Using Rational SoDA® for Word
Version 2000.02.10

Rational®
the e-development company™

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

IMPORTANT NOTICE

Contents

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” on page 6 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” on page 13 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 NT 4.0, Service Pack 3 or greater Windows 95, Service Pack 2 or greater Windows 98
Processor	166 MHz or greater
Memory	32 MB
Disk Space	50 MB
Word Processor	Microsoft Word 95, 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 NT system, you must have Windows NT 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 NT system, you must have Windows NT 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 for Word from the Windows Start menu. This command opens Word, enables the SoDA menu, and opens a blank document. Use one of the following procedures for working with a SoDA template.

New users

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 Getting Started Wizard guides you through selecting a SoDA template, saving it for your own use, identifying the source of information, and generating a report.
- 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 information components within that source for your report.

Experienced users

The SoDA code is loaded into Microsoft Word automatically, whenever you open an existing SoDA document. You can also create a new template 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 information components within that source for your report.
- Select **soda.dot** and click **OK**.

Understanding What SoDA Does

Software Documentation Automation (SoDA)

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 according to a template. SoDA comes with several predefined templates, or you can use its WYSIWYG template-building component to build templates for 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 look in **more detail** at each aspect of SoDA documentation automation:

- Information Retrieval
- Document Generation
- Report Generation
- Template Customization

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

Although SoDA comes with several templates, every project has different requirements for documentation. 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 new macro language to make template changes. You use the Microsoft Word's What-You-See-Is-What-You-Get (WYSIWYG) interface to specify structure and style, and you use dialog boxes to specify the SoDA

commands. For more on customizing templates, see “Template Customization Concepts” on page 45.

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.

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 styles, macros, etc. to that document. The soda.dot enables the SoDA environment in Word, including the SoDA menu and commands.

A **SoDA template** is a Word document, i.e., a .DOC file that is based on a soda.dot file. It also includes SoDA commands and additional text. 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.

The following sections list available SoDA templates 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 119.

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.

Generating a Web Page

To generate an HTML document:

1. Open the appropriate template.
2. Click **File** > **Save As** to save the template to another name and working directory.
3. Click **SoDA** > **Generate Web Page**.

   If this is the first time you've used this template, you may have to identify the location of the source data.

   The generated HTML document is displayed in Microsoft Word. You can regenerate the data in the HTML document by rerunning the Generate Web Page command from the SoDA menu. If you edit the generated HTML document, you cannot regenerate it.

   To save the HTML document in Word format, use the File > Save As command and select the Word document (.doc) file type option.

Generating a Report

Follow these steps to generate a report:

1. Open the appropriate template.
2. Click **File** > **Save As** to save the template to another name and working directory.
3. Click **SoDA** > **Generate Report**.

   If this is the first time you've used this template, you may have to identify the location of the source data.

   As report generation begins, you will see SoDA's progress indicator box in the lower-right corner of the screen. The Word window minimizes prior to the report generation and is not available for any other use.

Note To cancel your generation request before SoDA has completed generating your report, click the Cancel button on the SoDA progress indicator. Word may take some time to recover its normal state.

When generation is complete, Word is restored and the report is displayed.

Generation Tips

- The SoDA Generate Web Page command is a faster way to generate an HTML document using SoDA and Microsoft Word 2000. You can then use the File > Save As command to save the HTML file as a Word document (with a .doc extension).
- To increase the speed of report generation, be sure that all other document windows in Word are closed.
- Turn off Spelling and Grammar checking.
- When using the Rose domain, preload the Rose model in Rose and minimize the application.
- Reports with tables will take longer than reports without tables.

The Error Report

If there were errors during generation, a SoDA confirmation dialog box displaying this message appears: "Display error report?" If you choose Yes, an error report will be displayed, describing the sections where errors occurred. In some cases, you can double-click an error to access the source of the error.

Generating the Document

Follow these steps to generate a full document:

1. Open the appropriate document or template.
2. If this the initial generation from this template, use the **File** > **Save As** command to rename and relocate this template.
3. Click **SoDA** > **Generate Document**.
4. Do one of the following:
   - If this is the initial generation from this template, click **OK** in the SoDA Generator dialog box. It is not necessary to enable any options in the dialog box. If this is the first time you've used this template, you may have to identify the location of the source data.

- For subsequent generations, choose the generation options you need in the SoDA Generator dialog box. Click **OK**.

As document generation begins, SoDA's progress indicator box appears in the lower-right corner of the screen. The Word window will then minimize as the document is generated. If you wish to cancel your generation request before SoDA has completed generating your document, click the Cancel button.

When generation is complete, Word is restored and the document is displayed.

Generation Tips

- The SoDA Generate Web Page command is a faster way to generate an HTML document using SoDA and Microsoft Word 2000. You can then use the File > Save As command to save the HTML file as a Word document (with a .doc extension).
- The Generate Report option is significantly faster than Generate Document.
- Documents with tables will take longer than documents without tables.
- To increase the speed of document generation, be sure that all other document windows in Word are closed. In the Window menu, make sure there is only one numbered document (your template).

The Error Report

If there were errors during generation, a SoDA confirmation dialog displaying this message appears: "Display error report?" If you choose Yes, an error report will be displayed, describing the sections where errors occurred.

Browsing a Generated Document

When you have completed document generation, your SoDA document will include repeated sections and displayed values according to the commands stored in the template.

As you browse through a generated document, you will notice several differences from the original template:

- Displayed value placeholders have been replaced by actual values.
- The template information has been hidden.
- Repeated sections generated by SoDA have attached links, or pointers back to their sources.

It is possible that after you generate a document the first time you notice changes you wish to make to the template itself. In this case, close the generated document, and be sure the template is open. Then follow the procedures described in Customizing a SoDA Template.

It is much easier to change a document template than it is to change template information within a generated document.

Updating a Document

Once you have generated a document for the first time, you can update it in various ways. The following sections cover the process of updating and maintaining a SoDA document:

- Adding Supplemental Text
- Modifying Generated Text or Graphics
- Using Visit Source
- Regenerating a Document
- Regenerating a Section of a Document
- Maintaining Generated Documents

Adding Supplemental Text

Most document templates provide blank lines between the SoDA-generated values where you can enter information not stored in the source domain. After a document has been generated, you can start adding supplemental text:

1. Place the cursor on the line where you want to add text.
2. If you are unfamiliar with the SoDA template, display hidden text by pressing the Show/Hide toolbar button (¶). Turn off the hidden text prior to saving the document.

3 Verify that you are on a line with no annotations.

Note Be careful when editing text within annotations in a generated document. Do not modify or remove the hidden annotation text. Modifications to the hidden text prevent you from regenerating document. The most common error for this condition refers to "unbalanced annotations."

4 Type the descriptive text.

As you gain more expertise with SoDA you can add text in non-blank lines; just be sure you are within the LINK and ENDLINK annotations, and outside any DISPLAY..END DISP pairs.

Modifying Generated Text or Graphics

The Rule: Never make changes to generated text!

If you edit the text generated by SoDA, changes will be overwritten when the document is regenerated. If you see an error of any sort in generated text, correct it in the source.

To help you identify generated text, SoDA colors all generated text blue.

Using Visit Source

As you browse through a generated document, use the Visit Source command to navigate to the source that corresponds to a particular section. This provides hypertext-like links to the source, enabling you to make modifications to the data. The Visit Source feature provides compliance with the ISO 9000 requirement for single source information. Visit Source is implemented for ClearQuest, Rational Rose, Rose RealTime, and RequisitePro. All other domains are not supported.

To use this feature:

1 Place the cursor in a displayed value in the generated document.

2 Click **SoDA > Visit Source**.

Visit Source opens the domain application and the related source information.

Regenerating a Document

Whenever you make changes to the information in your source domains, you should regenerate your document. You can regenerate the entire document, or you can regenerate just one section.

During regeneration, SoDA reevaluates all data. In repeated sections, it determines whether objects have been added or removed from the source. If a new object is found, SoDA adds a new section. When an existing object is found, SoDA reevaluates the displayed values within that section. It compares the current value with the value stored in the document, and updates any changes.

To regenerate your entire document, follow these steps:

1. If you have made recent changes to your document(s), save your work using **File > Save**.
2. Be sure no text is selected, and click **SoDA > Generate Document**.
   The Generator dialog box appears.
3. Choose generation options and click **OK** to generate your document.

Regenerating a Section of a Document

If you know you only made changes to the source corresponding to a particular section of a document, you can regenerate just that section, and save some time in the process. Use the **SoDA > Regenerate Section** command:

- To regenerate a single displayed value, place the insertion point anywhere in the displayed text (the blue text).
- To regenerate a generated section, place the insertion point at the beginning of that section.
- To regenerate an entire repeated section, place the insertion point at the beginning of the first generated section.

When the insertion point is at the beginning of the first section and you click **SoDA > Regenerate Section**, SoDA will ask you

whether you want to regenerate the entire repeated section, or just the first generated section.

If you are not sure which sections of a document have changed, regenerate the entire document. It will be faster than using Check Consistency followed by regenerating the changed sections.

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. Open the template in Word.
4. Click **SoDA** > **Modify Command**. The OPEN Command dialog box appears.
5. 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.
6. 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.

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 ClearQuest, Rational Rose, Rational Rose RealTime, Rational RequisitePro, Rational TestStudio/TeamTest, 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 other SoDA 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 document 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. Master the key concepts for template customization.
2. Identify the domain that contains the information you need to document.
3. 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**.
4. Use the 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.

5. Save the template.
6. Test the template.
7. 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 after generation.

SoDA's Use of Annotations

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

- store the strings that hold the values of the SoDA 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 documents, it is recommended that you make a copy of a SoDA document and place your comments in the copy.

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 document you must determine what information will act as sources to the document.

See “Rational SoDA for Word Domains” on page 99 for more information about SoDA domains:

- Rose Domain
- Rose RealTime Domain
- RequisitePro Domain
- TeamTest Domain
- ClearQuest Domain

- 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 the SoDA 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 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 document.

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 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 document.

- 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 document 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 Command** 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.

To add a single command:

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**. 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 document: through the Template View or in the document itself. Both methods give the same results. Deleting commands in the SoDA document 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:
  - **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 will result in the template being in an inconsistent state.

Creating Hyperlink Documents and Web Pages

If you are using Word 97 or Word 2000 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 document 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 document. 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 document.

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 document.
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 document. When you have many OPEN commands stored in many places within one document, meaningful names will make it easier to define other elements.

   You can add multiple OPEN commands to your document. 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 document that will extract objects from the File System domain. You would like SoDA to create a document section for each directory in your projects directory.

First, you must "open" your home directory within your document, 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 (for each object found in the source) sections in your document. For example, you could use a REPEAT command 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 document before a REPEAT command can be created.

Adding a New REPEAT Command

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, modify any options, and click 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 – once you generate the document you cannot change the table style.

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:

=	exactly equal to (case sensitive)
!=	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 SoDA to create a document 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:

.	Match any single character
*	Match zero or more of the preceding character in the expression
^	Match the expression only at the beginning of the string (place at start of expression)
$	Match the expression only at the end of the string (place at end of expression)
[]	Match any one of the enclosed characters
\	The next character should not be interpreted as a special character

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.

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 document 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 document which will extract objects from the File System domain. You would like SoDA to create a document section for each directory in your project directory. You would also like the name of each directory to appear in the corresponding document-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 document 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 document 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 document 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 document which will extract objects from the File System domain. You would like SoDA to create a document 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.

   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 document which will extract objects from the File System domain. You would like SoDA to create a document 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 SoDA 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.

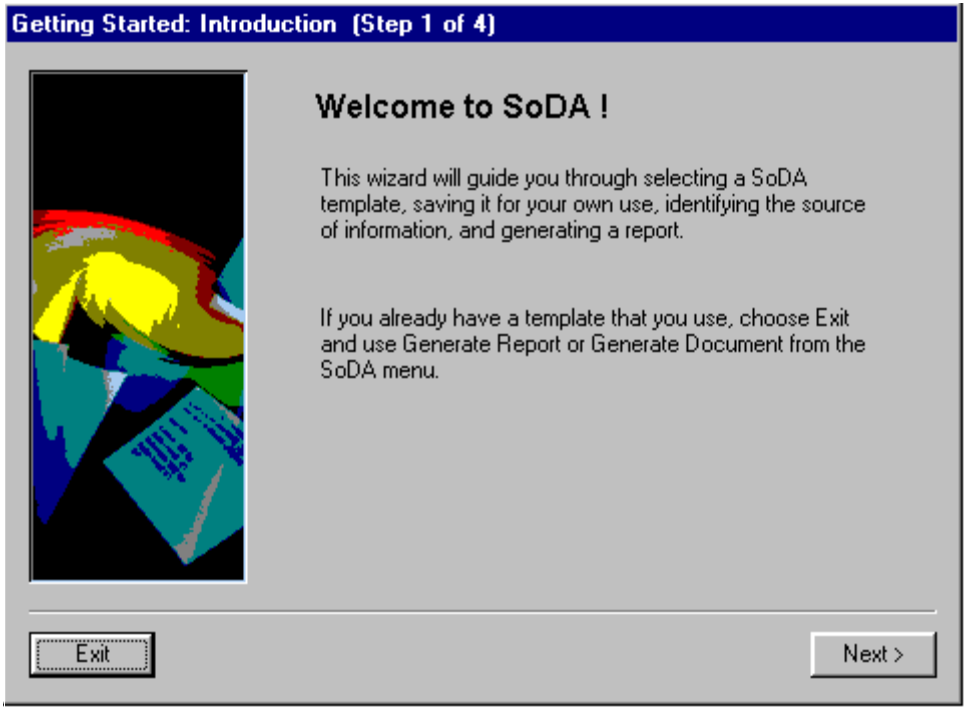
5 Wizards and Dialog Boxes

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

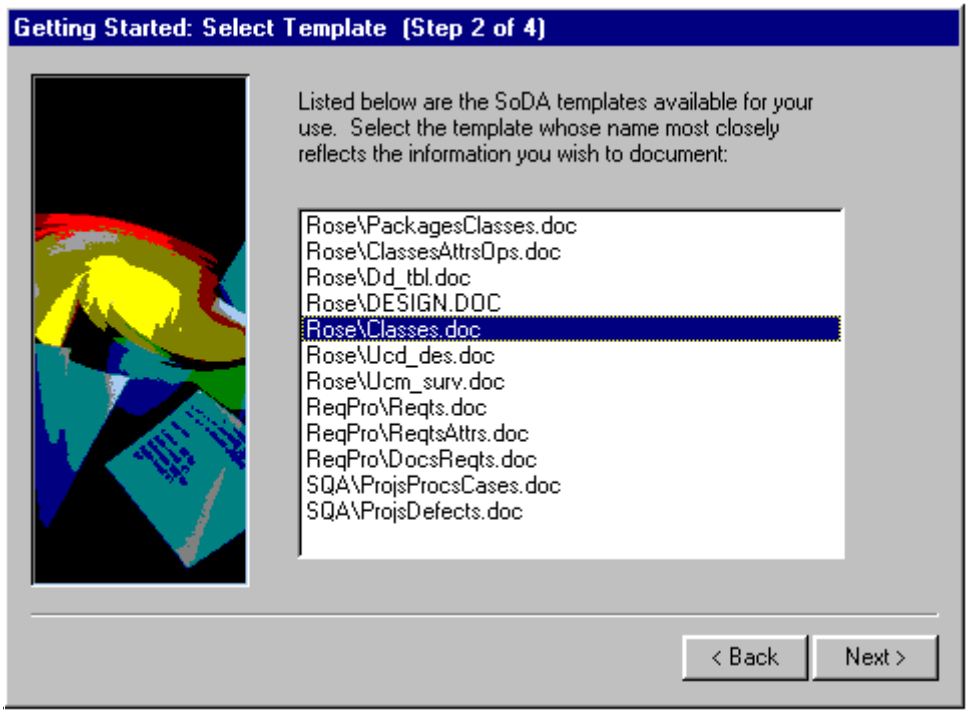
- Getting Started Wizard
- Template View
- SoDA Generator Dialog Box
- Identify the <Class> Dialog Box
- Select Command to Add Dialog Box
- OPEN Command Dialog Box
- REPEAT Command Dialog Box
- DISPLAY Command Dialog Box
- LIMIT Command Dialog Box
- Edit Link Dialog Box
- Adjust Links Dialog Box

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:

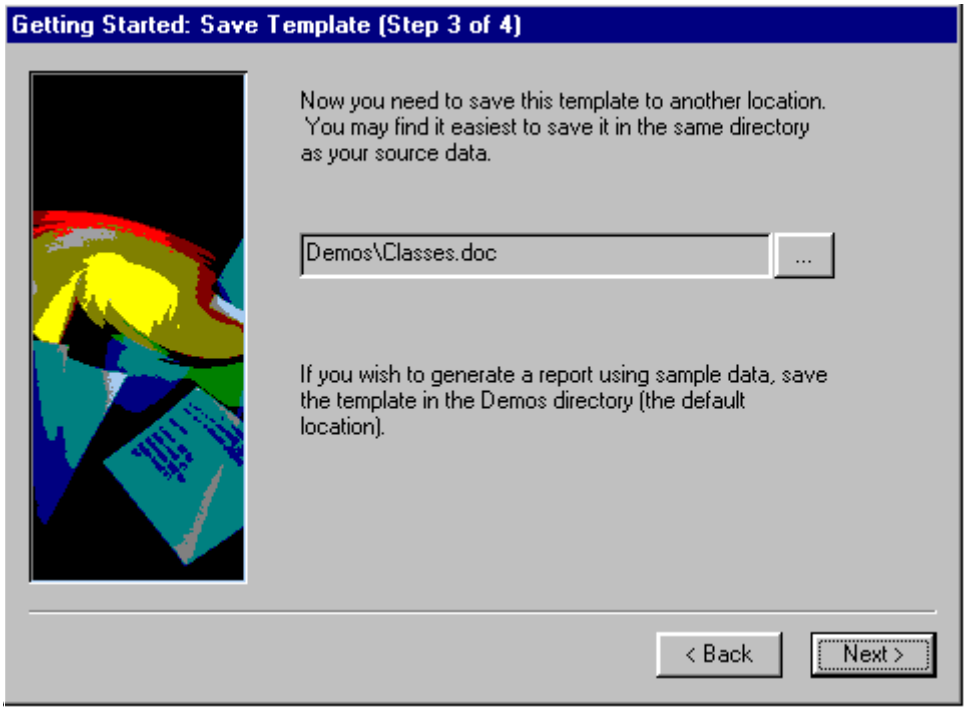


Once you have read the introduction, choose Next to display the list of available templates:



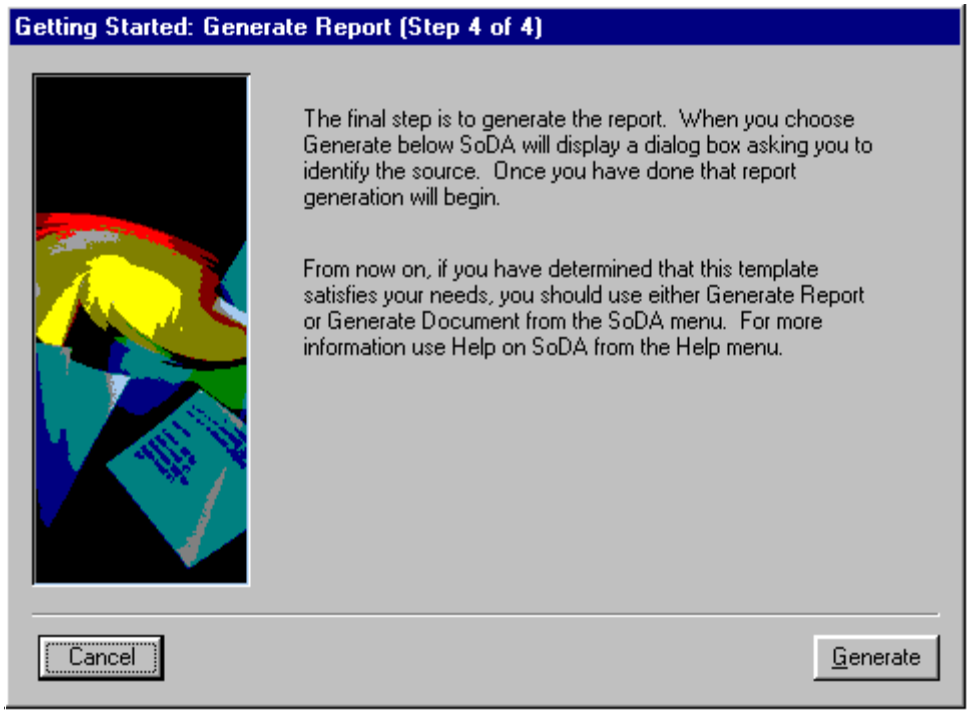
The name of the template should give you an idea about what the template documents. Select the one that most closely fits the type of report you wish to produce.

