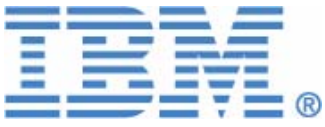
Rational. Statemate

Statemate User Guide

Rational Statemate User Guide



Before using the information in this manual, be sure to read the “Notices” section of the Help or the PDF file available from **Help > List of Books**.

This edition applies to IBM® Rational® Statemate® 4.6 and to all subsequent releases and modifications until otherwise indicated in new editions.

© Copyright IBM Corporation 1997, 2009.

U.S. Government Users Restricted Rights - Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Rational Statemate Overview 1

The Rational Statemate Development Environment 1

Rational Statemate Interface Features 1

The Rational Statemate Main Window 2
Tabs 4
Charts Tab 4
Files Tab 5
Databank Tab 5
Search Tab 10
Messages Tab 11
Log Tab 11
Right-click Menus 12
Toolbars 15
Edit Toolbar 15
Change Tracking Toolbar 15
Information Toolbar 16
Properties Toolbar 16
Tools Toolbar 17
Menus 18
File Menu 18
Edit Menu 19
View Menu 21
List Menu 21
Project Menu 22
Configuration Menu 22
Tools Menu 24
Utilities Menu 25
Windows Menu 25
Help Menu 25

Activity Interface Browser 26

Menus 27
Tool Bar 27

Working Environment 29

Projects 29
Creating a Project 30
Modifying a Project 32
Displaying Project Settings 34
Opening a Project 35
Deleting a Project 36
Closing a Project 36
Workareas 37
Creating a Workarea 37
Opening a Workarea 39
Moving a Workarea 39
Copying a Workarea 40
Reducing the Workarea 40
Deleting a Workarea 41
Sharing the Workarea 41
Sharing and Locking 42
Temporary Workarea Directory 42
Output Devices 43
Setting Up Output Devices 43
Modifying Output Devices 45
Deleting Output Devices 46
Preferences 46
Setting General Rational Statemate Preferences 46
Specifying Where Preferences are Applied 47
Specifying Access to Preferences 47
Preferences Descriptions 48
General Preferences 49
Preferences Management 52
Statechart Graphic Editor Preferences 52
Activity-Chart Graphic Editor Preferences 54
Module-chart Preferences 57
Flowchart Graphic Editor Preferences 58
Sequence Diagram Graphic Editor Preferences 59
Use-Case Diagram Graphic Editor Preferences 60
Panel Graphic Editor Preferences 61
Graphic Editors Preferences 62
Databank Browser Preferences 63
Simulation Preferences 64
Properties Preferences 66
Check Model Preferences 68
Prototype C Code Generator Preferences 69
MicroC Code Generator 71
Embedded Rapid Prototyper Preferences 72

RT Interface Preferences . . . 73
Panel Builder Preferences . . . 74
. . . 84
Activity Interface Browser and Reports Preferences . . . 85
Loading Predefined Preferences . . . 86
Setting Parameter Preferences . . . 87
Setting Preferences for Editors and Utilities . . . 89

Configuration Management . . . 90
Databank . . . 90
Elements . . . 91
Working with the Databank . . . 92
Automatic Databank Refresh . . . 93
Checking Out Databank Items . . . 93
Creating a New Configuration . . . 94
Locking Databank Items . . . 95
Exporting Charts and Files . . . 96
Error Handling when Loading Charts . . . 96
Checking In and Out Elements . . . 97
Tracking Changes . . . 98
Automatic Change Tracking . . . 98
Track Changes Preferences . . . 99
Types of Changes Tracked . . . 99
Track Changes Limitations . . . 99

Database Diagnostics . . . 100
Error Report . . . 100
View and Resolve Errors . . . 101

Plugins . . . 102

Using the Graphic Editors . . . 105

Overview of the Rational Statemate Graphic Editors . . . 106
Graphic Editor Icons . . . 107
Graphic Editor Menus . . . 109
File Menu . . . 110
Edit Menu . . . 112
View Menu . . . 113
Layout Menu . . . 116
Tools Menu . . . 118
Options Menu . . . 121

Working with Graphic Editors . . . 123
Starting a Graphic Editor . . . 123
Creating a New Chart or Diagram . . . 123
Creating a New Chart or Diagram from the Open Chart Window . . . 124

Creating a New Chart or Diagram with a Graphic Editor . . . 126
Drawing Operations in Graphic Editors . . . 127
Drawing Boxes . . . 127
Drawing Lines . . . 128
Drawing Connectors . . . 128
Editing Text . . . 128
Selecting Elements . . . 128
Labeling Elements . . . 129
Moving Elements . . . 130
Copying Elements . . . 130
Resizing Elements . . . 130
Deleting Elements . . . 131
Constraining Graphic Operations . . . 131
General Operations in Graphic Editors . . . 132
Opening the Properties Window for Elements . . . 132
Displaying Element Properties . . . 132
Opening the Properties Window for an Entire Chart . . . 132
Displaying Chart Properties . . . 132
Displaying Subroutine Properties . . . 133
Opening a Simulation Execution Window . . . 133
Invoking the Check Model Tool . . . 133
Invoking the RT Interface . . . 133
Closing a Chart . . . 133
Saving a Chart . . . 134
Opening a Parent Chart . . . 134
Opening a Sub-Chart . . . 134
Inserting a Chart . . . 135
Creating a Sub-Chart . . . 135
Exiting the Graphic Editor . . . 135

Working with Charts and Diagrams . . . 136
Activity Charts . . . 136
Accessing an Activity Chart . . . 136
Activity Chart Icons . . . 138
Module Charts . . . 142
Accessing a Module Chart . . . 142
Statecharts . . . 146
Accessing a Statechart . . . 147
Statechart Icons . . . 148
Associating a Statechart with an Activity . . . 151
“Only Once” Test Benches . . . 151
Use Case Diagrams . . . 152
Accessing a Use Case Diagram . . . 152
Use Case Diagram Icons . . . 154
Use Case Diagram Properties . . . 155

Linking Use Cases to Scenarios . . . 157
Sequence Diagrams . . . 159
Accessing a Sequence Diagram . . . 161
Sequence Diagram Icons . . . 162
Sequence Diagram Drawing Notes . . . 163
Lifeline Decomposition . . . 163
Integrating Sequence Diagrams with the Rational Statemate Model . . . 164
Generating a Sequence Diagram from an Activity Chart . . . 164
Using Properties with Sequence Diagrams . . . 166
Auto-Numbering in Sequence Diagrams . . . 167
Print Pagination . . . 167
Flowcharts . . . 168
Accessing a Flowchart . . . 168
Flowchart Icons . . . 168
Flowchart as Subroutine Implementation . . . 170

Panels . . . 173

Using the Panel Editor . . . 175
Accessing the Panel Editor . . . 175
Panel Editor Menus . . . 177
File Menu . . . 177
Edit Menu . . . 178
View Menu . . . 179
Layout Menu . . . 181
Transform Menu . . . 183
Group Menu . . . 184
Tools Menu . . . 184
Options Menu . . . 185
Panel Editor Icons . . . 186
Interactor Icons . . . 186
Drawing and Naming Icons . . . 187

Binding Interactors . . . 188
Individual Bindings . . . 191
Group Bindings . . . 191

Using the Panel Builder . . . 192

Element Properties . . . 195

Understanding Elements . . . 196
Textual Elements . . . 196
Textual Types, Sub-types, and Structures . . . 200
Default Values for Textual Elements . . . 201
Graphical Elements . . . 202

Chart Elements 204

Creating and Modifying Elements 207

Quick-Edit Mode 207

Elements in the Chart Hierarchy 207

Quick-Edit Mode Limitations 207

Invoking the Properties Dialog Box 208

Subroutine Properties 209

Editing Multiple Elements 209

Record/Union Field Properties 209

Cut, Copy and Paste Operations on Record/Union Fields 209

Properties Preference “Mass edit overwrite values” 209

Properties Window 210

Individual Property Fields Display 210

Toolbar Operations 210

Searching Charts 210

Property Information Displayed in Tabs 211

Creating and Modifying Elements 218

Resetting Default Values for Elements 243

Searching for Elements 244

Starting the Search Tool 244

Creating a List of Elements 246

Saving a List 247

Accessing a Stored List 248

Filtering a List of Elements 249

Appending to a List of Elements 249

Running an Advanced Query 250

Finding Where Elements are Referenced and Used 256

Finding Where Elements are Referenced 256

Finding Where Elements are Used 258

Libraries and Components 259

Working with Components 261

Creating a Component 261

Inserting a Component 263

Copying a Component 265

Previewing a Component 266

Deleting a Component 267

Managing Components 267

Working with Libraries 268

Adding Libraries to a Project 269

The Router Element 271

Router Element for the Activity Charts 271

Working with the Router 272

Drawing Router Blocks 273

Using Routers to Reduce Flow Lines 275

Router Rules 275

Compound Flow Lines through Routers 276

Defining Router Properties 277

Interface Reporting 279

Local Interface Report 279

Global Interface Report 280

Using Check Model with Router Blocks 282

Exporting Router Blocks to Rational DOORS 282

Setting Router Preferences 283

Internal Router Preferences 284

External Router Preferences 285

Global Definition Set Editor 287

Creating a New GDS 288

Editing an Existing GDS 291

GDS Properties 292

GDS Usage Property 292

GDS Visibility Mode Property 292

Reduced GDS 293

MicroC Code Generator 295

Scope Definition 295

Module Structure 295

Testbenches 296

Creating a Sample Profile 296

Invoking the Profile Editor 296

Defining Code Modules 298

Assigning Behavior to the Module 299

Splitting Activity Chart Hierarchy 300

Code Options 301

RESET_Data as Function 301

Ignore External Binding 301

Code Generation for Control Activities 301

Enhanced Generated Code-Level Readability and Documentation 301

Support Selective GBA 302

Byte Orientation Instrumentations . . . 302
Single-Bit Elements . . . 302
User-Code Generation . . . 302
Setting the Time Scale . . . 303
Setting the Time Expression Scale Preference . . . 303
Working with Multiple Counters . . . 304
Setting the Time Expression Scale . . . 304
Defining Counters in a cfg file . . . 305
Generation of Constant Elements with “const” Modifier . . . 306
Default Data Types . . . 306
Generating Code with Extended Documentation . . . 306
Dynamic Data Initialization . . . 307
OSEK GetResource Usage . . . 307

