documentation	Item	The documentation for the item
IsClass		TRUE if the ObjectInstance is a class
MultipleInstances		True if the Multiple Instances box is checked; otherwise False
MyClassName		The ObjectInstance class name
Name	Item	The name of the item
Persistence		Persistent, Static, or Transient depending on the value of the Persistence radio control in the object specification.
QualifiedName	Item	The qualified name of the item
Stereotype	Item	The item's stereotype
UniqueID	Item	The unique id of the item

Relationships Specific to ObjectInstance (see also class diagram above)

Name	Kind	Class	Documentation
Class	0..1	Class	The class of the object.
Links	0..n	Link	The links associated with the object.
ObjectFlows	0..n	ObjectFlow	The associated ObjectFlows with this ObjectInstance

Class: Operation (Rose Adapter)

Operations denote services provided by the class. Operations can be methods for accessing and modifying class fields or methods that implement characteristic behaviors of a class.

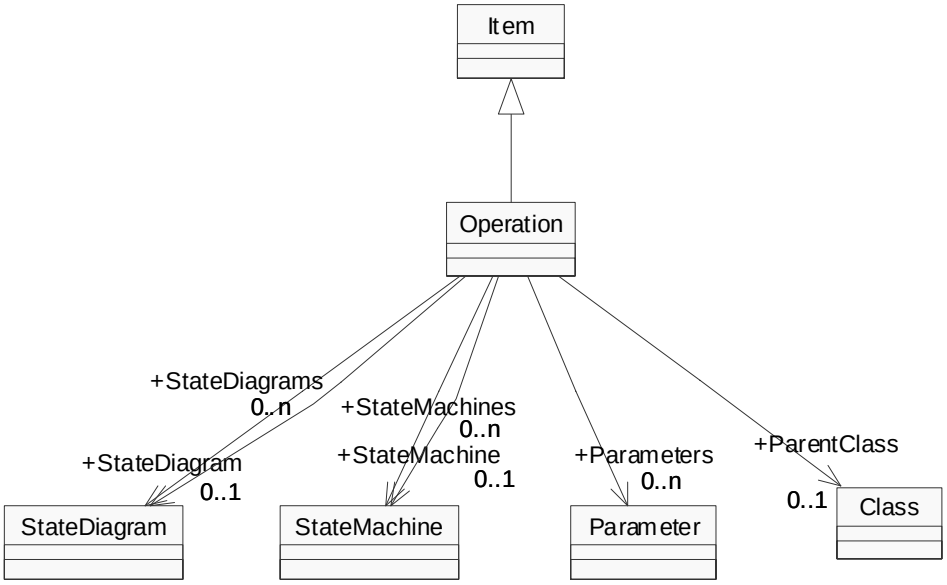
The operations of a class are listed in the Operations list box in the class specification. Rational Rose stores operation information in an operation specification. You can access operation specifications only through the class specification.

Class Hierarchy: Artifact>Item>Operation

SubClasses of Operation

Operation has no subclasses.

Class Diagram



Attributes Specific to Operation

Attributes	Inherited From	Description
AdaImage		An Ada code segment that represents the declaration of the operation. This image is derived from the operation name and the operation parameters. Although the AdaImage is semantically consistent with your actual code, it may differ in terms of format, depending on the rules and styles you use for code generation and/or reverse engineering.
COMImage		A COM code segment that represents the declaration of the operation.
Concurrency		Denotes the semantics of the operation in the presence of multiple threads of control. Returns Sequential, Guarded, or Synchronous, depending on the state of the Concurrency radio control in the More dialog of the operation specification.
CppImage		A C++ code segment that represents the prototype of the operation. This image is derived from the operation name and the operation parameters. Although the C++Image is semantically consistent with your actual code, it may differ in terms of format, depending on the rules and styles you use for code generation and/or reverse engineering.
Documentation	Item	The documentation for the item
Exceptions		Textual list of the exceptions that can be raised by the operation. The Exceptions text field appears in the More dialog of the operation specification.

Attributes	Inherited From	Description
ExportControl		Specifies the type of access allowed by the class for this operation. Will return Public, Protected, Private, or Implementation, depending on the state of the Export Control radio control in the operation specification.
HasStateDiagram		TRUE if the operation has an associated StateDiagram
JavaImage		A Java code segment that represents the declaration of the operation.
Name	Item	The name of the item
Postconditions		Text describing the post-conditions of the operation. The PostText is that text which appears in the Dynamic Semantics field of the operation specification when the Post radio button is selected.
Preconditions		Text describing the preconditions of the operation. The PreText is that text which appears in the Dynamic Semantics field of the operation specification when the Pre radio button is selected.
Protocol		The Protocol field lists a set of operations that a client may perform on an object and the legal orderings in which they may be invoked. The protocol of an operation has no semantic impact. The Protocol text field appears in the More dialog of the operation specification.
Qualification		Identifies language-specific features that allow you to qualify the method. The Qualification text field appears in the More dialog of the operation specification.

Attributes	Inherited From	Description
QualifiedName	Item	The qualified name of the item
ReturnType		For operations that are functions, refers to the class that is returned by the function. The ReturnClass text field appears in the Return Class field on the operation specification.
Semantics		Text describing the action of the main operation. The SemanticsText is that text which appears in the Dynamic Semantics field of the operation specification when the Semantics radio button is selected.
Size		Text describing the size of the class.
Stereotype	Item	The item's stereotype
Time		A statement about the relative or absolute time required to complete an operation. The Time text field appears in the More dialog of the operation specification.
UMLImage		The image of the operation and parameters using UML standard notation.
UniqueID	Item	The unique id of the item

Relationships Specific to Operation (see also class diagram above)

Name	Kind	Class	Documentation
Parameters	0..n	Parameter	The formal parameters of the operation. These appear in the Arguments list box in the operation specification.

Name	Kind	Class	Documentation
ParentClass	0..1	Class	The class to which this operation belongs.
StateMachine	0..1	StateMachine	The top-level state machine associated with this operation.
StateDiagram	0..1	StateDiagram	The top-level state diagram associated with this operation.
StateMachines	0..n	StateMachine	All state machines associated with this operation.
StateDiagrams	0..n	StateDiagram	All state diagrams associated with this operation.

Class: Package (Rose Adapter)

Packages serve to partition the logical model of a system. They are clusters of highly related classes that are themselves cohesive, but are loosely coupled relative to other such clusters. You can use packages to group classes and other packages. Rational Rose stores data describing the package in a package specification.

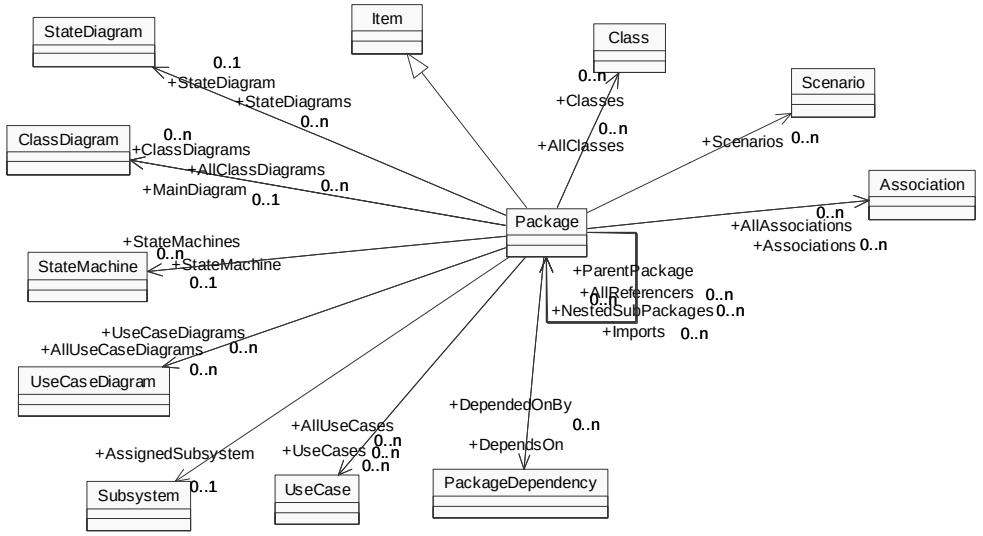
Note: When you create an OPEN command directly to a package, be sure to specify the name of the .mdl file and the name of the package, even if the package is contained in a separate .cat file.

Class Hierarchy: Artifact>Item>Package

SubClasses of Package

Package has no subclasses.

Class Diagram



Attributes Specific to Package

Attributes	Inherited From	Description
Documentation	Item	The documentation for the item
Global		True if the package is global, otherwise False.
HasAssignedSub-system		True if the package has a subsystem associated with it, otherwise False.
HasStateDiagram		True if the package has a state/activity diagram.
IsUseCasePack-age		True if the package is a descendent of the UseCase View package, otherwise False.
Name	Item	The name of the item
QualifiedName	Item	The qualified name of the item
Stereotype	Item	The item's stereotype
UniqueID	Item	The unique id of the item

Relationships Specific to Package (see also class diagram above)

Name	Kind	Class	Documentation
MainDiagram	0..1	ClassDiagram	The diagram specifically called "Main."
StateDiagram	0..1	StateDiagram	The top-level state/activity dia-gram associated with this pack-age.
StateDia-grams	0..n	StateDiagram	All state diagrams associated with this package.
StateMachine	0..1	StateMachine	The top-level state machine associated with this package.

Name	Kind	Class	Documentation
StateMachines	0..n	StateMachine	All state machines associated with this package.
Classes	0..n	Class	All classes that are immediate members of this package. All member classes are returned, regardless of whether they appear on any diagrams.
UseCases	0..n	UseCase	All use cases that are immediate members of this package.
UseCaseDiagrams	0..n	UseCaseDiagram	The use-case diagrams contained within this package.
ClassDiagrams	0..n	ClassDiagram	All class diagrams that are immediate members of this package.
AllUseCaseDiagrams	0..n	UseCaseDiagram	All use case diagrams that are defined in this package, or in any nested packages.
AllClassDiagrams	0..n	ClassDiagram	All class diagrams that are defined in this package, or in any nested packages.
Imports		Package	All packages that are imported by this package. Does not include indirect dependencies. For example if A imports B and B imports C, A does not directly import C.
AllReferencers		Package	All packages that import this package. Does not include indirect referencers.
Relationships	0..n	Relationship	The relationships defined within this package.

Name	Kind	Class	Documentation
NestedSub-Packages		Package	All packages that are descendents of this package.
AllUseCases	0..n	UseCase	All use cases that are defined in this package, or in any nested packages.
ParentPackage		Package	The enclosing package. This relationship will result in an error if applied to the TopLevelCategory.
Scenarios	0..n	Scenario	
AllAssociations	0..n	Association	All associations that are defined in this package, or in any nested packages.
AllClasses	0..n	Class	All classes that are defined in this package, or in any nested packages.
SubPackages		Package	All packages that are immediate children of this package.
Associations	0..n	Association	All associations that are immediate members of this package.
AssignedSubsystem	0..1	Subsystem	The subsystem associated with this package, as specified in the package specification.
DependsOn	0..n	PackageDependency	Associates this package as the receiver package in a package dependency
DependedOnBy	0..n	PackageDependency	Associates this package as the supplier package in a package dependency

Class: PackageDependency (Rose Adapter)

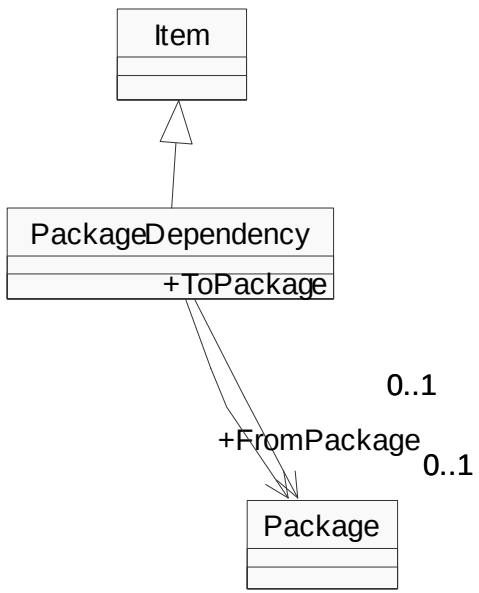
The package dependency indicates that one package in a model uses the services or facilities of another.

Class Hierarchy: Artifact>Item>PackageDependency

SubClasses of PackageDependency

PackageDependency has no subclasses.

Class Diagram



Attributes Specific to PackageDependency

Attributes	Inherited From	Description
Documentation	Item	The documentation for the item

Attributes	Inherited From	Description
Name	Item	The name of the item
QualifiedName	Item	The qualified name of the item
Stereotype	Item	The item's stereotype
SupplierName		The name of the package that is the supplier in the package dependency.
UniqueID	Item	The unique id of the item

Relationships Specific to PackageDependency (see also class diagram above)

Name	Kind	Class	Documentation
ToPackage	0..1	Package	The receiver package
FromPackage	0..1	Package	The supplier package

Class: Parameter (Rose Adapter)

Formal parameter of an operation, instantiated class, or instantiated class utility.

Class Hierarchy: Artifact>Item>Parameter

SubClasses of Parameter

Parameter has no subclasses.

Class Diagram

Attributes Specific to Parameter

Attributes	Inherited From	Description
Const		True if the parameter is constant; otherwise False
Documentation	Item	The documentation for the item
InitValue		The initial value of the parameter
Name	Item	The name of the item
QualifiedName	Item	The qualified name of the item
Stereotype	Item	The item's stereotype
Type		The type of the parameter.
UniqueID	Item	The unique id of the item

Relationships Specific to Parameter (see also class diagram above)

This class has no relationships.

Class: ParameterizedClass (Rose Adapter)

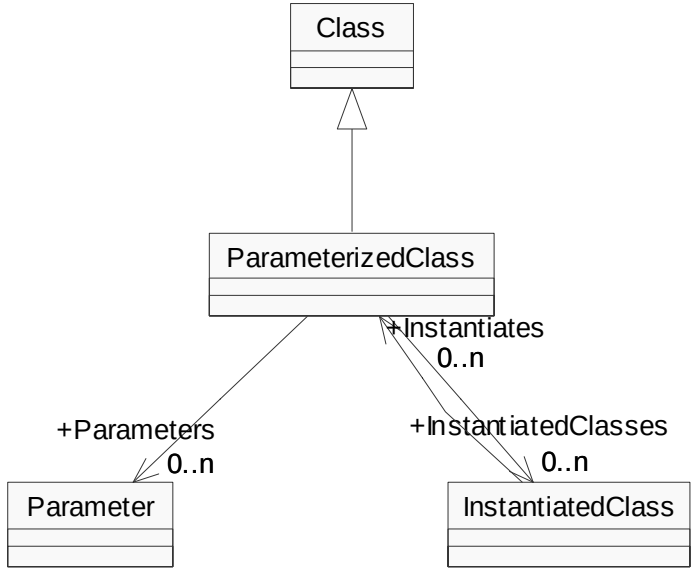
A parameterized class is a template for creating any number of instantiated classes that follow its format. A parameterized class declares formal parameters, which can be classes, objects, or operations.

Class Hierarchy: Item>Class>ParameterizedClass

SubClasses of ParameterizedClass

ParameterizedClass has no subclasses.

Class Diagram



Attributes Specific to ParameterizedClass

Attributes	Inherited From	Description
Abstract	Class	True if the Abstract check box is selected in the class specification, otherwise False.
Cardinality	Class	The string in the Cardinality field of the class specification.
Concurrency	Class	Returns Sequential, Guarded, Active, or Synchronous, depending on the value of the Concurrency radio control in the More dialog of the class specification.
Documentation	Item	The documentation for the item
ExportControl	Class	Returns Public or Implementation, depending on the value of the Export Control radio control in the class specification.
FundamentalType	Class	TRUE if this class is a fundamental type
HasStateDiagram	Class	True if the class has an associated state diagram, otherwise False.
IsNested	Class	True if the class is nested.
Kind	Class	The kind of class
Name	Item	The name of the item
Persistence	Class	This property is Persistent or Transient, depending on the value of the Persistence radio control in the More dialog of the class specification.
QualifiedName	Item	The qualified name of the item
Space	Class	The string in the Space field of the More dialog of the class specification.

Attributes	Inherited From	Description
Stereotype	Item	The item's stereotype
UniqueID	Item	The unique id of the item

Relationships Specific to ParameterizedClass (see also class diagram above)

Name	Kind	Class	Documentation
Parameters	0..n	Parameter	Formal, generic parameters declared by the parameterized class. The parameters appear in the Parameters list box in the More dialog of the class specification.
Instantiated-Classes	0..n	Instantiated-Class	All instantiated classes of this parameterized class.

Class: ParameterizedClassUtility (Rose Adapter)

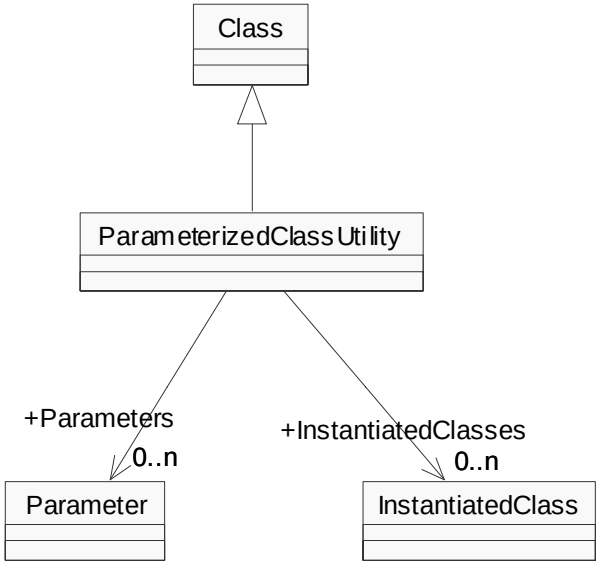
A parameterized class utility is a set of operations or functions that are not associated with a higher level class (free subprograms) and are defined in terms of formal parameters. Parameterized class utilities are used as templates for creating instantiated class utilities.

Class Hierarchy: Item>Class>ParameterizedClassUtility

SubClasses of ParameterizedClassUtility

ParameterizedClassUtility has no subclasses.