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

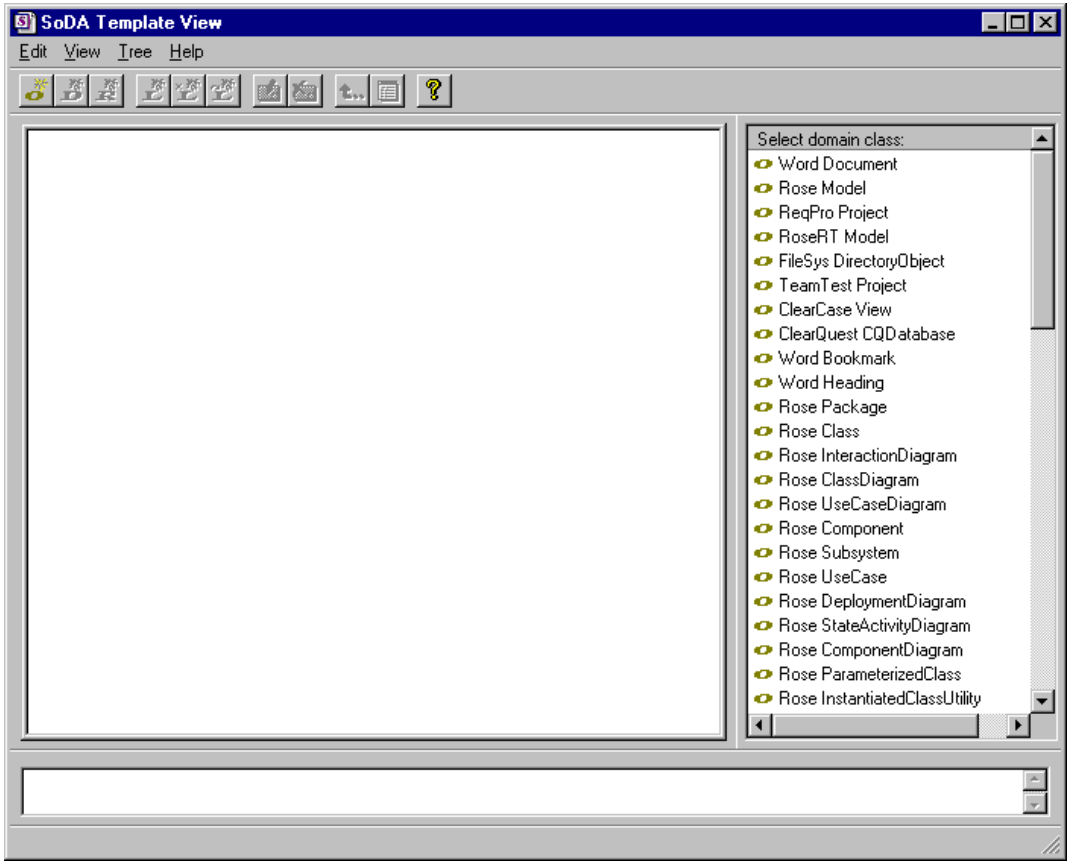
Generate Report Creates a printable Word document with the same text that is in the template view

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:

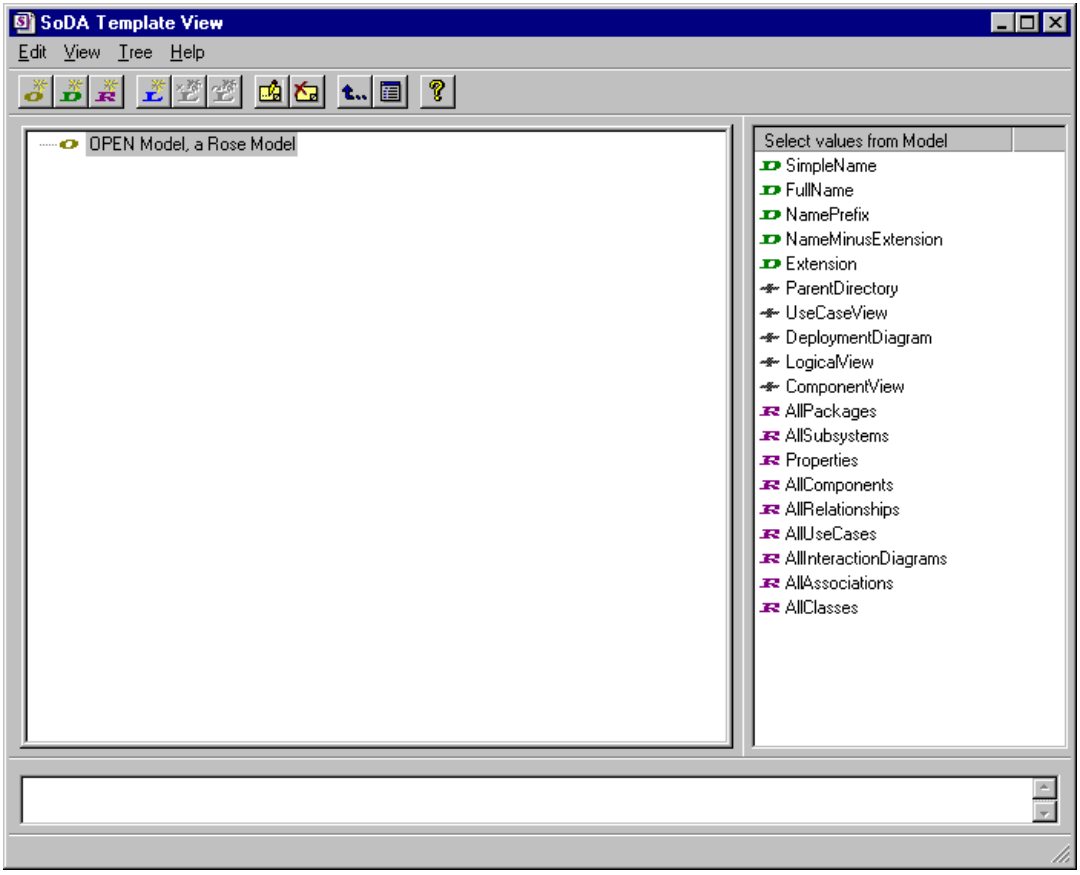


Here you select the starting point for the template. If you are unsure where you want to begin, find the first line that contains your source (Rose Model, ReqPro Project, ClearQuest database or TeamTest Repository).

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

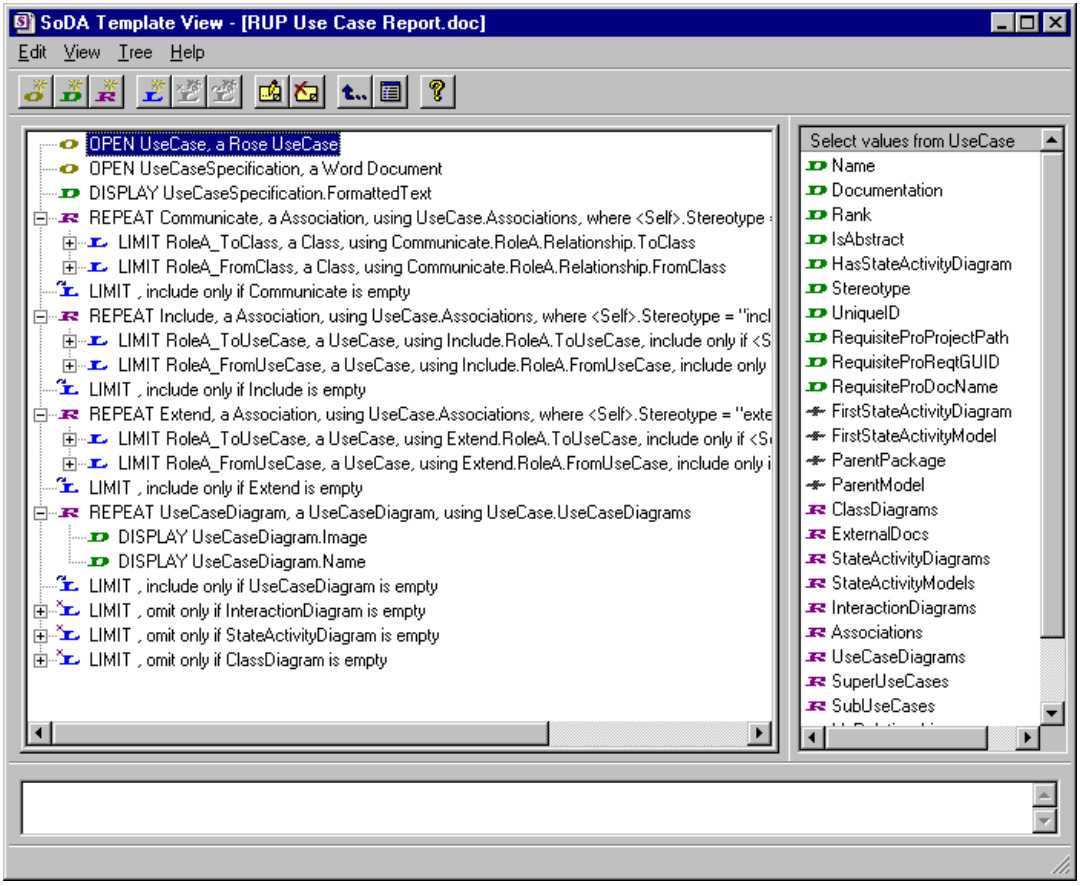
Once you have selected a source, or if you are adding a section to an existing template, you will see a panel that looks like this:



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

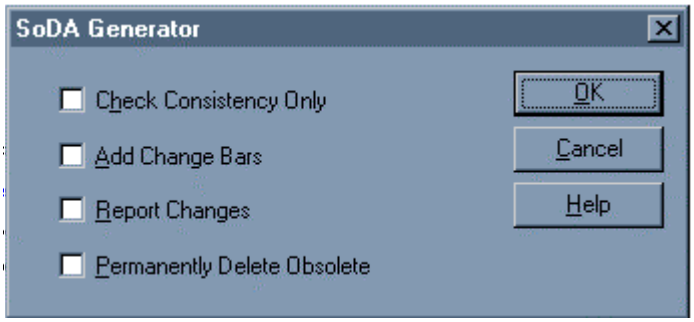
The toolbar in the Template View provides additional features:

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.

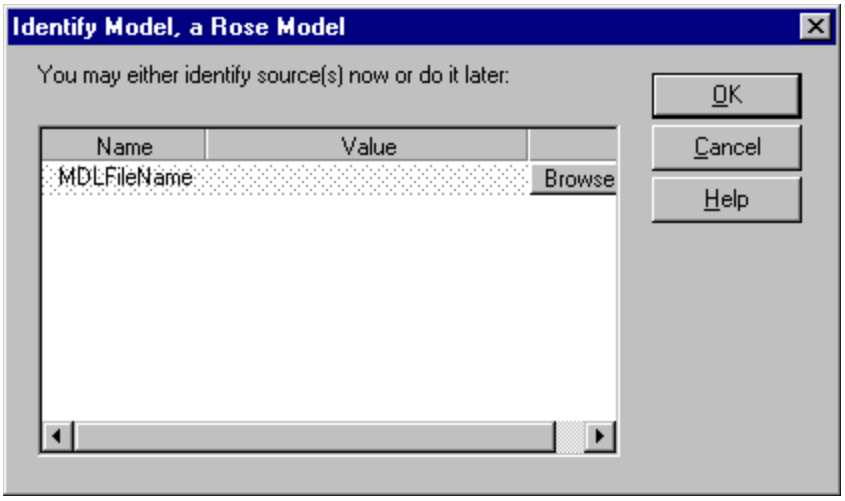
Because an object may be moved by accident (and not deleted), SoDA allows you to leave "deleted" text in your document. This allows you to edit a link to point to a moved object's new location or modify a query to compensate for some other change in the information source.

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.

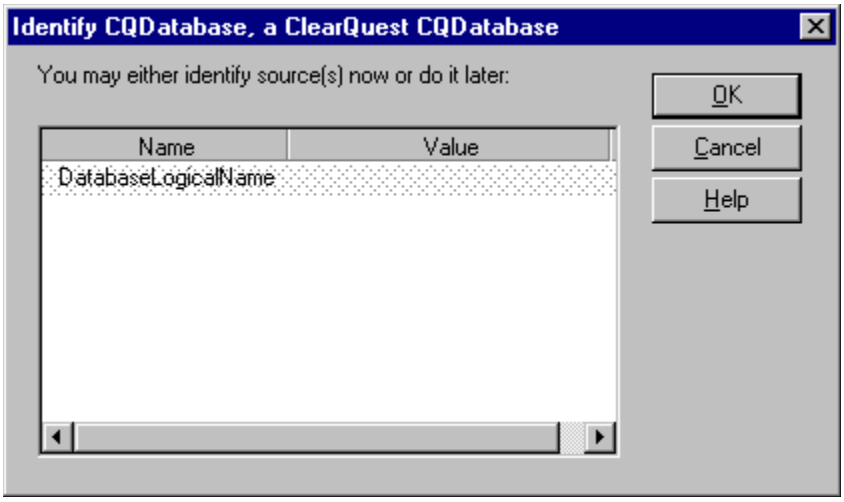
Identify the <Class> Dialog Box

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.

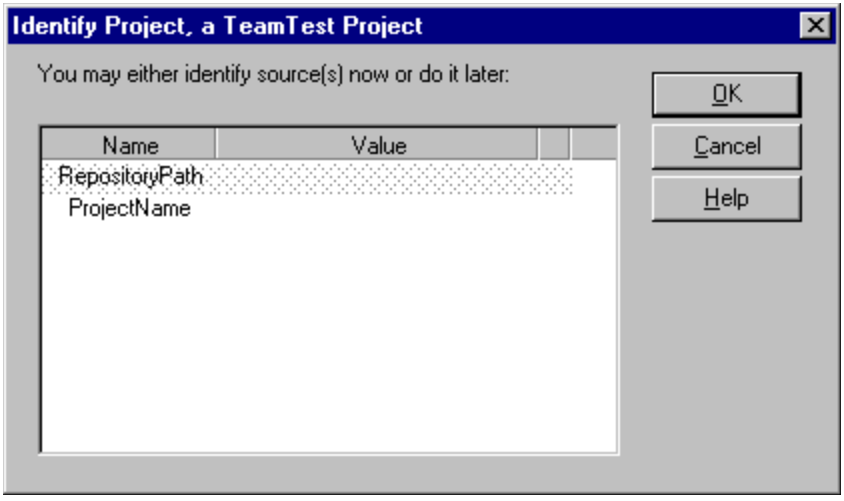


The grid area contains the specific details that identify a particular object of the specified class. In most cases, only one argument is required; however, sometimes there are two or more, depending on the class. If there is a Browse button, you can browse for a specific model, project, or file. Otherwise,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

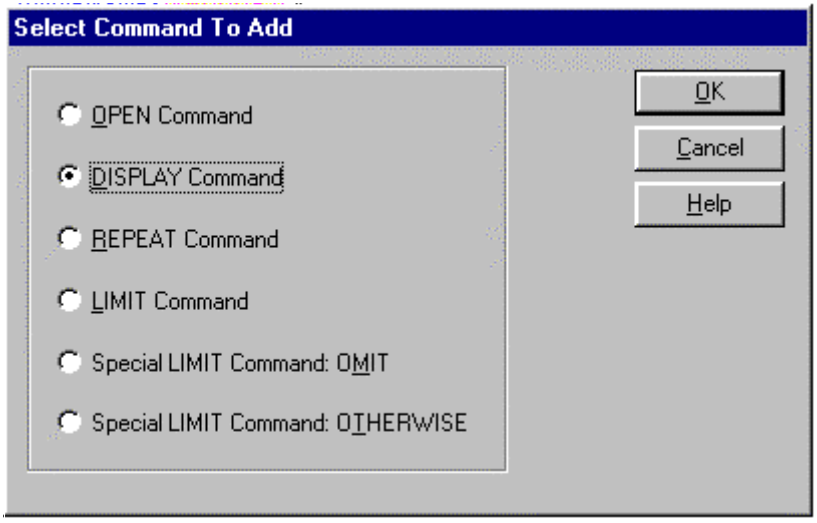


For TeamTest, enter the path for the repository. The repository must exist in the Rational Adminstrator application on your system. The ProjectName is defined in the repository. Type the project name in the applicable Value field. Be sure that spaces within the path and project name match the source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



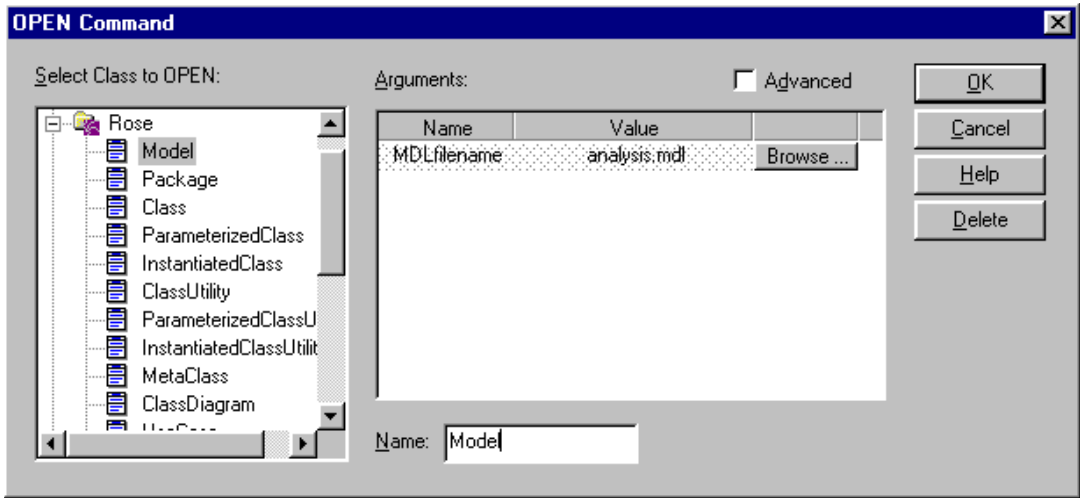
The Select Command to Add dialog box will display a list of the commands that can be added:

- 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.