Rational Statemate Block in a Rational Rhapsody Model . . . 308

Required Rational Statemate Model Characteristics . . . 308
Preparing the Rational Statemate Model . . . 308
Synchronizing Rational Statemate and Rational Rhapsody . . . 309
Troubleshooting Rational Statemate with Rational Rhapsody . . . 309

Code Optimizations . . . 311

Empty Overlapping Tests of State Hierarchy . . . 311
Generate All and Generate Only Used . . . 311
Optimization Algorithms . . . 311
Inline Setting of the “Need Another Step” Bit . . . 311
Inline Entering and Exiting Reactions . . . 312
Reuse of Timeout Variables . . . 313
Clutching Entrance to a State Hierarchy . . . 313
Additional Optimization Options for Code Generation . . . 313

OS Definition Tool . . . 314

Design Attributes . . . 314
Design Attribute Notation . . . 314
Inheritable Design Attributes . . . 315
Special Design Attributes . . . 315
Element Attributes . . . 316
Task Execution Mode API and Design Attributes . . . 316
Get-Set Functions for Buffered Access Data-Items . . . 316
OS Static Configuration . . . 317
Defining the Location of the CTD Directory . . . 317
APIs . . . 317
Customizable Timeouts using OSDT . . . 320
Support for Queues . . . 321
Task/ISR APIs . . . 321
Statecharts Functions . . . 323
APIs for Function-Declare-Style . . . 324

Customizable OSEK APIs 324
API Modification Rules 325
Upgrading an OSI 325
List Support in OSDT 326
Generated Data Declaration 328

Supported Targets 330

Utilities 330
Remote Panel Server Support 330
Using the Remote Panel Server 331
Invoking the Remote Panel 331
MicroC Design-Level Debugger 332

Rational DOORS RT Interface 333

How the RT Interface Works 334
Exporting Data 334
Re-Exporting and Synchronizing Data 335
Methodology Guidelines 336

Configuring the RT Interface 337
Preliminary Requirements 337
Configuring the RT Interface on Windows 338
Edit run_stmm.bat 339
Edit doorss.bat 340
Edit GetDoorsVer.bat file 340
Edit run_doors.bat 341

Working with the RT Interface 341
Associating a Rational Statemate Project with a Rational DOORS Project 342
Setting Preferences 343
Setting Preferences for Chart Plots 343
Setting Preferences for External Use-Case Files 346
Setting Preferences for Log Files 347
Setting Up a Default Configuration File 348
Exporting Rational Statemate Data to Rational DOORS 350
Preparing Rational Statemate Elements for Export to Rational DOORS 350
Configuring Attributes for Export 353
Configuring Filtering by Attribute 355
Configuring Linksets for Export 358
Creating Multiple Linksets in a Single Link Module 362
Exporting 363
Re-Exporting Rational Statemate Data to Rational DOORS 364
Using a Saved Configuration File 364
Rational DOORS Interface Support for Transitions 365

Truth Tables 367

Format and Content of Truth Tables 368

Special Characters 368
Input Columns 368
Valid Input Elements 369
Input Column Header Operators 370
Invalid Input Elements 370
Output Columns 371
Output Elements 371
Action Column 372

Executing Truth Tables 372

Default Row 372
Row Execution 373
Truth Table Contents for Activities and Actions 373
Truth Table Contents for Subroutines 374
Micro-step Execution of Procedure Truth Tables 374
Execution of Action Truth Tables 375
Factorization of Cells 375
Factorizing Inputs 375
Factorizing Outputs and Actions 377

Defining a Truth Table 378

Lookup Tables 381

Defining a Lookup Table 381

Example of a Lookup Table 383

Example Components 385

Overview 385

Example Component Library 386

STM_BRANCH_2 388
STM_BRANCH_3 391
STM_FORK_2 395
STM_FORK_3 397
STM_JOIN_2 399
STM_JOIN_3 402
STM_FIFO_ACTIVE 406
STM_FIFO_PASSIVE 410
STM_LIFO_ACTIVE 414
STM_LIFO_PASSIVE 418
STM_PMPT_ACTIVE 422
STM_PRTY_PASSIVE 426

STM_SINK 430
STM_SOURCE 432

AUTOSAR Generator 435

Overview of the AUTOSAR Interface 436
AUTOSAR Menus 436
AUTOSAR Toolbar Options 437
Scope Definition Area 437

Generating Code and XML Description 445

Timeouts 445

In-Out Elements 446

SAG Implementation of AUTOSAR Features 447
. . . . . Exclusive Areas447
Timing Events 447
Data Types 448
Example 1:Default behavior: 448
Example 2- Using a UDT: 449
Example 3 - Using an externally defined type: 450
Services 450
Data Send Points, Data Receive Points, Data Read Access, Data Write Access 451
Inter Runnable Variables 454
Mode Declaration Groups 455

The AUTOSAR RTE OS ImplementationsI 456

Important Notes 456

Creating an AUTOSAR Project 456

Technical Support 457

Contacting IBM Rational Software Support 457

Prerequisites 457

Contacting Support 458

Reporting Rational Statemate Problems from the Software 460

Glossary of Rational Statemate Terminology 461

Index 505

Rational Statemate Overview

IBM® Rational® Statemate® is a high-level graphical development environment that enables systems engineers to design, validate, and simulate models that clearly and precisely represent the intended functions and behavior of the system they are developing.

The Rational Statemate Development Environment

Rational Statemate encapsulates the design and simulation of simple or complex reactive systems, enabling systems engineers to flesh out problems and firm up their specifications during the design stage, rather than during implementation – or worse, after the system is in production or has shipped to customers.

Although problems discovered beyond the requirements-gathering and design stages cost roughly 100 times more to fix, almost half the work done on systems projects is devoted to reworking just these sorts of problems.

Rational Statemate is designed to eliminate this sort of costly re-engineering by increasing project and system clarity (which fosters cooperation and reduces misunderstandings between collaborating groups and suppliers), identifying ambiguous or conflicting requirements, targeting modules and sub-systems for reuse, and providing accurate simulation and rigorous validation - all of which translates into increased productivity, faster time-to-market, and an overall increase in enterprise Return On Investment (ROI).

Rational Statemate Interface Features

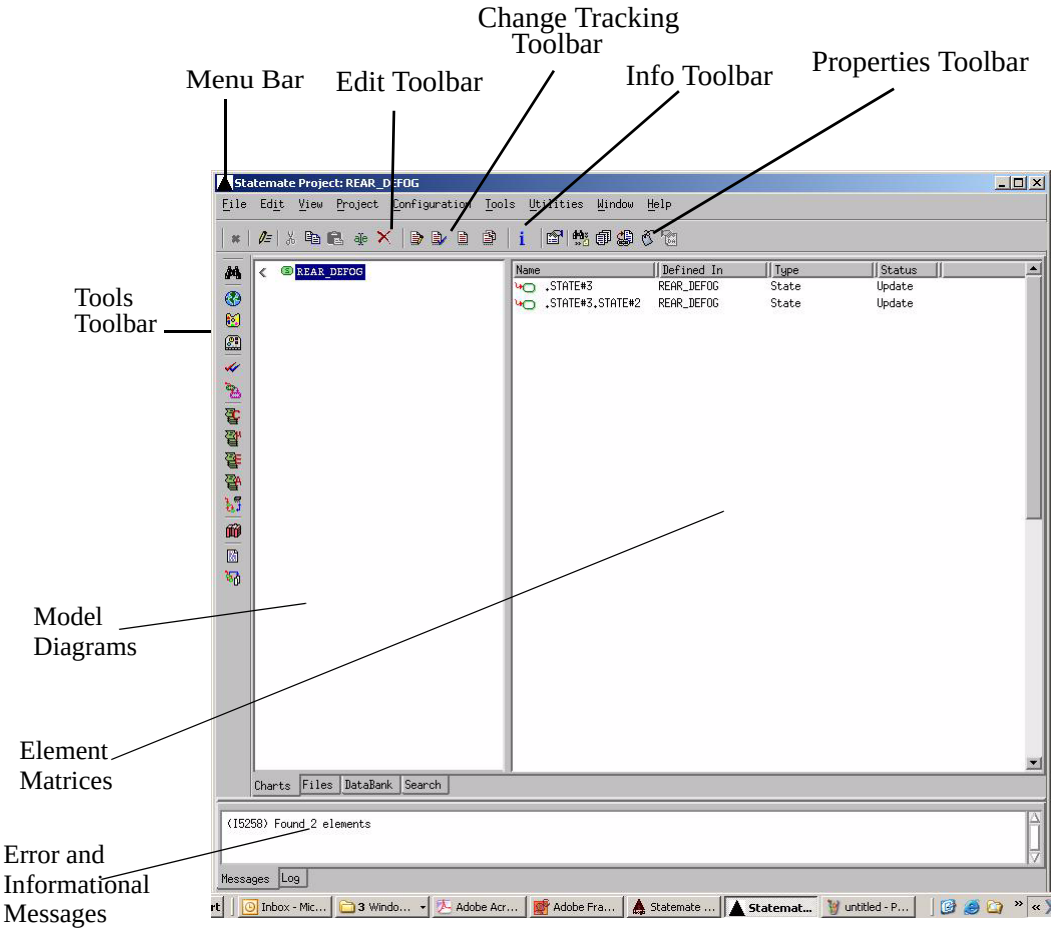
This section provides a description of these primary features of the Rational Statemate interface:

- **The Rational Statemate Main Window**
- **Tabs**
- **Toolbars**
- **Menus**

The Rational Statemate Main Window

Rational Statemate consists of a set of tools that interact with one another to provide a complete system design environment. From the Rational Statemate main window, Rational Statemate provides various browser views, windows and drawing areas, menus and toolbars to access and use its collection of system design, validation, simulation, and requirements-gathering utilities.

The Rational Statemate main window displays a hierarchy of your project, and provides easy access to the elements and diagrams it contains. The following figure shows the different areas of the Rational Statemate main window.



The Rational Statemate main window displays the current project name in the top status bar and includes tabs, toolbars, and menus to help you use Rational Statemate.

In the Rational Statemate main window, tabs at the bottom provide four views for looking at Rational Statemate project elements. The Rational Statemate main window provides the following views:

- **Charts Tab**
- **Files Tab**
- **Search Tab**

Below each of the four main tabs are a **Messages Tab** and a **Log Tab**, that display the status of your Rational Statemate operations in linear or searchable form.

A menu bar at the top of the window provides access to many Rational Statemate tools and features. In addition to the menu bar, the main window provides several toolbars:

- **Edit Toolbar**
- **Change Tracking Toolbar**
- **Information Toolbar**
- **Properties Toolbar**
- **Tools Toolbar**

Note

The graphic editors have additional toolbars and menus, which are described in **Using the Graphic Editors** and **Graphic Editor Menus**.

Note

All Statement windows allow standard multiple row selection operations using the CTRL and SHIFT keys as well as cut, copy and paste operations between windows.

The following matrix operations are available for elements on the right pane:

- Sort by column
- Scroll list on key-press
- Cut/Copy/Paste

The following sections describe the tabs, toolbars, and menus on the main window.

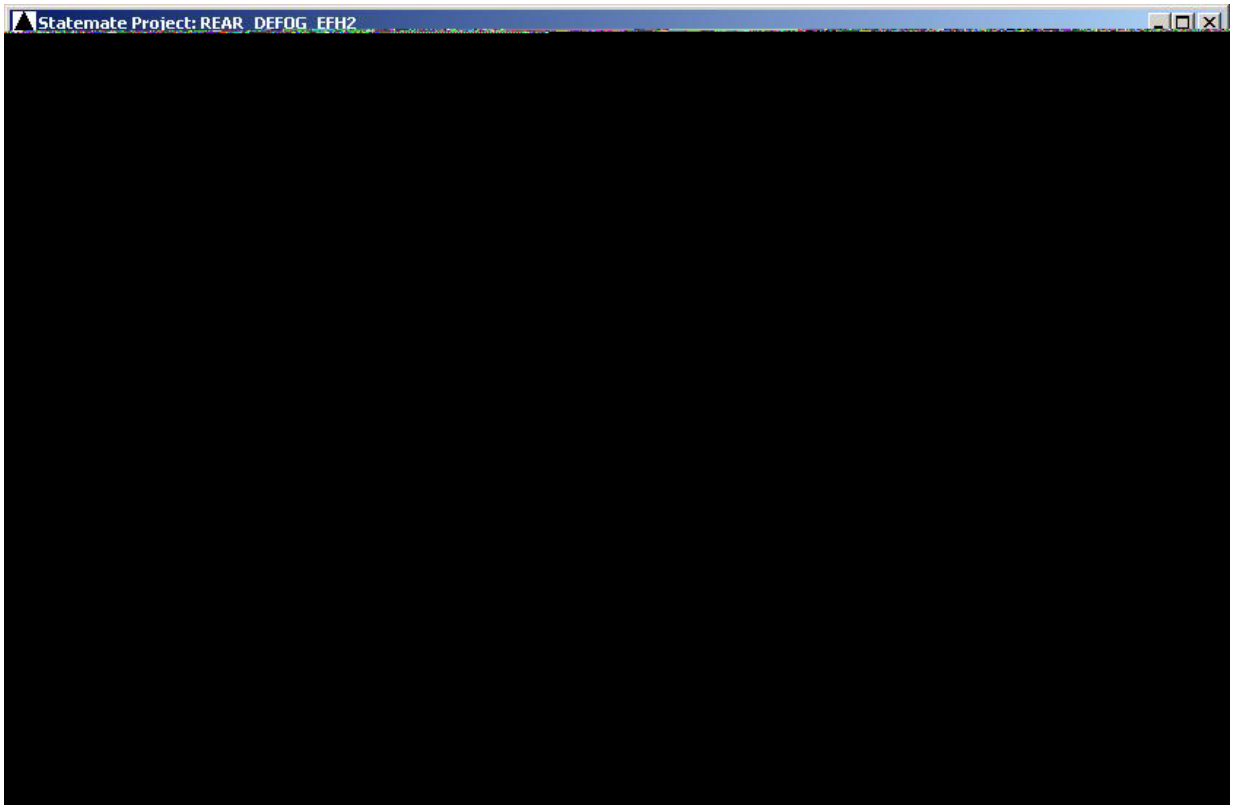
Tabs

The Rational Statemate main window contains four tabs that provide different views of Rational Statemate files, elements, and status.At the bottom of the window, there are two tabs that display error and informational messages.

Charts Tab

The Charts tab divides the Rational Statemate window into the left hand tree, which displays project charts and diagrams in their hierarchical tree structure, and the right hand matrix, which displays details of all charts and diagrams, sorted by name, definition location, type, and status.

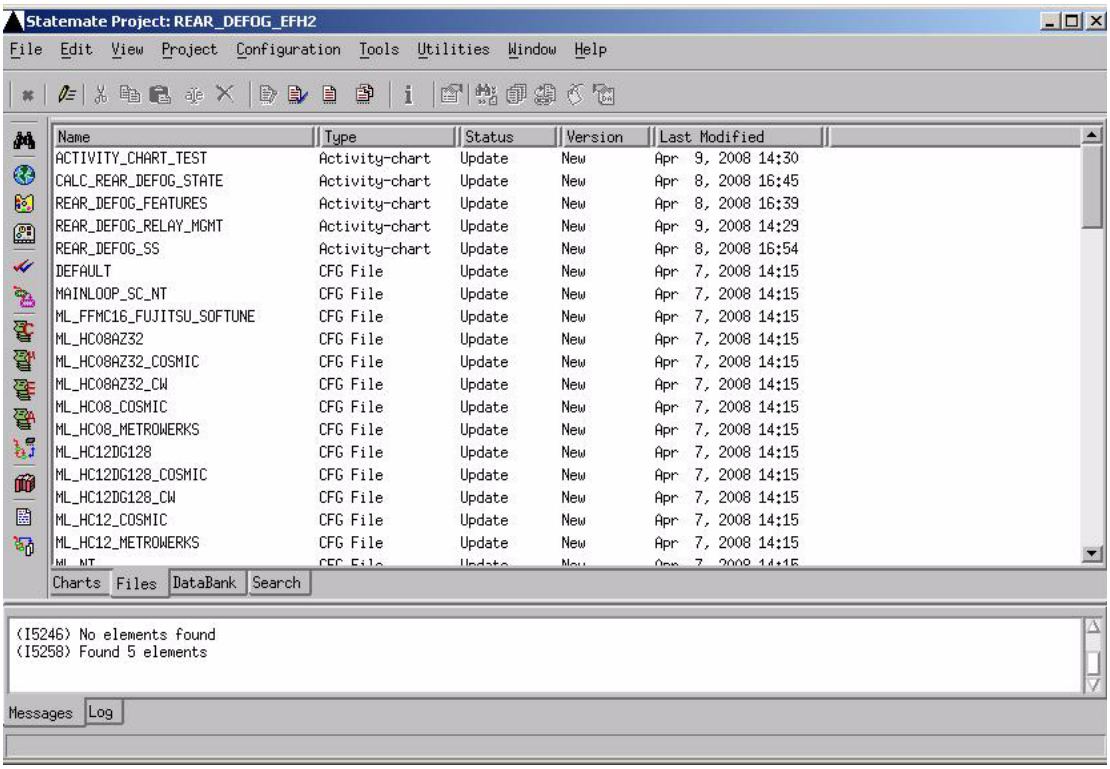
For more information about working with charts, see **Using the Graphic Editors**.