Class Diagram



Attributes Specific to ParameterizedClassUtility

Attributes	Inherited From	Description
Abstract	Class	True if the Abstract check box is selected in the class specification, otherwise False.
Cardinality	Class	The string in the Cardinality field of the class specification.
Concurrency	Class	Returns Sequential, Guarded, Active, or Synchronous, depending on the value of the Concurrency radio control in the More dialog of the class specification.
Documentation	Item	The documentation for the item
ExportControl	Class	Returns Public or Implementation, depending on the value of the Export Control radio control in the class specification.
FundamentalType	Class	TRUE if this class is a fundamental type
HasStateDiagram	Class	True if the class has an associated state diagram, otherwise False.
IsNested	Class	True if the class is nested.
Kind	Class	The kind of class
Name	Item	The name of the item
Persistence	Class	This property is Persistent or Transient, depending on the value of the Persistence radio control in the More dialog of the class specification.
QualifiedName	Item	The qualified name of the item
Space	Class	The string in the Space field of the More dialog of the class specification.

Attributes	Inherited From	Description
Stereotype	Item	The item's stereotype
UniqueID	Item	The unique id of the item

Relationships Specific to ParameterizedClassUtility (see also class diagram above)

Name	Kind	Class	Documentation
Parameters	0..n	Parameter	Formal, generic parameters declared by the parameterized class utility. The parameters appear in the Parameters list box in the More dialog of the class specification.
Instantiated-Classes	0..n	Instantiated-Class	All instantiated class utilities of this parameterized class utility.

Class: Process (Rose Adapter)

A process transforms data values. Lowest-level processes are pure functions without side effects.

Class Hierarchy: Artifact>Item>Process

SubClasses of Process

Process has no subclasses.

Class Diagram

Attributes Specific to Process

Attributes	Inherited From	Description
Documentation	Item	The documentation for the item
Name	Item	The name of the item
Priority		The priority of the process.
QualifiedName	Item	The qualified name of the item
Stereotype	Item	The item's stereotype
UniqueID	Item	The unique id of the item

Relationships Specific to Process (see also class diagram above)

This class has no relationships.

Class: Processor (Rose Adapter)

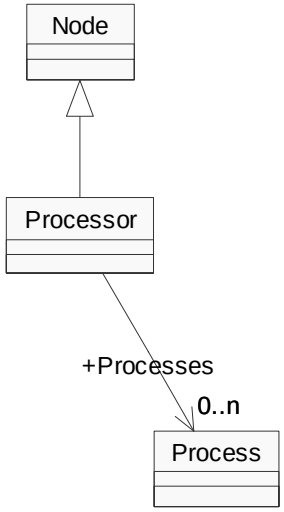
A processor is a hardware component capable of executing programs.

Class Hierarchy: Item>Node>Processor

SubClasses of Processor

Processor has no subclasses.

Class Diagram



Attributes Specific to Processor

Attributes	Inherited From	Description
Characteristics	Node	The characteristics of the processor or device.

Attributes	Inherited From	Description
Documentation	Item	The documentation for the item
Name	Item	The name of the item
QualifiedName	Item	The qualified name of the item
Scheduling		The text in the Scheduling field of the processor specification.
Stereotype	Item	The item's stereotype
UniqueID	Item	The unique id of the item

Relationships Specific to Processor (see also class diagram above)

Name	Kind	Class	Documentation
Processes	0..n	Process	The processes defined by this processor.

Class: Property (Rose Adapter)

A code-generation property associated with the model, a package, a subsystem, a class, an association, a has relationship, an attribute, a module, or an operation.

Class Hierarchy: Artifact>Property

SubClasses of Property

Property has no subclasses.

Class Diagram

Attributes Specific to Property

Attributes	Inherited From	Description
Name		The name of the property.
ParentUID		The unique id of this property's parent artifact type
ToolName		The name of the tool, or tab, for the property, such as cg or DDL .
Value		The string equivalent of the value associated with the property.

Relationships Specific to Property (see also class diagram above)

This class has no relationships.

Class: RealizeRelationship (Rose Adapter)

A realize relationship between a logical class and a component class shows that the component class realizes the operations defined by the logical class.

Class Hierarchy: Item>Relationship>RealizeRelationship

SubClasses of RealizeRelationship

RealizeRelationship has no subclasses.

Class Diagram

Attributes Specific to RealizeRelationship

Attributes	Inherited From	Description
ClientCardinality	Relationship	Indicates the number of possible links from an instance of the client class to an instance of the supplier class. Can be the same values as those listed in CardinalityFrom above.
Documentation	Item	The documentation for the item
ExportControl	Relationship	Specifies the type of access allowed between classes. Returns Public, Protected, Private, or Implementation, depending on the state of the Access radio control on the relationship specification. Access is also shown by adornments on relationships in diagrams.
Kind	Relationship	Kind of the relationship, which will be one of: AggregateRole, AssociationRole, HasRelationship, InheritsRelationship or UsesRelationship.
Name	Item	The name of the item
QualifiedName	Item	The qualified name of the item
Stereotype	Item	The item's stereotype

Attributes	Inherited From	Description
SupplierCardinal-ity	Relationship	Indicates the number of possible links from an instance of the supplier class to an instance of the client class. Can be one the following values: n, 1, 0..n, 1..n, 0..1, <literal>, <literal>..n, or <lit-eral>..<literal>.
SupplierName	Relationship	The name of the supplier class or use case.
UniqueID	Item	The unique id of the item

Relationships Specific to RealizeRelationship (see also class diagram above)

This class has no relationships.

Class: Relationship (Rose Adapter)

A semantic connection between two classes. Rational Rose stores relationship information in a relationship specification.

Class Hierarchy: Relationship

Subclasses of Relationship:

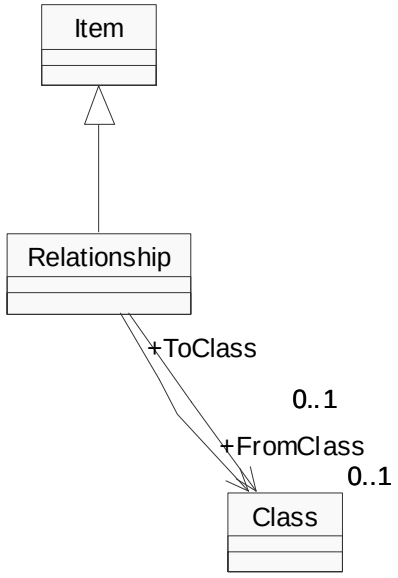
HasRelationship, InheritsRelationship, Role, UsesRelationship, Rose RealizeRelationship Class.

Class Hierarchy: Artifact>Item>Relationship

SubClasses of Relationship

HasRelationship InheritRelationship ObjectFlow RealizeRelationship Role UsesRelationship

Class Diagram



Attributes Specific to Relationship

Attributes	Inherited From	Description
ClientCardinality		Indicates the number of possible links from an instance of the client class to an instance of the supplier class. Can be the same values as those listed in CardinalityFrom above.
Documentation	Item	The documentation for the item
ExportControl		Specifies the type of access allowed between classes. Returns Public, Protected, Private, or Implementation, depending on the state of the Access radio control on the relationship specification. Access is also shown by adornments on relationships in diagrams.
Kind		Kind of the relationship, which will be one of: AggregateRole, AssociationRole, HasRelationship, InheritsRelationship or UsesRelationship.
Name	Item	The name of the item
QualifiedName	Item	The qualified name of the item
Stereotype	Item	The item's stereotype
SupplierCardinality		Indicates the number of possible links from an instance of the supplier class to an instance of the client class. Can be one the following values: n, 1, 0..n, 1..n, 0..1, <literal>, <literal>..n, or <literal>..<literal>.
SupplierName		The name of the supplier class or use case.
UniqueID	Item	The unique id of the item

Relationships Specific to Relationship (see also class diagram above)

Name	Kind	Class	Documentation
ToClass	0..1	Class	The supplier class. For example, if A Has a B, B is the supplier, or To class.
FromClass	0..1	Class	The client class. For example, if A Has a B, A is the client, or From class.

Class: Role (Rose Adapter)

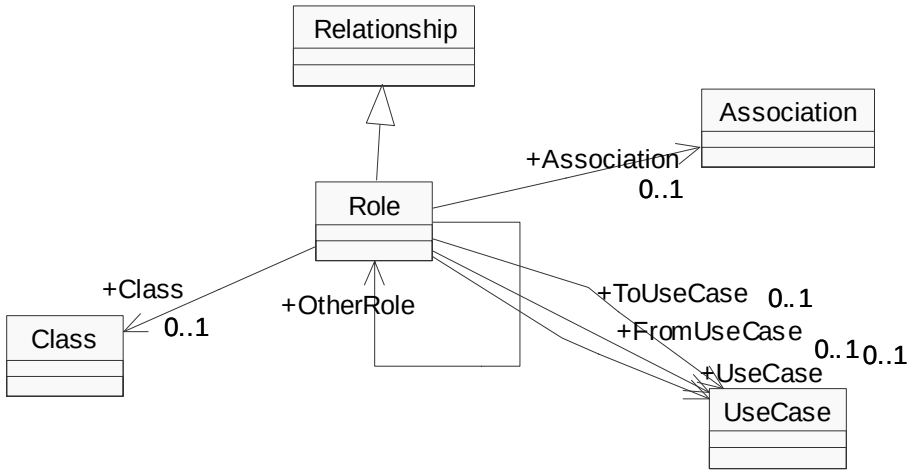
The purpose or capacity where one class associates with another.

Class Hierarchy: Item>Relationship>Role

SubClasses of Role

Role has no subclasses.

Class Diagram



Attributes Specific to Role

Attributes	Inherited From	Description
Aggregate		True if the role is an aggregate relationship.
Cardinality		The cardinality of this role

Attributes	Inherited From	Description
ClientCardinality	Relationship	Indicates the number of possible links from an instance of the client class to an instance of the supplier class. Can be the same values as those listed in CardinalityFrom above.
Constraints		The text of the Contraints field in the role specification.
Containment		Specifies the physical containment of the role. Returns Value, Reference, or Unspecified, depending on the state of the Containment radio control on the role specification.
Documentation	Item	The documentation for the item
ExportControl	Relationship	Specifies the type of access allowed between classes. Returns Public, Protected, Private, or Implementation, depending on the state of the Access radio control on the relationship specification. Access is also shown by adornments on relationships in diagrams.
Friend		True if the Friend check box is selected in the role specification, otherwise False.
Kind	Relationship	Kind of the relationship, which will be one of: AggregateRole, AssociationRole, HasRelationship, InheritsRelationship or UsesRelationship.
Name	Item	The name of the item
Navigable		True if the Navigable check box is selected, otherwise False.
QualifiedName	Item	The qualified name of the item

Attributes	Inherited From	Description
Static		True if the Static check box is selected in the role specification, otherwise False.
Stereotype	Item	The item's stereotype
SupplierCardinal-ity	Relationship	Indicates the number of possible links from an instance of the supplier class to an instance of the client class. Can be one the following values: n, 1, 0..n, 1..n, 0..1, <literal>, <literal>..n, or <lit-eral>..<literal>.
SupplierName	Relationship	The name of the supplier class or use case.
UniqueID	Item	The unique id of the item

Relationships Specific to Role (see also class diagram above)

Name	Kind	Class	Documentation
ToUseCase	0..1	UseCase	The client use case of the inher-its relationship, if it is a use case.
Keys	0..n	(Attribute)	Each key is an attribute that uniquely defines a single target object.
Class	0..1	Class	The class associated with this role
UseCase	0..1	UseCase	The use case associated with this role
FromUseCase	0..1	UseCase	The supplier use case of the role, if it is a use case.

Name	Kind	Class	Documentation
OtherRole		Role	The role at the other end of the association.
Association	0..1	Association	The association that this role is a part of.

Class: Scenario (Rose Adapter)

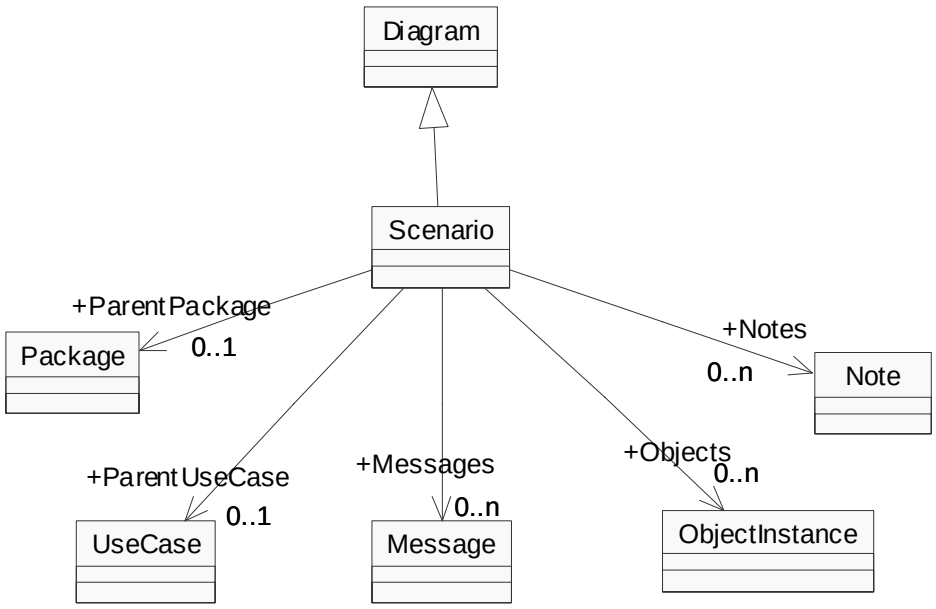
A scenario is an instance of a use case; it is an outline of events that occur during system execution.

Class Hierarchy: Artifact>Diagram>Scenario

SubClasses of Scenario

Scenario has no subclasses.

Class Diagram



Attributes Specific to Scenario

Attributes	Inherited From	Description
DiagramType		The diagram type of this scenario

Attributes	Inherited From	Description
Documentation	Diagram	The documentation text associated with the diagram.
MappedPoints	Diagram	
Name	Diagram	The name of the diagram.
ParentKind		The parent diagram of this scenario
QualifiedName	Diagram	The qualified name of the diagram
UniqueID	Diagram	The unique id for the diagram

Relationships Specific to Scenario (see also class diagram above)

Name	Kind	Class	Documentation
ParentPackage	0..1	Package	The parent package of this scenario
ParentUseCase	0..1	UseCase	The parent use case of this scenario
Objects	0..n	ObjectInstance	The object instances associated with the scenario
Messages	0..n	Message	The messages associated with this scenario
Notes	0..n	Note	The notes associated with this scenario

Class: State (Rose Adapter)

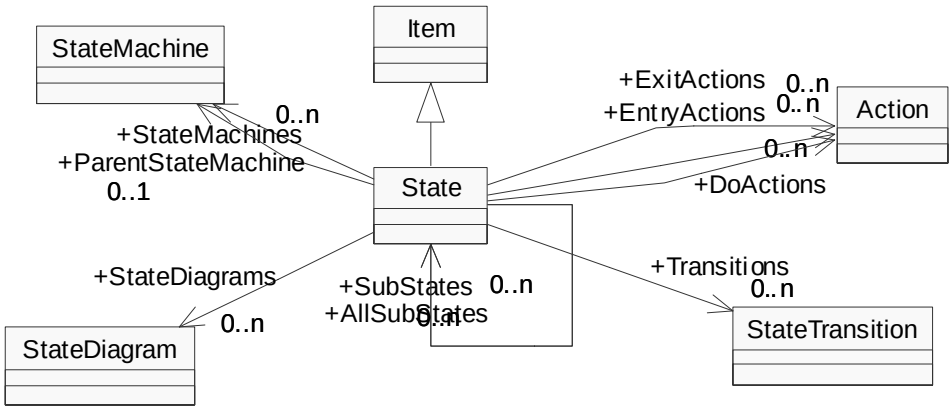
The state of an object represents the cumulative history of its behavior. State encompasses all of the object's static properties and the current values of each property.

Class Hierarchy: Artifact>Item>State

SubClasses of State

State has no subclasses.

Class Diagram



Attributes Specific to State

Attributes	Inherited From	Description
Documentation	Item	The documentation for the item
History		The text in the History field of the state specification.
Name	Item	The name of the item
QualifiedName	Item	The qualified name of the item

Attributes	Inherited From	Description
StateKind		One of Start, Normal or Stop.
Stereotype	Item	The item's stereotype
UniqueID	Item	The unique id of the item

Relationships Specific to State (see also class diagram above)

Name	Kind	Class	Documentation
Transitions	0..n	StateTransition	The transitions that exit from this state.
SubStates		State	The states that are part of this state.
SubActivities	0..n	(Activity)	The activities that are part of this state.
AllSubStates		State	All states that are associated with this state.
AllSubActivities	0..n	(Activity)	All activities that are associated with this state.
StateMachines	0..n	StateMachine	All state machines associated with this scenario
StateDiagrams	0..n	StateDiagram	All state diagrams associated with this scenario
EntryActions	0..n	Action	The Entry actions for this activity.
DoActions	0..n	Action	The Do actions for this activity.
ExitActions	0..n	Action	The Exit actions for this activity.

Name	Kind	Class	Documentation
ParentState-Machine	0..1	StateMachine	The top-level state machine associated with this scenario

Class: StateDiagram (Rose Adapter)

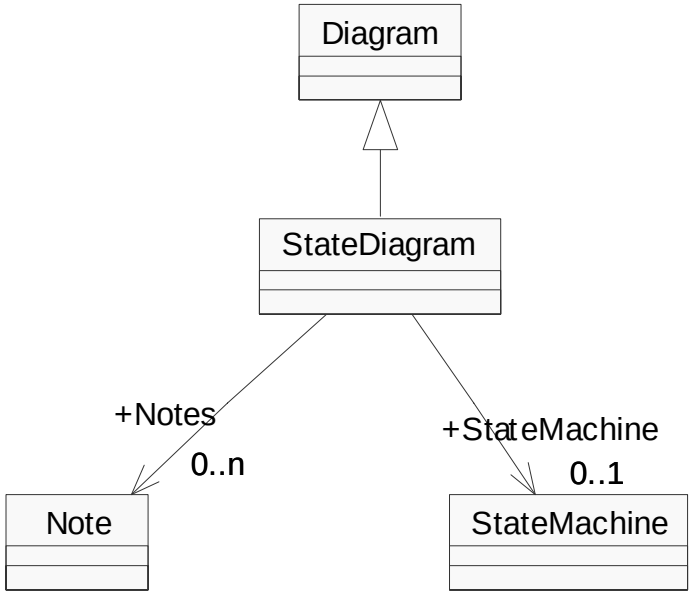
Depicts significant event-ordered behavior of a particular class. Each class may have one state diagram to describe its behavior.

Class Hierarchy: Artifact>Diagram>StateDiagram

SubClasses of StateDiagram

StateDiagram has no subclasses.