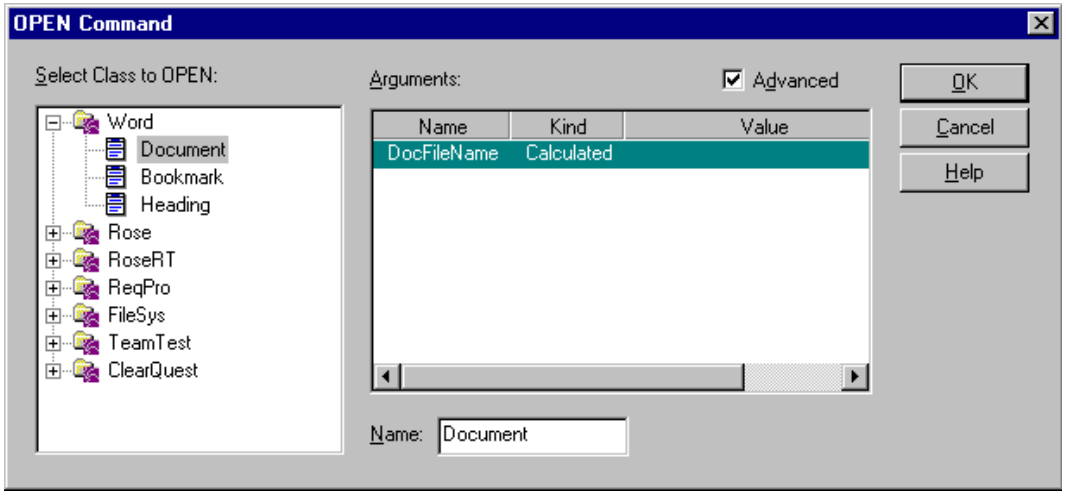
- In the tree control, select **ExternalDoc > Value**.

- 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.

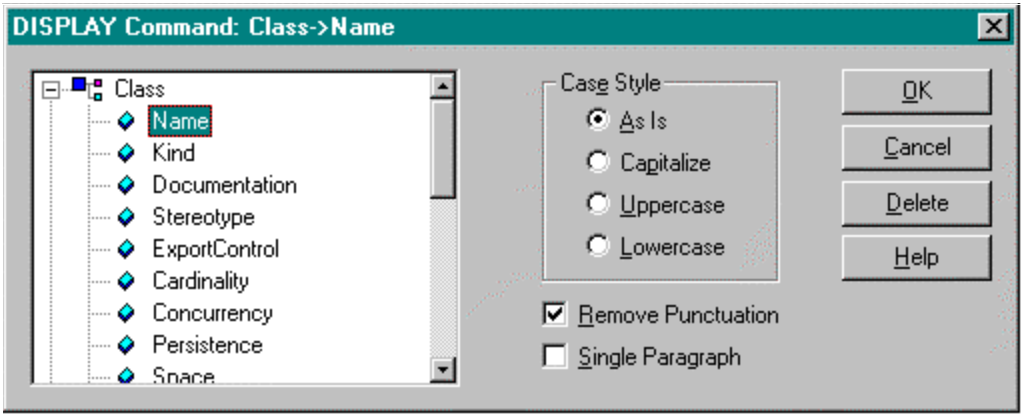
8 In the Select Attribute area, choose **WordFile > FormattedText**.

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.

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.

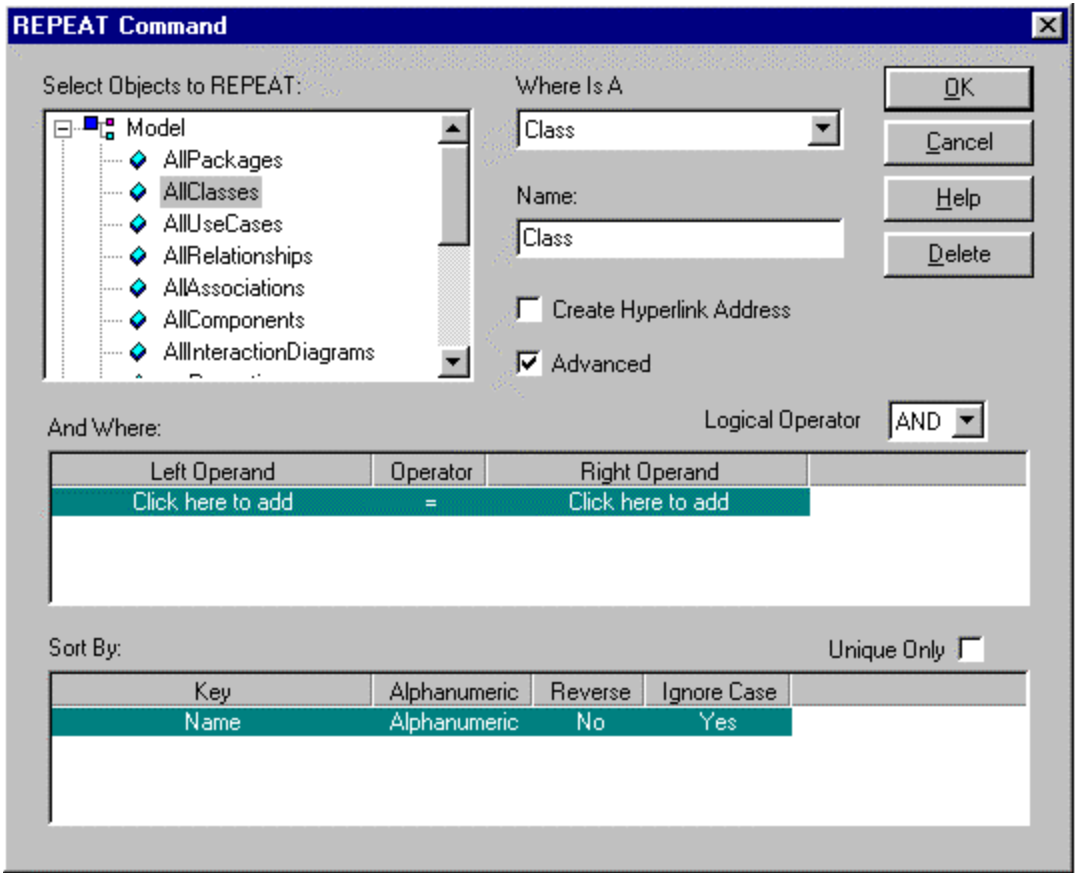
Width and Height

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.

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.

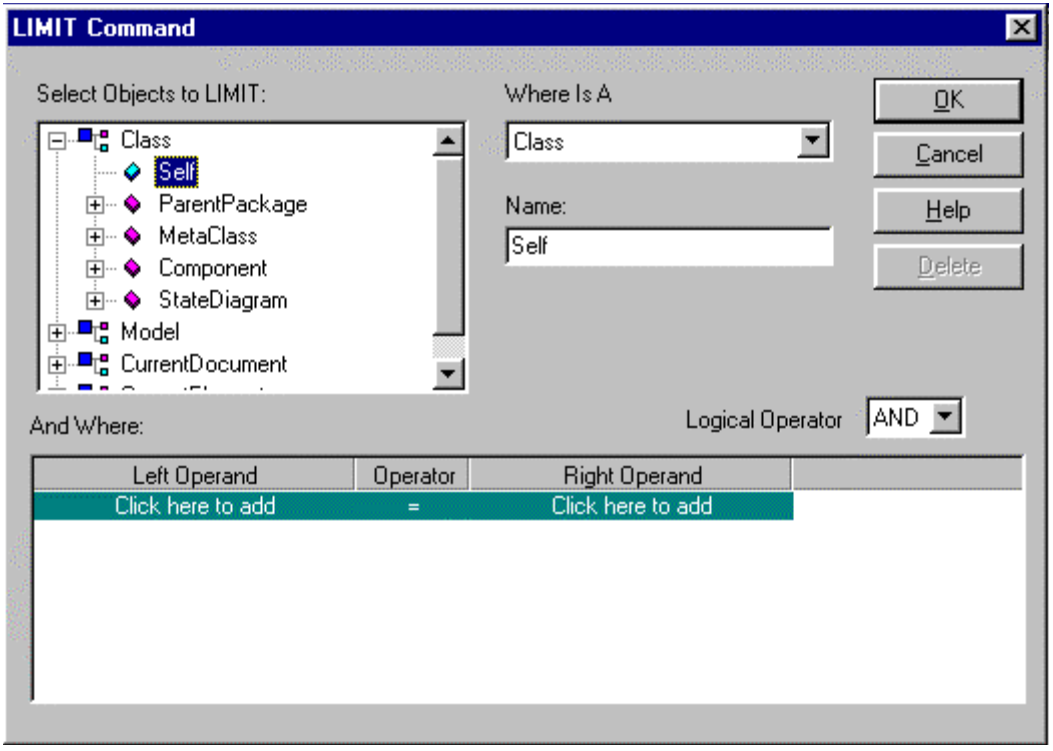
Choose "Reverse Order" for reverse alphabetical or numeric order.

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.

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.

6 Rational SoDA for Word Domains

This chapter describes the SoDA domains and their classes. The overview includes a description of:

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

For information on specific domains, refer to the following sections:

Overview

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 domain (.dom) description file in the SODA_HOME\domains 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.dom 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. SoDA 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
```

Apex NT Domain

The Apex domain allows you to incorporate information from Ada units, Configurations, Views, and Subsystems into your SoDA documents.

Many Apex-domain classes are derived from the File System (FileSys) domain classes File and Directory, and therefore inherit the attributes and relationships of those classes.

Classes that represent structure within Ada units are based on the ASIS interface. By default, SoDA opens and closes ASIS libraries as the units with those libraries are accessed. Performance when building a document from a number of large libraries can therefore suffer. Performance can be improved by defining the scope of the project.

Apex NT Domain Classes

Apex NT CompositeType Class

An array or record type declaration.

CompositeType is a subclass of Type.

Attributes specific to CompositeType

None

Relationships specific to CompositeType

None

Apex NT CompUnit Class

An Ada compilation unit, i.e., a file ending in **.ada**.

CompUnit is a subclass of Apex.File.

Subclasses of CompUnit: UnitSpec, UnitBody, SubunitBody.

Attributes specific to CompUnit

Name – Kind	Description
AdaName – text	The name assigned to the unit in the source code. If the unit is a subunit, also includes the name of the parent(s). For example, the AdaName of `package Text_Io` is `Text_Io`. The AdaName of the subunit `Integer_Io` is `Text_Io.Integer_Io`.
AttachedComments – text	Comments preceding or following the unit declaration and not separated from it by a blank line. See also Declaration.AttachedComments .
PrecedingComments – text	Comments that come before the unit declaration and are not separated from it by a blank line
FollowingComments – text	Comments that come after the unit declaration and are not separated from it by a blank line.
IsPrivate – text	**True** if the unit is a private library unit; otherwise **False**.
IsChild – text	**True** if the unit is a child library unit; otherwise **False**.
HasChildren – text	**True** if the unit has child library units; otherwise **False**.

Relationships specific to CompUnit

Name – Kind	Class	Description
ChildUnits – n	CompUnit	The child library units for this compilation unit, if any.
DescendantUnits – n	CompUnit	The child library units for this compilation unit, if any.
ParentCompUnit – 1	CompUnit	If the unit is a child library unit, this relationship will return the parent.
UnitDeclaration – 1	Declaration	The declaration of the compilation unit.
WithedUnits – n	UnitSpec	All unit specs upon which this compilation unit directly depends, i.e., the contents of the *with* clauses of the unit.

Apex NT Configuration Class

A configuration file contains a list of views, one per subsystem. Configurations are typically used to specify a set of views that make up a particular version of a system.

Configuration is a subclass of FileSys.File.

Attributes specific to Configuration

None

Relationships specific to Configuration

Name – Kind	Class	Description
AllViews – n	View	All the views listed in the configuration file.
AllSubsystems – n	Subsystem	The enclosing subsystems of all the views listed in the configuration file.
ImportedViews – n	View	All the views imported by any view in the configuration but not themselves listed in the configuration.

Apex NT Declaration Class

Any Ada declaration.

Subclasses of Declaration:

PackageSpec, PackageBody, SubprogramSpec, SubprogramBody, Parameter, Object, Type, Task, Entry, Exception.

Attributes specific to Declaration

Name – Kind	Description
IsGeneric – text	**True** if the declaration is an Ada generic, otherwise **False**.
IsVisible – text	**True** if the declaration is visible to other compilation units, otherwise **False**.
IsAbstract – text	**True** if the declaration (type or subprogram) is abstract, otherwise **False**.
IsAliased – text	**True** if the declaration is aliased, otherwise **False**.
SimpleName – text	The identifier of the declaration. For example, the SImpleName of the `declaration procedure Put (C : Character);` is `Put`
MyKind – text	The Apex domain class name of the declaration. For example, MyKind of a package specification returns **PackageSpec**.

Keyword – text	The Ada keyword associated with the declaration. For example, the keyword of a package specification (or body) is **package**. Other possible responses are **procedure**, **function**, **type**, and **task**. Returns a null string for declarations that do not have a keyword.
Text – text	The full image of the declaration.
AttachedComments – text	Comments preceding or following the declaration and not separated from it by a blank line. The comment characters themselves (i.e., "**--**) are not included. For example, given the declaration: -- Character output put (C : Character); -- Provides output of a single character AttachedComments returns: Character output. Provides output of a single character.
PrecedingComments – text	Comments that come before the declaration and are not separated from it by a blank line. See also Declaration.AttachedComments.
FollowingComments – text	Comments that come after the declaration and are not separated from it by a blank line. See also Declaration.AttachedComments.

Relationships specific to Declaration

Name – Kind	Class	Description
ParentCompUnit – 1	CompUnit	The compilation unit in which the declaration is found.
Parent – 1	Declaration	The declaration, if any, that encloses this declaration. For example, given the code: package One procedure A (X : Integer); end One; **One** is the parent of **A**, and **A** is the parent of **X**.
VisibleDeclarations – n	Declaration	The declarations in a package spec that can be referenced by clients of that spec (i.e., declarations not in the private part). Returns a null list if the declaration is not a package spec.
PrivateDeclarations – n	Declaration	The declarations in a package spec that cannot be referenced by clients of that spec (i.e., declarations in the private part). Returns a null list if the declaration is not a package spec.
AllDeclarations – n	Declaration	All declarations in the public or private parts of a package spec or within a package body.
GenericParameters – n	Declaration	The generic formal parameters of a generic declaration. Returns a null list if the declaration is not generic.

Apex NT Entry Class

An Ada task entry.

Entry is a subclass of Declaration.

Attributes specific to Entry

None

Relationships specific to Entry

Name – Kind	Class	Description
Parameters -- n	Parameter	The formal parameters of this task entry declaration.

Apex NT Exception Class

An Ada exception.

Exception is a subclass of Declaration.

Attributes specific to Exception

None

Relationships specific to Exception

None

Apex NT File Class

Any file contained within an Apex View or a subdirectory of that view. In addition to normal File System domain attributes, Apex CMVC information is available for objects of this class.

File is a subclass of FileSys.File.

Subclasses of File: CompUnit.

Attributes specific to File

Name – Kind	Description
IsControlled – text	**True** if the file is under Apex CMVC control, otherwise **False**.
VersionHistory – text	The CMVC version history name. Returns a null string if the file is not controlled.
VersionNumber – text	The CMVC version number. Returns a null string if the file is not controlled.

Relationships specific to File

Name – Kind	Class	Description
EnclosingView – 1	View	The View containing the file.

Apex NT FunctionBody Class

An Ada function body.

FunctionBody is a subclass of SubprogramBody.

Attributes specific to FunctionBody

None

Relationships specific to FunctionBody

Name – Kind	Class	Description
ReturnType – 1	Type	The declaration of the type returned by the function.

Apex NT FunctionSpec Class

An Ada function specification.

FunctionSpec is a subclass of SubprogramSpec.

Attributes specific to FunctionSpec

None

Relationships specific to FunctionSpec

Name – Kind	Class	Description
ReturnType – 1	Type	The declaration of the type returned by the function.

Apex NT Object Class

An Ada variable or constant declaration.

Object is a subclass of Declaration.

Attributes specific to Object

None

Relationships specific to Object

Name – Kind	Class	Description
MyType – 1	Type	The declaration of the object's type.

Apex NT PackageBody Class

An Ada package body.

PackageBody is a subclass of Declaration.

Attributes specific to PackageBody

None

Relationships specific to PackageBody

Name – Kind	Class	Description
MySpec – 1	PackageSpec	The package specification corresponding to this body.

Apex NT PackageSpec Class

An Ada package specification.

PackageSpec is a subclass of Declaration.

Attributes specific to PackageSpec

None

Relationships specific to PackageSpec

Name – Kind	Class	Description
MyBody – 1	PackageBody	The package body corresponding to this specification.

Apex NT Parameter Class

An Ada formal parameter declaration. Can be part of a procedure, function, or task entry declaration.

Parameter is a subclass of Declaration.

Attributes specific to Parameter

Name – Kind	Description
Mode – text	The mode of the parameter: **in**, **out**, or **in out**.

Relationships specific to Parameter

Name – Kind	Class	Description
MyType – 1	Type	The declaration of the parameter's type.

Apex NT PrimitiveType Class

A discrete, real, or fixed type declaration.

PrimitiveType is a subclass of Type.

Attributes specific to PrimitiveType

None

Relationships specific to PrimitiveType

None

Apex NT ProtectedType Class

A protected array or record type declaration.

ProtectedType is a subclass of Type.

Attributes specific to ProtectedType

None

Relationships specific to ProtectedType

None

Apex NT Statement Class

An Ada statement.

Attributes specific to Statement

Name – Kind	Description
MyKind – text	The statement kind, as defined by ASIS. Examples: AN_IF_STATEMENT, A_CASE_STATMENT, A_BLOCK_STATEMENT.
Text – text	The full image of the statement.
CommentsWithin – text	All comments that appear after the first line of the statement, but before the last line.
PrecedingComments – text	Comments that come before the statement and are not separated from it by a blank line.
FollowingComments – text	Comments that come after the statement and are not separated from it by a blank line.

Relationships specific to Statement

Name – Kind	Class	Description
ParentDeclaration -- 1	Declaration	The subprogram body declaration that encloses this statement.

Apex NT SubprogramBody Class

An Ada procedure or function body.

SubprogramBody is a subclass of Declaration.

Subclasses of SubprogramBody: FunctionBody.

Attributes specific to SubprogramBody

None

Relationships specific to SubprogramBody

Name – Kind	Class	Description
MySpec – 1	SubprogramSpec	The specification corresponding to this subprogram body.
Parameters – n	Parameter	The formal parameters of this subprogram body.
Statements – n	Statement	The first-level statements declared in the subprogram body block.

Apex NT SubprogramSpec Class

An Ada procedure or function specification.

SubprogramSpec is a subclass of Declaration.

Subclasses of SubprogramSpec: FunctionSpec.

Attributes specific to SubprogramSpec

None

Relationships specific to SubprogramSpec

Name – Kind	Class	Description
MyBody – 1	SubprogramBody	The subprogram body corresponding to this specification.

Apex NT Subsystem Class

Subsystem Class is an Apex subsystem, i.e., a directory ending in .ss and containing zero or more views.

Subsystem is a subclass of FileSys.Directory .

Attributes specific to Subsystem

None

Relationships specific to Subsystem

Name – Kind	Class	Description
AllViews – n	View	All the views contained within this subsystem, i.e., directories ending in **.wrk** or **.rel**.

Apex NT SubunitBody Class

A CompUnit that is an Ada subunit.

SubunitBody is a subclass of CompUnit.

Attributes specific to SubunitBody

None

Relationships specific to SubunitBody

Name – Kind	Class	Description
EnclosingUnit – 1	UnitBody	The unit body in which this subunit is declared.
AllDependencies – n	CompUnit	All units upon which this body depends for compilation. Includes units in the *with* clauses of the specification and enclosing unit bodies.
MySubunits – n	SubunitBody	All subunits declared within this subunit.

Apex NT Task Class

An Ada task object or type.

Task is a subclass of Declaration.

Subclasses of Task: TaskType.

Attributes specific to Task

None

Relationships specific to Task

Name – Kind	Class	Description
EntryPoints – n	Entry	The entry points of the task.

Apex NT TaskType Class

An Ada task type

TaskType is a subclass of Type and a subclass of Task.

Attributes specific to TaskType

None

Relationships specific to TaskType

None

Apex NT Type Class

An Ada type declaration

Type is a subclass of Declaration.

Subclasses of Type: PrimitiveType, CompositeType, ProtectedType, TaskType.

Attributes specific to Type

Name – Kind	Description
IsTagged – text	**True** if this is a tagged type, otherwise **False**.

Relationships specific to Type

None

Apex NT UnitBody Class

A CompUnit that is an Ada body.

UnitBody is a subclass of CompUnit .

Attributes specific to UnitBody

None

Relationships specific to UnitBody

Name – Kind	Class	Description
MySpec (1)	UnitSpec	The compilation unit specification that corresponds to this body.
AllDependencies – n	CompUnit	All units upon which this body depends for compilation. Includes units in the *with* clauses of the specification.
MySubunits – n	SubunitBody	All subunits declared within this body.

Apex NT UnitSpec Class

A CompUnit that is an Ada specification.

UnitSpec is a subclass of CompUnit.

Attributes specific to UnitSpec

Name – Kind	Description
HasBody – text	**True** if the unit has a corresponding body; otherwise **False**.

Relationships specific to UnitSpec

Name – Kind	Class	Description
MyBody – 1	UnitBody	The compilation unit body that corresponds to this specification.
ReferencingUnits – n	CompUnit	All compilation units that reference (i.e., *with*) this specification. The scope of units considered in evaluating this relationship is controlled by the **SODA_APEX_LIBRARY** environment variable.

Apex NT View Class

An Apex view, i.e., a directory ending in **.wrk** or **.rel** and contained within a Subsystem.

View is a subclass of FileSys.Directory.

Attributes specific to View

None

Relationships specific to View

Name – Kind	Class	Description
EnclosingSubsystem – 1	Subsystem	The Subsystem containing this view.
ImportedViews – n	View	The Views imported by this view.
Subdirectories – n	ViewDirectory	The directories immediately contained within this view.
AllSubdirectories – n	ViewDirectory	All directories contained within this view or, recursively, within those directories, their subdirectories, etc.

Files – n	File	The files immediately contained in the view. Returns objects of the Apex domain File class, which differ from File System (FileSys) domain files in that Apex CMVC information is available for them.
AllFiles – n	CompUnit	All files contained within this view or, recursively, within the subdirectories of the view, their subdirectories, etc. Returns objects of the Apex domain File class, which differ from File System (FileSys) domain files in that Apex CMVC information is available for them.
AllCompUnits – n	CompUnit	All Ada units contained in this view or, recursively, within the subdirectories of the view, their subdirectories, etc.
UnitSpecs – n	UnitSpec	The Ada unit specifications immediately contained in this view, i.e., files ending in **.1.ada**.
AllUnitSpecs – n	UnitSpec	All Ada unit specifications contained in this view or, recursively, within the subdirectories of the view, their subdirectories, etc.
ImportedUnits – n	UnitSpec	All units from other views visible from within this view.
ExportedUnits – n	UnitSpec	All units in this view that are visible to other views.

Apex NT ViewDirectory Class

A directory within an Apex view.

ViewDirectory is a subclass of FileSys.Directory.

Attributes specific to ViewDirectory

None

Relationships specific to ViewDirectory

Name – Kind	Class	Description
EnclosingView – 1	View	The View containing this directory.
Subdirectories – n	ViewDirectory	The ViewDirectories immediately contained within this directory.
AllSubdirectories – n	ViewDirectory	All ViewDirectories contained within this directory or, recursively, contained in its directories, their subdirectories, etc.

Files – n	File	The files immediately contained in the directory. Returns objects of the Apex domain File class, which differ from File System (FileSys) domain files in that Apex CMVC information is available for them.
AllFiles – n	File	All files contained within this directory or, recursively, within its subdirectories, their subdirectories, etc. Returns objects of the Apex domain File class, which differ from File System (FileSys) domain files in that Apex CMVC information is available for them.
CompUnits – n	CompUnit	The Ada units immediately contained within this directory, i.e., files ending in **.ada**.
AllCompUnits – n	CompUnit	All Ada units contained in this directory or, recursively, within its subdirectories, their subdirectories, etc.
UnitSpecs – n	UnitSpec	The Ada unit specifications immediately contained in this directory, i.e., files ending in **.1.ada**.
AllUnitSpecs – n	UnitSpec	All Ada unit specifications contained in this directory or, recursively, within its subdirectories, their subdirectories, etc.

Subsystem Structure for Apex NT Templates

SoDA templates relating to Rational Apex NT are located in the following directory:

```
<install drive>\Program Files\Rational\SoDAWord\template\Apex
```

Note: SoDA's Apex templates are designed specifically for use in a particular hierarchy of Apex subsystems and view. For more details, see Apex NT Subsystem Structure.

Apex NT Subsystem Structure

SoDA's Apex 498 templates assume a certain structure for the subsystems in which they reside and from which documents will be generated. This default structure is described below. If you want, you can alter the templates to use a structure of your choosing.

```
a_system\
. interface_views.cfg
. a_system.ss\view.wrk\
.               .       .       irs\
.               .       .       idd\
.               .       .       .       interface_diagram.*
.               cscis\a_csci\
.               .       .       a_csci.ss\view.wrk\
.               .       .       .       .       srs\                    #
.               .       .       .       .       sdd\                    #
.               .       .       .       .       .       system_architecture.*
.               .       .       .       .       .       csci_architecture.*
.               .       .       .       .       .       states_and_modes.*
.               .       .       all_views.cfg
.               .       .       exported_views.cfg
.               .       .       local_views.cfg
.               .       .       source_subsystem.ss\view.wrk\
.               .       .       .       .sdd\                           #
.               .       .       .       .       .preliminary\           #
.               .       .       .       .       .       .architecture.*
.               .       .       .       .       .       .scenarios\     #
.               .       .       .       .       .       .       .scenario_n.*
.               .       .       .       .       .detailed\              #
.               .       .       .       .       .       .architecture.*
.               .       .       .       .       .       .instance_diagrams\     #
.               .       .       .       .       .       .       .instance_n.*
.               .       .       .       .       .       .scenarios\     #
.               .       .       .       .       .       .       .scenario_n.*
```

Note: The directories marked with a (**#**) can be empty, but they must exist for the documents that reference them to generate without errors.

Apex NT Subsystem Directories

SoDA System Directory

A 498 system is represented with a directory containing a subdirectory named **cscis** and an Apex configuration named **interface_views.cfg**. The **cscis** directory contains all the CSCI directories for the system. The **interface_views.cfg** configuration contains the views that are external interfaces of the system.

SoDA CSCI Directories

A 498 CSCI is represented with a directory containing three Apex configu ration files **all_views.cfg**, **exported_views.cfg**, and **local_views.cfg**. Each of these files contains a list of Apex views, one per subsystem.

SoDA Document Subsystems

The SoDA templates assume that documents will be located within Apex subsystem views with a specific relationship to their information sources.

When you use the standard SoDA templates, you will create one additional subsystem to contain the system level documents and an additional subsystem for each CSCI in the system.

System-Level Documents

You create a subsystem in the root directory of the project to hold the project-wide documents (the IRS and IDD). This subsystem can have any name, but a good convention is to give it the same name as the project root directory but with the **.ss** extension. For example:

```
a_system\a_system.ss\view.wrk
```

The IRS and IDDdocuments are located in subdirectories within this subsystem's views.

CSCI-Level Documents

A directory is used to represent a CSCI. You create a subsystem within that directory to hold the CSCI-level documents (the SRS and SDD). This subsystem can have any name but a good convention is to give it the same name as the CSCI's directory but with the **.ss** extension. For example:

```
a_system\cscis\a_csci\a_csci.ss\view.wrk
```

The SRS and SDD documents are located in subdirectories within this subsystem's views.

SoDA Document Directories

Each of SoDA's 498 templates is, by convention, located in its own subdirec tory. In some cases, the templates expect these directories to contain addi tional subdirectories and files.

Requirements Documents

For requirements, SoDA supports a project-wide IRS and CSCI-level SRSs.

Interface Requirements Specification (IRS)

The IRS template is located in an **irs** subdirectory of the project-wide document subsystem view. For example:

```
a_system\a_system.ss\view.wrk\irs
```

The IRS template is primarily used as a template in which to manually enter the document content.

Software Requirements Specification (SRS)

The SRS template is located in an **srs** subdirectory of each CSCI's document subsystem view. For example:

```
a_system\cscis\a_csci\a_csci.ss\view.wrk\srs
```

The SRS template is primarily used as a template in which to manually enter the document content.

Design Documents

For design, SoDA supports a project-wide IDD and CSCI-level SDDs.

Interface Design Document IDD)

The IDD template is located in an **idd** subdirectory of the project-wide document subsystem view. For example:

```
a_system\a_system.ss\view.wrk\idd
```

The IDD template contains SoDA commands to automatically generate document sections for interfaces that are to be implemented as Ada program calls. Sections for interfaces implemented in other ways (such as pipes, sockets, etc.) are entered directly into the document.

The information sources for the automatically generated document sections are the exported program units of the project's code-containing subsystems. If a file named **interface_diagram.*** is present in the IDD document direc tory it will be included in the document. This file can be in any graphic format imported by Word.

Software Design Document (SDD)

The SDD template is located in an **sdd** subdirectory of each CSCI's document subsystem view. For example:

```
a_system\cscis\a_csci\a_csci.ss\view.wrk\sdd
```

The SDD template contains SoDA commands to generate most of the docu ment. The information sources for the SDD are the code-containing subsystems of the CSCI. In addition, if files named **system_architecture.***, **csci_architecture.***, and/or **states_and_modes.*** are found in the SDD document directory they will be included in the document.

SoDA Source Subsystems

The code-containing subsystems of the project are the information source for the automatically generated sections of the IDD and SDD. To generate either of these documents, you need to create specifications for all units that are to appear in the document. You do not need to do anything special to the Ada units for SoDA to generate these documents. However, if you follow the convention of attaching comments to declarations, those comments will appear in the documents. A comment is considered attached to a declara tion if there are no blank spaces between the element and the comment. For example:

```
-- this comment is attached to A_Package
package A_Package is
-- this comment is attached to A_Package as well

    -- this comment is attached to A_procedure
    procedure A_Procedure;
    --
    -- this comment is attached to A_Procedure as well

    -- this comment is attached to A_Function
```

```
function A_Function return Integer;
procedure Another_Procedure (

        A_Parameter : Integer
        -- this comment is attached to A_Parameter but
        -- not Another_Procedure