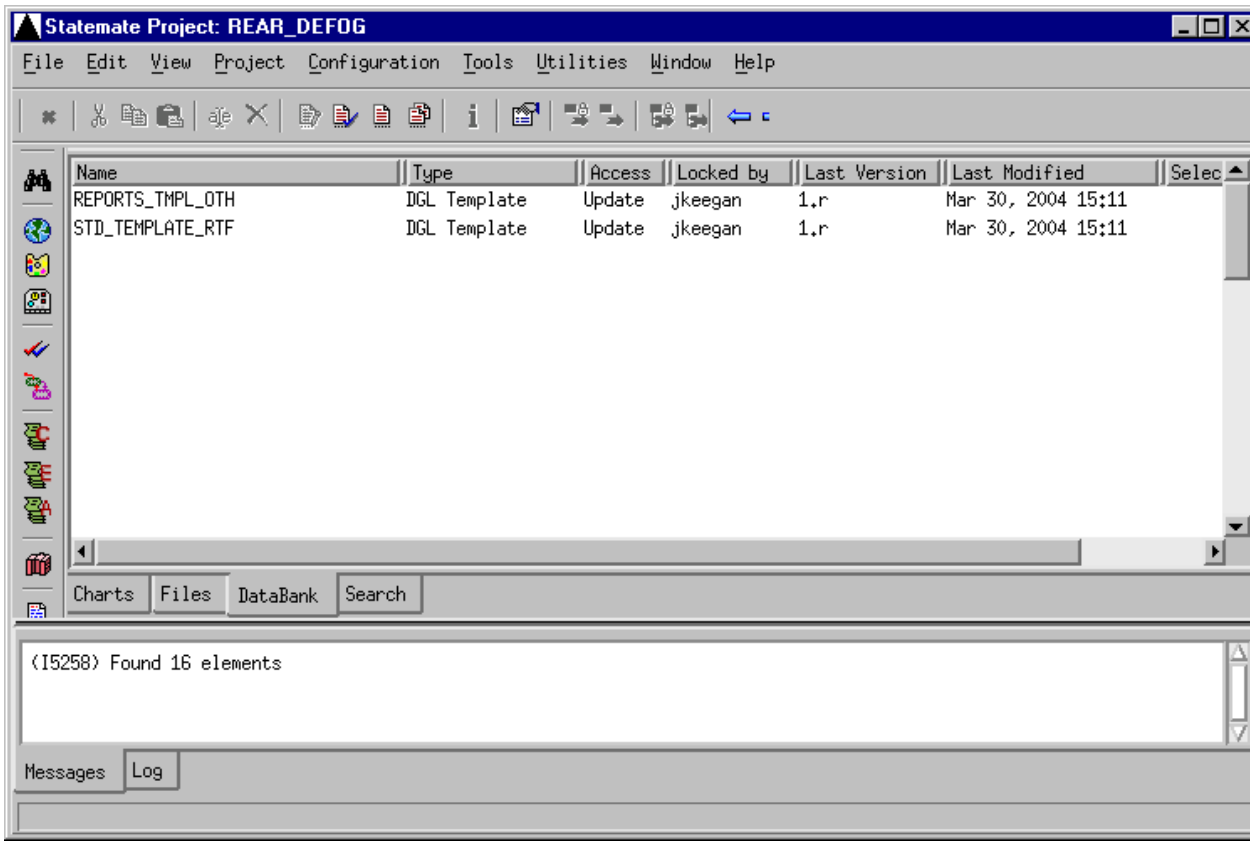
Files Tab

The Files tab displays details of all the files in a project sorted by the file name, type, mode, version number, modification status, and date modified.

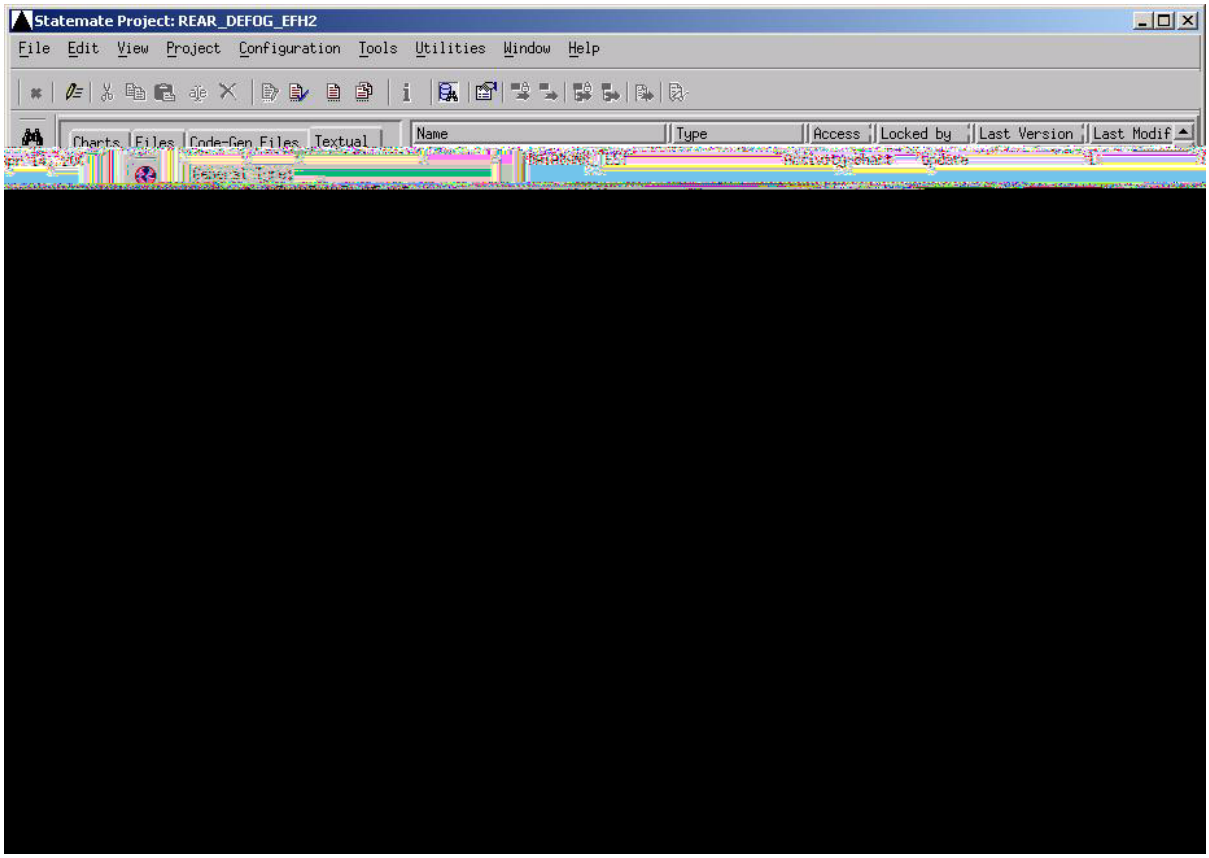
Databank Tab



The DataBank tab displays details of all the data elements in a project sorted by name, type, access status, current lock ownership, version number, and selected versions.


Statemate Project: REAR_DEFOG
File Edit View Project Configuration Tools Utilities Window Help
Name Type Access Locked by Last Version Last Modified Selec
REPORTS_TMPL_OTH DGL Template Update jkeegan 1.r Mar 30, 2004 15:11
STD_TEMPLATE_RTF DGL Template Update jkeegan 1.r Mar 30, 2004 15:11
Charts Files DataBank Search
(I5258) Found 16 elements
Messages Log

To enable the search feature, click . The following figure displays the DataBank with the Search feature enabled:



These tabs allow you to enter a **Name Pattern** using alpha-numeric characters, the underscore, and wildcards.There are two search options, **New List** and **Append to List**. Click **Search Now** to begin the search. Click **Stop Search** to end the search.

Note

Wildcards in the **Name Pattern** field are not allowed in the Textual tab.

The following table lists the features of each Databank search tab.