Class Diagram



Attributes Specific to StateDiagram

Attributes	Inherited From	Description
Documentation	Diagram	The documentation text associated with the diagram.
HasStateMachine		True if the diagram includes a state activity model.
IsActivityDiagram		TRUE if the StateDiagram is an activity diagram
MappedPoints	Diagram	
Name	Diagram	The name of the diagram.
QualifiedName	Diagram	The qualified name of the diagram
UniqueID	Diagram	The unique id for the diagram

Relationships Specific to StateDiagram (see also class diagram above)

Name	Kind	Class	Documentation
StateMachine	0..1	StateMachine	The top-level state machine associated with this diagram.
Notes	0..n	Note	The notes that appear in the diagram.

Class: StateMachine (Rose Adapter)

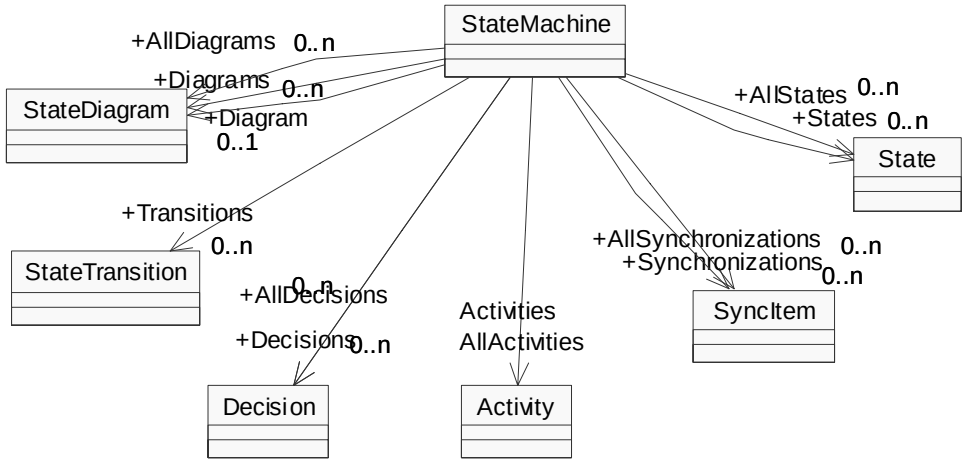
A state machine can be defined as a behavior that specifies the valid sequences of activities that an object or interaction goes through during its life in response to events, together with its responses and actions.

Class Hierarchy: Artifact>StateMachine

SubClasses of StateMachine

StateMachine has no subclasses.

Class Diagram



Attributes Specific to StateMachine

Attributes	Inherited From	Description
Documentation		The documentation for the StateMachine

Attributes	Inherited From	Description
HasDiagram		True if the state activity model has at least one state or activity diagram.
Name		The name of the state activity model.
Stereotype		The stereotype of the StateMachine
UniqueID		The internal unique identifier of the state activity model.

Relationships Specific to StateMachine (see also class diagram above)

Name	Kind	Class	Documentation
Diagram	0..1	StateDiagram	The (first) state or activity diagram associated with this state activity model.
Diagrams	0..n	StateDiagram	The state or activity diagrams associated with this state activity model.
AllDiagrams	0..n	StateDiagram	The diagrams defined in both this state activity model and all nested state activity models.
Transitions	0..n	StateTransition	The transitions that are part of this state activity model.
States	0..n	State	The states that are part of this state activity model.
Activities	0..n	Activity	The activities defined in this state activity model.
Decisions	0..n	Decision	The decisions defined in this state activity model.
Synchronizations	0..n	SyncItem	The synchronizations defined in this state activity model.

Name	Kind	Class	Documentation
AllStates	0..n	State	All states that are associated with this state activity model.
AllActivities	0..n	Activity	The activities defined in both this state activity model and all nested state activity models.
AllDecisions	0..n	Decision	The decisions defined in both this state activity model and all nested state activity models.
AllSynchroni-zations	0..n	SyncItem	The synchronizations defined in both this state activity model and all nested state activity models.
		Activity	

Class: StateTransition (Rose Adapter)

A state transition is a change of state caused by an event. Use state transitions to connect two states in a state diagram or show state transitions from a state to itself.

Class Hierarchy: Artifact>Item>StateTransition

SubClasses of StateTransition

StateTransition has no subclasses.

Class Diagram

Attributes Specific to StateTransition

Attributes	Inherited From	Description
CausingArguments		The arguments that accompany the causing event.
CausingEventName		The name of the event that causes this transition.
Documentation	Item	The documentation for the item
GuardCondition		
Name	Item	The name of the item
QualifiedName	Item	The qualified name of the item
SendArguments		The arguments that accompany the trigger event.
SendEventName		The name of the event triggered by the transition.
SendTarget		The name of the object that will receive the transition event.
Stereotype	Item	The item's stereotype
SupplierName		The name of the object that supplies the transition event.

Attributes	Inherited From	Description
UniqueID	Item	The unique id of the item

Relationships Specific to StateTransition (see also class diagram above)

Name	Kind	Class	Documentation
FromState	0..1	State	The state that this transition emanates from.
FromActivity	0..1	Activity	The activity that this transition emanates from.
SendAction	0..1	Action	The send action of this transition.
TriggerAction	0..1	Action	The action that triggers this transition.
ToState	0..1	State	The state that this transition leads to.
ToActivity	0..1	Activity	The activity that this transition leads to.
		Activity	

Class: Subsystem (Rose Adapter)

Subsystems represent clusters of logically related components. They parallel the role played by packages for class diagrams, allowing you to partition the physical model of the system.

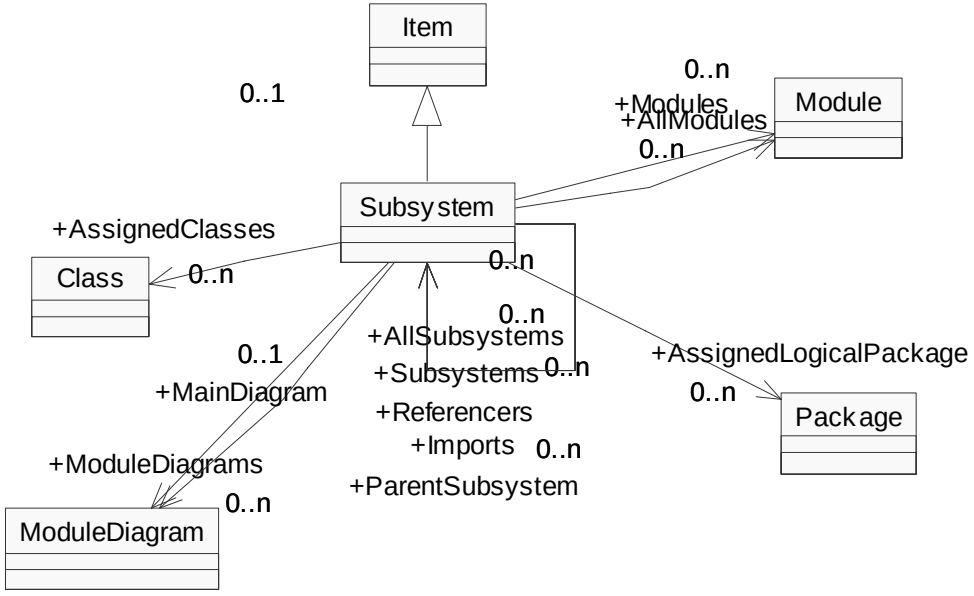
Each subsystem can contain components and other subsystems. Each module in your system must reside in a single subsystem or at the Component View of the model.

Class Hierarchy: Artifact>Item>Subsystem

SubClasses of Subsystem

Subsystem has no subclasses.

Class Diagram



Attributes Specific to Subsystem

Attributes	Inherited From	Description
Documentation	Item	The documentation for the item
Name	Item	The name of the item
QualifiedName	Item	The qualified name of the item
Stereotype	Item	The item's stereotype
UniqueID	Item	The unique id of the item

Relationships Specific to Subsystem (see also class diagram above)

Name	Kind	Class	Documentation
Assigned-Classes	0..n	Class	The classes assigned to this subsystem.
Imports		Subsystem	All other subsystems that this subsystem directly depends on. Does not include indirect dependences. For example if A imports B and B imports C, A does not directly import C.
ModuleDiagrams	0..n	ModuleDiagram	All module diagrams contained in this Subsystem.
Subsystems		Subsystem	The subsystems contained in this subsystem
AllModules	0..n	Module	All modules associated with this subsystem
MainDiagram	0..1	ModuleDiagram	All main module diagram contained in this Subsystem.
ParentSubsystem		Subsystem	The parent subsystem of this subsystem

Name	Kind	Class	Documentation
AllSubsystems		Subsystem	All subsystems associated with this subsystem
Referencers		Subsystem	All other subsystems that directly depend on this subsystem. Does not include indirect referencers. For example if A imports B and B imports C, A is not a direct referencer of C.
AssignedLogicalPackages	0..n	Package	The logical packages assigned to this subsystem.
Modules	0..n	Module	The models contained in this subsystem

Class: SyncItem (Rose Adapter)

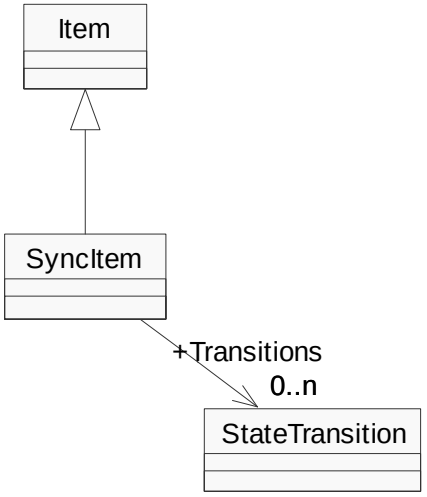
The SyncItem class is an abstract class that exposes Rose s synchronization functionality in the extensibility interface.

Class Hierarchy: Artifact>Item>SyncItem

SubClasses of SyncItem

SyncItem has no subclasses.

Class Diagram



Attributes Specific to SyncItem

Attributes	Inherited From	Description
Documentation	Item	The documentation for the item
Name	Item	The name of the item

Attributes	Inherited From	Description
QualifiedName	Item	The qualified name of the item
Stereotype	Item	The item's stereotype
UniqueID	Item	The unique id of the item

Relationships Specific to SyncItem (see also class diagram above)

Name	Kind	Class	Documentation
Transitions	0..n	StateTransi-tion	The state transitions associated with this SyncItem.

Class: UseCase (Rose Adapter)

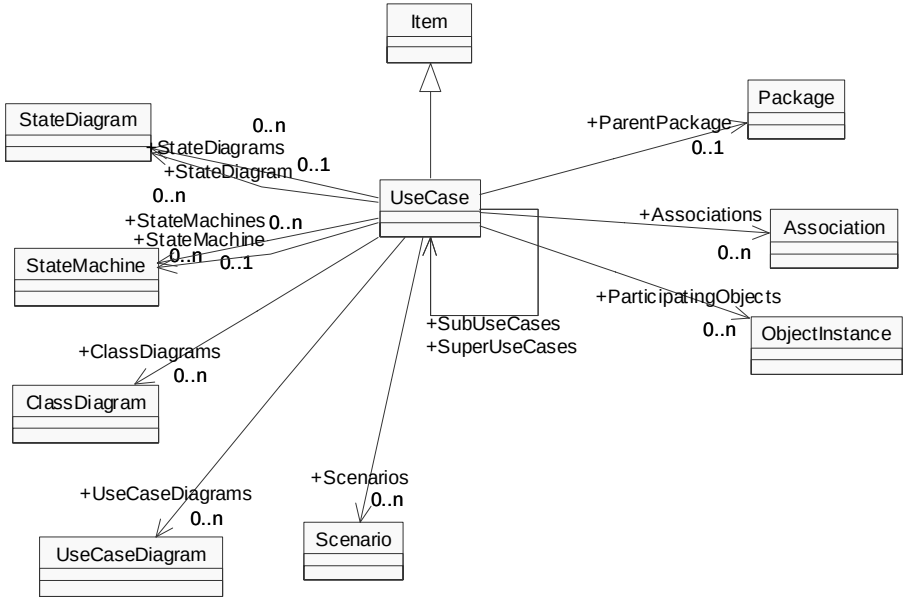
A use case is a sequence of transactions performed by a system in response to a triggering event initiated by an actor to the system. A full use case should provide a measurable value to an actor when the actor is performing a certain task. A use case contains all the events that can occur between an actor-use case pair, not necessarily the ones that will occur in any particular scenario. A use case contains a set of scenarios that explain various sequences of interaction within the transaction.

Class Hierarchy: Artifact>Item>UseCase

SubClasses of UseCase

UseCase has no subclasses.

Class Diagram



Attributes Specific to UseCase

Attributes	Inherited From	Description
Abstract		True if the abstract check-box is checked.
Documentation	Item	The documentation for the item
HasStateDiagram		True if the use case has an associated state diagram.
Name	Item	The name of the item
QualifiedName	Item	The qualified name of the item
Rank		The rank of the use case.
RequisiteProDoc-Name		The associated ReqPro ReqDocument name
RequisitePro-ProjectPath		The associated ReqPro Project path
RequisiteProReqt-GUID		The associated ReqPro Requirement GUID
Stereotype	Item	The item's stereotype
UniqueID	Item	The unique id of the item

Relationships Specific to UseCase (see also class diagram above)

Name	Kind	Class	Documentation
StateDiagram	0..1	StateDiagram	The top-level state diagram associated with this use case.
StateMachine	0..1	StateMachine	The top-level state machine associated with this use case.
ParentPack-age	0..1	Package	The enclosing package.

Name	Kind	Class	Documentation
ClassDia-grams	0..n	ClassDiagram	The class diagrams included in this use case.
StateDia-grams	0..n	StateDiagram	All state diagrams associated with this use case.
StateMa-chines	0..n	StateMachine	All state activity models associ-ated with this use case.
Scenarios	0..n	Scenario	The scenarios by this use case
Associations	0..n	Association	The associations where this use case plays a role.
UseCaseDia-grams	0..n	UseCaseDia-gram	The use-case diagrams associ-ated with this use case.
SuperUse-Cases		UseCase	The use cases that this use case inherits from directly.
SubUseCases		UseCase	The use cases that inherit from this use case.
MyRelation-ships	0..n	(Relationship)	The inherits and role relation-ships defined by this use case.
Participatin-gObjects	0..n	ObjectIn-stance	The objects included in scenar-ios defined by this use case.

Class: UseCaseDiagram (Rose Adapter)

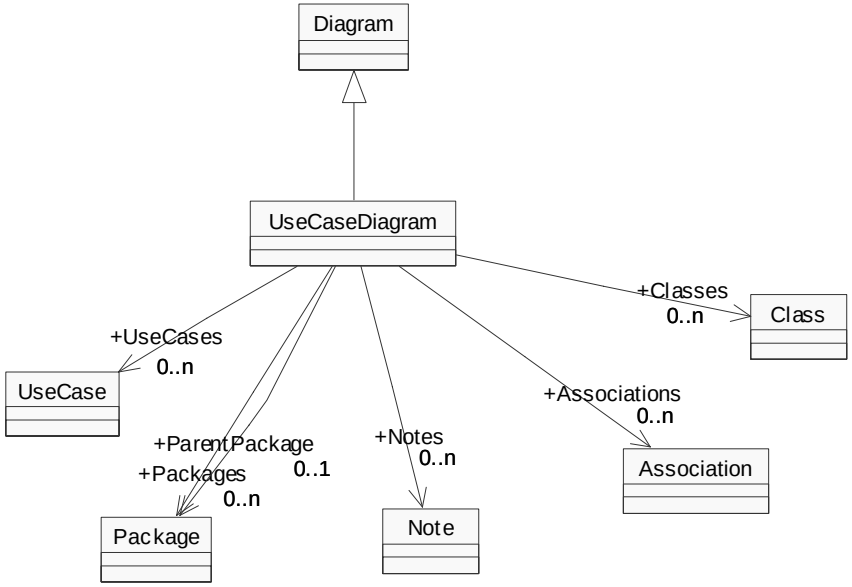
A use-case diagram shows the relationships between use cases and actors. Use-case diagrams can be considered as filtered views into the model. They do not necessarily depict all the use cases or relationships in the model. For example, iterating over all the use cases in the main diagram of a package will not necessarily return all the use cases defined in that package.

Class Hierarchy: Artifact>Diagram>UseCaseDiagram

SubClasses of UseCaseDiagram

UseCaseDiagram has no subclasses.

Class Diagram



Attributes Specific to UseCaseDiagram

Attributes	Inherited From	Description
Documentation	Diagram	The documentation text associated with the diagram.
MappedPoints	Diagram	
Name	Diagram	The name of the diagram.
QualifiedName	Diagram	The qualified name of the diagram
UniqueID	Diagram	The unique id for the diagram

Relationships Specific to UseCaseDiagram (see also class diagram above)

Name	Kind	Class	Documentation
UseCases	0..n	UseCase	All of the use cases that appear on the diagram.
Packages	0..n	Package	All of the packages that appear on the diagram.
Classes	0..n	Class	All of the classes that appear on the diagram.
Relationships	0..n	(Relationship)	All of the relationships that appear on the diagram.
Associations	0..n	Association	The associations where this use case diagram plays a role.
ParentPack-age	0..1	Package	The package that contains the diagram, if applicable.
Notes	0..n	Note	All of the notes associated with the diagram.

Class: UsesRelationship (Rose Adapter)

Indicates that the client class depends on the supplier class to provide certain services, such as:

The client class accesses a value (constant or variable) defined in the supplier class

Operations of the client class invoke operations of the supplier class

Operations of the client class have signatures whose return class or arguments are instances of the supplier class

Class Hierarchy: Item>Relationship>UsesRelationship

SubClasses of UsesRelationship

UsesRelationship has no subclasses.

Class Diagram

Attributes Specific to UsesRelationship

Attributes	Inherited From	Description
ClientCardinality	Relationship	Indicates the number of possible links from an instance of the client class to an instance of the supplier class. Can be the same values as those listed in CardinalityFrom above.
Documentation	Item	The documentation for the item
ExportControl	Relationship	Specifies the type of access allowed between classes. Returns Public, Protected, Private, or Implementation, depending on the state of the Access radio control on the relationship specification. Access is also shown by adornments on relationships in diagrams.