);
```

In addition to extracting information from subsystems and Ada units, SoDA looks for graphics in the following subdirectories of each source subsystem view:

- A CSC interface architecture diagram in SDD section 4.1.X from:

  `<view>\sdd\preliminary\architecture.*`

- Object scenario diagrams in SDD section 4.1.X from:

  `<view>\sdd\preliminary\scenarios\*`

- A CSC implementation architecture diagram in SDD section 5.X from:

  `<view>\sdd\detailed\architecture.*`

- Object scenario diagrams in SDD section 5.1.X from:

  `<view>\sdd\detailed\scenarios\*`

- Object instance diagrams in SDD section 5.1.X from:

  `<view>\sdd\detailed\instance_diagrams\*`

Note: The subdirectory names for the location of graphics in the source subsystem views are defined in domain extensions. For more details, see SoDA's Domain Extension File.

ClearCase Domain

SoDA's new source domain for ClearCase enables extraction of version control, activity management, workspace management, and VOB meta-data information from ClearCase 4.0 for inclusion in your SoDA documents. For details on using the ClearCase domain, please refer to the SoDA online help.

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 SoDA 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 Open commands for the following ClearCase objects request a VOBIdentifier instead of a VOB tag. (VOBIdentifier signifies that either a VOB tag or a pathname \ VOB tag can be entered.)

- VOB
- Activity
- AttributeType
- BranchType
- HyperlinkType
- LabelType
- TriggerType

ClearCase Domain Classes

ClearCase Activity Class

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.

Activity is a subclass of VOBObject Class.

Attributes specific to Activity

Name - Kind	Description
Headline - text	The title of this activity.
LockedBy – text	The name of the user who locked the activity.
LockedDescription - text	The description of the lock on this activity.
LockedOn – text	The date the activity was locked.
Master - text	The master replica for this activity.
State - text	The state of the lock on this activity.

Relationships specific to Activity

Name - Kind	Class	Description
ChangeSet – n	Version	The versions in this activity's change set.
ExcludedUsers - n	Name	The list of users exempted from the lock.
Lock – 1	Lock	The lock on this activity.
NameResolverView – 1	View	A "best guess" view for resolving the names of versions in a change set.
VOB - 1	VOB	The VOB containing the activity.

ClearCase Attribute Class

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.

Attributes specific to Attribute

Name - Kind	Description
Value - text	The attribute value.

Relationships specific to Attribute

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

ClearCase AttributeType Class

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).

AttributeType is a subclass of VOBObject Class.

Attributes specific to AttributeType

Name - Kind	Description
Constraint – text	The constraint for this attribute type.
DefaultValue – text	The default value for this attribute type.
Group – text	The group to which this attribute type belongs.
HasSharedMastership - text	Whether this attribute type is shared or can be mastered.
LockedBy – text	The name of the user who locked this attribute type.
LockedDescription – text	The user comment for the lock
LockedOn - text	The date this attribute type was locked.
LowerIsInRange – text	Whether or not the lower value is in the range of legal values for this attribute type.

LowerValue – text	The lower value for this attribute type.
Master - text	The master replica for this attribute type.
NumberOfEnumValues – text	The number of enumerated values for this attribute type.
Owner – text	The owner of this attribute type.
Scope – text	The scope of this attribute type (for example, local to this VOB).
State - text	The state of the lock on this attribute type.
UpperIsInRange – text	Whether or not the upper value is in the range of legal values for this attribute type.
UpperValue - text	The upper value for this attribute type.
ValueType – text	The value type for this attribute.

Relationships specific to AttributeType

Name - Kind	Class	Description
EnumValues - n	Value	The enumerated values for this attribute type.
ExcludedUsers - n	Name	The list of users who are exempt from the lock.
Lock - 1	Lock	The lock on this attribute type.
VOB - 1	VOB	The VOB containing this attribute type.

ClearCase Branch Class

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.

Branches have both names and paths. The SoDA Browse feature facilitates accessing the path of the Branch. 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.

Branch is a subclass of VOBObject Class.

Attributes available for Branch

Name - Kind	Description
BranchPath - text	The pathname of this branch.
ExtendedPath - text	Extended pathname of the branch.
LockedBy – text	The name of the user who locked the branch.
LockedDescription – text	The description of the current lock on the branch.
LockedOn - text	The date the branch was locked.
Master - text	The master replica for this branch.
State - text	The state of the lock on this branch.

Relationships available for Branch

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

ClearCase BranchType Class

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

BranchType is a subclass of VOBObject Class.

Attributes available for BranchType

Name - Kind	Description
Constraint - text	The constraint for this branch type.
Group - text	The group to which this branch type belongs.
LockedBy – text	The name of the user who locked this branch type.

LockedDescription – text	The user comment for the lock on this branch type.
LockedOn - text	The date on which this branch type was locked.
Master - text	The master replica for this branch type.
Owner - text	The owner of this branch type.
Scope - text	The scope of this branch type (for example, local to this VOB)
State - text	The state of the lock on this branch type.

Relationships available for BranchType

Name - Kind	Class	Description
ExcludedUsers - n	Name	The list of users who are exempt from the lock on this branch type.
Lock - 1	Lock	The lock on this branch type.
VOB - 1	VOB	The VOB containing this branch type.

ClearCase CheckedOutFile Class

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).

CheckedOutFile is a subclass of Version Class.

Attributes specific to CheckedOutFile

Name - Kind	Description
IsReserved - text	Whether or not this checkout is reserved.

Relationships specific to CheckedOutFile

Name - Kind	Class	Description
ByView – 1	View	The view to which this file is checked out.

ClearCase Element Class

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.

Element is a subclass of File Class.

Attributes available for Element

Name - Kind	Description
ElementType - text	The element type of this element.
LockedBy – text	The use who locked this element.
LockDescription - text	A comment associated with the history record for the lock.
LockedOn - text	The date the element was locked.
Group - text	The group to which this element belongs.
Master - text	The master replica for this element.
Owner - text	The owner of the element.
State - text	The current state of the lock on this element.

Relationships available for Element

Name - Kind	Class	Description
AllBranches - n	branch	All branches in the version tree for this element.
AllCheckedOutFiles - n	CheckedOutFile	The versions of the element checked out to any view.
AllVersions - n	Version	Versions in the version tree for this element.
ExcludedUsers - n	Name	Array of string values containing the names of users exempted from the lock being created.
CheckedOutFile – 1	CheckedOutFile	The version of the element checked out to the associated view.
GroupPermissions – n	Name	The group permissions of the element (users within the same group have these permissions).
Lock – 1	Lock	The lock on this element.
OtherPermissions – n	Name	The other permissions of the element (all users).
OwnerPermissions – n	Name	The owner permissions of the element (the owner has these permissions).

ParentDirectory - 1	Element	This element's parent directory element.
RootVersion - 1	Version	The particular version of this element specified by the version selector.
Triggers – n	Trigger	The collection of triggers attached to this file of directory element.

ClearCase File Class

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.

File is a subclass of VOBObject Class.

Attributes specific to File

Name - Kind	Description
ExtendedPath - text	The VOB-extended pathname of this file system object.
Extension - text	The file extension (the portion after the final dot).
IsDirectory - text	Whether or not the file is a directory.
NameMinusExtension - text	The simple name of the file without the extension and final.dot.
Path - text	The pathname to this file system object.
SimpleName - text	The simple name of the file, i.e., the name of the file without the path.

Relationships specific to File

Name - Kind	Class
View – 1	View
VOB – 1	VOB

ClearCase HistoryRecord Class

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.

Attributes available for HistoryRecord

Name - Kind	Description
Comment - text	The comment associated with the operation indicated by this history record.
Date - string	The date and time the operation was executed.
EventKind - text	Indicates the type of operation that was executed.
Group - text	The name of the login group that performed the operation indicated by this history record.
Host - text	The name of the host machine from which the operation indicated by this history record was executed.
UserFullName - text	The full name of the user who performed the operation indicated by this history record.
UserLoginName - text	The login name of the user who performed the operation indicated by this history record.

Relationships available for HistoryRecord

Name - Kind	Class	Description
VOB - 1	VOB	The VOB containing the object to which the operation was applied.

ClearCase Hyperlink Class

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.

Hyperlink is a subclass of VOBObject Class.

Attributes available for Hyperlink

Name - Kind	Description
FromText - text	The from-text on the from-object of the hyperlink.
Group - text	The group to which this hyperlink belongs.
IDString - text	The string identifying the hyperlink (*type-name@id@vob-selector*).
IsUnidirectional - text	Whether of not the hyperlink object can be navigated only in one direction (from-object -> to-object).
Master - text	The master replica for this hyperlink.
Owner - text	The owner of this hyperlink.
ToText - text	The to-text on the to-object of the hyperlink.

Relationships available for Hyperlink

Name - Kind	Class	Description
FromObject - 1	VOBObject	The from-object of the hyperlink.
ToObject - 1	VOBObject	The to-object of the hyperlink.
Type - 1	HyperlinkType	The hyperlink type of this hyperlink.
VOB – 1	VOB	The VOB containing this hyperlink.

ClearCase HyperlinkType Class

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

HyperlinkType is a subclass of VOBObject Class.

Attributes available for HyperlinkType

Name - Kind	Description
Group – text	The group to which this hyperlink type belongs.
HasSharedMastership - text	Whether this hyperlink type is shared or can be mastered.
LockedBy – text	The name of the user who locked this hyperlink type.
LockedDescription – text	The comment of the user who locked this hyperlink type.
LockedOn - text	The date on which this hyperlink type was locked.
Master – text	The master replica for this hyperlink type.

Owner – text	The owner of this hyperlink type.
Scope – text	The scope of this hyperlink type (for example, local to this VOB).
State - text	The state of the lock on this hyperlink type.

Relationships available for HyperlinkType

Name - Kind	Class	Description
ExcludedUsers - n	Name	The list of users who are exempt from the lock.
Lock - 1	Lock	The lock on this hyperlink type.
VOB - 1	VOB	The VOB containing this hyperlink type.

ClearCase Label Class

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.

Attributes available for Label

None

Relationships available for Attribute

Name - Kind	Class	Description
Type – 1	LabelType	The label type of this label.
VOB - 1	VOB	The VOB containing the labeled version.

ClearCase LabelType Class

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

LabelType is a subclass of VOBObject Class.

Attributes specific to LabelType

Name - Kind	Description
Constraint – text	The constraint for this label type.
Group – text	The group to which this label type belongs.
HasSharedMastership - text	Whether this label type is shared or can be mastered.
LockedBy – text	The name of the user who locked this label type.
LockedDescription – text	The comment of the user who locked this label type.

LockedOn - text	The date on which this label type was locked.
Master – text	The master replica for this label type.
Owner – text	The owner of this label type.
Scope – text	Whether this label type is global for VOBs using this as an admin VOB or local to this VOB.
State - text	The state of the lock on this label type.

Relationships specific to LabelType

Name - Kind	Class	Description
ExcludedUsers - n	Name	The list of users who are exempt from the lock on this label type.
Lock - 1	Lock	The lock on this label type.
VOB - 1	VOB	The VOB containing this label type.

ClearCase Lock Class

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

Attributes available for Lock

Name - Kind	Description
CreatedBy – text	The name of the user who created the lock.
CreatedOn – text	The date the lock was created.
Description - text	The user's comment for the lock.
IsObsolete - text	Whether the locked object is marked as obsolete.
NumberOfExemptUsers - text	The number of users who are exempt from this lock.

Relationships available for Lock

Name - Kind	Class	Description
CreationRecord - 1	HistoryRecord	The creation record for this lock.
ExemptUsers - n	Name	The users who are exempt from this lock.
LockedObject - 1	VOBObject	The object held by this lock.
VOB - 1	VOB	The VOB in which this lock resides.

ClearCase Name Class

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

Attributes specific to Name

Name - Kind	Description
Name - text	Simple name string.

Relationships specific to Name

None

ClearCase Region Class

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.

Attributes specific to Region

Name - Kind	Description
Region - text	The name of the region.

Relationships specific to Region

Name - Kind	Class	Description
Views - n	View	Views contained within the region.
VOBs - n	VOB	VOBs contained within the region.

ClearCase Trigger Class

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.

Attributes available for Trigger

Name - Kind	Description
IsOnAttachedList - text	Whether this trigger is on the attached list of the element.
IsOnInheritanceList - text	Whether this trigger is on the inheritance list of an element, if the element is a directory element.

Relationships available for Trigger

Name - Kind	Class	Description
Type - 1	TriggerType	The trigger type of this element trigger.
VOB - 1	VOB	The VOB containing this element trigger.

ClearCase TriggerType Class

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.

TriggerType is a subclass of VOBObject Class.

Attributes specific to TriggerType

Name - Kind	Description
DebugPrinting - text	Whether or not debug printing happens when the trigger fires.
FiringOrder - text	The trigger type firing order, before or after the operation (pre-op or post-op)
Group – text	The group to which this trigger type belongs.
KindOfTrigger - text	The kind of trigger for this trigger type.
LockedBy – text	The user who locked this trigger type.
LockedDescription – text	The comment of the user who locked this trigger type.
LockedOn - text	The date on which this trigger type was locked.
NumberOfActions – text	The number of actions for this trigger type.
NumberOfExemptUsers – text	The number of users for whom this trigger type does not fire.
NumberOfInclusions – text	The number of inclusions for this element trigger type.
NumberOfOperationKinds – text	The number of operation kinds which fire this trigger type.

NumberOfRestrictions - text	The number of restrictions for this trigger type.
Owner - text	The owner of this trigger type.
State - text	The state of the lock on this trigger type.

Relationships specific to TriggerType

Name - Kind	Class	Description
Actions - n	Name	An array of action/value pairs for this trigger type (that is, a type followed by one or two values).
ExcludedUsers - n	Name	A list of users who are exempt from this trigger type.
ExemptUsers - n	Name	The users exempted from the firing of triggers for this trigger type.
Inclusions - n	Name	The inclusion list for this trigger type.
Lock - 1	Lock	The lock on this trigger type.
OperationKinds - n	Name	An array of kinds of operations which fire this trigger type.
Restrictions - n	Name	The restriction list for this element trigger type.
VOB - 1	VOB	The VOB containing this trigger type.

ClearCase Value Class

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

Attributes specific to Value

Name - Kind	Description
Value - text	Simple value string.

Relationships specific to Value

None

ClearCase Version Class

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.

Subclasses of Version Class: CheckedOutFile.

Version is a subclass of File Class.

Attributes available for Version

Name - Kind	Description
Identifier - text	The version's identifier string.
IsCheckedOut - text	Whether or not this object represents a checked-out file.
IsDifferent - text	Whether or not this version is different from its predecessor.
IsHijacked - text	Whether or not this version is hijacked.
IsLatest - text	Whether or not this version is the latest on its branch.
VersionNumber - text	This version's version number.

Relationships available for Version

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

ClearCase View Class

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

Attributes specific to View

Name - Kind	Description
ConfigSpec - text	A displayable form of the config spec for this view.
CreatesShareableDerivedObjects - text	Whether or not this view builds non-shareable derived objects.
Host - text	The host on which the storage area for this view resides.
IsSnapShot - text	Whether or not this view is a snapshot view.
IsStarted - text	Whether or not the view is started on the local machine.
TagName - text	The view-tag name.

Relationships specific to View

None

ClearCase VOB Class

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.

VOB is a subclass of VOBObject Class.

Attributes available for VOB

Name - Kind	Description
Group - text	The group to which this VOB belongs.
HasMSDOSTextMode - text	Whether or not this VOB has MSDOS text mode enabled.
Host - text	The host on which the storage area for this VOB resides.

IsMounted - text	Whether or not the VOB is mounted.
IsReplicated - text	Whether or not this VOB is replicated.
LockedBy – text	The name of the user who locked the VOB.
LockedDescription – text	The description of the lock for the VOB.
LockedOn - text	The date the VOB was locked.
Owner - text	The owner of the VOB.
NumberOfAdditionalGroups - text	The number of additional groups to which this VOB belongs.
NumberOfReplicas - text	The number of replica names for the VOB family of this VOB, if this VOB is replicated.
State - text	The state of the lock on the VOB.
TagName - text	The VOB-tag name.
ThisReplica – text	The replica name for this VOB, if the VOB is replicated.

Relationships available for VOB

Name - Kind	Class	Description
AdditionalGroups - n	Name	Additional groups to which this VOB belongs.
AllAttributeTypes - n	AttributeType	All existing attribute types in the VOB, including obsolete types.
AllBranchTypes – n	BranchType	All existing branch types in the VOB, including obsolete types.
AllHyperlinkTypes - n	HyperlinkType	All existing hyperlink types in the VOB, including obsolete types.
AllLabelTypes – n	LabelType	All existing label types in the VOB, including obsolete types.
AllLocks – n	Lock	An enumeration of all the locks in this VOB, including obsolete locks.
AllTriggerTypes – n	TriggerType	All existing trigger types in the VOB, including obsolete types.
AttributeTypes – n	AttributeType	All existing attribute types in the VOB.
BranchTypes – n	BranchType	All existing branch types in the VOB.
ExcludedUsers - n	Name	The list of users exempted from the lock on the VOB.
HyperlinkTypes - n	HyperlinkType	All existing hyperlink types in the VOB.
LabelTypes – n	LabelType	All existing label types in the VOB.

Lock – 1	Lock	The lock on this VOB, if there is one.
Locks – n	Lock	An enumeration of all the locks in this VOB.
Replicas – n	Name	The array of replica names for the VOB family of this VOB, if this VOB is replicated.
TriggerTypes - n	TriggerType	All existing trigger types in the VOB.

ClearCase VOBObject Class

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.

Subclasses of VOBObject Class:

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

Attributes specific to VOBObject

Name - Kind	Description
CreatedBy – text	The user who created the object.
Created On - text	The date the object was created.
Description - text	The comment associated with the VOB object.
Name - text	The name of the versioned object.
OID - text	The object identifier for the VOB object.
VOBFamilyUUID - text	The VOB family UUID for the VOB of this VOB object.

Relationships specific to VOBObject

Name - Kind	Class	Description
Attributes - n	Attribute	The collection of attributes associated with this VOB object
CreationRecord – 1	HistoryRecord	The creation record for the VOB object
HistoryRecords - n	HistoryRecord	The collection of history records for this object
Hyperlinks - n	Hyperlink	The collection of hyperlinks associated with this VOB object

ClearQuest Domain

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.

ClearQuest Domain Classes

The ClearQuest domain provides information regarding the following classes of objects:

- ClearQuest Attachments Class
- ClearQuest CQDatabase Class
- ClearQuest Groups Class
- ClearQuest History Class
- ClearQuest Users Class

Remember: There will likely be additional classes (or differenct classes) once you OPEN a specific database.

ClearQuest Attachments Class

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

Attributes available for Attachments

Name - Kind	Description
Dbid - text	The internal database ID
Description - text	The description of the attachment
Entity_dbid - text	
Entity_fileddef_id - text	
Filesize - text	The size of the attachment
Is_active - text	True if the record is active
Lock_version -text	The version of the locking mechanism
Locked_by - text	The user who has locked this record
Version - text	

Relationships available for Attachments

Name - Kind	Class	Description
History - n	History	The history records for this attachment

ClearQuest CQDatabase Class

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

Attributes available for CQDatabase

Name – Kind	Description
Description – text	The description of the database
Name – text	The logical name of the database

Relationships available for CQDatabase

Name – Kind	Class	Description
Attachments - n	Attachments	All attachments stored in the database
Groups – n	Groups	All groups stored in the database
History – n	History	The history records for this database
Users – n	Users	All users stored in the database

When you create an OPEN command for a specific ClearQuest database, SoDA automatically creates a set of N-ary relationships under the database class, one for each accessible query. One relationship is included for each public query as well as one for each private query owned by you. The class of each relationship is the same as the class of object returned by the query.

For example, if you create a personal query called 'MyDefects' in ClearQuest, which retrieves all defects assigned to you, SoDA creates a relationship called 'PersonalQueriesMyDefects' under the Database class. The class of object returned by that relationship is 'defect'. This relationship enables you to use a REPEAT to iterate over the results of the query within a SoDA template.

Although it is possible to retrieve the same information as provided by this query by adding a REPEAT over all defects, it is highly recommended that you make use of ClearQuest queries whenever possible. Performance of the ClearQuest domain is significantly improved when a query is used.

ClearQuest Groups Class

A group is a list of users with similar privileges.

Attributes available for Groups

Name - Kind	Description
Dbid - text	The internal database ID
Is_active - text	True if the group is active
Lock_version - text	Version of the locking mechanism
Locked_by - text	The name of the user who is locking the database
Master_dbid - text	
Name - text	The name of the group
Version - text	

Relationships available for Groups

Name - Kind	Class	Description
History - n	History	The history records of this group

ClearQuest History Class

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

Attributes available for History

Name - Kind	Description
Action_name - text	The action that was entered
Action_timestamp - text	The time the action was entered
Comments - text	Any comments associated with the event
Dbid - text	The internal database ID