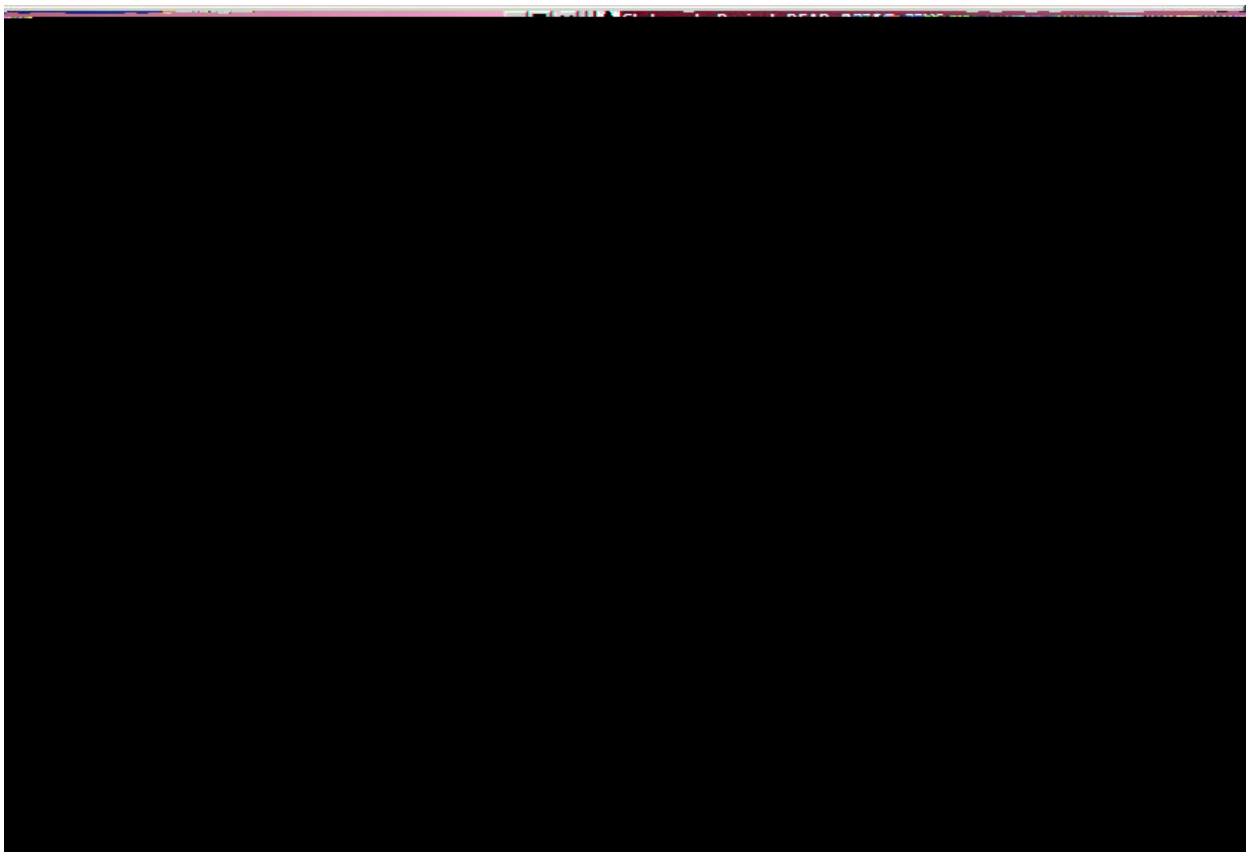
Tab	Options
Charts	Select the type of chart you want to search for: • Statecharts • Activity-charts • Use-Case Diagrams • Sequence Diagrams • Flowcharts • Module-charts • Global Definition Sets • All Charts
Files	Select the files from the following options: • Analysis Profiles • Monitor Files • SCL & Test Files • Waveform Profiles • Status Files • Check Model Profiles Panels • Component Configuration Files • Plugin Files • DGL Templates • Include Files • Configuration Files • All Files
Code-Gen Files	Select the Code-Gen files from the following options: • Target Files • Card Files • Makefiles • OIL Files • CFG Files • Source(c) Files • Source(h) Files • Rational Statemate Prototype Comp. Profiles • Rapid Prototype Comp. Files • Rational Statemate MicroC Comp. Profiles

Tab	Options
Textual	Select the type of elements to search for in the databank: • **General Type** - Select from the pull-down list. Options are All, Textual, or Graphical • **Type** - Select the Type from the pull-down list. Note that this option is disabled if All is selected for the General Type. Options for Textual Type are All, Data-Item, UserDefType, Condition, Event, Action, Info-flow, Subroutine, Field. Options for Graphical Type are All, State, Activity, Module, Actor, Use-Case, Boundary Box, Charts instance. • **Search In** - Options are All charts, Selected Charts, Included In Configuration Files. • **Match** - Options are Complete or Partial (complete or partial match to the “Name Pattern”)

Search Tab

The Search tab enables you to perform a variety of search operations on project elements.

For more information on searching for element properties, see **Searching for Elements**.



Messages Tab

Right-click Menus

Two right-click menus are defined for items in the main window tabs. Right-click an item to display the menu. The following figures and tables describe these menus.

Note

The left pane menu is available for the Charts and Files tab only. Not all functions are available for both tabs. The Charts tab version is shown.

Left Pane Menu

Menu Item	Description
New/Open	Opens a new or existing Chart, GDS, Profile or Panel.
Edit	Opens the selected item for editing.
Show in Activity Interface Browser	Enables you to browse the model and navigate through the information flows across the hierarchy of the chart. For more information, see **Activity Interface Browser**.
Rename	Opens the Rename <xx> dialog box, enabling you to rename the selected item.
Copy	Places a copy of the selected items on the clipboard.
Paste	Copies from the clipboard to the selected item.

Menus

Menu	Options
File Menu	• **New Window** - Opens another Activity Interface Browser. • **Close** - Closes the Activity Interface Browser
View Menu	• **Refresh** - Refreshes the Activity Interface Browser information. • **Expand All/Collapse All** - Expands or collapses the tree structure.
Tools Menu	• **Show Interface** - Show interface of selected activity • **Show in Model** - Opens the Graphical Editor. • **Properties** - Opens the Properties dialog box. • **Properties with Content** - Opens the properties dialog for the selected activity and it's interface elements • **Info** - displays version informations, date, mode, created by, and created on information. • **Textual Reports->Local Interface Report -** Show the selected activity's input/output signals and their source/target activities inside current chart. • **Textual Reports->Global Interface Report (Activities) -** Show the selected activity's input/output signals and their source/target activities throughout model hierarchy, sorted by the source/target activities names. • **Textual Reports->Global Interface Report (Elements) -** Show the selected activity's input/output signals and their source/target activities throughout model hierarchy, sorted by signal name. • **Options** - Select Graphical Interface or Functional Interface. • **Options->Show Source/Target in LCA chart** - Setting to display the basic source/target activity or its ancestor instance in the Least Common Ancestor chart

Tool Bar

Icon	Function
	Displays interface of selected activity
	Opens the Graphical Editor.
	Opens the Properties dialog box.
	Displays version informations, date, mode, created by, and created on information.
	Displays previous/next interface list.

Working Environment

This section describes the Rational Statemate working environment. The topics are as follows:

- **Projects**
- **Workareas**
- **Output Devices**
- **Preferences**
- **Configuration Management**
- **Database Diagnostics**
- **Plugins**

Projects contain workareas for each member of the project and each project (and its various charts and components) can be directed to a variety of output devices for distribution.

Additionally, users and project managers can set preferences for Rational Statemate and the various tools and utilities, and store elements in a databank.

Projects

A *project* is the main unit for organizing work in Rational Statemate. A project consists of data and users who can access that data. In general, project members have access to all data in a project, but any (or all) specific elements can be protected from write or read access.

A project includes the following:

- Name
- Manager
- Databank
- CM Tool
- RT project (optional)
- Libraries containing components (optional)
- Members

Note

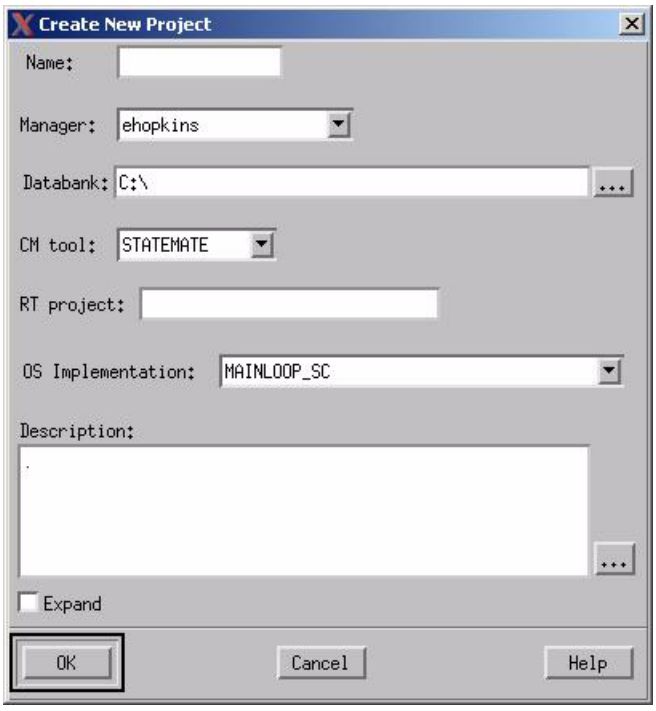
A project is intended to be accessed by multiple users. A workarea is intended to be accessed by a single user.

Creating a Project

To create a project:

1. From the Rational Statemate main window, select **File > New Project** or **Project > Project Management**, and then click **New**.

   The **Create New Project** dialog box displays.



Note: Check **Expand** to view Libraries and Project Members fields.

2. Specify the following values:

   - **Name** - Enter a name for the project.

     Project names must be unique at your site. Project names must begin with a letter, and can only consist of letters, numbers, and underscores (_). Lowercase letters are automatically converted to uppercase. Names in Rational Statemate, including project names, are not case sensitive and cannot contain spaces. A Rational Statemate project name can be up to 64 characters.