Attributes	Inherited From	Description
InvolvesFriend-ship		Indicates whether the supplier class grants rights to the client class to access its non-public parts. Returns True, if the Friendship required check box is checked on the relationship specification. Otherwise, returns False.
Kind	Relationship	Kind of the relationship, which will be one of: AggregateRole, AssociationRole, HasRelationship, InheritsRelationship or UsesRelationship.
Name	Item	The name of the item
QualifiedName	Item	The qualified name of the item
Stereotype	Item	The item's stereotype
SupplierCardinal-ity	Relationship	Indicates the number of possible links from an instance of the supplier class to an instance of the client class. Can be one the following values: n, 1, 0..n, 1..n, 0..1, <literal>, <literal>..n, or <literal>..<literal>.
SupplierName	Relationship	The name of the supplier class or use case.
UniqueID	Item	The unique id of the item

Relationships Specific to UsesRelationship (see also class diagram above)

This class has no relationships.

Rose Realtime (not an RSE adapter)

For this release, the Rose Realtime domain behaves exactly the same as in previous releases.

The Rose RealTime source domain allows you to incorporate textual and graphical information from Rational Rose RealTime models. To extract information from a Rose RealTime model, you would typically create an OPEN command to the model specifying its filename. Once this command provides context for the model, you can traverse through the various components.

Before generating a document from a Rose RealTime model, you need to save the model. When you generate the document, the Rose RealTime domain will create a directory named <document name prefix>.dia and will fill it with .WMF files for each diagram requested from the model.

The Rose RealTime domain uses aliases to support multiple notations. SoDA is delivered with UML aliases. To change to use aliases for another notation or another language, simply modify the RoseRT.dom file.

Displaying the Contents of Files Referenced by ExternalDocs

The Files tab in most Rose RealTime specifications is for External Documents. In this tab you can identify one or more documents that further describe the model element. Follow these steps to include the contents of these documents in your SoDA document or report.

1. Within the context of a model element, create a **REPEAT** command and select the **ExternalDocs** relationship; set the **Name** to **ExternalDoc**.
2. Just inside the **REPEAT** command, create an **OPEN** command.
3. In the **Select Class** area, choose **Word** -> **WordFile**.
4. Click the **Advanced** button.
5. In the **Argument** area, click **Filename** twice to show a tree control next to the argument.
6. In the tree control, select **ExternalDoc** -> **Value**; click **OK** to create the **OPEN** command.
7. Just to the right of the **OPEN** command, create a **DISPLAY** command.
8. In the **Select Attribute** area, choose **WordFile** -> **FormattedText**.

Class: Action (Rose RealTime)

Action is a subclass of ModelElement Class.

Subclasses of Diagram Class:

LocalState Class, RequestAction Class, ResponseAction Class, Coregion Class, CreateAction Class, DestroyAction Class, TerminateAction Class, UninterpretedAction Class.

Attributes specific to Action

Name - Kind
Kind - text
Time - text

Relationships specific to Action

Name - Kind	Class
Arguments - n	SString
ParentMessage - 1	Message
ParentState - 1	State
ParentTransition - 1	Transition

Class: Association (Rose RealTime)

A RealTime association represents a semantic connection between two classes. Associations are bi-directional; they are the most general of all relationships and the most semantically weak.

Association is a subclass of ModelElement Class.

Subclasses of Association Class: AssociationRole.

Attributes specific to Association

Name – Kind	Description
IsDerived – text	True if the association is derived; otherwise False.

Relationships specific to Association

Name - Kind	Class	Description
AssociationClass - 1	Class	
EndA - 1	AssociationEnd	The first role defined in the association.
EndB - 1	AssociationEnd	The second role defined in the association.

Class: AssociationEnd (Rose RealTime)

AssociationEnd is a subclass of Relationship.

Attributes specific to AssociationEnd

Name – Kind
Constraints – text
Containment – text
IsAggregrate – text
IsFriend – text
IsNavigable – text
IsStatic – text
Multiplicity – text
Visibility – text

Relationships specific to AssociationEnd

Name – Kind	Class
Association – 1	Association
FromElement – 1	ModelElement FromElement
Classifier – 1	Classifier
From – 1	Classifier
Keys – n	Attribute
OtherAsociationEnd – 1	AsociationEnd
To - 1	Classifier
UseCase - 1	UseCase

Class: AssociationRole (Rose RealTime)

AssociationRole is a subclass of Association Class.

Attributes specific to AssociationRole

Name – Kind

BaseName – text

Multiplicity – text

Relationships specific to AssociationRole

Name - Kind	Class	Description
Base - 1	Association	
EndA - 1	AssociationEndRole	The first role defined in the association.
EndB - 1	AssociationEndRole	The second role defined in the association.
ParentCollaboration - 1	Collaboration	

Class: AssociationEndRole (Rose RealTime)

AssociationEndRole is a subclass of AssociationRole Class.

Attributes specific to AssociationEndRole

Name – Kind
Multiplicity – text

Relationships specific to AssociationEndRole

Name - Kind	Class
AssociationRole - 1	AssociationRole
Base - 1	AssociationRole

Class: Attribute (Rose RealTime)

Attributes are data members of a class whose type is not another class.

Attribute is a subclass of ModelElement Class.

Attributes specific to Attribute

Name – Kind	Description
Containment – text	Specifies the physical containment of the attribute. Returns Value, Reference, or Unspecified, depending on the state of the Containment radio control on the attribute specification.
InitialValue – text	The initial value of the attribute.
Derived – text	True if the Derived check box is selected in the attribute specification, otherwise False.
Scope – text	
Type – text	The type of the attribute.
Visibility – text	

Relationships specific to Attribute

Name - Kind	Class	Description
ParentClassifier - 1	Classifier	The class in which this attribute is defined.

Class: Class (Rose RealTime)

A class captures the common structure and common behavior of a set of objects. A class is an abstraction of real-world items. When these items exist in the real world, they are instances of the class, and referred to as objects. Rational Rose RealTime stores class information in a class specification.

Class is a subclass of Classifier Class.

Subclasses of Class:

ParameterizedClass, InstantiatedClass, ClassUtility, ParameterizedClassUtility, InstantiatedClassUtility, MetaClass.

Attributes specific to Class

Name – Kind	Description
Concurrency – text	Returns Sequential, Guarded, Active, or Synchronous, depending on the value of the Concurrency radio control in the More dialog of the class specification.
IsFundamentalType – text	
IsNestedClass – text	True if the class is nested.
Multiplicity – text	
Persistence – text	Returns Persistent or Transient, depending on the value of the Persistence radio control in the More dialog of the class specification.
Space – text	The string in the Space field of the More dialog of the class specification.
Type – text	

Relationships specific to Class

Name – Kind	Class	Description
AppearsIn – n	ClassDiagram	The class diagrams where this class appears.
Instances AppearIn – n	Interaction Diagram	The interaction diagrams that include instances of this class.

InstantiateRelationships – n	InstantiateRelationship	
NestedClasses – n	Class	The classes that are nested within this class.
ParentClass – 1	Class	The parent class of this class, if it is nested.

Class: ClassDiagram (Rose RealTime)

A RealTime class diagram shows the relationships between packages and classes; the essential relationships include association, inherits, has, and uses. Each class diagram provides a logical view of the current model.

Class diagrams contain icons representing packages and classes. Class diagrams can be considered as filtered views into the model. They do not necessarily depict all the classes or relationships in the model. For example, iterating over all the classes in the main diagram of a package will not necessarily return all the classes defined in that category.

ClassDiagram is a subclass of Diagram Class.

Attributes specific to ClassDiagram

None

Relationships specific to ClassDiagram

Name – Kind	Class	Description
Classes - n	Class	All of the classes that appear on the diagram.
Packages - n	LogicalPackage	
ParentPackage – 1	LogicalPackage	
UseCases - n	UseCase	All of the use cases that appear on the diagram.

Class: Classifier (Rose RealTime)

Classifier Class serves to partition the logical model of a system. They are clusters of highly related classes that are themselves cohesive, but are loosely coupled relative to other such clusters. You can use packages to group classes and other packages. Rational Rose RealTime stores data describing the package in a package specification.

Note: When you create an OPEN command directly to a package, be sure to specify the name of the .mdl file and the name of the package, even if the package is contained in a separate .cat file.

Classifier is a subclass of ModelElement Class.

Subclasses of Classifier Class:

Capsule, Class, Protocol, UseCase.

Attributes specific to Classifier

Name – Kind
IsAbstract – text
HasStateDiagram – text
IsSystemClass – text
Language – text
QualifiedName – text
Visibility – text

Relationships specific to Classifier

Name - Kind	Class	Description
AllAssociations - n	Association	All associations where this class plays a role, including those inherited from other classes.
AllAttributes - n	Attribute	All attributes of this class, including those inherited from other classes.
AllCollaborations - n	Collaboration	
AllOperations - n	Operation	All operations of this class, including those inherited from other classes.
AllRelationships - n	Relationship	All relationships of this class, including those inherited from other classes.

AllSubClasses - n	Classifier	All classes in the lineage of this class. For example, if A inherits from B and B inherits from C, then AllSubClasses of C would include B and A.
AllSuperClasses - n	Classifier	All classes in the ancestry of this class. For example, if A inherits from B and B inherits from C, then AllSuperClasses of A would include B and C.
Associations - n	Association	The associations where this class plays a role.
Attributes - n	Attribute	
Collaborations - n	Collaboration	
Instances - n	Object	The object instances of this class.
Operations - n	Operation	
ParentPackage - 1	Package	The enclosing package.
Relationships - n	Relationship	
StateDiagram - 1	StateDiagram	
StateMachine - 1	StateActivityModel	
SubClasses - n	Classifier	
SuperClasses - n	Classifier	

Class: ClassifierRole (Rose RealTime)

ClassifierRole is a subclass of ModelElement Class.

Subclasses of ClassifierRole Class: CapsuleRole Class.

Attributes specific to ModelElement

Name – Kind
ClassifierName – text
Multiplicity – text

Relationships specific to ModelElement

Name – Kind	Class
Classifier – 1	Classifier
ParentCollaboration – 1	Collaboration

Class: CallAction (Rose RealTime)

CallAction is a subclass of RequestAction Class.

Attributes specific to CallAction

Name - Kind
Operation - text

Relationships specific to CallAction

None

Class: Capsule (Rose RealTime)

Capsule is a subclass of Classifier Class.

Attributes specific to Capsule

None

Relationships specific to Capsule

Name - Kind	Class
Structure - 1	CapsuleStructure

Class: CapsuleRole (Rose RealTime)

CapsuleRole is a subclass of ClassifierRole Class.

Attributes specific to CapsuleRole

Name – Kind
Cardinality – text
Genericity – text
IsSubstitiutable – text

Relationships specific to CapsuleRole

Name – Kind	Class
Capsule – 1	Capsule
PortRoles – n	PortRole

Class: CapsuleStructure (Rose RealTime)

ModelElement is a subclass of Collaboration Class.

Attributes specific to CapsuleStructure

None

Relationships specific to CapsuleStructure

Name - Kind	Class
CapsuleRoles - n	CapsuleRole
Ports - n	Port

Class: ChoicePoint (Rose RealTime)

ChoicePoint is a subclass of StateVertex Class.

Attributes specific to ChoicePoint

Name – Kind
Condition – text

Relationships specific to ChoicePoint

Name - Kind	Class
FALSETransition - 1	Transition
InTransition - 1	Transition
TRUETransition - 1	Transition

Class: ClassUtility (Rose RealTime)

A class utility is a set of operations that provide additional functions for classes. Class utilities are used to:

- Denote one or more free subprograms
- Name a class that only provides static members and/or static member functions.

ClassUtility is a subclass of Class.

Attributes specific to ClassUtility

None

Relationships specific to ClassUtility

None

Class: Collaboration (Rose RealTime)

Collaboration is a subclass of ModelElement Class.

Attributes specific to Collaboration

None

Relationships specific to Collaboration

Name - Kind	Class
AssociationRoles - n	AssociationRole
ClassifierRoles - n	ClassifierRole
Connectors - n	Connectors
Diagram - 1	CollaborationDiagram
Interactions - n	Interaction
ParentClassifier - 1	Classifier
ParentLogicalPackage - 1	LogicalPackage

Class: CollaborationDiagram (Rose RealTime)

CollaborationDiagram is a subclass of Diagram Class.

Attributes specific to CollaborationDiagram

None

Relationships specific to CollaborationDiagram

None

Class: Component (Rose RealTime)

A building block for the physical structure of a system. A component can be one of the following: Main Program, Package Body, Subprogram, Package, Task Body, Generic Package, Task, Subprogram Body.

Component is a subclass of ModelElement Class.

Attributes specific to Component

Name – Kind	Description
CodeGenMakeDocumentation - text	
CodeGenMakeFlags – text	
CodeGenMakeName – text	
CodeGenMakeOverrides - text	
CodeGenMakeType – text	
CompilationMakeDocumentation - text	
CompilationMakeFlags – text	
CompilationMakeName – text	
CompilationMakeOverrides - text	
CompilationMakeType – text	
CompilerDocumentation – text	
CompilerFlags – text	
CompilerLibrary – text	
CompilerOverride – text	
DefaultArgs – text	
ExecutableFileName – text	
IsMultiThreaded – text	
LinkerDocumentation – text	
LinkerFlags – text	
LinkerOverride – text	
OutputPath – text	
Platform – text	
RTSDocumentation – text	Text from the Documentation field of the component specification.

RTSType – text	
TargetDescription – text	
TargetServicesLibrary – text	

Relationships specific to Component

Name – Kind	Class
ClassifierReferences – n	Classifier
Inclusions – n	Sstring
InclusionPaths – n	Sstring
PackageReferences – n	LogicalPackages
ParentComponentPackage – 1	ComponentPackage
Relationships – n	Relationship
TopCapsule – 1	Capsule
UserLibraries – n	Sstring
UserLibraryPaths – n	Sstring
UserSourceFiles – n	Sstring
UserObjectFiles – n	Sstring

Class: ComponentAggregation (Rose RealTime)

ComponentAggregation is a subclass of Relationship.

Attributes specific to ComponentAggregation

None

Relationships specific to ComponenetAggregation

Name - Kind	Class
From - 1	Component
To - 1	Component

Class: ComponentDependency (Rose RealTime)

ComponentDependency is a subclass of Relationship.

Attributes specific to ComponentDependency

None

Relationships specific to ComponentDependency

Name - Kind	Class
From - 1	Component
FromClass - 1	Class
FromComponentPackage - 1	ComponentPackage
To - 1	Component
ToClass - 1	Class
ToComponentPackage - 1	ComponentPackage

Class: ComponentDiagram (Rose RealTime)

A component diagram shows relationships between subsystems and components. Each component diagram provides a physical view of the current model. Each component diagram is contained by the subsystem enclosing the components it depicts.

ComponentDiagram is a subclass of Diagram Class.

Attributes specific to ComponentDiagram

None

Relationships specific to ComponentDiagram

Name - Kind	Class	Description
Components - n	Component	The components contained in the diagram.

Class: ComponentInstance (Rose RealTime)

ComponentInstance is a subclass of ModelElement Class.

Attributes specific to ComponentInstance

Name – Kind
AttachTargetObservability – text
ConsolePort – text
LoadDelay – text
LoadOrder – text
LogsPort – text
OperationMode – text
TargetObservabilityPort – text
UserParameters – text

Relationships specific to ComponentInstance

Name - Kind	Class
Component - 1	Component

Class: ComponentPackage (Rose RealTime)

ComponentPackage is a subclass of ModelElement Class.

Attributes specific to ComponentPackage

Name – Kind
IsRootPackage – text

Relationships specific to ComponentPackage

Name - Kind	Class
AllComponents - n	Component
AllComponentPackages - n	ComponentPackage
Components - n	Component
ComponentDiagrams - n	ComponentDiagram
ComponentPackages - n	ComponentPackage
ParentComponentPackage - 1	ComponentPackage

Class: Connector (Rose RealTime)

Connector is a subclass of ModelElement Class.

Attributes specific to Connector

Name – Kind
Cardinality – text
Delay – text

Relationships specific to Connector

Name - Kind	Class
Port1 - 1	Port
Port2 - 1	Port
PortRole1 - 1	PortRole
Port Role2 - 1	PortRole

Class: Coregion (Rose RealTime)

Coregion is a subclass of Action Class.

Attributes specific to Coregion

None

Relationships specific to Coregion

Name - Kind	Class
Messages - n	Message

Class: CreateAction (Rose RealTime)

CreateAction is a subclass of Action Class.

Attributes specific to CreateAction

Name - Kind
Operation - text

Relationships specific to CreateAction

None

Class: DeploymentDiagram (Rose RealTime)

A deployment diagram shows the allocation of processes to processors in the physical design of a system. A deployment diagram may represent all or part of the process architecture of a system.

DeploymentDiagram is a subclass of Diagram Class.

Attributes specific to DeploymentDiagram

None

Relationships specific to DeploymentDiagram

Name	Class	Description
Processors - n	Processor	The processors contained in the diagram.
Devices - n	Device	The devices contained in the diagram.

Class: DeploymentPackage (Rose RealTime)

DeploymentPackage is a subclass of ModelElement Class.

Attributes specific to DeploymentPackage

None

Relationships specific to DeploymentPackage

Name - Kind	Class
AllDevices - n	Device
AllProcessors - n	Processor
DeploymentDiagram - n	DeploymentDiagram

Class: DestroyAction (Rose RealTime)

DestroyAction is a subclass of Action Class.

Attributes specific to DestroyAction

None

Relationships specific to DestroyAction

None

Class: Device (Rose RealTime)

A device is a hardware component with no computing power.

Device is a subclass of ModelElement Class.

Attributes specific to Device

Name – Kind
Characteristics – text

Relationships specific to Device

Name - Kind	Class
ConnectedDevices - n	Device
ConnectedProcessors - n	Processor

Class: Diagram (Rose RealTime)

Diagram is a subclass of Element Class.

Subclasses of Diagram Class:

ClassDiagram, SequenceDiagram, DeploymentDiagram, CollaborationDiagram, StateDiagram, ComponentDiagram.

Attributes specific to Diagram

Name - Kind
Diagram graphic - image

Relationships specific to Diagram

Name – Kind	Class
ModelElements - n	ModelElement
NoteViews - n	NoteView Notes

Class: Element (Rose RealTime)

A Rose RealTime element file. A model file contains a Rose RealTime model, which describes your problem domain and system software. Model files use the default extension .mdl. Models are the highest hierarchical elements of the Rose RealTime source domain. Most templates will start with connections to a Model.

Subclasses of Element Class:

ModelElement, Diagram, StateMachine, Trigger.

Attributes specific to Element

Name – Kind
Name – text
UniqueID – text

Relationships specific to Element

Name – Kind	Class	Description
AllProperties - n	Property	
Model – 1	Model	
Properties - n	Property	The code-generation properties associated with the model.