Entity_dbid - text	
Entitydef_id - text	
Entitydef_name – text	
Expired_timestamp - text	The time the history expires
Is_active - text	True if the record is active
Lock_version - text	The version of the locking mechanism

Locked_by - text	The user who locked the record
New_state - text	The state of the record following the action
Old_state - text	The state of the record prior to the action
User_name - text	The user who triggered the action
Version - text	

Relationships available for History

None

ClearQuest Users Class

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

Attributes available for Users

Name - Kind	Description
Dbid - text	The internal database ID
Email - text	The email address of the user
Encrypted_password - text	The password of the user
Fullname - text	
Is_active - text	True if the user is active
Is_appbuilder - text	True if the user has AppBuilder privileges
Is_superuser - text	True if the user is a superuser
Is_user_maint - text	True if the user has maintenance privileges
Lock_version - text	The version of the locking mechanism
Locked_by - text	The user that has locked the user database
Login_name - text	
Master_dbid - text	
Misc_info - text	Miscellaneous information
Phone - text	The phone number of the user
Version - text	

Relationships available for Users

Name - Kind	Class	Description
Groups - n	Groups	The groups which this user is a member
History - n	History	The history records for this user record

File System Domain

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.

The File System domain provides information regarding the following classes of objects:

- File System DirectoryObject Class
- File System Directory Class
- File System File Class
- File System FileRecord Class

File System Domain Classes

File System DirectoryObject Class

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

Subclasses of DirectoryObject:

- Directory
- File

Attributes available for DirectoryObject

Name (Kind)	Description
Extension (text)	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 (text)	The complete pathname of an object. For example, c:\bill\file.txt
NameMinusExtension (text)	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.
SimpleName (text)	The context-independent portion of an object's name. For example, the SimpleName of c:\bill\file.txt is file.txt.

Relationships available for DirectoryObject

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

File System Directory Class

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

Directory is a subclass of DirectoryObject.

Attributes specific to Directory

None

Relationships specific to Directory

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

File System File Class

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.

File is a subclass of DirectoryObject.

Attributes specific to File

Name (Kind)	Description
Graphic (graphic)	The diagram contained in the file. The file can be text (such as encapsulated PostScript) or binary (such as bitmap files). Your Microsoft Word documentation describes the file formats available.
Text (text)	The complete contents of an ASCII text file. Undefined for other file types.

Relationships specific to File

Name (Kind)	Class	Description
Records (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.

File System FileRecord Class

ASCII text files can be further 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.

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).

Attributes available for FileRecord

Name (Kind)	Description
Field01 .. Field30 (text)	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.
Filename (text)	The full pathname of the file that contains this record.
Text (text)	Full text of the entire record.
UniqueKey (text)	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 available for FileRecord

None

RequisitePro Domain

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 Generation 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" on page 60 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.

RequisitePro Domain Classes

The RequisitePro domain provides information regarding the following classes of objects:

- RequisitePro Project Class
- RequisitePro Document Class
- RequisitePro Requirement Class
- RequisitePro <Project-Specific Type>Requirement Class
- RequisitePro Relationship Class
- RequisitePro AttributeValue Class
- RequisitePro DocumentType Class
- RequisitePro RequirementType Class
- RequisitePro Discussion Class
- RequisitePro Response Class
- RequisitePro User Class
- RequisitePro Group Class
- RequisitePro Revision Class

RequisitePro AttributeValue Class

Attributes are descriptive information attached to a requirement that provide important details about that requirement, such as priority, cost, or difficulty. Be sure to see the section titled Accessing Project-specific Attributes.

When using RequisitePro to view attributes and their related values, often the value has been left blank with no value set. The RequisitePro view shows nothing or a blank in the attribute value field for that requirement. However, when accessing the blank attribute value from SoDA, the SoDA report returns a value of "(null)". It should be noted that the true value of the attribute in RequisitePro is blank and not "(null)". SoDA currently designates a null value with the string "(null)".

Attributes available for AttributeValue

Name (Kind)	Description
DataType (text)	
Label (text)	The name of the attribute
Value (text)	The value of the attribute

Relationships available for AttributeValue

None

RequisitePro Discussion Class

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.

Attributes available for Discussion

Name (Kind)	Description
DateTime (text)	When the discussion was created
HasResponses (text)	True if someone has responded to this discussion
Message (text)	The text of the discussion
Priority (text)	The priority of the discussion: High, Medium, or Low
Restricted (text)	True if the discussion is restricted to the listed participants
Status (text)	The status of the discussion: Open or Closed
Subject (text)	The subject of the discussion

Relationships available for Discussion

Name (Kind)	Class	Description
Author (1)	User	The user who created this reponse
Participants (n)	User	
ParticipantGroups (n)	Group	
Responses (n)	Response	The responses to this response

RequisitePro Document Class

Although a requirements document looks like a normal Word document, RequisitePro has added the capability to create requirements. The descriptions, attribute values, and traceability relationships for the requirements become part of the project database.

Attributes available for Document

Name (Kind)	Description
Description (text)	A description of the document
Extension (text)	
Filename (text)	The simple filename for the document
FullPath (text)	The full pathname of the document
Name (text)	A descriptive name for the document
Path (text)	

Relationships available for Document

Name (Kind)	Class	Description
DocType (1)	DocumentType	
LatestRevision - 1	Revision	Accesses the latest revision of the document.
Requirements (n)	Requirement	The requirements contained within this document
Revisions (n)	Revision	The revisions (history) of the document

RequisitePro DocumentType Class

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.

Attributes available for DocumentType

Name (Kind)	Description
Description (text)	The purpose and content of the document type
Extension (text)	The file extension applied to all documents associated with this document type
Name (text)	The name of the document type

TemplateDescription (text)	A description of the template, or outline, for this document type
TemplateFilename (text)	
TemplateName (text)	The template, or outline, used when documents of this type are created

Relationships available for DocumentType

Name (Kind)	Class	Description
DefaultRequirementType (1)	RequirementType	The default type of requirement stored in this type of document

RequisitePro Project Class

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.

When creating an OPEN command to a ReqPro project, you must supply the name of the project file (the file with the extension .RQS).

Attributes available for Project

Name (Kind)	Description
Description (text)	Optional information describing the purpose and content of the project
FileName (text)	
Name (text)	The name of the project
Path (text)	
Prefix (text)	The prefix that is prepended to requirement tags when using external projects

Relationships available for Project

Name (Kind)	Class	Description
<View Name Requirement Type> (n)	Requirement	If a template contains an OPEN command to a specific project, SoDA will create a relationship for each attribute matrix view. It will return the requirements based on the query stored in the view. The column information is not used.
Author (1)	User	The creator of the project
CurrentDocument (1)	Document	The document that is currently open (if any) in RequisitePro. No document is returned if the open document is not in the open project.

DocTypes (n)	DocumentType	The document types defined in this project
Documents (n)	Document	The set of documents associated with this project
ExternalProjects (n)	Project	The external projects that have been attached to this project
Groups (n)	Group	The security groups defined in this project
ReqTypes (n)	RequirementType	The requirement types defined in this project.
Requirements (n)	Requirement	All requirements stored in the project database
Revisions (n)	Revision	The historical data of the project
SecurityGroups (n)	Group	The security groups defined in this project
SelectedRequirements (n)	Requirement	The requirements that are currently selected in the current View in RequisitePro. Only requirements in the open project are included.
Users (n)	User	All users who are registered in this project

RequisitePro Relationship Class

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

Attributes available for Relationship

Name (Kind)	Description
Direction (text)	The direction of the relationship: TraceTo, TraceFrom, Parent, or Child
Suspect (text)	True if the relationship is suspect; otherwise, False
Type (text)	The type of the relationship, either Hierarchical or Traceability

Relationships available for Relationship

Name (Kind)	Class	Description
RelatedReq (1)	Requirement	The associated requirement

RequisitePro Requirement Class

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.

Attributes available for Requirement

Name (Kind)	Description
Bookmark (text)	The name of the Word bookmark associated with this requirement
DocPosition (text)	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 (text)	The full tag of the requirement, such as PR1
GUID - text	
HasChildren (true)	True if this requirement has child requirements; otherwise False.
HasParent (true)	True if this requirement has a parent requirement. This would be the same as Level > 0.
Level (text)	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.
TagNumber (text)	The number of the tag, such as 1
TagPrefix (text)	The prefix of the tag, such PR
Text (text)	The text of the requirement.

Relationships available for Requirement

Name (Kind)	Class	Description
Attributes (n)	Attribute	The attributes of the requirement
Children (n)	Relationship	The children of this requirement. 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.
Discussions (n)	Discussion	The discussions attached to this requirement
Document (1)	Document	The document where the requirement is stored
LatestRevision (1)	Revision	The current status of this requirement
Parent (1)	Relationship	The parent of this requirement. If a template contains an OPEN command to a specific project, you will also see the ParentRequirement option, which lets you go directly to the requirement.

ParentProject (1)	Project	The project that this requirement is contained in. This relationship is especially useful when doing cross-project traceability.
ReqType (1)	RequirementType	The type of this requirement
Revisions (n)	Revision	The revisions (history) of this requirement
TracesFrom (n)	Relationship	The relationships traced out of this requirement
TracesTo (n)	Relationship	The relationships traced in to this requirement

<Project-Specific Type>Requirement Class

When you create an OPEN command to a specific project, SoDA automatically creates a series of new classes that are subclasses of the Requirement Class . The name of the class will be 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.

Attributes specific to <Project-Specific Type>Requirement

Name (Kind)	Description
<Attribute Name> (text)	There will be one attribute for each attribute name defined by the requirement type, such as Status, Priority, Build_Number, etc.

Relationships specific to <Project-Specific Type>Requirement

Name (Kind)	Class	Description
ChildRequirements (n)	(the same project-specific requirement class)	The child requirements for this requirement.
ParentRequirement (1)	(the same project-specific requirement class)	The parent requirement of this requirement

RequisitePro RequirementType Class

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. Be sure to see the section titled Accessing Project-specific Attributes.

Attributes available for RequirementType

Name (Kind)	Description
Description (text)	A general description of the requirement type
Name (text)	The name of the requirement type
ReqColor (text)	
ReqPrefix (text)	The prefix of the type, such as SR
ReqStyle (text)	The Word style applied to requirements of this type; one of Normal, SmallCapitals, or DoubleUnderline

Relationships available for RequirementType

None

RequisitePro Reply Class

Replies are responses to a discussion.

Attributes available for Reply

Name (Kind)	Description
DateTime (text)	When the reply was created
HasReplies (text)	True if someone has replied to this reply
Message (text)	The text of the reply
Subject (text)	The subject of the reply

Relationships available for Reply

Name (Kind)	Class	Description
Author (1)	User	The user who created this reply
Replies (n)	Reply	The responses to this reply

RequisitePro Revision Class

The Revision class lets you document revision information about a requirement, document or project.

Attributes available for Revision

Name (Kind)	Description
DateTime (text)	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 (text)	The change description field
Label (text)	Text associated with a revision number
Number (text)	The revision number, incremented automatically for each revision

Relationships available for Revision

Name (Kind)	Class	Description
Author (1)	User	The user that made the change

RequisitePro User Class

Users are people who have access to project information. RequisitePro tracks which users make changes to project and requirement information.

Attributes available for User

Name (Kind)	Description
Name (text)	The name for this user

Relationships available for User

Name (Kind)	Class	Description
Group (1)	UserGroup	The group this user belongs to

RequisitePro Group Class

A user group is a set of users. Groups are used for project security.

Attributes available for Group

Name (Kind)	Description
DefAttrRights (text)	The privileges this group has to create or modify attributes
DefDocTypeRights (text)	The privileges this group has to create or modify document types

DefListItemRights (text)	The privileges this group has to create or modify list items
Name (text)	The name of the user group
DefProjRights (text)	The privileges this group has to create or modify privileges
DefReqTypeRights (text)	The privileges this group has to create or modify requirements

Relationships available for Group

None

Rose Domain

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.

How to Display 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 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**.

Rose Domain Classes

Rose Action Class

Attributes available for Action

Name - Kind	Description
Documentation - text	The text from the Documentation field of this transition.
Arguments - text	The arguments that accompany the trigger event.
Name - text	The name of the event.
Target - text	The name of the event object.
Stereotype - text	The stereotype of the transition.
UniqueID - text	

Relationships available for Action

None

Rose Association Class

An 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.

Attributes available for Association

Name - Kind	Description
Constraints - text	The text from the Constraints field in the association specification.
Documentation - text	Text from the Documentation field in the association specification.
HasLinkElement - text	True if the association has an attached association class, otherwise False.
IsDerived - text	True if the association is derived; otherwise False.
Name - text	The name of the association.
Stereotype - text	The stereotype of the association.
UniqueID -text	The internal unique identifier of the association.

Relationships available for Association

Name - Kind	Class	Description
LinkElement - 1	Class	The linked class attached to the association.
Properties - n	Property	The code-generation properties associated with the association.
RoleA - 1	Role	The first role defined in the association.
RoleB - 1	Role	The second role defined in the association.

Rose Activity Class

Attributes available for Activity

Name - Kind	Description
Documentation - text	The text from the documentation field of the activity.
Name - text	The name of the activity.
Stereotype - text	The stereotype of the activity.
UniqueID - text	The internal unique identifier of the activity.

Relationships available for Activity

Name - Kind	Class	Description
AllSubActivities - n	Activity	
AllSubStates - n	State	
DoActions - n	Action	The Do actions for this activity.
EntryActions - n	Action	The Entry actions for this activity.
ExitActions - n	Action	The Exit actions for this activity.
ExternalDocs - n	String	The external documents attached to this activity.
ParentStateActivityModel - 1	StateActivityModel	The parent state machine associated with this activity.
Properties - n	Property	The properties attached to this activity.
StateActivityDiagrams - n	StateActivityDiagram	The state or activity diagrams internal to this activity.
StateActivityModels - n	StateActivityModel	The state or activity models internal to this activity.
SubActivities - n	Activity	The activities that are part of this activity.
SubStates - n	State	The states that are part of this activity.
Transitions - n	StateTransition	The transitions that exit from this activity.

Rose Attribute Class

Attributes are data members of a class whose type is not another class.

Attributes available for 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.
Documentation - text	Text from the Documentation field in the attribute specification.
ExportControl - text	The export control of the attribute. Returns Public, Protected, Private, or implementation.
InitialValue - text	The initial value of the attribute.
IsDerived - text	True if the Derived check box is selected in the attribute specification, otherwise False.

IsStatic - text	True if the Static check box is selected in the attribute specification, otherwise False.
Name - text	The name of the attribute.
Stereotype - text	The stereotype of the attribute.
Type - text	The type of the attribute.
UniqueID - text	The internal unique identifier of the attribute.

Relationships available for Attribute

Name - Kind	Class	Description
ParentClass - 1	Class	The class in which this attribute is defined.
Properties - n	Property	The code-generation properties of the attribute.

Rose Class Class

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.

Attributes available for Class

Name - Kind	Description
Cardinality - text	The string in the Cardinality field of the class specification.
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.
Documentation - text	Text from the Documentation field in the class specification.
ExportControl - text	Returns Public or Implementation, depending on the value of the Export Control radio control in the class specification.
FundamentalType - text	
HasStateActivityDiagram - text	True if the class has an associated state diagram, otherwise False.
IsAbstract - text	True if the Abstract check box is selected in the class specification, otherwise False.

IsNested - text	True if the class is nested.
Kind - text	The Rose source domain class name of the declaration. For example, Kind of a class utility returns ClassUtility.
Name - text	The name of the class.
Persistence - text	Returns Persistent or Transient, depending on the value of the Persistence radio control in the More dialog of the class specification.
QualifiedName - text	The name of the class along with its parent categories, such as AncestorCat:ParentCat:ClassName
Space - text	The string in the Space field of the More dialog of the class specification.
Stereotype - text	The stereotype of the class.
UniqueID - text	The internal ID of the class.

Relationships available for Class

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.
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	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.
AllSuperClasses - n	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.
AppearsIn - n	ClassDiagram	The class diagrams where this class appears.
Associations - n	Association	The associations where this class plays a role.
Components - n	Component	The components associated with this class, as specified in the class specification.
ExternalDocs - n	String	The external documents attached to this class.

FirstStateActivityDiagram - 1	StateActivityDiagram	The (first) state/activity diagram associated with this class.
FirstStateActivityModel - 1	StateActivityModel	The (first) state/activity model associated with this class.
Instances - n	Object	The object instances of this class.
Instances AppearIn - n	Interaction Diagram	The interaction diagrams that include instances of this class.
Module - 1	Module	
MyAttributes - n	Attribute	The attributes that are defined by this class. Does not include inherited attributes.
MyOperations - n	Operation	The operations that are defined by this class. Does not include inherited operations.
MyRelationships - n	Relationship	The relationships that are defined by this class. Does not include inherited relationships.
MySubClasses - n	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.
MySuperClasses - n	Class	The classes that this class directly inherits from. Only includes immediate superclasses. For example, if A inherits from B and B inherits from C, then MySuperClasses of A would include B but not C.
NestedClasses - n	Class	The classes that are nested within this class.
ParentClass - 1	Class	The parent class of this class, if it is nested.
ParentModel - 1	Model	
ParentPackage - 1	Package	The enclosing package.
Properties - n	Property	The code-generation properties of this class.
StateActivityDiagrams - n	StateActivityDiagram	All state/activity diagrams associated with this class.
StateActivityModels - n	StateActivityModel	The state/activity models associated with this class.

Rose ClassDiagram Class

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.

Attributes available for ClassDiagram

Name - Kind	Description
Documentation - text	
Image - graphic	The graphical representation of the diagram.
Name - text	The name of the diagram.
UniqueID - text	

Relationships available for ClassDiagram

Name - Kind	Class	Description
Classes - n	Class	All of the classes that appear on the diagram.
ExternalDocs - n	String	The external documents attached to this diagram.
Notes - n	Note	
Packages - n	Package	
ParentModel - 1	Model	
ParentPackage - 1	Package	
Relationships - n	Relationship	All of the relationships that appear on the diagram.
UseCases - n	UseCase	All of the use cases that appear on the diagram.

Rose ClassUtility Class

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

Rose Component Class

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.

Attributes available for Component

Name - Kind	Description
AssignedLanguage - text	
Declarations - text	
Documentation - text	Text from the Documentation field of the component specification.
Name - text	The name of the component.
Part - text	
Path - text	
Stereotype - text	
Type - text	
UniqueID - text	

Relationships available for Component

Name - Kind	Class	Description
AssignedClasses - n	Class	
ParentModel - 1	Model	The model that contains this component.
ParentSubsystem - 1	Subsystem	Enclosing subsystem of this component.
VisibilityRelationships - n	ModuleVisibility Relationship	

Rose ComponentDiagram Class

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.

Attributes available for ComponentDiagram

Name - Kind	Description
Image - graphic	The graphical representation of the diagram.
Name - text	The name of the diagram.
Documentation - text	
UniqueID - text	

Relationships available for ComponentDiagram

Name - Kind	Class	Description
Modules - n	Component (Module)	The components contained in the diagram.
ExternalDocs - n	String	The external documents attached to this diagram.
Notes - n	Note	
ParentModel - 1	Model	

Rose Decision Class

Attributes available for Decision

Name - Kind	Description
Documentation - text	The text from the documentation field for this decision.
Name - text	The name of the decision.