   - Do not use Caps Lock.
   - **Manager** - Enter the name of the project manager. You can select a name from the pull-down list.

     Each project can only have one project manager associated with it. By default, the project manager is also a project member. The project manager does not need to be an Rational Statemate administrator.

   - **Databank** - Select an area on a shared drive to hold the project files. Click **...** to browse for a directory.
   - **CM Tool** - Select a configuration management utility. Rational Statemate is selected by default though several widely-used, third-party CM tools are also supported. If you prefer a different tool, templates are available so that you can create your own script-based interface to your tool of choice. For more information, contact your sales representative.
   - **RT Project** (optional) - Enter the name of the Rational DOORS project, if the project uses Rational DOORS. For more information on Rational DOORS, see **Rational DOORS RT Interface**.
   - **OS Implementation** - Select the Operating System implementation from the pull-down list.
   - **Description** - Provide a brief description of the project for future reference. The description is intended primarily for libraries.
   - **Defined As Library** (optional) - If this option is selected, the project will be defined as a library, enabling its contents to be shared by other projects.
   - **Insert Libraries** (optional) - To include libraries in the project, click **Insert** to the right of the Libraries area and browse for libraries to include.
   - **Insert Project Members** (optional) - To add members to the project, click **Insert** to the right of the Project Members area. The Select member dialog box displays. Select the members to add. The application lists all the user accounts on the current system.

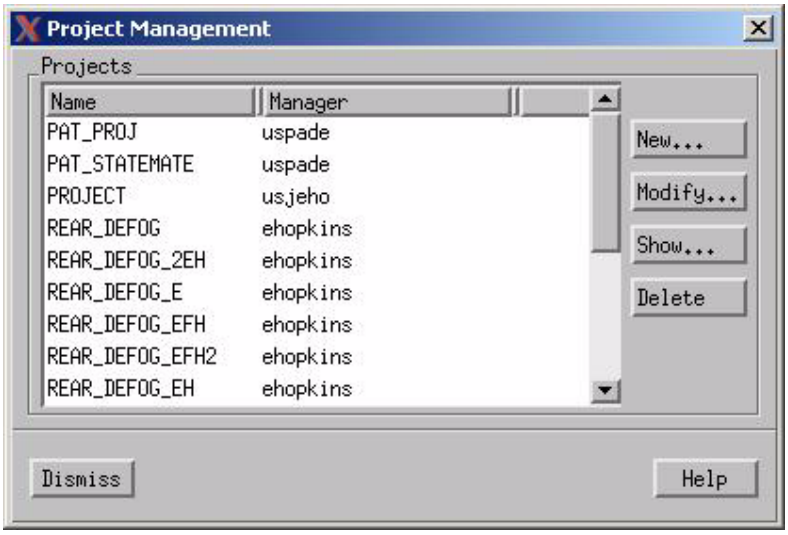
   Note: To view Defined as Library, Insert Libraries, and Insert Project Members, you must check the Extend box.

3. Click **OK** to create the project.

Modifying a Project

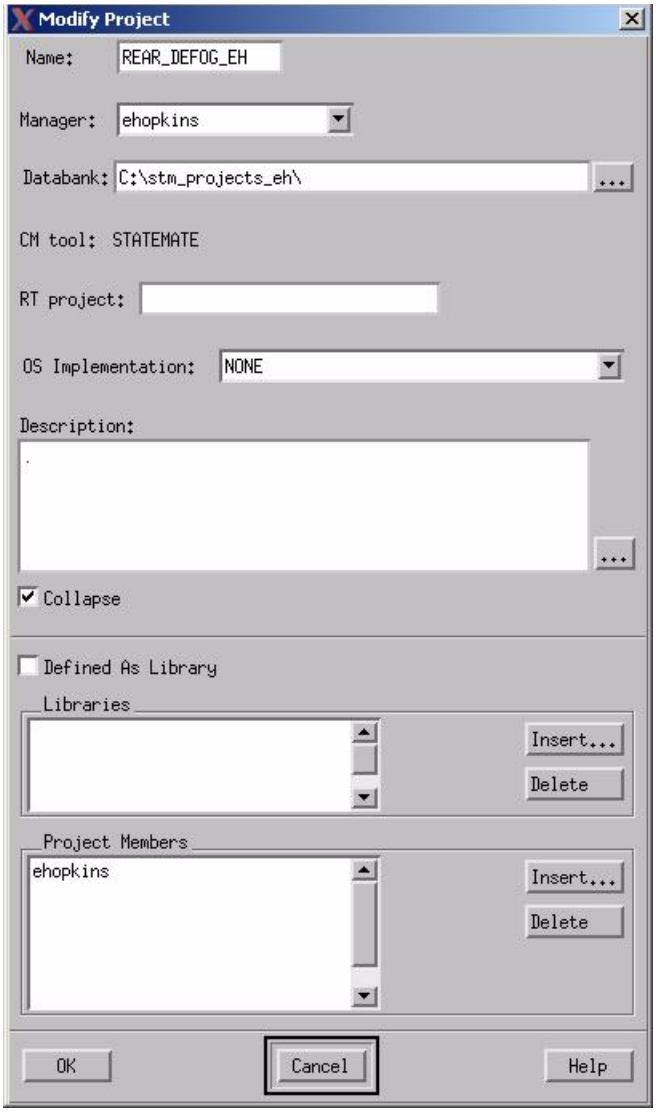
To modify a project:

1. From the Rational Statemate main window, select **Project > Project Management**. The **Project Management** dialog box displays a list of all projects and their corresponding managers.



2. Select the project to be modified.

3. Click **Modify**. The Modify Project dialog box displays. The **Modify Project** dialog box displays (shown expanded).



4. Make the necessary modifications (described in the table in **Creating a Project**, Step 2).

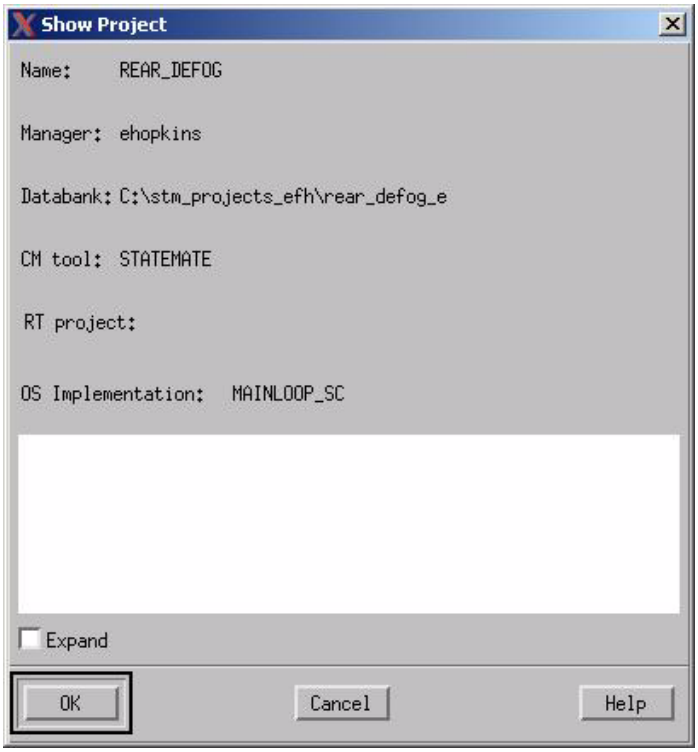
5. Click **OK**.

Displaying Project Settings

To display the settings for a project:

1. From the Rational Statemate main window, select **Project** > **Project Management**. The Project Management dialog box displays.
2. Select the project to display and click **Show**. The **Show Project** dialog box displays.

Note: Check **Expand** to view the Libraries and Project Members fields.

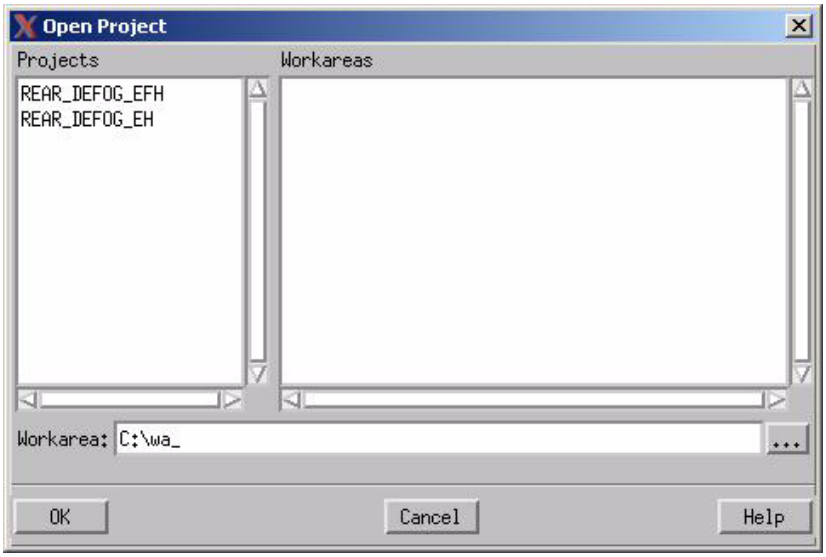


Opening a Project

To open a project:

1. From the Rational Statemate main window, select **File > Open Project**. The **Open Project** dialog box displays.

   Note: You can only open one project at a time.