Class: Environment (Rose RealTime)

Environment is a subclass of InteractionInstance Class.

Attributes specific to Environment

None

Relationships specific to Environment

None

Class: File (Rose RealTime)

Attributes specific to File

Name – Kind
IsURL – text
Value – text

Relationships specific to File

Name - Kind	Class
ParentLogicalPackage - 1	Package

Class: FinalState (Rose RealTime)

FinalState is a subclass of StateVertex Class.

Attributes specific to FinalState

None

Relationships specific to FinalState

None

Class: Generalization (Rose RealTime)

Generalization is a subclass of Relationship.

Attributes specific to Generalization

Name – Kind
FriendshipRequired – text
Visibility – text

Relationships specific to Generalization

Name - Kind	Class
From - 1	Classifier
To - 1	Classifier

Class: InitialPoint (Rose RealTime)

InitialPoint is a subclass of StateVertex Class.

Attributes specific to InitialPoint

None

Relationships specific to InitialPoint

None

Class: InstantiatedClass (Rose RealTime)

A class which instantiates a parameterized class. Instantiated classes are created by supplying the actual values for the formal parameters of the parameterized class. An instantiated class is concrete, meaning that its implementation is complete, and it may have object instances.

InstantiatedClass is a subclass of Class.

Attributes specific to InstantiatedClass

None

Relationships specific to InstantiatedClass

None

Class: InstantiatedClassUtility (Rose RealTime)

A class utility which instantiates a parameterized class utility. Instantiated class utilities are created by supplying the actual values for the formal parameters of the parameterized class utility.

InstantiatedClassUtility is a subclass of Class.

Attributes specific to InstantiatedClassUtility

None

Relationships specific to InstantiatedClassUtility

None

Class: InstantiateRelationship (Rose RealTime)

InstantiateRelationship is a subclass of Relationship.

Attributes specific to InstantiateRelationship

None

Relationships specific to InstantiateRelationship

Name - Kind	Class
From - 1	Class
To - 1	Class

Class: Interaction (Rose RealTime)

Interaction is a subclass of ModelElement Class.

Attributes specific to Interaction

None

Relationships specific to Interaction

Name - Kind	Class
Instances - n	InteractionInstance
Messages - n	Message
ParentCollaboration - 1	Collaboration
ParentProtocol - 1	Protocol
SequenceDiagram - 1	SequenceDiagram

Class: InteractionInstance (Rose RealTime)

InteractionInstance is a subclass of ModelElement Class.

Subclasses of InteractionInstance Class: Environment Class.

Attributes specific to InteractionInstance

None

Relationships specific to InteractionInstance

Name - Kind	Class
Messages - n	Message
Path - n	ClassifierRole
ParentInteraction - 1	Interaction

Class: JunctionPoint (Rose RealTime)

JunctionPoint is a subclass of StateVertex Class.

Attributes specific to JunctionPoint

Name – Kind
Continuation – text
IsEntry – text
IsExit – text
IsExternallyVisible – text

Relationships specific to JunctionPoint

None

Class: LocalState (Rose RealTime)

LocalState is a subclass of Action Class.

Attributes specific to LocalState

None

Relationships specific to LocalState

None

Class: Message (Rose RealTime)

Any message associated with an object.

Message is a subclass of ModelElement Class.

Attributes specific to Message

None

Relationships specific to Message

Name - Kind	Class	Description
Action - 1	Action	
Activator - 1	Message	
ParentInteraction - 1	Interaction	
Receiver - 1	InteractionInstance	The object that receives the message.
Sender - 1	Object	The object that sends the message.

Class: MetaClass (Rose RealTime)

A metaclass is a class whose instances are classes rather than objects. Metaclasses provide operations for initializing class variables and serve as repositories to hold class variables where a single value will be required by all objects of a class. Smalltalk and CLOS support the use of metaclasses. C++ does not directly support metaclasses.

MetaClass is a subclass of Class.

Attributes specific to MetaClass

None

Relationships specific to MetaClass

None

Class: Model (Rose RealTime)

A Rose RealTime model file. A model file contains a Rose RealTime model, which describes your problem domain and system software. Model files use the default extension .mdl. Models are the highest hierarchical elements of the Rose RealTime source domain. Most templates will start with connections to a Model.

Model is a subclass of File System File Class.

Attributes specific to Model

Name - Kind
FileName - text
Name – text
UniqueID - text
Documentation - text
Stereotype - text

Relationships specific to Model

Name – Kind	Class	Description
AllAssociations - n	Association	All associations in the model.
AllCapsules - n	Capsule	
AllClasses - n	Class	All classes in the model, including actors.
AllComponentPackages - n	ComponentPackage	
AllComponents - n	Component	All components in the model (including subsystems).
AllPackages - n	LogicalPackage	All packages in the model, including use-case packages (but not including subsystems in the Component View).
AllProperties - n	Property	
AllProtocols – n	Protocol	
AllRelationships - n	Relationship	All relationships in the model.
AllUseCases - n	UseCase	All use cases in the model.
ComponentView - 1	ComponentPackage	The highest-level subsystem in the model; its name is Component View. All other subsystems are nested beneath it.

DeploymentDiagram - 1	DeploymentPackage	The deployment diagram (process diagram) for the model.
Deployment View - 1	DeploymentPackage	
ExternalDocuments - n	ExternalDocument	
LogicalView - 1	LogicalPackage	The highest-level package in the model; its name is Logical View. All other packages are nested beneath it.
Model - 1	Model	
Properties - n	Property	The code-generation properties associated with the model.
UseCaseView - 1	Package	The root use-case package in the model; its name is Use Case View. All other use-case packages are nested beneath it.

Class: ModelElement (Rose RealTime)

A Rose RealTime model element file. A model file contains a Rose RealTime model, which describes your problem domain and system software. Model files use the default extension .mdl. Models are the highest hierarchical elements of the Rose RealTime source domain. Most templates will start with connections to a Model.

ModelElement is a subclass of Element Class.

Subclasses of ModelElement Class:

Action Class, Association Class, Attribute Class, Classifier Class, ClassifierRole Class, Component Class, ComponentPackage Class, ComponentInstance Class, Connector Class, DeploymentPackage Class, Device Class, Interaction Class, InteractionInstance Class, Message Class, Operation Class, Package Class, Parameter Class, PortRole Class, Processor Class, Relationship Class, Signal Class, StateVertex Class, Transition Class.

Attributes specific to ModelElement

Name – Kind
Documentation – text
Stereotype – text

Relationships specific to ModelElement

Name – Kind	Class
ExternalDocuments - n	ExternalDocument

Class: NoteView (Rose RealTime)

Attributes specific to NoteView

Name – Kind
Text – text
Type – text

Relationships specific to NoteView

Name - Kind	Class
ModelElement - 1	ModelElement
ParentDiagram - 1	Diagram

Class: Operation (Rose RealTime)

Operations denote services provided by the class. Operations can be methods for accessing and modifying class fields or methods that implement characteristic behaviors of a class.

The operations of a class are listed in the Operations list box in the class specification. Rational Rose RealTime stores operation information in an operation specification. You can access operation specifications only through the class specification.

Operation is a subclass of ModelElement Class.

Attributes specific to Operation

Name – Kind	Description
AdaImage – text	An Ada code segment that represents the declaration of the operation. This image is derived from the operation name and the operation parameters. Although the AdaImage is semantically consistent with your actual code, it may differ in terms of format, depending on the rules and styles you use for code generation and/or reverse engineering.
C++Image – text	A C++ code segment that represents the prototype of the operation. This image is derived from the operation name and the operation parameters. Although the C++Image is semantically consistent with your actual code, it may differ in terms of format, depending on the rules and styles you use for code generation and/or reverse engineering.
Concurrency – text	Denotes the semantics of the operation in the presence of multiple threads of control. Returns Sequential, Guarded, or Synchronous, depending on the state of the Concurrency radio control in the More dialog of the operation specification.
Exceptions – text	Textual list of the exceptions that can be raised by the operation. The Exceptions text field appears in the More dialog of the operation specification.
IsAbstract – text	
IsQuery – text	
IsVirtual – text	
PostConditions – text	Text describing the post-conditions of the operation. The PostText is that text which appears in the Dynamic Semantics field of the operation specification when the Post radio button is selected.
PreConditions – text	Text describing the preconditions of the operation. The PreText is that text which appears in the Dynamic Semantics field of the operation specification when the Pre radio button is selected.

Protocol – text	The Protocol field lists a set of operations that a client may perform on an object and the legal orderings in which they may be invoked. The protocol of an operation has no semantic impact. The Protocol text field appears in the More dialog of the operation specification.
Qualification – text	Identifies language-specific features that allow you to qualify the method. The Qualification text field appears in the More dialog of the operation specification.
ReturnClass – text	For operations that are functions, refers to the class that is returned by the function. The ReturnClass text field appears in the Return Class field on the operation specification.
Semantics – text	Text describing the action of the main operation. The SemanticsText is that text which appears in the Dynamic Semantics field of the operation specification when the Semantics radio button is selected.
Size – text	Text describing the size of the class.
Time – text	A statement about the relative or absolute time required to complete an operation. The Time text field appears in the More dialog of the operation specification.
UMLImage – text	The image of the operation and parameters using UML standard notation.
Visibility – text	

Relationships specific to Operation

Name - Kind	Class
Parameters - n	Parameters
ParentClassifier - 1	Classifier

Class: Package (Rose RealTime)

Packages serve to partition the logical model of a system. They are clusters of highly related classes that are themselves cohesive, but are loosely coupled relative to other such clusters. You can use packages to group classes and other packages. Rational Rose RealTime stores data describing the package in a package specification.

Note: When you create an OPEN command directly to a package, be sure to specify the name of the .mdl file and the name of the package, even if the package is contained in a separate .cat file.

Package is a subclass of ModelElement Class.

Attributes specific to Package

Name – Kind	Description
HasAssignedComponentPackage – text	True if the package has a subsystem associated with it, otherwise False.
IsGlobal – text	
IsRootPackage – text	
IsUseCasePackage – text	True if the package is a descendent of the Use Case View package, otherwise False.

Relationships specific to Package

Name – Kind	Class	Description
AllAssociations – n	Association	All associations that are defined in this package, or in any nested packages.
AllClasses – n	Class	All classes that are defined in this package, or in any nested packages.
AllUseCases – n	UseCase	All use cases that are defined in this package, or in any nested packages.
AssignedComponentPackage – 1	ComponentPackage	
Associations – n	Association	
Capsules – n	Capsule	
ClassDiagrams – n	ClassDiagram	All class diagrams that are immediate members of this package.

Classes - n	Class	All classes that are immediate members of this package. All member classes are returned, regardless of whether they appear on any diagrams.
Collaborations – n	Collaboration	
Imports – n	Package	All packages that are imported by this package. Does not include indirect dependencies. For example if A imports B and B imports C, A does not directly import C.
PackageDependencies	LogicalPackageDependencies	
MainDiagram - 1	ClassDiagram	The diagram specifically called “Main”.
NestedSubPackages – n	Package	All packages that are descendents of this package.
ParentPackage - 1	Package	The enclosing package. This relationship will result in an error if applied to the TopLevelCategory.
Protocols – n	Protocol	
Referencers – n	Package	All packages that import this package. Does not include indirect referencers.
SubPackages – n	Package	All packages that are immediate children of this package.
UseCases - n	UseCase	

Class: PackageDependency (Rose RealTime)

PackageDependency is a subclass of Relationship.

Attributes specific to PackageDependency

None

Relationships specific to PackageDependency

Name - Kind	Class
From - 1	Package
To - 1	Package

Class: Parameter (Rose RealTime)

Formal parameter of an operation, instantiated class, or instantiated class utility.

Parameter is a subclass of ModelElement Class.

Attributes specific to Parameter

Name – Kind	Description
InitValue – text	The initial value of the parameter
IsConst – text	True if the parameter is constant; otherwise False
Type – text	The type of the parameter.

Relationships specific to Parameter

None

Class: ParameterizedClass (Rose RealTime)

A parameterized class is a template for creating any number of instantiated classes that follow its format. A parameterized class declares formal parameters, which can be classes, objects, or operations.

ParameterizedClass is a subclass of Class.

Attributes specific to ParameterizedClass

None

Relationships specific to ParameterizedClass

Name - Kind	Class	Description
FormalArguments - n	Parameter	Formal, generic parameters declared by the parameterized class. The parameters appear in the Parameters list box in the More dialog of the class specification.

Class: ParameterizedClassUtility (Rose RealTime)

A parameterized class utility is a set of operations or functions that are not associated with a higher level class (free subprograms) and are defined in terms of formal parameters. Parameterized class utilities are used as templates for creating instantiated class utilities.

ParameterizedClassUtility is a subclass of Class.

Attributes specific to ParameterizedClassUtility

None

Relationships specific to ParameterizedClassUtility

Name - Kind	Class	Description
FormalArguments - n	Parameter	Formal, generic parameters declared by the parameterized class utility. The parameters appear in the Parameters list box in the More dialog of the class specification.

Class: Port (Rose RealTime)

Port is a subclass of ClassifierRole Class.

Attributes specific to Port

Name - Kind
Cardinality - text
Genericity - text
IsConjugated - text
IsEndPort - text
IsNotified - text
IsWired - text
RegistrationMode - text
RegistrationString - text
Visibility - text

Relationships specific to Port

Name - Kind	Class
Protocol - 1	Protocol

Class: PortRole (Rose RealTime)

PortRole is a subclass of ModelElement Class.

Attributes specific to PortRole

None

Relationships specific to PortRole

Name - Kind	Class
ParentCapsuleRole - 1	CapsuleRole
Port - 1	Port

Class: Processor (Rose RealTime)

A processor is a hardware component capable of executing programs.

Processor is a subclass of ModelElement Class.

Attributes specific to Processor

Name – Kind
Address – text
CPU – text
OS – text
ServerAddress – text
UserScriptDirectory – text

Relationships specific to Processor

Name - Kind	Class
ComponentInstances - n	ComponenetInstances
ConnectedDevices - n	Device
ConnectedProcessors - n	Processor

Class: Property (Rose RealTime)

A code-generation property associated with the model, a package, a subsystem, a class, an association, a relationship, an attribute, a module, or an operation.

Attributes specific to Property

Name – Kind	Description
Name – text	The name of the property.
PropertyType – text	
ToolName – text	The name of the tool, or tab, for the property, such as “cg” or “DDL”.
Type – text	
Value – text	The string equivalent of the value associated with the property.

Relationships specific to Property

None

Class: Protocol (Rose RealTime)

Protocol is a subclass of Classifier Class.

Attributes specific to Protocol

None

Relationships specific to Protocol

Name - Kind	Class
InSignals - n	Signal
Interactions - n	Interaction
OutSignals - n	Signal

Class: RealizeRelatio(Rose RealTime)

A realize relationship between a logical class and a component class shows that the component class realizes the operations defined by the logical class.

RealizeRelationship is a subclass of Relationship.

Attributes specific to RealizeRelationship

None

Relationships specific to RealizeRelationship

Name - Kind	Class
FromCapsule - 1	Class
FromClass - 1	Class
FromProtocol - 1	Class
ToClass - 1	Class
ToUseCase - 1	Class

Class: Relationship (Rose RealTime)

A semantic connection between two classes. Rational Rose RealTime stores relationship information in a relationship specification.

Relationship is a subclass of ModelElement Class.

Subclasses of Relationship Class:

UsesRelationship, RealizeRelationship, InstantiateRelationship, Generalization, PackageDependency, ComponentDependency, ComponentAggregation, AssociationEnd.

Attributes specific to Relationship

Name – Kind	Description
Kind – text	Kind of the relationship, which will be one of: AggregateRole, AssociationRole, HasRelationship, InheritsRelationship or UsesRelationship.
ToName – text	

Relationships specific to Relationship

Name – Kind	Class	Description
From - 1	ModelElement	The client class. For example, if A Has a B, A is the client, or From class.
To - 1	ModelElement	The supplier class. For example, if A Has a B, B is the supplier, or To class.

Class: ReplyAction (Rose RealTime)

ReplyAction is a subclass of ResponseAction Class.

Attributes specific to ReplyAction

Name - Kind
Data - text
Signal - text

Relationships specific to ReplyAction

None

Class: RequestAction (Rose RealTime)

RequetAction is a subclass of Action Class.

Subclasses of RequestAction Class:

CallAction Class, SendAction Class.

Attributes specific to RequestAction

Name - Kind
Mode - text

Relationships specific to RequestAction

Name - Kind	Class
Return - 1	ResponseAction

Class: ResponseAction (Rose RealTime)

ResponseAction is a subclass of Action Class.

Subclasses of ResponseAction Class:

ReturnAction Class, ReplyAction Class.

Attributes specific to ResponseAction

None

Relationships specific to ResponseAction

Name - Kind	Class
Request - 1	RequestAction

Class: ReturnAction (Rose RealTime)

ReturnAction is a subclass of ResponseAction Class.

Attributes specific to ReturnAction

None

Relationships specific to ReturnAction

None

Class: SendAction (Rose RealTime)

CallAction is a subclass of RequestAction Class.

Attributes specific to SendAction

Name - Kind
DeliveryTime - text
Priority - text
ReceiverPort - text
SenderPort - text
Signal - text

Relationships specific to SendAction

None

Class: SequenceDiagram (Rose RealTime)

SequenceDiagram is a subclass of Diagram Class.

Attributes specific to SequenceDiagram

None

Relationships specific to SequenceDiagram

None

Class: Signal (Rose RealTime)

Signal is a subclass of ModelElement Class.

Attributes specific to Signal

Name – Kind
DataClassName – text

Relationships specific to Signal

Name - Kind	Class
DataClass - 1	Class
ParentProtocol - 1	Protocol

Class: State (Rose RealTime)

The state of an object represents the cumulative history of its behavior. State encompasses all of the object's static properties and the current values of each property.

State is a subclass of StateVertex Class.

Attributes specific to State

None

Relationships specific to State

Name – Kind	Class	Description
EntryAction – 1	UniterpretedAction	
ExitAction - 1	UninterpretedAction	
States - n	StateVertex	
SubDiagram - 1	StateDiagram	The subdiagram associated with a CompositeState (alias State)
Transitions - n	Transition	The transitions that exit from this state.

Class: StateDiagram (Rose RealTime)

Depicts significant event-ordered behavior of a particular class. Each class may have one state diagram to describe its behavior.

StateDiagram is a subclass of Diagram Class.

Attributes specific to StateDiagram

Name - Kind	Description
HasStateMachine - text	True if the diagram includes a state machine.