Stereotype - text	The stereotype of the decision.
UniqueID - text	The internal unique identifier of the decision.

Relationships available for Decision

Name - Kind	Class
Transitions - n	StateTransition

Rose DeploymentDiagram Class

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.

Attributes available for DeploymentDiagram

Name - Kind	Description
Image - graphic	The image of the deployment diagram.
Name - text	
Documentation - text	
UniqueID - text	

Relationships available for DeploymentDiagram

Name - Kind	Class	Description
Nodes - n	Node	The processors and devices contained in the diagram.
Notes - n	Note	
ParentModel - 1	Model	

Rose Device Class

A device is a hardware component with no computing power.

Device is a subclass of Node.

Attributes specific to Device

None

Relationships specific to Device

None

Rose HasRelationship Class

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.

HasRelationship is a subclass of Relationship.

Attributes specific to HasRelationship

Name – Kind	Description
Containment – text	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.
Static – text	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.

Relationships specific to HasRelationship

None

Rose InheritsRelationship Class

InheritsRelationship is a subclass of Relationship.

Attributes specific to InheritsRelationship

Name – Kind	Description
FriendshipRequired - 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.
IsVirtual - text	

Relationships specific to InheritsRelationship

Name – Kind	Class	Description
FromUseCase – 1	UseCase	The supplier use case of the inherits relationship, if it is a use-case.
ToUseCase – 1	UseCase	The client use case of the inherits relationship, if it is a use case.

Rose InstantiatedClass Class

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

Name - Kind	Class	Description
Instantiates - 1	ParameterizedClass	The parameterized class that this instantiated class instantiates.

Rose InstantiatedClassUtility Class

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

Name - Kind	Class	Description
Instantiates - 1	ParameterizedClassUtility	The parameterized class utility that this instantiated class utility instantiates.

Rose InteractionDiagram Class

An interaction diagram is an important sequence of interactions between Objects. Rose enables you to capture, view, and manipulate both collaboration diagrams and sequence diagrams. Collaboration and sequence diagrams express the same information, but with different notations.

Attributes available for InteractionDiagram

Name - Kind	Description
DiagramType - text	The type of the diagram, either CollaborationDiagram or SequenceDiagram
Documentation - text	The documentation text associated with the interaction diagram.
Image - graphic	The graphical representation of the interaction diagram.

Name - text	The name of the interaction diagram.
ParentKind - text	Type of the parent, either UseCase or ClassCategory
UniqueID - text	

Relationships available for InteractionDiagram

Name - Kind	Class	Description
ExternalDocs - n	String	The external documents attached to this diagram.
Messages - n	Message	The messages that appear in the diagram.
Notes – n	Note	The notes that appear in the diagram.
Objects - n	Object	The objects that appear in the diagram.
ParentModel - 1	Model	The model that this diagram is contained in.
ParentPackage - 1	Package	The package that this diagram is contained in, if applicable.
ParentUseCase - 1	UseCase	The use case that this diagram is contained in, if applicable.

Rose Link Class

A link is an instance of an association.

Attributes available for Link

Name - Kind	Description
ClientIsShared - text	True if the Shared box is checked on the client side; otherwise False.
ClientVisibility - text	One of Unspecified, Field, Parameters, Local, or Global.
IsLinkToSelf - text	True if the link goes from an object to itself.
Name - text	The name of the link.
SupplierIsShared - text	True if the Shared box is checked on the supplier side; otherwise False.
SupplierVisibility - text	One of Unspecified, Field, Parameters, Local, or Global.
UniqueID - text	

Relationships available for Link

Name - Kind	Class	Description
Client - 1	Object	The client object instance (role) of the link.
Messages - n	Message	The messages associated with the link.
Supplier - 1	Object	The supplier object instance (role) of the link.

Rose Message Class

Any message associated with an object.

Attributes available for Message

Name - Kind	Description
Documentation - text	Text from the Documentation field in the message specification.
Frequency - text	The frequency of the message.
HierarchicalSeqNumber - text	The hierarchical sequence number of the message.
Name - text	The name of the message.
NameWithoutParentheses - text	The name of a message without the parenthesized parameters from Rose that are added when a message is associated with a class operation.
SeqNumber - text	The sequence number of the message.
SimpleName - text	The name of the message, excluding any parameters
Stereotype - text	The stereotype of the message.
Synchronization - text	The concurrency semantics for the operation named in the Operations Field; one of Simple, Synchronous, Balking, Timeout or Asynchronous.
UniqueID - text	

Relationships available for Message

Name - Kind	Class	Description
Properties - n	Property	Displays the properties of the message.
Receiver - 1	Object	The object that receives the message.
Sender - 1	Object	The object that sends the message.

Rose MetaClass Class

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

Rose Model Class

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 will start with connections to a Model.

Model is a subclass of File System File Class.

Attributes specific to Model

None

Relationships specific to Model

Name - Kind	Class	Description
AllAssociations - n	Association	All associations in the model.
AllClasses - n	Class	All classes in the model, including actors.
AllComponents - n	Module	All components in the model (including subsystems).
AllInteractionDiagrams - n	InteractionDiagram	All interaction diagrams in the model.
AllPackages - n	Package	All packages in the model, including use-case packages (but not including subsystems in the Component View).
AllRelationships - n	Relationship	All relationships in the model.
AllSubsystems - n	Subsystem	All subsystem components in the model.
AllUseCases - n	UseCase	All use cases in the model.

ComponentView - 1	Subsystem	The highest-level subsystem in the model; its name is Component View. All other subsystems are nested beneath it.
DeploymentDiagram - 1	DeploymentDiagram	The deployment diagram (process diagram) for the model.
LogicalView - 1	Package	The highest-level package in the model; its name is Logical View. All other packages are nested beneath it.
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 UseCase View. All other use-case packages are nested beneath it.

Rose ModuleVisibilityRelationship Class

Attributes available for ModuleVisibilityRelationship

Name - Kind
Documentation - text
Name - text
UniqueID - text

Relationships available for ModuleVisibilityRelationship

Name - Kind	Class
ContextModule - 1	Module
SupplierModule - 1	Module

Rose Node Class

Node is an abstract class for processors and devices.

Subclasses of Node:

Processor, Device

Attributes available for Node

Name - Kind	Description
Characteristics - text	The characteristics of the processor or device.
Documentation - text	The text in the Documentation field of the processor or device specification
Stereotype - text	
Name - text	The name of the component.
UniqueID - text	

Relationships available for Node

None

Rose Object Class

Any object. Its structure and behavior will be defined by its class.

Attributes available for Object

Name - Kind	Description
ClassName - text	
Documentation - text	Text from the Documentation field in the object specification.
IsClass - text	
MultipleInstances - text	True if the Multiple Instances box is checked; otherwise False
Name - text	The name of the object.
Persistence - text	Persistent, Static, or Transient depending on the value of the Persistence radio control in the object specification.
Stereotype - text	The stereotype of the object.
UniqueID - text	

Relationships available for Object

Name - Kind	Class	Description
Class - 1	Class	The class of the object.
Links - n	Link	The links associated with the object.

Rose Operation Class

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.

Attributes available for 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.
Documentation - text	Text from the Documentation field in 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.
ExportControl - text	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.

JavaImage - text	A Java code segment that represents the declaration of the operation.
Name - text	The name of the operation. (Does not include the function parentheses.)
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.
Stereotype - text	The stereotype of the operation.
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.
UniqueID - text	The internal ID for this operation.

Relationships available for Operation

Name - Kind	Class	Description
Arguments - n	Parameter	The formal parameters of the operation. These appear in the Arguments list box in the operation specification.
ExternalDocs - n	String	The external documents attached to this operation.
Properties - n	Property	The code-generation properties of the operation.
ParentClass - 1	Class	The class to which this operation belongs.

Rose Package Class

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.

Attributes available for Package

Name - Kind	Description
Documentation - text	Text from the Documentation field in the package specification.
Global - text	True if the package is global, otherwise False.
HasAssignedSubsystem - text	True if the package has a subsystem associated with it, otherwise False.
HasStateActivityDiagram - text	True if the package has a state/activity diagram.
IsUseCasePackage - text	True if the package is a descendent of the UseCase View package, otherwise False.
Name - text	The name of the package.
Stereotype - text	The stereotype of the package.
UniqueID - text	The internal unique identifier for the package.

Relationships available for Package

Name - Kind	Class	Description
AllAssociations - n	Association	All associations that are defined in this package, or in any nested packages.
AllClassDiagrams - n	ClassDiagram	

AllClasses - n	Class	All classes that are defined in this package, or in any nested packages.
AllReferencers - n	Package	All packages that import this package. Does not include indirect referencers.
AllUseCaseDiagrams - n	UseCaseDiagram	
AllUseCases - n	UseCase	All use cases that are defined in this package, or in any nested packages.
AssignedSubsystem - 1	Subsystem	The subsystem associated with this package, as specified in the package specification.
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.
DependedOnBy - n	PackageDependency	
DependsOn - n	PackageDependency	
ExternalDocs - n	String	The external documents associated with this package.
FirstStateActivityDiagram - 1	StateActivityDiagram	
FirstStateActivityModel - 1	StateActivityModel	
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.
InteractionDiagrams - n	InteractionDiagram	All interaction scenario diagrams that are immediate members of this package.
MainDiagram - 1	ClassDiagram	The diagram specifically called “Main”.
MyAssociations - n	Association	All associations that are immediate members of this package.
MyUseCases - n	UseCase	All use cases that are immediate members of this package.
NestedSubPackages - n	Package	All packages that are descendents of this package.
ParentModel - 1	Model	The model that contains this package. This relationship is used primarily when a template OPENs a package directly.

ParentPackage - 1	Package	The enclosing package. This relationship will result in an error if applied to the TopLevelCategory.
Properties - n	Property	The code-generation properties associated with this package.
Relationships - n	Relationship	The relationships defined within this package.
StateActivityDiagrams - n	StateActivityDiagram	The state/activity diagram associated with this package.
StateActivityModels - n	StateActivityModel	The state/activity model for this package.
SubPackages - n	Package	All packages that are immediate children of this package.
UseCaseDiagrams - n	UseCaseDiagram	The use-case diagrams contained within this package.

Rose PackageDependency Class

Attributes specific to PackageDependency

Name - Kind
SupplierName - text
Documentation - text
Name -text
Stereotype - text
UniqueID - text

Relationships specific to PackageDependency

Name - Kind	Class
ToPackage - 1	Package
FromPackage - 1	Package

Rose Parameter Class

Formal parameter of an operation, instantiated class, or instantiated class utility.

Attributes available for Parameter

Name - Kind	Description
Documentation - text	Text from the Documentation area when a parameter specification is visible.
InitValue - text	The initial value of the parameter
Const - text	True if the parameter is constant; otherwise False
Name - text	The name of the parameter.
Stereotype - text	The stereotype of the parameter.
Type - text	The type of the parameter.
UniqueID - text	

Relationships available for Parameter

None

Rose ParameterizedClass Class

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
InstantiatedClasses - n	InstantiatedClass	All instantiated classes of this parameterized class.
Parameters - 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.

Rose ParameterizedClassUtility Class

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
InstantiatedClasses - n	InstantiatedClassUtility	All instantiated class utilities of this parameterized class utility.
Parameters - 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.

Rose Process Class

A process transforms data values. Lowest-level processes are pure functions without side effects.

Attributes available for Process

Name - Kind	Description
Documentation - text	The text in the Documentation field of the process specification.
Name - text	The name of the process.
Priority - text	The priority of the process.
Stereotype - text	
UniqueID - text	

Relationships available for Process

None

Rose Processor Class

A processor is a hardware component capable of executing programs.

Processor is a subclass of Node.

Attributes specific to Processor

Name - Kind	Description
Scheduling - text	The text in the Scheduling field of the processor specification.

Relationships specific to Processor

Name - Kind	Class	Description
Processes - n	Process	The processes defined by this processor.

Rose Property Class

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.

Attributes available for Property

Name - Kind	Description
Name - text	The name of the property.
ToolName - text	The name of the tool, or tab, for the property, such as “cg” or “DDL”.
Value - text	The string equivalent of the value associated with the property.

Relationships available for Property

None

Rose RealizeRelationship Class

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 available for RealizeRelationship

None

Relationships available for RealizeRelationship

None

Rose Relationship Class

A semantic connection between two classes. Rational Rose stores relationship information in a relationship specification.

Subclasses of Relationship:

HasRelationship, InheritsRelationship, Role, UsesRelationship, Rose RealizeRelationship Class.

Attributes available for Relationship

Name - Kind	Description
ClientCardinality - text	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 - text	Text from the Documentation field of the relationship specification.
ExportControl - text	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 - text	Kind of the relationship, which will be one of: AggregateRole, AssociationRole, HasRelationship, InheritsRelationship or UsesRelationship.
Name - text	The name of the relationship.
Stereotype – text	The stereotype of the relationship.
SupplierCardinality - text	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 - text	The name of the supplier class or use case.
UniqueID –text	The internal unique identifier of the relationship.

Relationships available for Relationship

Name – Kind	Class	Description
FromClass – 1	Class	The client class. For example, if A Has a B, A is the client, or From class.
ToClass – 1	Class	The supplier class. For example, if A Has a B, B is the supplier, or To class.
Properties – n	Property	The properties associated with the relationship.

Rose Role Class

The purpose or capacity where one class associates with another.

Role is a subclass of Relationship.

Attributes specific to Role

Name – Kind	Description
Cardinality - text	
Constraints – text	The text of the Contraints field in the role specification.
Containment – text	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.
ExportControl - text	
Friend – text	True if the Friend check box is selected in the role specification, otherwise False.
IsAggregate – text	True if the role is an aggregate relationship.
IsNavigable – text	True if the Navigable check box is selected, otherwise False.
IsStatic – text	True if the Static check box is selected in the role specification, otherwise False.
UniqueID - text	

Relationships available for Role

Name – Kind	Class	Description
Keys – n	Attribute	Each key is an attribute that uniquely defines a single target object.
FromUseCase – 1	UseCase	The supplier use case of the role, if it is a use case.
Association – 1	Association	The association that this role is a part of.
Class - 1	Class	
OtherRole – 1	Role	The role at the other end of the association.
ToUseCase – 1	UseCase	The client use case of the inherits relationship, if it is a use case.
UseCase -- 1	UseCase	

Rose State Class

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.

Attributes available for State

Name - Kind	Description
Documentation - text	The text in the Documentation field of the state specification.
History - text	The text in the History field of the state specification.
Name - text	The name of the state.
StateKind - text	One of Start, Normal or Stop.
Stereotype - text	
UniqueID - text	

Relationships available for State

Name - Kind	Class	Description
AllSubActivities - n	Activity	
AllSubStates - n	State	
DoActions - n	Action	
EntryActions - n	Action	
ExitActions - n	Action	
ExternalDocs - n	String	
ParentStateActivityModel - 1	StateActivityModel	The parent state machine associated with this state.
Properties - n	Property	The properties attached to this state.
StateActivityDiagrams - n	StateActivityDiagram	
StateActivityModels - n	StateActivityModel	
SubActivities - n	Activity	
SubStates - n	State	
Transitions - n	StateTransition	The transitions that exit from this state.

Rose StateActivityDiagram Class

Attributes available for StateActivityDiagram

Name - Kind	Description
Documentation - text	The documentation attached to the diagram.
HasStateActivityModel - text	True if the diagram includes a state activity model.
IsActivityDiagram - text	
Image - graphic	The graphical representation of the diagram.
Name - text	The name of the diagram.
UniqueID - text	

Relationships available for StateActivityDiagram

Name - Kind	Class	Description
ExternalDocs - n	String	The external documents attached to this diagram.
Notes - n	Note	The notes visible on this diagram.
ParentModel - 1	Model	
StateActivityModel - 1	StateActivityModel	The top-level state activity model associated with this diagram.

Rose StateDiagram Class

Depicts significant event-ordered behavior of a particular class. Each class may have one state diagram to describe its behavior.

Attributes available for StateDiagram

Name - Kind	Description
Documentation - text	The documentation attached to the diagram.
HasStateMachine - text	True if the diagram includes a state machine.
Image - graphic	The graphical representation of the diagram.
Name - text	The name of the diagram.

Relationships available for StateDiagram

Name - Kind	Class	Description
ExternalDocs - n	String	The external documents attached to this diagram.
Notes - n	Note	The notes visible on this diagram.
StateMachine - 1	StateMachine	The top-level state machine associated with this diagram.

Rose StateActivityModel Class

Attributes available for StateActivityModel

Name - Kind	Description
HasStateActivityDiagram - text	True if the state activity model has at least one state or activity diagram.
Name - text	The name of the state activity model.
UniqueID - text	The internal unique identifier of the state activity model.

Relationships available for StateActivityModel

Name - Kind	Class	Description
Activities - n	Activity	The activities defined in this state activity model.
AllActivities - n	Activity	The activities defined in both this state activity model and all nested state activity models.
AllDecisions - n	Decision	The decisions defined in both this state activity model and all nested state activity models.
AllStateActivityDiagrams - n	StateActivityDiagram	The diagrams defined in both this state activity model and all nested state activity models.
AllStates - n	State	
AllSynchronizations - n	Synchronization	The synchronizations defined in both this state activity model and all nested state activity models.
Decisions - n	Decision	The decisions defined in this state activity model.
FirstStateActivityDiagram - 1	StateActivityDiagram	The (first) state or activity diagram associated with this state activity model.
StateActivityDiagrams - n	StateActivityDiagram	The state or activity diagrams associated with this state activity model.

States - n	State	The states that are part of this state activity model.
Synchronizations - n	Synchronization	The synchronizations defined in this state activity model.
Transitions - n	StateTransition	The transitions that are part of this state activity model.

Rose StateTransition Class

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.

Attributes available for StateTransition

Name - Kind	Description
CausingArguments - text	The arguments that accompany the causing event.
CausingEventName - text	The name of the event that causes this transition.
Documentation - text	The text from the Documentation field of this transition.
GuardCondition - text	
SendArguments - text	The arguments that accompany the trigger event.
SendEventName - text	The name of the event triggered by the transition.
SendTarget - text	The name of the object that will receive the transition event.
Stereotype - text	The stereotype of the transition.
SupplierName - text	
UniqueID - text	

Relationships available for StateTransition

Name - Kind	Class	Description
FromActivity - 1	Activity	
FromState - 1	State	The state that this transition emanates from.
Properties - n	Property	The properties attached to this transition.
SendAction - 1	Action	The send action of this transition.
ToActivity - 1	Activity	
ToState - 1	State	The state that this transition leads to.
TriggerAction - 1	Action	The action that triggers this transition.

Rose String Class

The Rose 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 167.

Rose Subsystem Class

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.

Attributes available for Subsystem

Name - Kind	Description
Documentation - text	Text from the Documentation field of the subsystem specification.
Name - text	The name of the subsystem.
Stereotype - text	The stereotype, if any, of the subsystem.
UniqueID - text	

Relationships available for Subsystem

Name - Kind	Class	Description
AllComponents - n	Component	All components nested in this Subsystem, including recursively nested components. For example, if compnent C is nested in Subsystem B is nested in Subsystem A, then AllComponents of A would return B and C.
AllSubsystems - n	Subsystem	
AssignedClasses - n	Class	The classes assigned to this subsystem.
AssignedLogicalPackages - n	Package	
Components - n	Module	

Imports - n	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.
ComponentDiagrams - n	ComponentDiagram	All component diagrams contained in this Subsystem.
MainDiagram - 1	ModuleDiagram	
ParentModel - 1	Model	
Properties - n	Property	
ParentSubsystem - 1	Subsystem	
Referencers - n	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.
Subsystems - n	Subsystem	

Rose Synchronization Class

Attributes available for Synchronization

Name - Kind	Description
Documentation - text	The text from the documentation field for this synchronization.
Name - text	The name of the synchronization.
Stereotype - text	The stereotype of the synchronization.
UniqueID - text	The internal unique identifier of the synchronization.

Relationships available for Synchronization

Name - Kind	Class
Transitions - n	StateTransition

Rose UseCase Class

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.

Attributes available for UseCase

Name - Kind	Description
Documentation - text	Text from the Documentation field of the use-case specification
HasStateActivityDiagram - text	True if the use case has an associated state diagram.
IsAbstract - text	True if the abstract check-box is checked.
Name - text	The name of the use case.
Rank - text	The rank of the use case.
RequisiteProDocName - text	
RequisiteProProjectPath - text	
RequisiteProReqtGUID - text	
Stereotype - text	The stereotype of the use case.