2. Select the project to open in the Projects panel.

   Note: A list of workareas is displayed for the selected project in the Workareas panel. More than one workarea can be associated with a project.

3. Double-click the workarea you want to use or create a new workarea.

4. Click **OK**. The **Open Project** dialog box closes and the main window is redisplayed with the selected project.

Deleting a Project

Note

- To delete a project, you must be the project manager and the project must be closed.
- When you delete a project, only the reference to the project is deleted. The databank, files, and workareas connected to the project are not deleted.

To delete a project:

1. From the Rational Statemate main window, select **Project** > **Project Management**.
2. Select the project to delete in the Project Management dialog box.
3. Click **Delete**. Rational Statemate prompts you to confirm the deletion.
4. Click **Yes** to confirm your choice.

Closing a Project

To close a project, from Rational Statemate main window, select **File** > **Close Project**.

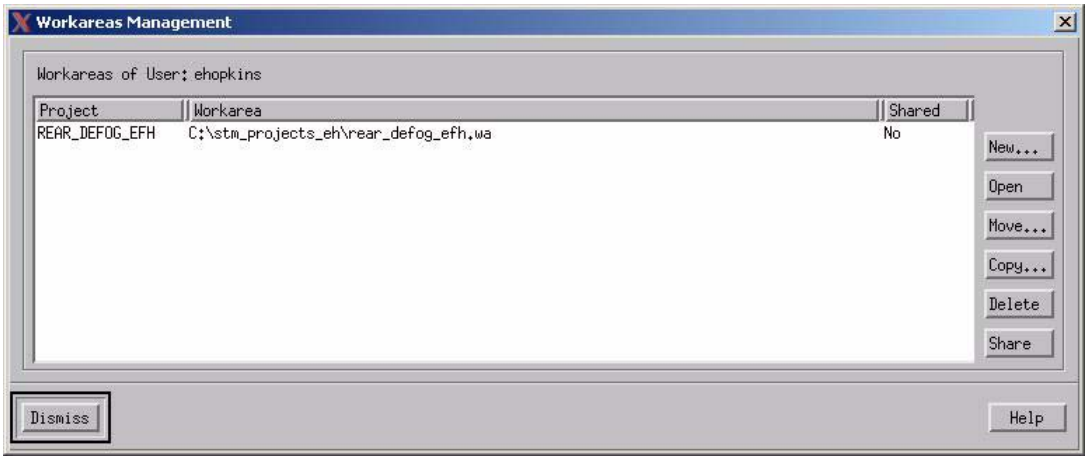
Workareas

A *workarea* is private directory structure associated with a user and a project. This workarea enables each project member to work independently. You can design and redesign, without making irrevocable changes to the current working or released design. As you rework your charts, modifications are made within your workarea.

Creating a Workarea

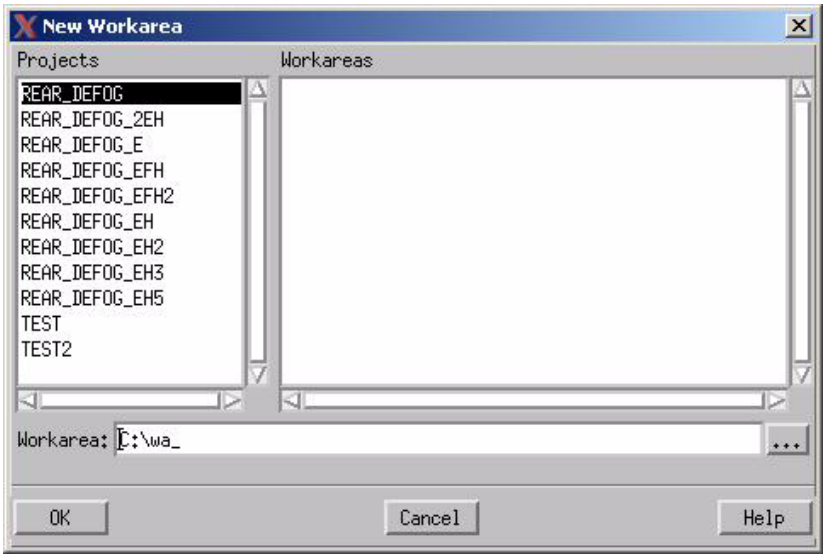
To create a new workarea:

1. From the Rational Statemate main window, select **Project** > **Workarea Management.** The **Workareas Management** dialog box displays.



2. Click **New**. The **New Workarea** dialog box displays. A list of projects and their corresponding workareas displays.

   Note: If another workarea is already active, this option is disabled.



3. In the **Projects** list, select a project with which to associate a new workarea.

4. In the **Workarea** box, enter the name and absolute path of the new workarea, or click ... to browse to a directory.

   Note: The absolute path to the directory where you want the workarea to reside must already exist; Rational Statemate does not create the absolute path.

5. Click **OK**.

Opening a Workarea

To open a workarea:

1. From the Rational Statemate main window, select **Project > Workarea Management.**
2. In the **Workareas Management** dialog box, select the project and workarea to be opened.
3. Click **Open**.

Note: If another workarea is already active, this option is disabled.

The **Workarea Management** dialog box closes and the main window is redisplayed with the new workarea.

When the **Open Last Wa Used** preference is set to yes, the tool is opened at the last visited project and workarea.

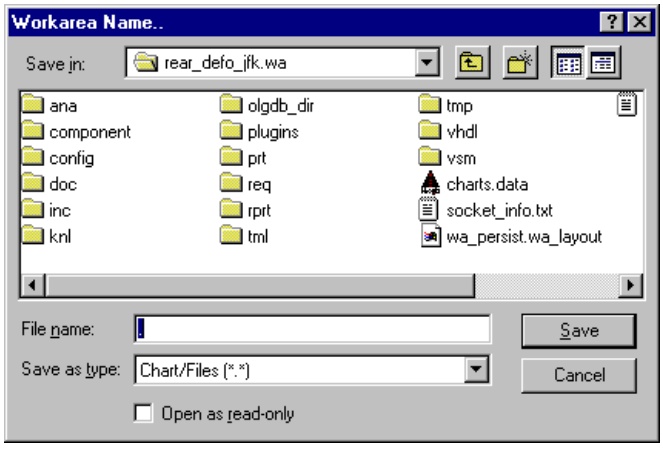
Moving a Workarea

To move a workarea:

1. From the Rational Statemate main window, select **Project > Workarea Management**.
2. In the **Workareas Management** dialog box, select the project and workarea to be moved.
3. Click **Move**.

Note: If another workarea is already active, the **Move** button is disabled.

The **Workarea Name** dialog box displays. It displays the existing directory structure. The current location of the workarea displays in the **Save In** box.



4. Select a new location to move the workarea, if appropriate.

   Note: The name of the workarea displays in the **File Name** box.

5. Modify the file name, as needed.

6. Click **Save**.

Copying a Workarea

To copy a workarea:

1. From the Rational Statemate main window, select **Project > Workarea Management**.

2. In the **Workareas Management** dialog box, select the project and workarea to be copied.

3. Click **Copy**. The **Workarea Name** dialog box displays. It displays the existing directory structure. The current location of the workarea displays in the **Save In** text box.

   Note: If another workarea is already active, the **Copy** button is disabled.

4. Select a new location to copy the workarea, if appropriate.

   Note: The name of the workarea displays in the **File Name** box.

5. Modify the file name, as needed.

6. Click **Save**.

Reducing the Workarea

To improve the workflow when dealing with reduce charts and filtered check outs, you may quickly eliminate unnecessary items and, thus, reduce the size of the workarea with the Reduce Workarea feature. This feature supports interactive work by allowing the users to work with only the minimal necessary definitions.

The Reduce Workarea operation performs this clean-up with one selection and eliminates the time-consuming process of deleting unused elements manually. To reduce the definitions in the workarea automatically, in the **Charts** view, select the **Configuration** > **Reduce Workarea**.

This operation then collects and automatically deletes the following from the workarea:

- All unused elements set as read only
- **Reduced GDS** elements (read-only)
- Activity charts

Deleting a Workarea

To delete a workarea:

1. Close the workarea, if it is open.
2. From the Rational Statemate main window, select **Project** > **Workarea Management**.
3. From the list of projects, select the project that is associated with the workarea you want to delete.
4. Select the workarea to be deleted.
5. Click **Delete**. A Question dialog box displays. Click **Yes** to confirm the deletion.

Note: If the workarea is used to store files that are not Rational Statemate-configurable items (for example, user-written code), make sure these files are moved to a safe place before you delete a workarea.

Sharing the Workarea

If a team wants to view the complete model, without each user having to check-out all charts and files into his own personal workarea, they should use a shared workarea. A shared workarea is a workarea that is available for all team members to examine in read-only mode.

To create a shared workarea:

1. One of the project members creates a new workarea, and checks-out the full model in read-only mode.
2. The owner of the new workarea clicks the **Share/UnShare** toggle button to make the workarea shared (the text on the button changes according to the toggle selection).

Note

When no workarea is selected, the Share/UnShare toggle button is dimmed, so be certain to select a workarea before clicking the toggle.