Relationships specific to StateDiagram

Name - Kind	Class	Description
StateMachine - 1	StateMachine	The top-level state machine associated with this diagram.

Class: StateMachine (Rose RealTime)

Defines event-ordered behavior of a class.

StateMachine is a subclass of Element Class.

Attributes specific to StateMachine Class

None

Relationships specific to StateMachine Class

Name - Kind	Class	Description
AllStates - n	StateVertex	All states that are part of this state machine
ParentClassifier – 1	Classifier	
StateDiagram - 1	StateDiagram	The (first) state diagram associated with this state machine.
Top - 1	CompositeSite	

Class: StateVertex (Rose RealTime)

StateVertex is a subclass of ModelElement Class.

Subclasses of StateVertex Class:

State, InitialPoint, JunctionPoint, ChoicePoint, FinalState.

Attributes specific to StateVertex

Name – Kind
StateKind – text

Relationships specific to StateVertex

Name - Kind	Class
IncomingTransitions - n	Transition
OutgoingTransitions - n	Transition
ParentState - 1	CompositeState
ParentStateMachine - 1	StateMachine

Class: String (Rose RealTime)

The Rose RealTime String class is used to store the names of external documents.

It has one attribute, Value - text.

For more information on including external document contents in your SoDA document or report, see “How to Display the Contents of Files Referenced by ExternalDocs” on page 202.

Class: TerminateAction (Rose RealTime)

TerminateAction is a subclass of Action Class.

Attributes specific to TerminateAction

None

Relationships specific to TerminateAction

None

Class: Transition (Rose RealTime)

Transition is a subclass of ModelElement Class.

Attributes specific to Transition

Name – Kind
IsInternal – text
SourceRegion – text

Relationships specific to Transition

Name - Kind	Class
Action - 1	UninterpretedAction
ParentState - 1	CompositeState
ParentStateMachine - 1	StateMachine
Source - 1	StateVertex
Target - 1	StateVertex
Triggers - n	EventGuard

Class: Trigger (Rose RealTime)

Trigger is a subclass of Element Class.

Attributes specific to Trigger

Name - Kind
Guard - text

Relationships specific to Trigger

Name - Kind	Class
ParentTransition - 1	Transition
Ports - n	Port
Signals - n	Signal

Class: UninterpretedAction (Rose RealTime)

UninterpretedAction is a subclass of Action Class.

Attributes specific to UninterpretedAction

Name – Kind
Code – text
Effect – text

Relationships specific to UninterpretedAction

None

Class: UseCase (Rose RealTime)

A use case is a sequence of transactions performed by a system in response to a triggering event initiated by an actor to the system. A full use case should provide a measurable value to an actor when the actor is performing a certain task. A use case contains all the events that can occur between an actor-use case pair, not necessarily the ones that will occur in any particular scenario. A use case contains a set of scenarios that explain various sequences of interaction within the transaction.

UseCase is a subclass of Classifier Class.

Attributes specific to UseCase

Name – Kind	Description
Rank – text	The rank of the use case.

Relationships specific to UseCase

Name – Kind	Class	Description
ClassDiagrams – n	ClassDiagram	The class diagrams included in this use case.
SuperUseCases – n	UseCase	The use cases that this use case inherits from directly.
UseCaseDiagrams – n	ClassDiagram	The use-case diagrams associated with this use case.

Class: UsesRelationship (Rose RealTime)

Indicates that the client class depends on the supplier class to provide certain services, such as:

- The client class accesses a value (constant or variable) defined in the supplier class
- Operations of the client class invoke operations of the supplier class
- Operations of the client class have signatures whose return class or arguments are instances of the supplier class

UsesRelationship is a subclass of Relationship.

Attributes specific to UsesRelationship

Name – Kind	Description
FromCardinality – text	
InvolvesFriendship – text	Indicates whether the supplier class grants rights to the client class to access its non-public parts. Returns True, if the Friendship required check box is checked on the relationship specification. Otherwise, returns False.
ToCardinality – text	

Relationships specific to UsesRelationship

Name - Kind	Class
From – 1	Classifier
To - 1	Classifier

RSE Adapter: TeamTest

The RSE adapter for Rational Test Manager.

The following classes are available through the TeamTest adapter:

Build
Computer
ConfiguredTestCase
Group
Iteration
Log
LogEvent
LogFolder
NewClass
Project
Requirement
Script
Session
TestCase
TestCaseFolder
TestCaseResult
TestInput
TestPlan
UseCase
User
Variant
VerificationPoint

Class: Build (TeamTest Adapter)

A build is a version of the application under test. Typically, engineers add new features or enhancements to each incremental build. You use Rational TestManager to manage builds.

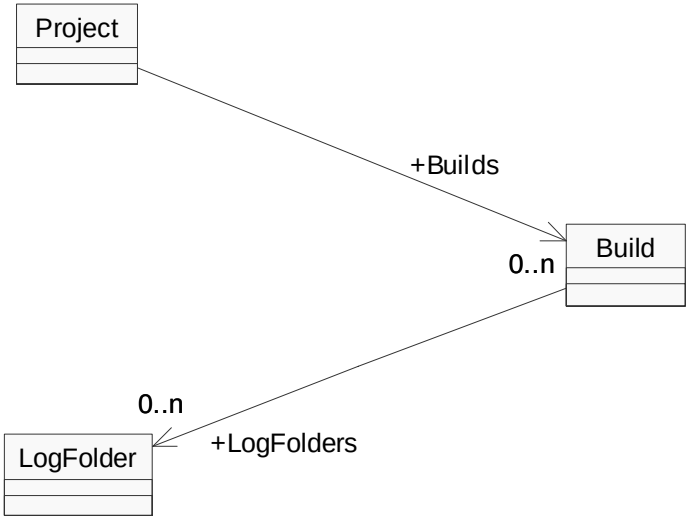
A Build contains a collection of Log Folder artifacts which in turn contain log artifacts with actual test results.

Class Hierarchy: Artifact>Build

SubClasses of Build

Build has no subclasses.

Class Diagram



Attributes Specific to Build

Attributes	Inherited From	Description
CreatedBy		The user that created the build.
CreationDate		The date the build was created
Description		The description of the build (from the General tab)
LastModifiedBy		The user that last modified the build.
ModificationDate		The date the build was last modified
Name		The name of the build (from the General tab)
Owner		The owner of the build (from the General tab)
ProjectName		The name of the project
Status		The status of the build (from the General tab)
UID		The unique ID of the build

Relationships Specific to Build (see also class diagram above)

Name	Kind	Class	Documentation
LogFolders	0..n	LogFolder	The log folders that are included with this build.

Class: Computer (TeamTest Adapter)

You coordinate the activities of all your test scripts from a single NT computer where TestManager is running, known as the Local computer. From the Local computer, you create, run, and monitor suites.

During the execution of a test, you play back test scripts on the Local computer, or on computers that you have designated as Agent computers.

Class Hierarchy: Artifact>Computer

SubClasses of Computer

Computer has no subclasses.

Class Diagram

Attributes Specific to Computer

Attributes	Inherited From	Description
Description		A description for the computer
IPAddress		The IP address of the computer
Name		The name of the computer
UID		The unique ID of the computer

Relationships Specific to Computer (see also class diagram above)

This class has no relationships.

Class: ConfiguredTestCase (TeamTest Adapter)

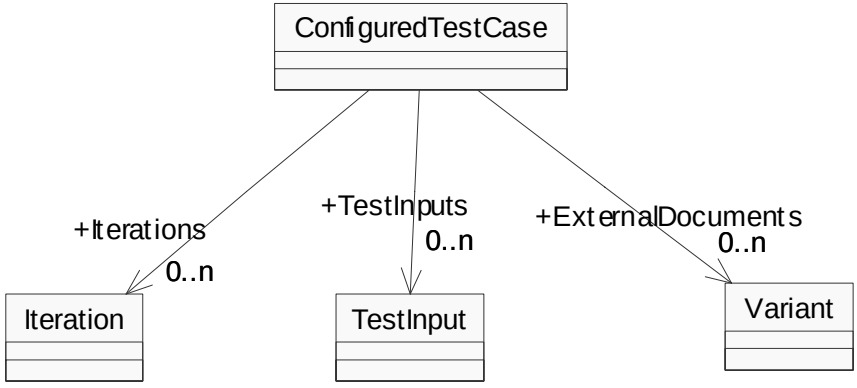
Configurations specify on what hardware and software configurations test cases must be run. A Configured Test Case is similar to a Test Case except that it is associated with a single Configuration. A Configured Test Case may have zero or more Iterations and zero or more Configurations.

Class Hierarchy: Artifact>ConfiguredTestCase

SubClasses of ConfiguredTestCase

ConfiguredTestCase has no subclasses.

Class Diagram



Attributes Specific to ConfiguredTestCase

Attributes	Inherited From	Description
AcceptanceCriteria		The expected results or performance characteristics that define whether or not the configured test case passed or failed. For example: The response time range should be between 0.5 and 2.0 seconds for pass.
Configured		TRUE if there is a configuration associated with this c onfigured test case.
CreatedBy		The user who created the configured test case.
CreationDate		The date the configured test case was created.
Custom1		Used to add a custom user-definable value
Custom2		Used to add a custom user-definable value
Custom3		Used to add a custom user-definable value
Description		A description for this configured test case.
LastModifiedBy		The user who last modified the configured test case.
ModificationDate		The date the configured test case was last modified.
Name		The name of the configured test case.
Owner		The owner of the configured test case.

Attributes	Inherited From	Description
Postconditions		Any cleanup steps that must be performed after the configured test case is run to bring the system back to a known state. For example, after you login and successfully verify the test case, you need to logout (or bring the system back into a known state for the tests that follow).
Preconditions		Any setup dependency that is required for the configured test case to run. For example: You must have the proper user ID login available in the system and the system must be in a logged out state.
Purpose		The stated purpose for the test
Suspect		
UID		The unique ID of the configured test case.

Relationships Specific to ConfiguredTestCase (see also class diagram above)

Name	Kind	Class	Documentation
Iterations	0..n	Iteration	The iterations asociated with a configured test case.
TestInputs	0..n	TestInput	The test inputs associated with a configured test case.

Name	Kind	Class	Documentation
ExternalDocuments	0..n	Variant	Other associated files to the configured test case. The variants represent an array of strings, but since you cannot have a relationship to string(s) you need an artifact type, hence the Variant.

Class: Group (TeamTest Adapter)

A user group is a basic building block for all performance testing suites. A user group is a collection of virtual testers that perform the same activity. Administrators and Public are the default groups.

Class Hierarchy: Artifact>Group

SubClasses of Group

Group has no subclasses.

Class Diagram

Attributes Specific to Group

Attributes	Inherited From	Description
Description		A description of the group
IsDefault		TRUE if the group is the default group
Name		The name of the group

Relationships Specific to Group (see also class diagram above)

Name	Kind	Class	Documentation
Users	0..n	(User)	The users in the group

Class: Iteration (TeamTest Adapter)

Iterations specify when a test case must pass. An iteration is a defined span of time during a project.The end of an iteration is a milestone.An iteration says that at some point in time,the product has to meet a certain quality standard to reach a milestone.The quality standard is defined by the test cases that must pass.

An Iteration may be assigned to Test Cases and/or a Configured Test Cases. This indicates that the Test Case or Configured Test Case is expected to be executed and pass for that iteration.

Class Hierarchy: Artifact>Iteration

SubClasses of Iteration

Iteration has no subclasses.

Class Diagram

Attributes Specific to Iteration

Attributes	Inherited From	Description
CreatedBy		The creator of the iteration
CreationDate		The date the iteration was created.
Description		The description of the iteration
EndDate		The end date of the iteration
LastModifiedBy		The user who last modified the iteration
ModificationDate		The date the iteration was modified
Name		The name of the iteration
Owner		The owner of the iteration
StartDate		The start date of the iteration
UID		The unique ID of the iteration

Relationships Specific to Iteration (see also class diagram above)

This class has no relationships.

Class: Log (TeamTest Adapter)

A log is a file that contains the record of events that occur while playing back a script or running a schedule. A log contains the results of all verification points executed as well as performance data. A Log contains a collection of Test Case Result artifacts. A Log also contains a collection of Load Test Report Output objects.

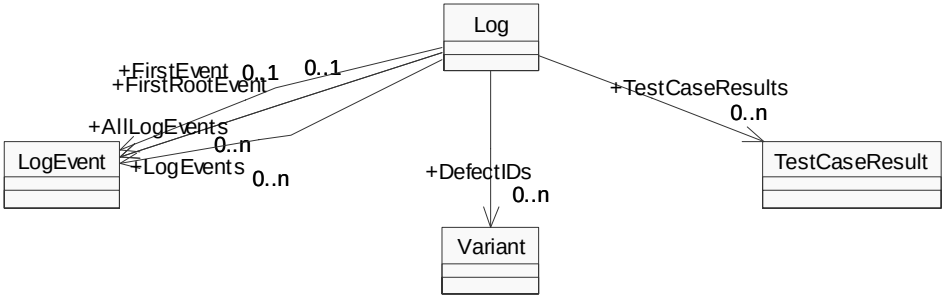
A Log contains the results of a specific test or series of tests. The log contains a hierarchy of log events AND a collect of Test Case Results.

Class Hierarchy: Artifact>Log

SubClasses of Log

Log has no subclasses.

Class Diagram



Attributes Specific to Log

Attributes	Inherited From	Description
AgentLogFiles-Path		The location of log files on the agent computer
CreatedBy		The user that created this log

Attributes	Inherited From	Description
CreationDate		The date the log was created
Description		A description of the log
LastModifiedBy		The ID of the person who last modified the log
MasterLogFile-Path		The location of the log files on the master computer
ModificationDate		The date the log was last modified
Name		The name of the log
Owner		The owner of the log
PerformanceData-Path		The location of the performance data
ProjectName		The test manager project name
Suite		The test suite
UAWPath		The location of unexpected active window data
UID		The unique ID of the log
VPPath		The location of verification point data

Relationships Specific to Log (see also class diagram above)

Name	Kind	Class	Documentation
FirstEvent	0..1	LogEvent	The first event contained in this log
Fir-stRootEvent	0..1	LogEvent	The root event for the events in this log
AllLogEvents	0..n	LogEvent	All associated log events

Name	Kind	Class	Documentation
LogEvents	0..n	LogEvent	The events contained in this log
DefectIDs	0..n	Variant	The defect ids
TestCaseResults	0..n	TestCaseResult	The associated test case results to the log.

Class: LogEvent (TeamTest Adapter)

Log events are generated when you run a test script, test case, or suite. Log events include script start and end, verification points, manual steps, and unexpe ted active windows.

LogEvent displays the type of event, the date and time the event was recorded, the s ript name, result information (if any), and other information about a log event.

Class Hierarchy: Artifact>LogEvent

SubClasses of LogEvent

LogEvent has no subclasses.

Class Diagram

Attributes Specific to LogEvent

Attributes	Inherited From	Description
EndDateTime		The time the event ended
EventCategory-Text		The event category for the log event
EventTypeText		The event type for the log event
FailureDescrip-tion		The failure description (from the Result tab)
FailureReason-Text		The failure reason (from the Result tab)
HasChildren		TRUE if the log event has children log events
ResultText		The location of the actual results
StartDateTime		The start date and time (from the Gen-eral tab)
UID		The unique id for the log event

Relationships Specific to LogEvent (see also class diagram above)

Name	Kind	Class	Documentation
Next		LogEvent	Iterates to the next log event
Parent		LogEvent	The name of the parent log
FirstChild		LogEvent	The first child log event
LastChild		LogEvent	The last child log event
NextSibling		LogEvent	Iterates to the next sibling log event
PreviousSibling		LogEvent	Iterates to the previous sibling log event
Siblings		LogEvent	The associated sibling log events
Children		LogEvent	The associated children log events

Class: LogFolder (TeamTest Adapter)

A log folder is a directory that contains test logs. Contains a collection of Log artifacts.

Class Hierarchy: Artifact>LogFolder

SubClasses of LogFolder

LogFolder has no subclasses.

Class Diagram

Attributes Specific to LogFolder

Attributes	Inherited From	Description
Name		The name of the log folder
ProjectName		The test manager project name

Relationships Specific to LogFolder (see also class diagram above)

Name	Kind	Class	Documentation
Logs	0..n	Log	The logs contained in this folder
SubFolders		LogFolder	The subfolders (LogFolders) contained in this folder

Class: NewClass (TeamTest Adapter)

SubClasses of NewClass

NewClass has no subclasses.

Class Diagram

Attributes Specific to NewClass

This class has no attributes

Relationships Specific to NewClass (see also class diagram above)

This class has no relationships.

Class: Project (TeamTest Adapter)

A project is a collection of data, including test assets, defects and requirements, that can facilitate the testing of one or more software components. Projects are managed primarily by the Rational Administrator.

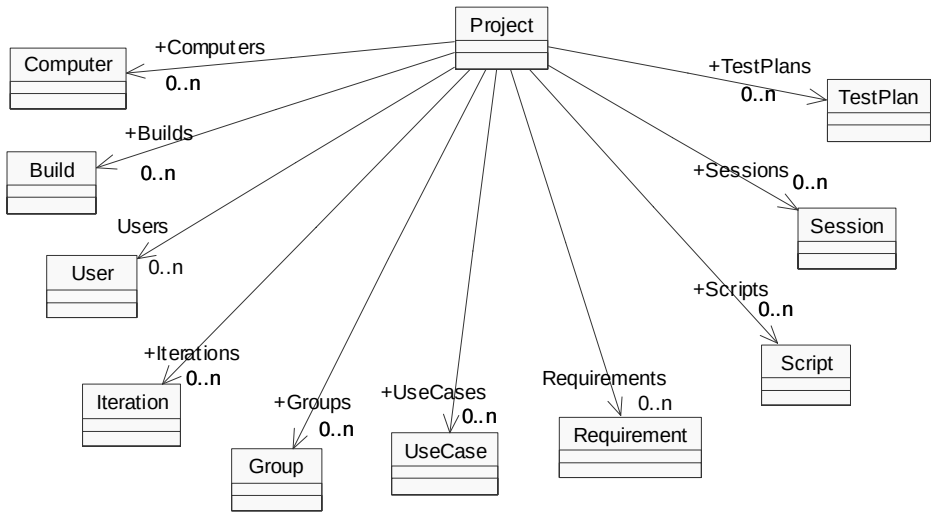
Projects contain multiple test plans.

Class Hierarchy: Artifact>Project

SubClasses of Project

Project has no subclasses.