UniqueID - text	

Relationships available for UseCase

Name - Kind	Class	Description
Associations - n	Association	The associations where this use case plays a role.
Properties - n	Property	The properties associated with this use case.
ClassDiagrams - n	ClassDiagram	The class diagrams included in this use case.
ExternalDocs - n	String	The names of the external document associated with this use case.
FirstStateActivityDiagram - 1	StateActivityDiagram	
FirstStateActivityModel - 1	StateActivityModel	
InteractionDiagrams - n	InteractionDagram	The interaction diagrams defined by this use case.
MyRelationships - n	Relationship	The inherits and role relationships defined by this use case.
ParentModel - 1	Model	
ParentPackage - 1	Package	The enclosing package.
ParticipatingObjects - n	Object	The objects included in scenarios defined by this use case.
Properties - n	Property	The properties attached to this use case.
StateActivityDiagrams - n	StateActivityDiagram	The state diagram associated with this use case.
StateActivityModels - n	StateActivityModel	

SubUseCases - n	UseCase	
SuperUseCases - n	UseCase	The use cases that this use case inherits from directly.
UseCaseDiagrams - n	UseCaseDiagram	The use-case diagrams associated with this use case.

Rose UsesRelationship Class

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
ExportControl - text
InvolvesFriendship - text

Relationships specific to UsesRelationship

None

Rose UseCaseDiagram Class

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.

Attributes available for UseCaseDiagram

Name - Kind	Description
Documentation - text	
Image - graphic	The graphical representation of the diagram.
Name - text	The name of the diagram.
UniqueID - text	

Relationships available for UseCaseDiagram

Name - Kind	Class	Description
Classes - n	Class	All of the classes that appear on the diagram.
ExternalDocs - n	String	The external documents attached to this diagram.
ParentModel - 1	Model	The model that contains the diagram.
Notes - n	Note	
ParentPackage - 1	Package	The package that contains the diagram, if applicable.
Packages - n	Package	
Relationships - n	Relationship	All of the relationships that appear on the diagram.
UseCases - n	UseCase	All of the use cases that appear on the diagram.

Rose RealTime Domain

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.

How to Display 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**.

Rose RealTime Domain Classes

Rose RealTime Action Class

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

Rose RealTime Association Class

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.

Rose RealTime AssociationEnd Class

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

Rose RealTime AssociationRole Class

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	

Rose RealTime AssociationEndRole Class

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

Rose RealTime Attribute Class

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.

Rose RealTime Class Class

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.

Rose RealTime ClassDiagram Class

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.

Rose RealTime Classifier Class

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	

Rose RealTime ClassifierRole Class

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

Rose RealTime CallAction Class

CallAction is a subclass of RequestAction Class.

Attributes specific to CallAction

Name - Kind
Operation - text

Relationships specific to CallAction

None

Rose RealTime Capsule Class

Capsule is a subclass of Classifier Class.

Attributes specific to Capsule

None

Relationships specific to Capsule

Name - Kind	Class
Structure - 1	CapsuleStructure

Rose RealTime CapsuleRole Class

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

Rose RealTime CapsuleStructure Class

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

Rose RealTime ChoicePoint Class

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

Rose RealTime ClassUtility Class

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

Rose RealTime Collaboration Class

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

Rose RealTime CollaborationDiagram Class

CollaborationDiagram is a subclass of Diagram Class.

Attributes specific to CollaborationDiagram

None

Relationships specific to CollaborationDiagram

None

Rose RealTime Component Class

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

Rose RealTime ComponentAggregation Class

ComponentAggregation is a subclass of Relationship.

Attributes specific to ComponentAggregation

None

Relationships specific to ComponenetAggregation

Name - Kind	Class
From - 1	Component
To - 1	Component

Rose RealTime ComponentDependency Class

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

Rose RealTime ComponentDiagram Class

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.

Rose RealTime ComponentInstance Class

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

Rose RealTime ComponentPackage Class

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

Rose RealTime Connector Class

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

Rose RealTime Coregion Class

Coregion is a subclass of Action Class.

Attributes specific to Coregion

None

Relationships specific to Coregion

Name - Kind	Class
Messages - n	Message

Rose RealTime CreateAction Class

CreateAction is a subclass of Action Class.

Attributes specific to CreateAction

Name - Kind
Operation - text

Relationships specific to CreateAction

None

Rose RealTime DeploymentDiagram Class

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.

Rose RealTime DeploymentPackage Class

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

Rose RealTime DestroyAction Class

DestroyAction is a subclass of Action Class.

Attributes specific to DestroyAction

None

Relationships specific to DestroyAction

None

Rose RealTime Device Class

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

Rose RealTime Diagram Class

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

Rose RealTime Element Class

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.

Rose RealTime Environment Class

Environment is a subclass of InteractionInstance Class.

Attributes specific to Environment

None

Relationships specific to Environment

None

Rose RealTime File Class

Attributes specific to File

Name – Kind
IsURL – text
Value – text

Relationships specific to File

Name - Kind	Class
ParentLogicalPackage - 1	Package

Rose RealTime FinalState Class

FinalState is a subclass of StateVertex Class.

Attributes specific to FinalState

None

Relationships specific to FinalState

None

Rose RealTime Generalization Class

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

Rose RealTime InitialPoint Class

InitialPoint is a subclass of StateVertex Class.

Attributes specific to InitialPoint

None

Relationships specific to InitialPoint

None

Rose RealTime InstantiatedClass Class

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

Rose RealTime InstantiatedClassUtility Class

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

Rose Realtime InstantiateRelationship Class

InstantiateRelationship is a subclass of Relationship.

Attributes specific to InstantiateRelationship

None

Relationships specific to InstantiateRelationship

Name - Kind	Class
From - 1	Class
To - 1	Class

Rose RealTime Interaction Class

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

Rose RealTime InteractionInstance Class

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

Rose RealTime JunctionPoint Class

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

Rose RealTime LocalState Class

LocalState is a subclass of Action Class.

Attributes specific to LocalState

None

Relationships specific to LocalState

None

Rose RealTime Message Class

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.

Rose RealTime MetaClass Class

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

Rose RealTime Model Class

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.

Rose RealTime ModelElement Class

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

Rose RealTime NoteView Class

Attributes specific to NoteView

Name – Kind
Text – text
Type – text

Relationships specific to NoteView

Name - Kind	Class
ModelElement - 1	ModelElement
ParentDiagram - 1	Diagram

Rose RealTime Operation Class

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

Rose RealTime Package Class

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	

Rose RealTime PackageDependency Class

PackageDependency is a subclass of Relationship.

Attributes specific to PackageDependency

None

Relationships specific to PackageDependency

Name - Kind	Class
From - 1	Package
To - 1	Package

Rose RealTime Parameter Class

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

Rose RealTime ParameterizedClass Class

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.

Rose RealTime ParameterizedClassUtility Class

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.

Rose RealTime Port Class

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

Rose RealTime PortRole Class

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

Rose RealTime Processor Class

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

Rose RealTime Property Class

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

Rose RealTime Protocol Class

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

Rose RealTime RealizeRelationship

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

Rose RealTime Relationship Class

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.

Rose RealTime ReplyAction Class

ReplyAction is a subclass of ResponseAction Class.

Attributes specific to ReplyAction

Name - Kind
Data - text
Signal - text

Relationships specific to ReplyAction

None

Rose RealTime RequestAction Class

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

Rose RealTime ResponseAction Class

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

Rose RealTime ReturnAction Class

ReturnAction is a subclass of ResponseAction Class.

Attributes specific to ReturnAction

None

Relationships specific to ReturnAction

None

Rose RealTime SendAction Class

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

Rose RealTime SequenceDiagram Class

SequenceDiagram is a subclass of Diagram Class.

Attributes specific to SequenceDiagram

None

Relationships specific to SequenceDiagram

None

Rose RealTime Signal Class

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

Rose RealTime State Class

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.

Rose RealTime StateDiagram Class

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.

Rose RealTime StateMachine Class

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	

Rose RealTime StateVertex Class

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

Rose RealTime String Class

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 203.

Rose RealTime TerminateAction Class

TerminateAction is a subclass of Action Class.

Attributes specific to TerminateAction

None

Relationships specific to TerminateAction

None

Rose RealTime Transition Class

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

Rose RealTime Trigger Class

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

Rose RealTime UninterpretedAction Class

UninterpretedAction is a subclass of Action Class.

Attributes specific to UninterpretedAction

Name – Kind
Code – text
Effect – text

Relationships specific to UninterpretedAction

None

Rose RealTime UseCase Class

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.

Rose RealTime UsesRelationship Class

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

TeamTest Domain

TeamTest Domain Classes

TeamTest Project Class

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.

Attributes available for Project

Name - Kind	Description
ClearQuestDatabaseName	The name of the associated ClearQuest database
Directory - text	The directory that contains the project
Name - text	The name of the project
Path - text	The full path of the project
RequisiteDBPath - text	The full path of the associated RequisitePro project

Relationships available for Project

Name - Kind	Class	Description
Builds - n	Build	The builds defined in the project.
Computers - n	Computer	The computers defined in the project
Requirements - n	Requirement	
Schedules - n	Schedule	The schedules created in the project
Scripts - n	Script	All scripts included in the project
TestDocuments - n	TestDocument	The test documents associated with the project
Users - n	User	

TeamTest Build Class

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.

Attributes available for Build

Name - Kind	Description
CreationDate - text	
Description - text	The description of the build (from the General tab)
ModificationDate - text	
Name - text	The name of the build (from the General tab)
Notes - text	Related notes for the build (from the Specifications tab)
Owner - text	The owner of the build (from the General tab)
State - text	The state of the build (from the General tab)

Relationships available for Build

Name - Kind	Class	Description
Creator - 1	User	The user that created the build.
LogFolders - n	LogFolder	The log folders that are included with this build.
Owner - 1	User	
ScriptsUserd - n	Script	

TeamTest Computer Class

A computer is a machine involved in the testing process.

Attributes available for Computer

Name - Kind	Description
Description - text	A description of the computer
IsClient – text	
IsGUIAgent – text	
IsServer - text	
IsVUAgent - text	
Name - text	The name of the computer
NetworkName - text	The network name of the computer
OperatingSystem - text	The operating system of the computer

Relationships available for Computer

Name - Kind	Class
Ports - n	Port

TeamTest Event Class

An event is an object in a schedule upon which another object is dependent.

Attributes available for Event

Name - Kind	Description
ActualResultsFile - text	The location of the actual results
AdditionalInformation - text	Any additional information about the result (from the Result tab)
AgentLogFile - text	The location of the log on the agent machine
BaselineResultsFile - text	The location of the baseline results
ComputerName - text	The name of the computer the script was run on (from the Configuration tab)
ConfigurationSettingCount - text	The number of configuration settings
DisplayLevel - text	
EndDateTime - text	The time the event ended
FailureDescription - text	The failure description (from the Result tab)
FailureReason - text	The failure reason of the even (from the Result tab)
Index - text	
Output - text	
ParentLogFilename - text	The name of the parent log
Result - text	The result of the event (from the Result tab)
ScriptLineNumber - text	The script line number (from the General tab)
StartDateTime - text	The start date and time (from the General tab)
TimerName - text	
UAWResponse - text	
UserErrorFile - text	
UserLogFile - text	
UserOutputFile - text	

UserType - text	
VPName - text	The name of the verificaiton point
VPType - text	The type of the verification point
VPTypeName - text	

Relationships available for Event

Name - Kind	Class	Description
ConfigurationLogEvent - 1	Event	The associated configuration event
Schedule - 1	Schedule	The associated schedule, if any
Script - 1	Script	The associated script, if any
User - 1	User	The creator of the event

TeamTest Group Class

A group is a collection of users that execute similar tasks.

Attributes available for Group

Name - Kind	Description
Description - text	The description of the group
Name - text	The name of the group

Relationships available for Group

None

TeamTest Log Class

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.

Attributes available for Log

Name - Kind	Description
AgentLogFilesPath - text	The location of log files on the agent computer
CreationDate - text	The date the log was created
Description - text	A descripton of the log
LastModifiedBy - text	The ID of the person who last modified the log

LogFileListPath - text	The location of the log file list
MasterLogFilePath - text	The location of the log files on the master computer
ModificationDate - text	The date the log was last modified
Name - text	The name of the log
PerformanceDataPath - text	The location of the performance data
UAWPath - text	The location of unexpected active window data
UserLogFilePath - text	The location of user data
VPPath - text	The location of verification point data

Relationships available for Log

Name - Kind	Class	Description
Creator - 1	User	The user that created this log
Events - n	Event	The events contained in this log

TeamTest LogFolder Class

A log folder is a directory that contains test logs.

Attributes available for LogFolder

Name - Kind	Description
CreationDate - text	The date the folder was created
Description - text	A description of the folder
LastModifiedBy - text	The ID of the person who last modified the folder
Name - text	The name of the log folder

Relationships available for LogFolder

Name - Kind	Class	Description
Creator - 1	User	The user that created the log folder
Logs - n	Log	The logs contained in this folder

TeamTest Name Class

Attributes available for Name

Name - Kind
Value - text

Relationships available for Name

None

TeamTest Port Class

Attributes available for Port

Name - Kind	Description
Number - text	The number of the port
Name - text	The name of the port

Relationships available for Port

None

TeamTest Requirement Class

Attributes available for Requirement

Name - Kind
FullTag - text
Text - text
Type - text
VersionDateTime - text
VersionLabel - text

VersionNumber - text
VersionReason - text
VersionUser - text

Relationships available for Requirement

Name - Kind	Class
Attributes - n	Attribute
Children - n	Requirement
Scripts - n	Script
Schedules - n	Schedule

TeamTest ReqtAttribute Class

Attributes available for ReqtAttribute

Name - Kind
Name - text
Value - text

Relationships available for ReqtAttribute

None

TeamTest Schedule Class

A schedule specifies how LoadTest should perform tests.

Attributes available for Schedule

Name - Kind	Description
Custom1 - text	The first custom field for the schedule
Custom2 - text	The second custom field for the schedule
Custom3 - text	The third custom field for the schedule
Description - text	The description of the schedule
Developed - text	True if the schedule was developed

Name - text	The name of the schedule
Notes - text	Any notes associated with the schedule
RequirementDBKey - text	The ID of the associated RequisitePro requirement
SpecificationFilePath - text	The location of the specification file
Type - text	The type of the schedule

Relationships available for Schedule

Name - Kind	Class	Description
Creator - 1	User	The user who created this schedule
Owner - 1	User	The user who owns this schedule

TeamTest Script Class

A script is a file of SQABasic or VU commands.

Attributes available for Script

Name - Kind	Description
BinaryFilePath - text	The location of the binary file
Custom1 - text	The value of the Custom 1 field (from the Custom tab)
Custom2 - text	The value of the Custom 2 field (from the Custom tab)
Custom3 - text	The value of the Custom 3 field (from the Custom tab)
Description - text	The description of the script (from the General tab)
DLLPath - text	The location of the DLL
Environment - text	The operating environment for the script (from the General tab)
FilePath - text	The location of the script
IncludePath - text	The location of included files
IsDeveloped - text	Yes if developed; otherwise No (from the General tab)
LLSPath - text	
Name - text	The name of the script (from the General tab)
Notes - text	Related notes for the script (from the Specifications tab)
Purpose - text	The purpose of the script (from the General tab)

SpecificationFilePath - text	The path to the specification file (from the Specifications tab)
Type - text	The type of the script, GUI or VirtualUser (from the General tab)
VPPath - text	The location of the verification points

Relationships available for Script

Name - Kind	Class	Description
Creator - 1	User	The user that created the script
Defines - n	Name	
Libraries - n	Name	
Owner - 1	User	The owner of the script
VerificationPoints - n	VerificationPoint	The verification points included in the script

TeamTest TestDocument Class

A test document is a test plan, project schedule, resource allocation, or any other documents that are important to your project.

Attributes available for TestDocument

Name - Kind	Description
Description - text	The description of the test document
DocumentPath - text	The location of the test document
Name - text	The name of the test document

Relationships available for TestDocument

Name - Kind	Class	Description
Creator - 1	User	The user that created the document

TeamTest User Class

A user is a person who can log in to the TeamTest environment.

Attributes available for User

Name - Kind	Description
Company - text	The company this user works for
Department - text	The department this user works in
EmailAddress - text	The e-mail address of this user
FirstName - text	The first name of the user
ID - text	
LastName - text	The last name of the user
PhoneNumber - text	The phone number of the user
Title - text	The job title of the user

Relationships available for User

None

TeamTest VerificationPoint Class

A verification point is a point in an SQABasic that confirms the state of one or more objects.

Attributes available for VerificationPoint

Name - Kind	Description
BaselineFilePath - text	The associated baseline verification point
MetadataFilePath - text	
Name - text	The name of the verification point
Type - text	The type of the verification point

Relationships available for VerificationPoint

None

Word Domain

The Word domain lets you insert the contents of other Word documents into your SoDA document.

The Word domain includes the following classes:

Word Document Class

Word Paragraph Class

Word Heading Class

Word Bookmark Class

Word Domain Classes

Word Document Class

This class, a subclass of FileSys File class, represents a Word document.

Attributes specific to Document

Name (Kind)	Description
FormattedText (text)	The complete contents of the Word document, pasted as formatted text.
FormattedTextAfterLastBookmark (text)	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.
Text (text)	The complete contents of the Word document, pasted as a string.

Relationships specific to Document

Name (Kind)	Class	Description
Bookmarks (n)	Bookmark	The bookmarks defined in this document
Headings (n)	Heading	All headings found in this document
Paragraphs (n)	Paragraph	All paragraphs, including headings, found in this document.

Word Paragraph Class

Paragraphs are available only in the Word 97 or Word 2000 version of the Word domain.

Attributes available for Paragraph

Name (Kind)	Description
FormattedText (text)	The complete contents of the Word document, pasted as formatted text.
Position (text)	The character position of the first character of the paragraph
Style (text)	The style of the paragraph, such as “Normal”
Text (text)	The complete contents of the Word document, pasted as a string.

Relationships available for Paragraph

Name (Kind)	Class	Description
NextParagraph (1)	Paragraph	The next paragraph in the document
ParentHeading (1)	Heading	The nearest heading above the current paragraph
PreviousParagraph (1)	Paragraph	The previous paragraph in the document

Word Heading Class

Headings are paragraphs with a style “Heading1”, “Heading2”, and so on. Headings are available only in the Word 97 or Word 2000 version of the Word domain.

Attributes available for Heading

Name (Kind)	Description
FormattedText (text)	The text of the heading, pasted as formatted text.
Label (text)	The label of the heading, such as “1.1.2”
Position (text)	The character position of the first character of the paragraph
Style (text)	The style of the heading, such as “Normal Arial 10"
Text (text)	The text of the heading, pasted as a string.

Relationships available for Heading

Name (Kind)	Class	Description
ChildHeadings (n)	Heading	The headings contained within this heading, one level in
ChildParagraphs (n)	Paragraph	The paragraphs contained within this heading, including headings
NextHeading (1)	Heading	The next heading at the same level
ParentHeading (1)	Heading	The parent heading for this heading (one level up)
PreviousHeading (1)	Heading	The previous heading at the same level

Word Bookmark Class

Bookmarks are available only in the Word 97 or Word 2000 version of the Word domain.

Attributes available for Bookmark

Name (Kind)	Description
EndPosition (text)	Allows sorting of bookmarks by end position in the document.
FormattedText (text)	The text of the area defined by the bookmark, pasted as formatted text.
FormattedTextBefore (text)	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 (text)	The name of the bookmark.
StartPosition (text)	Allows sorting of bookmarks by start position in the document.
Text (text)	The text of the area defined by the bookmark, pasted as a string.

Relationships available for Bookmark

Name (Kind)	Class	Description
ParentParagraph (1)	Paragraph	The paragraph that contains the first character of the bookmark.

Index

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W