Class Diagram



Attributes Specific to Project

Attributes	Inherited From	Description
Directory		The directory that contains the project

Attributes	Inherited From	Description
Name		The name of the project
Path		The full path of the project

Relationships Specific to Project (see also class diagram above)

Name	Kind	Class	Documentation
Scripts	0..n	Script	All scripts included in the project
Sessions	0..n	Session	All sessions included in the project
TestPlans	0..n	TestPlan	The test plans associated with the project
Iterations	0..n	Iteration	The iterations of the project
Builds	0..n	Build	The builds defined in the project.
Computers	0..n	Computer	The computers associated with the project
Users	0..n	User	The users of the project
Groups	0..n	Group	The groups associated with the project
Requirements	0..n	Requirement	The requirements in the project
UseCases	0..n	UseCase	The use cases associated with the project
		User	
		Requirement	

Class: Requirement (TeamTest Adapter)

Represents a referenced RequisitePro Requirement.

Class Hierarchy: Artifact>Requirement

SubClasses of Requirement

Requirement has no subclasses.

Class Diagram

Attributes Specific to Requirement

Attributes	Inherited From	Description
DBName		The ReqPro requirements database name
FullTag		The unique identifier for the requirement.
Name		The name of the Requirement, identified by the value of NodeID.
NodeID		The unique ID of the requirement. The ID of the requirement for which the client wants to determine the parental status. A NodeID is a GUID for a ReqPro requirement.
SourceUID		The input source ID. A handle, provided by the adapter, that identifies the connection to the ReqPro Project.
Text		The text of the requirement

Relationships Specific to Requirement (see also class diagram above)

This class has no relationships.

Class: Script (TeamTest Adapter)

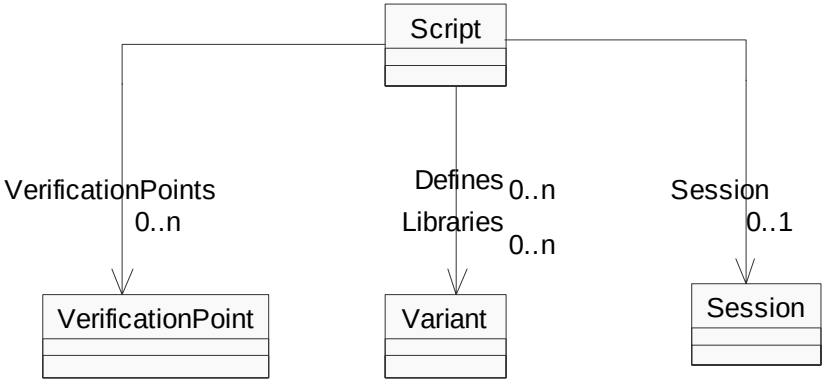
A script is a file of SQABasic or VU commands. Scripts may contain VerificationPoint.

Class Hierarchy: Artifact>Script

SubClasses of Script

Script has no subclasses.

Class Diagram



Attributes Specific to Script

Attributes	Inherited From	Description
BinaryFilePath		The location of the binary file
CreatedBy		The user that created the script
CreationDate		The creation date of the script.
Custom1		The value of the Custom 1 field (from the Custom tab)

Attributes	Inherited From	Description
Custom2		The value of the Custom 2 field (from the Custom tab)
Custom3		The value of the Custom 3 field (from the Custom tab)
Description		The description of the script (from the General tab)
Environment		The operating environment for the script (from the General tab)
FilePath		The location of the script
LastModifiedBy		The user who last modified the script
ModificationDate		The date of the script modification
Name		The name of the script (from the General tab)
Notes		Related notes for the script (from the Specifications tab)
Owner		The owner of the script
ProjectName		The test manager project name
Purpose		The purpose of the script (from the General tab)
ScriptType		The script type
SpecFilePath		The path to the specification file (from the Specifications tab)
Text		The script text
UID		The unique id of the script

Relationships Specific to Script (see also class diagram above)

Name	Kind	Class	Documentation
Session	0..1	Session	An associated session to the script
Defines	0..n	Variant	The Defines group is used for adding C-preprocessor directives, such as #define, #include, #ifdef, and #if to VU test scripts.
Libraries	0..n	Variant	The external C Libraries group is used to reference user-written external C libraries that you want to include when you compile VU test scripts.
Verification-Points	0..n	Verification-Point	The verification points included in the script
		Variant	
		Verification-Point	
		Session	

Class: Session (TeamTest Adapter)

A session is a recording of network or API traffic. Scripts may reference sessions and session may reference scripts.

A session is the period of time required to play back a suite. A session thead begins when the first test script of a suite starts execution and ends when the last script finishes.

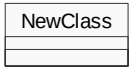
When you use Rational Robot to record a script, Robot records activities in a session, and then automatically creates a test script that represents the user's interactions with the server, as well as all queries and responses. If you have recorded a session in Robot,you can play back the test scripts in the session through TestManager.

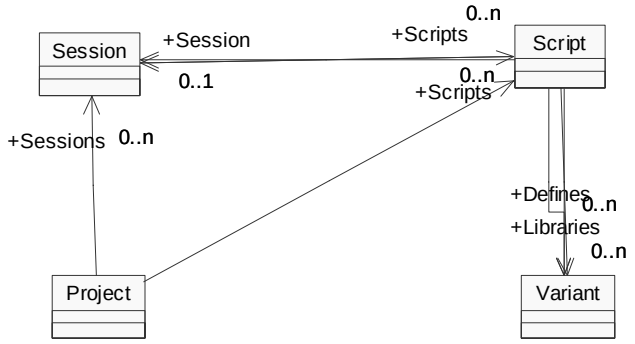
Class Hierarchy: Artifact>Session

SubClasses of Session

Session has no subclasses.

Class Diagram





Attributes Specific to Session

Attributes	Inherited From	Description
CreatedBy		The creator of the session
CreationDate		The session creation date
Custom1		The value of the Custom 1 field (from the Custom tab)
Custom2		The value of the Custom 2 field (from the Custom tab)
Custom3		The value of the Custom 3 field (from the Custom tab)
Description		A description of the session

Attributes	Inherited From	Description
LastModifiedBy		The user who last modified the session
ModificationDate		The date of the session modification
Name		The name of the session
Owner		The owner of the session
UID		The unique id of the session

Relationships Specific to Session (see also class diagram above)

Name	Kind	Class	Documentation
Scripts	0..n	Script	The scripts associated with this session

Class: TestCase (TeamTest Adapter)

Users plan what is to be tested and receive report results using test cases. A TestCase describes a specific flow of events through a feature.

Test cases validate that the system is working the way that it's supposed to work. The test case is the artifact in TestManager that answers the question,"What do I need to test?" You develop test cases to validate particular behaviors. Each test case is owned by or assigned to a team member.This answers the question, "Who will do the testing?"

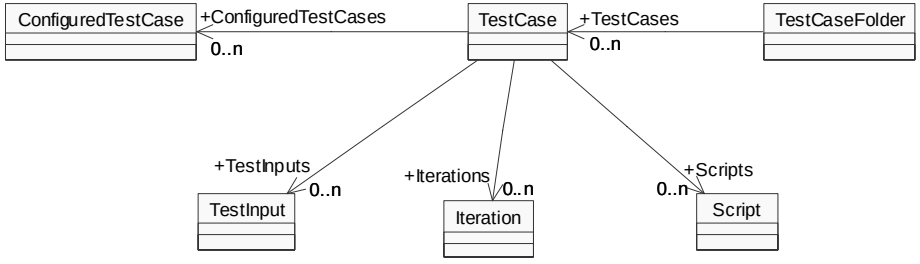
A test plan contains TestCases. You can create test cases within test case folders to organize your test cases hierarchically. A Test Case contains zero or more Configured Test Cases. A Test Case may have pointers to Iteration atifacts.

Class Hierarchy: Artifact>TestCase

SubClasses of TestCase

TestCase has no subclasses.

Class Diagram



Attributes Specific to TestCase

Attributes	Inherited From	Description
AcceptanceCrite-ria		The acceptance criteria indicates what needs to be true in order for a particular test case to pass.
Configured		TRUE if the test case has been config-ured
CreatedBy		The user that created the test case
CreationDate		The creation date of the TestCase
Custom1		The value of the Custom 1 field (from the Custom tab)
Custom2		The value of the Custom 2 field (from the Custom tab)
Custom3		The value of the Custom 3 field (from the Custom tab)
Description		A description of this test case
LastModifiedBy		The user who last modified the TestCase
ModificationDate		The date of the TestCase modification
Name		The name of the TestCase
Owner		The owner of this test case
Postconditions		The postconditions of the test case
Preconditions		The preconditions of the test case
Purpose		The purpose of this test case
Suspect		
UID		The unique id of the TestCase

Relationships Specific to TestCase (see also class diagram above)

Name	Kind	Class	Documentation
Config-uredTestCases	0..n	Config-uredTestCase	The associated configured test case
Iterations	0..n	Iteration	The iterations for this test case
TestInputs	0..n	TestInput	The test inputs for this test case
Scripts	0..n	Script	The scripts associated with this test case
ExternalDocu-ments	0..n	Variant	Other associated files to the test case. The variants represent an array of strings, but since you cannot have a relationship to string(s) you need an artifact type, hence the Variant.

Class: TestCaseFolder (TeamTest Adapter)

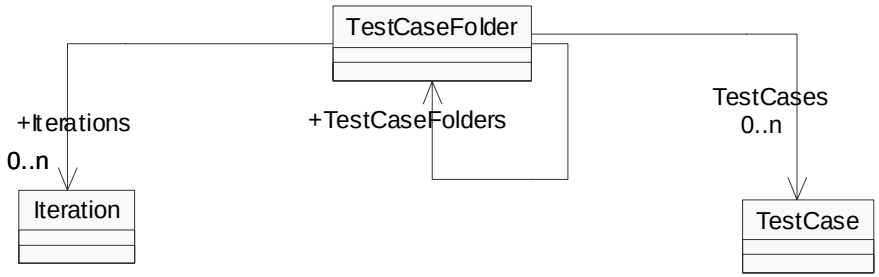
TestCaseFolders organize Test Cases. A TestCaseFolder is a directory that contains Test Cases. Test Case Folders can contain other Test Case Folders as well as Test Cases.

Class Hierarchy: Artifact>TestCaseFolder

SubClasses of TestCaseFolder

TestCaseFolder has no subclasses.

Class Diagram



Attributes Specific to TestCaseFolder

Attributes	Inherited From	Description
CreatedBy		The user that created the test case folder

Attributes	Inherited From	Description
CreationDate		The creation date of the TestCaseFolder
Description		A description of this test case folder
LastModifiedBy		The user who last modified the TestCaseFolder
ModificationDate		The date of the TestCaseFolder modification
Name		The name of the TestCaseFolder
Owner		The owner of this test case folder
UID		The unique id of the TestCaseFolder

Relationships Specific to TestCaseFolder (see also class diagram above)

Name	Kind	Class	Documentation
TestCaseFolders		TestCaseFolder	The test case folders associated with this test case folder
TestCases	0..n	TestCase	The test cases contained in this test case folder
Iterations	0..n	Iteration	The iterations contained in this test case folder
		TestCase	

Class: TestCaseResult (TeamTest Adapter)

A TestCaseResult is generated from a LogEvent that is recorded during test execution.

Class Hierarchy: Artifact>TestCaseResult

SubClasses of TestCaseResult

TestCaseResult has no subclasses.

Class Diagram

Attributes Specific to TestCaseResult

Attributes	Inherited From	Description
ActualResult		The actual result
InterpretedResult		The interpreted result
IsPromoted		TRUE if the test passes, FALSE if the test fails.
Name		The name of the TestCaseResult
Notes		The text for this TestCaseResult
UID		The unique id of the TestCaseResult

Relationships Specific to TestCaseResult (see also class diagram above)

Name	Kind	Class	Documentation
TestCase	0..1	TestCase	The associated TestCase for the TestCaseResult
LogEvent	0..1	LogEvent	The associated LogEvent for this TestCaseResult

Class: TestInput (TeamTest Adapter)

A TestInput is any requirement, use case, change request, or other input that requires validation by a Test Case. Test inputs are anything that the test designer uses to determine what needs to be tested.

Class Hierarchy: Artifact>TestInput

SubClasses of TestInput

TestInput has no subclasses.

Class Diagram

Attributes Specific to TestInput

Attributes	Inherited From	Description
Arg1		Used to add a custom user-definable argument
Arg2		Used to add a custom user-definable argument
Arg3		Used to add a custom user-definable argument
Arg4		Used to add a custom user-definable argument
Arg5		Used to add a custom user-definable argument
CollIndex		
IsContainer		TRUE if the test input contains a model, a use case, or a requirement
Kind		The kind of test input
Name		The name of the TestInput
NeedsValidation		TRUE if this test input needs to be validated

Attributes	Inherited From	Description
SubType		
Type		The test input type

Relationships Specific to TestInput (see also class diagram above)

This class has no relationships.

Class: TestPlan (TeamTest Adapter)

A TestPlan contains information about the purpose and goals of testing within the project, and the strategies to be used to implement and execute testing. Projects can contain multiple test plans.

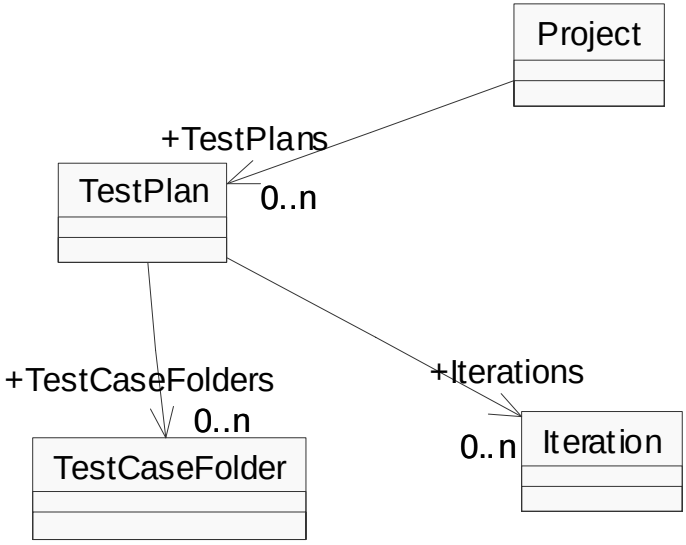
Each test plan can contain test case folders and test cases. A test plan may contain zero or more Test Case Folders, Iterations, and Configurations.

Class Hierarchy: Artifact>TestPlan

SubClasses of TestPlan

TestPlan has no subclasses.

Class Diagram



Attributes Specific to TestPlan

Attributes	Inherited From	Description
CreatedBy		The creater of the TestPlan
CreationDate		The creation date of the TestPlan
Custom1		Used to add a custom user-definable value
Custom2		Used to add a custom user-definable value
Custom3		Used to add a custom user-definable value
Description		A description for the TestPlan
LastModifiedBy		The user who last modified the TestPlan
ModificationDate		The most recent modification date of the TestPlan
Name		The name of the TestPlan
Owner		The owner of the TestPlan
UID		The unique ID of the TestPlan

Relationships Specific to TestPlan (see also class diagram above)

Name	Kind	Class	Documentation
TestCaseFolders	0..n	TestCaseFolder	The TestCaseFolders in this TestPlan
Iterations	0..n	Iteration	The Iterations associated for this TestPlan

Class: UseCase (TeamTest Adapter)

Represents a referenced Rose Use Case

Class Hierarchy: Artifact>UseCase

SubClasses of UseCase

UseCase has no subclasses.

Class Diagram

Attributes Specific to UseCase

Attributes	Inherited From	Description
Name		The name of the Use Case, identified by the value of NodeID.
NodeID		The unique ID of the Use Case. The ID of the Use Case for which the client wants to determine the parental status.
QualifiedName		The Rose Use Case qualified name
SourceUID		The input source ID. A handle, provided by the adapter, that identifies the connection to the Rose Use case.

Relationships Specific to UseCase (see also class diagram above)

This class has no relationships.

Class: User (TeamTest Adapter)

A user is an individual tester. Users are members of groups. The default user is admin.

Class Hierarchy: Artifact>User

SubClasses of User

User has no subclasses.

Class Diagram

Attributes Specific to User

Attributes	Inherited From	Description
Company		The user's company
Department		The user's department
Email		The user's email address
First		The user's first name
Last		The user's last name
Phone		The user's telephone number
Title		The user's title
UserID		The user ID

Relationships Specific to User (see also class diagram above)

Name	Kind	Class	Documentation
Groups	0..n	Group	The groups this user is a member of.
		Project	

Class: Variant (TeamTest Adapter)

Variant is an internal type used for representing relationships between artifact types and simple data types. You cannot create these directly, you can only resolve them through relationships.

Class Hierarchy: Artifact>Variant

SubClasses of Variant

Variant has no subclasses.

Class Diagram

Attributes Specific to Variant

Attributes	Inherited From	Description
CollIndex		Column index
IntValue		Integer value
RelName		Relationship type name
StrValue		String value

Relationships Specific to Variant (see also class diagram above)

This class has no relationships.

Class: VerificationPoint (TeamTest Adapter)

A VerificationPoint is a point in an SQABasic test script that confirms the state of one or more objects. Verification points capture some aspect of the application or system under test and store it away for later comparison to the actual state of the system or application.

Class Hierarchy: Artifact>VerificationPoint

SubClasses of VerificationPoint

VerificationPoint has no subclasses.

Class Diagram

Attributes Specific to VerificationPoint

Attributes	Inherited From	Description
BaselineFilePath		The associated baseline verification point
DataType		
MetadataFilePath		
Name		The name of the verificaiton point
Type		The type of the verification point

Relationships Specific to VerificationPoint (see also class diagram above)

Name	Kind	Class	Documentation
		Script	

RSE Adapter: Word

The RSE adapter for Microsoft Word.

The following classes are available through the Word adapter:

Bookmark

Document

Heading

Paragraph

Class: Bookmark (Word Adapter)

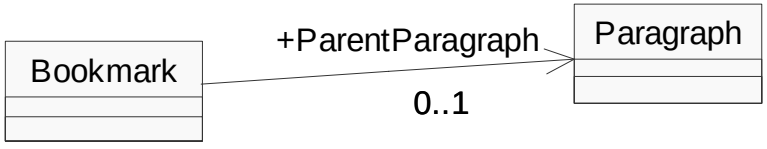
Bookmarks found in a Word document. Bookmarks are available in the Word 97 or Word 2000 version of the Word adapter.

Class Hierarchy: Artifact>Bookmark

SubClasses of Bookmark

Bookmark has no subclasses.

Class Diagram



Attributes Specific to Bookmark

Attributes	Inherited From	Description
EndPosition		The end position of the bookmark. Allows sorting of bookmarks by the end position in the document.
FormattedText		The text of the area defined by the bookmark, pasted as formatted text.
FormattedTextBefore		The formatted text that precedes the beginning of the bookmark and follows the end of the previous bookmark or the beginning of the document if there is no such bookmark.
Name		The name of the bookmark
StartPosition		The start position of the bookmark. Allows sorting of bookmarks by the start position in the document.
Text		The text of the area defined by the bookmark, pasted as a string.

Relationships Specific to Bookmark (see also class diagram above)

Name	Kind	Class	Documentation
ParentParagraph	0..1	Paragraph	The paragraph that contains the first character of the bookmark.

Class: Document (Word Adapter)

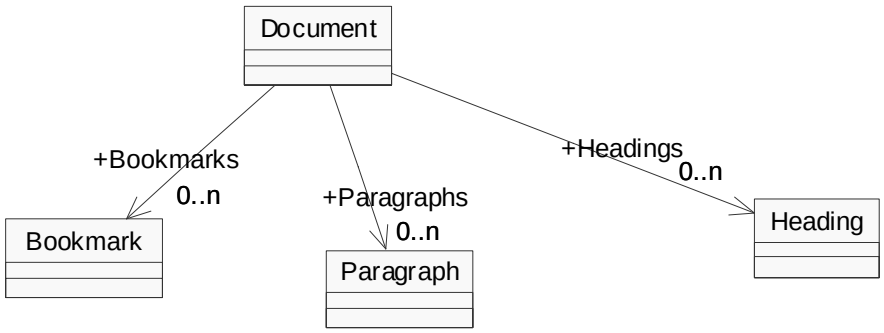
A Word document.

Class Hierarchy: Artifact>Document

SubClasses of Document

Document has no subclasses.

Class Diagram



Attributes Specific to Document

Attributes	Inherited From	Description
FormattedText		The complete contents of the Word document, pasted as formatted text.
FormattedTextAfterLastBookmark		The formatted text beginning at the end of the last bookmark and ending at the end of the document. If no bookmarks are found, the entire document is returned.
FullName		The fullname of the document

Attributes	Inherited From	Description
Name		The name of the document
Text		The complete contents of the Word document, pasted as a string.

Relationships Specific to Document (see also class diagram above)

Name	Kind	Class	Documentation
Bookmarks	0..n	Bookmark	The bookmarks defined in this document
Headings	0..n	Heading	All headings found in this document
Paragraphs	0..n	Paragraph	All paragraphs, including headings, found in this document.

Class: Heading (Word Adapter)

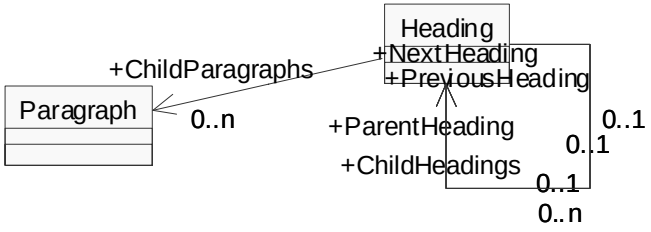
Headings are paragraphs with a style "Heading1", "Heading2", and so on.

Class Hierarchy: Artifact>Heading

SubClasses of Heading

Heading has no subclasses.

Class Diagram



Attributes Specific to Heading

Attributes	Inherited From	Description
FormattedText		The text of the heading, pasted as formatted text.
Label		The label of the heading. For example, "1.1.2"

Attributes	Inherited From	Description
Position		The character position of the first character of the paragraph.
StyleDescription		A description of the style.
StyleName		The style of the heading. For example, "Normal Arial 10."
Text		The text of the heading, pasted as a string.

Relationships Specific to Heading (see also class diagram above)

Name	Kind	Class	Documentation
ParentHeading		Heading	The parent heading for this heading, one level up.
PreviousHeading		Heading	The previous heading at the same level.
NextHeading		Heading	The next heading at the same level.
ChildHeadings		Heading	The headings contained within this heading, one level in.
ChildParagraphs	0..n	Paragraph	The paragraphs contained within this heading, including headings.

Class: Paragraph (Word Adapter)

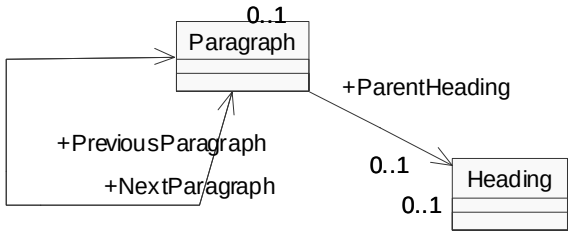
Paragraphs in a Word document.

Class Hierarchy: Artifact>Paragraph

SubClasses of Paragraph

Paragraph has no subclasses.

Class Diagram



Attributes Specific to Paragraph

Attributes	Inherited From	Description
FormattedText		The complete contents of the Word document, pasted as formatted text.
Position		The character position of the first character of the paragraph.
StyleDescription		A description of the paragraph style.
StyleName		The style name of the paragraph. For example, "Normal."
Text		The complete contents of the Word document, pasted as a string.

Relationships Specific to Paragraph (see also class diagram above)

Name	Kind	Class	Documentation
ParentHeading	0..1	Heading	The nearest heading above the current paragraph.
PreviousParagraph		Paragraph	The previous paragraph in the document.
NextParagraph		Paragraph	The next paragraph in the document.

B SoDA Template Library

The following sections list templates that come with every SoDA installation. The templates are divided into sections by domain.

Apex NT Templates

SoDA for Word includes the following Apex NT templates:

File Name	Description
Apex\498idd.doc	Interface Design Description compliant with MIL-STD-498
Apex\498irs.doc	Interface Requirements Specification compliant with MIL-STD-498
Apex\498ocd.doc	Operational Concept Description compliant with MIL-STD-498
Apex\498sdd.doc	Software Design Description compliant with MIL-STD-498
Apex\498sdp.doc	Software Development Plan compliant with MIL-STD-498
Apex\498srs.doc	Software Requirements Specification compliant with MIL-STD-498
Apex\498sss.doc	System/Subsystem Specification compliant with MIL-STD-498
Apex\InProcess.doc	Software Design document for a single Apex subsystem/view

See also “Subsystem Structure for Apex NT Templates” on page 115.

ClearCase Templates

SoDA for Word includes the following ClearCase templates:

File Name	Description
Activity.doc	Activity description report.

Version.doc	Detailed report on a file or directory version.
VOB.doc	VOB and meta-data detail report.
Region.doc	List of VOB and views by region.
Element.doc	Element and version history report.

Rose and Rose RealTime Templates

SoDA for Word includes the following Rose and Rose RealTime templates:

File Name	Template Name	Description	Conditions
498idd.doc	498 IDD	Interface Design Description: scope, referenced documents, interface design, and requirements traceability.	Assumes there is a Rose model that represents the 498 System. Internal interface diagrams are class diagrams attached to the Logical View with the word “Internal” somewhere in the name. The CSCIs are packages in the Logical View, and the key interfaces for each CSCI are classes. The details of each CSCI (described in SDDs) are in separate models
498irs.doc	498 IRS	Interface Requirements Specification: scope, referenced documents, requirements, qualification provisions, and requirements traceability.	Assumes there is a Rose model that represents the 498 System. In the Use Case View is a package called “Interfaces.” The use cases in that package become the interface requirements.

498ocd.doc	498 OCD	Operational Concept Description: scope, referenced documents, requirements, qualification provisions, and requirements traceability.	Assumes there is a Rose model that represents the 498 System. In the Use Case View is a package called “Interfaces.” The use cases in that package become the interface requirements. There is another package in the Use Case View called “Operational Scenarios.” The use cases and their interaction diagrams become the operational scenarios.
498sdd.doc	498 SDD	Software Design Description: scope, referenced documents, CSCI-wide decisions, detailed design, and requirements traceability.	Assumes there is a Rose model that represents each CSCI in the system. The Logical View should contain two diagrams: “System”, for the system architecture, and “Main”, for the CSCI architecture. Packages in the Logical View, representing CSCs, should each have a “Main” diagram as well. CSCs must be stereotyped as <<imported>> or <<exported>> to be documented as imported or exported CSCs.
498sdp.doc	498 SDP	Software Development Plan: scope, referenced documents, overview of required work, plans for general and detailed software development activities, schedules and activity network, project organization and resources.	This template contains no SoDA commands. It is included in the template set for consistency and completeness.

498srs.doc	498 SRS	Software Requirements Specification: scope, referenced documents, requirements, qualification provisions, and requirements traceability.	Assumes there is a Rose model that represents the CSCI. In the Use Case View is a use case called “CSCI.” The state diagram for this use case becomes the required states for the CSCI. The Use Case View should also have two packages: “Capabilities” and “Interfaces.” Use cases defined within these packages will become the capability and interface requirements.
498sss.doc	498 SSS	System/Subsystem Specification: scope, referenced documents, requirements, qualification provisions, and requirements traceability.	Assumes there is a Rose model that represents the System. In the Use Case View is a use case called “System.” The state diagram for this use case becomes the required states for the System. The Use Case View should also have two packages: “Capabilities” and “Interfaces.” Use cases defined within these packages will become the capability and interface requirements.
Classes.doc	Data Dictionary of Classes	Rose model report: class names and descriptions.	Classes must be defined in the model.
ClassesAttrsOps.doc	Data Dictionary of Classes with Attributes and Operations	Rose model report: class names, descriptions, attributes, and operations.	Classes must be defined in the model.

ClassesAttrsOpsTable.doc	Data Dictionary of Classes with Attributes / Operations in tables	Rose model report: class names, descriptions, attributes, and operations formatted in a table.	Classes must be defined in the model.
Design.doc	Software Design Document	Design document for the system: scope, referenced documents, architectural goals and constraints, logical architecture, and interaction diagrams.	Must have packages in the Logical View.
LogicalViewFull.doc	Detail of all Attributes and Operations by Class by Package	Rose model report: logical view, including package names, class names, public and private properties (attributes) and methods (operations), and package structure.	Must have packages and classes within those packages in the Logical View.
LogicalViewPublic.doc	Summary of Packages, Classes, and Public Attributes / Operations	Rose model report: logical view, including package names. class names, public properties and methods, and package structure.	Must have packages and classes within those packages in the Logical View.
LogicalViewSimple.doc	Components in the Model and their associated Classes	Rose model report: logical view, including Package names, class names, and package structure.	Must have packages and classes within those packages in the Logical View.
PackagesClasses.doc	Summary of Packages with Diagrams and Class Descriptions	Package report with description, class diagram, and classes.	Must have packages in the Logical View and classes within those packages.

PhysicalViewFull.doc	Physical View summary	Rose model report: component view (also known as Physical View), including Package Name, component name, attached class names with public and private properties (attributes) and methods (operations) and package structure.	Must have packages and components in the Component View, classes must be attached to the component.
PhysicalViewPublic.doc	Physical View with Public Operations and Attributes	Rose model report: component view.	Must have packages and components in the Component View, classes must be attached to the component.
PhysicalViewSimple.doc	Components in the Model and their associated Classes	Rose model report: physical view (with packages, components, and classes) and package structure with component views.	Must have packages and components in the Component View, classes must be attached to the component.
RUP Actor Report.doc	Rational Unified Process Actor Report	Actor report: brief description, characteristics, relationships, and state diagram.	External generation: Requires the Model (name and path), Parent Package, and Class (Actor) name. Rose Tight Integration: Open the Class Diagram, select the Actor, run SoDA Report.

RUP Business Entity Report.doc	Rational Unified Process Business Entity Report	Business entity report for a class: brief description, responsibilities, relationships, operations, attributes, state and class diagrams.	External generation: Requires the Model (name and path), Parent Package, and Class name. Rose Tight Integration: Open the Class Diagram, select the Class, run SoDA Report. **Only Operations stereotyped as <<responsibility>> are documented in the Responsibility Section.
RUP Business Object Model Survey.doc	Rational Unified Process Business Object Model Survey	Business object model survey.	Must have a package called “Business Object Model” in the Logical View. **Only classes stereotyped as <<business worker>> or <<business entity>> are documented in the Member Business Worker or Entities sections.
RUP Business Use Case Model Survey.doc	Rational Unified Process Business Use Case Model Survey	Business use-case model survey of actors, business use cases (including use-case diagrams), and views.	Must have a package in the Use Case View called Business Use-Case Model. External generation: Requires the Model (name and path), package name, and use case name. Tight Integration: Open the Class diagram, select the Business Use-Case, run SoDA Report. **Only classes under this package, stereotyped as <<business actor>>, are documented in the Business Actor section. **Only use-cases under this package, stereotyped as <<business use-case>>, are documented in the Business Use-Case section.

RUP Business Use Case Realization Report.doc	Rational Unified Process Business Use Case Realization Report	Use case realization report: brief description, flow of events, interaction diagrams, participating business objects, class diagrams, and derived requirements.	External generation: Requires the Model (name and path), Parent Package, and Use Case name. Tight Integration: Open the Use Case Diagram, select the Use Case, run SoDA Report.
RUP Business Worker Report.doc	Rational Unified Process Business Worker Report	Business worker report for a class: brief description of class, responsibilities, relationships, operations, attributes, competence requirements, state and class diagrams.	External generation: Requires the Model (name and path), Parent Package, and Class name. Tight Integration: Open the Class Diagram, select the Class, run SoDA Report. Retrieves External Word Docs into the Responsibilities and Competence Requirements sections. You must begin the file name with “Responsibilities” or “Competence”. Note: External document file names are case sensitive.
RUP Class Report.doc	Rational Unified Process Class Report	Class report: brief description, responsibilities, operations, attributes, relationships, and state diagram.	External generation: Requires the Model (name and path), Parent Package, and Class name. Tight Integration: Open the Class Diagram, select the Class, run SoDA Report. Only Operations, stereotyped as <<responsibility>>, are documented in the Responsibility Section.
RUP Design Model Survey.doc	Rational Unified Process Design Model Survey	Design model hierarchy with classes, packages, and class diagrams at each level.	A package called “Design Model” must exist in the Logical View.

RUP Software Architecture Document.doc	Rational Unified Process Software Architecture Document	Software architectural representation, goals, and constraints. Includes architecturally significant aspects of the views: use case, logical, process, deployment, and implementation. Logical view includes model elements, package and subsystem layering. Sections on size, performance, and quality are manually maintained.	This template works best when the model is structured as described in the Rational Unified Process, Rose Model Template. Section 5 requires a package named “Use Cases” under the Use Case View and a diagram with “Significant” in the name. Section 6.2 requires a Class Diagram with “Layering” in the name. Section 7 requires a package under Logical View named “Process View.” Section 9 requires a subsystem named “Implementation Model.”
RUP Use Case Model Survey.doc	Rational Unified Process Use Case Model Survey	Use-case model survey of actors, use cases (including use-case diagrams), and views.	Requires a package under Use Case View named “Use-Case Model.” Only classes, stereotyped as <<actor>>, are documented in section 2, Actors.
RUP Use Case Realization Report.doc	Rational Unified Process Use Case Realization Report	Use case realization report: brief description, flow of events, interaction diagrams, participating objects, class diagrams, and derived requirements.	External generation: Requires the Model (name and path), Parent Package, and Use Case name. Tight Integration: Open the Use Case Diagram, select the Use Case, run SoDA Report.

RUP Use Case Report.doc	Rational Unified Process Use Case Report	Use case report: relationships, with diagrams: use-case, interaction, state, class.	External generation: Requires the Model (name and path), Parent Package, and Use Case name. Tight Integration: Open the Use Case Diagram, select the Use Case. A Word document must exist for the Use Case Specification (this has the Title page, TOC, etc.) Only inherited relationships, stereotyped as <<uses>> or <<extends>>, are documented in either Uses or Extends Relationship sections.
RUP Use Case Storyboard Report.doc	Rational Unified Process Use Case Storyboard Report	Use case storyboard report: brief description, storyboard flow of events, usability requirements, references to user interface prototype, interaction diagrams, participating objects, class diagrams.	Requires a Rose model and a related use case.

RequisitePro Templates

SoDA for Word includes the following RequisitePro templates:

File Name	Template Name	Description	Conditions
DocsReqts.doc	Requirements in a Project, sorted by Document	A list of those requirements contained in documents and displayed in document order.	Must have document-based requirements in the RequisitePro project.
Reqts.doc	Summary of Requirements in a Project	A list of all requirements in a RequisitePro project.	Must have requirements define in a RequisitePro project.

ReqtsAttrs.doc	All Requirements and their Attributes in a Project	A list of all requirements and their related attributes and current values in a RequisitePro project.	Must have requirements and related attributes with values in a RequisitePro project.
ReqtsTraces.doc	Requirement Hierarchy and Traceability Summary	A list of all requirement tags of the project in a hierarchical display, and the text of all requirements and the traceability relationships for each requirement.	Must have hierarchical requirements and traceability relationships defined in the RequisitePro project.
ReqtsUseCases.doc	Use-Case Requirements with Rose Diagrams	A list of all RequisitePro Use-Case requirements and their associated Rose Use Case diagrams.	Must have an associated Rose Model for the RequisitePro project. The requirement text must match the Use Case name in Rose.

TeamTest Templates

SoDA for Word includes the following TeamTest templates:

File Name	Template Name	Description	Conditions
BuildDetail.doc	Build Detail Report	Report details build information such as the build name, state, owner, description, creator name and any related notes.	Must have at least one build specified in the project.
Build Summary.doc	Build Summary Report	Report summarizes build information and includes build name, state and description.	Must have at least one build specified in the project.

ComputerDetail.doc	Computer Detail Report	Report details computer information such as computer name, network name or IP address, operating system, and description.	Must have at least one computer set up through Rational Administrator.
ComputerSummary.doc	Computer Summary Report	Report summarizes computer information and includes computer name, operating system and network name or IP address.	Must have at least one computer set up through Rational Administrator.
ScriptDetail.doc	Script Summary Report	Report details script information such as script owner, type, description, specification file path, developed, purpose, script creator and notes.	Must have at least one test script planned or developed through Rational Robot or Rational Test Manager.
ScriptSummary.doc	Script Summary Report	Report summarizes script information and includes script name, type and description.	Must have at least one test script planned or developed through Rational Robot or Rational Test Manager.
TestDocDetail.doc	Test Document Detail Report	Report details indicated test document information such as document name, description, path of the document and the document creator.	Must have at least one test document developed and attached to the project through Rational Test Manager.
TestDocSummary.doc	Test Document Summary Report	Report summarizes indicated test document information including document name and description.	Must have at least one test document developed and attached to the project through Rational Test Manager.