The owner of a shared workarea sees it displayed as a regular workarea while the other team members view it as a shared workarea and in read-only mode. The other team members see it listed in the Project Workareas list, but, of course, they cannot modify it.

When working in a shared workarea, the following operations are not allowed (for users other than the owner):

- Check out files
- Execute configuration.
- Modify charts or files

Sharing and Locking

To store your changes permanently and to enable others to share them, save the modified charts to the databank. Through a locking mechanism, you are ensured that your work will not conflict with that of the other project members.

A user can have multiple workareas associated with any project, but any one workarea can only be associated with one project.

To share or stop sharing a Workarea, select **Project > Workarea Management** and then select **Share** or **UnShare**.

Temporary Workarea Directory

Users can set temporary workarea directories for their Rational Statemate work. Edit the `STM_TMP_DIR` environment variable for the location of the temporary directory.

Output Devices

Output devices, printers and plotters, for example, are used to obtain hardcopy legacy documents for projects and their components. The following sections explain how to configure and use output devices in Rational Statemate.

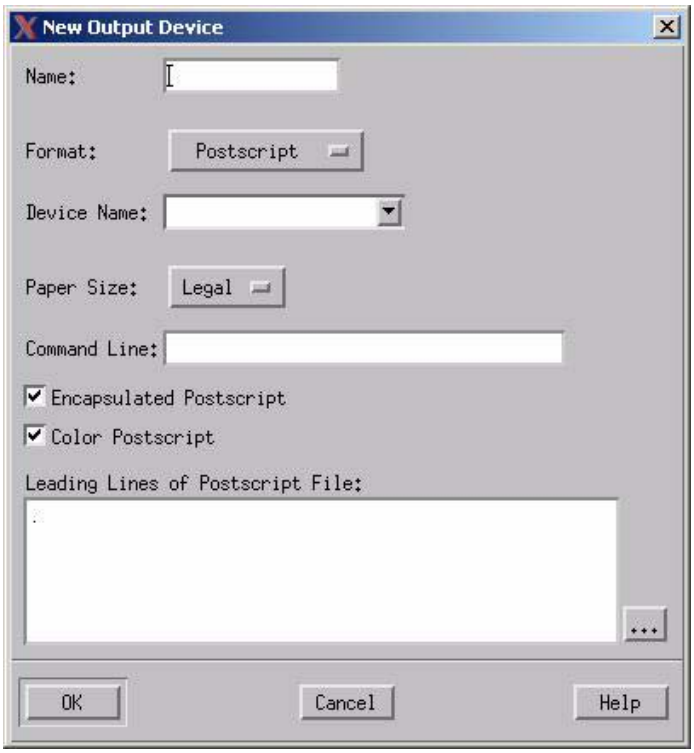
Setting Up Output Devices

To set up output devices:

1. From the Rational Statemate main window, select **Utilities > Output Devices**. The **Output Devices** dialog box displays.

   The **Output Devices** dialog box displays a list of all current plotters and printers on your system with their corresponding formats.

2. Click **New**. The **New Output Device** dialog box displays.



3. Specify the following values:

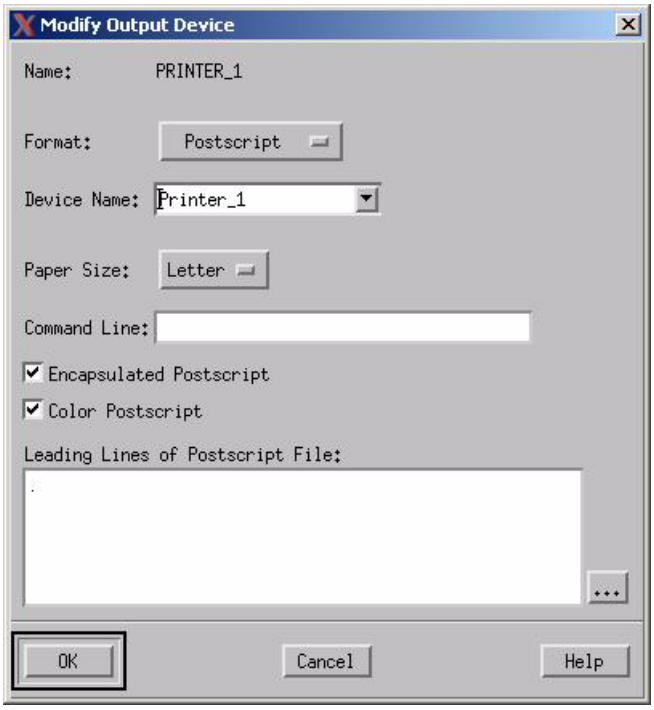
Field	Description
Name	Enter a name for the output device.
Format	Select the format for the output device. The default is **PostScript**.
Device Name	Select a name for the output device.
Paper Size	Select the correct paper size for the output device.
Command Line	Enter any special commands that need to be sent to the output device.
Encapsulated and Color PostScript	Select to enable (default) or clear to disable.
Leading Lines of Postscript file	Enter any leading lines that the PostScript may require. Click to use a text editor

4. Click **OK**.

Modifying Output Devices

To modify output devices:

1. From the Rational Statemate main window, select **Utilities** > **Output Devices**.
2. In the **Output Devices** dialog box, select an output device from the list of devices in the left frame.
3. Click **Modify**. The **Modify Devices** dialog box displays.



4. Modify the settings for the output device (for more information, see the table in **Setting Up Output Devices**, Step 3).
5. Click **OK**.

Deleting Output Devices

To delete output devices:

1. From the Rational Statemate main window select **Utilities > Output Devices.**
2. In the Output Devices window, select an output device from the list of devices in the left frame.
3. Click **Delete** to delete the selected output device. Click **Yes** to confirm your selections.

Preferences

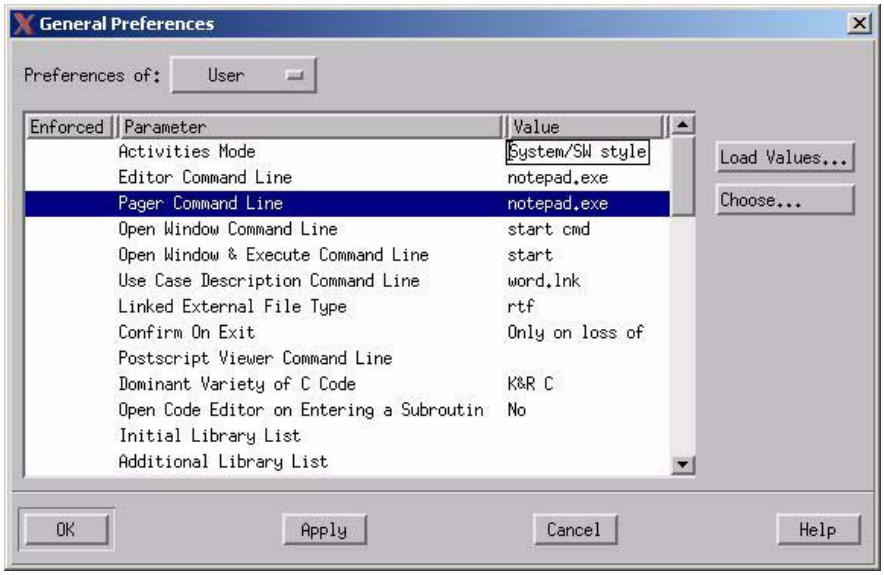
You can set *preferences* for Rational Statemate, as well as all editors and utilities that Rational Statemate supports. You can set preferences individually or by loading pre-established settings.

Project managers and system managers can set preferences that are enforced for users at your site. When you set your own preferences in a specific area, you can obtain information on which preferences have been enforced at your site.

Setting General Rational Statemate Preferences

To set general preferences for Rational Statemate:

1. From the Rational Statemate main window, select **Project > General Preferences**. The **General Preferences** dialog box displays.



2. Using this window, you can do the following:

   - Use the **Choose** option to specify where you want the preferences applied.
   - Use the **User** option to specify who can have access to these preferences. This option is the **Group** option if you are an administrator.
   - Use the **Load Values** option to load predefined preferences.
   - Set preferences for individual parameters.

Specifying Where Preferences are Applied

To determine where you want the preferences applied, select from the following options on the **Preferences of** pull-down menu:

- **System** - System-wide preferences. Only the system manager can set preferences applicable to all users at a site. Users, including project managers, cannot override these settings.
- **Project** - Preferences for a specific project, which you select from a list. Only the project manager can set preferences applicable to all members of a project. Other project members cannot override these settings.
- **User** - Preferences for a specific user, which you select from a list.
- **Default** - Rational Statemate default preferences.
- **Current** - Currently selected preferences.

Note: You might not have access to all the preference options. For example, the system administrator can reserve the right to change the System preferences.

Specifying Access to Preferences

If you are the project manager, you can use the **Group** button to determine who can have access to these preferences. (The **Group** button is available only if you are the project manager who set up this project in the databank.)

Preferences Descriptions

Parameter	Description
Preferences Of:	• **System** - Tool-wide preferences • **Project** - Set by project manager • **User** - Personal options • **Default** - Software defaults • **Current** - What is set in current chart
Enforced	Indicates which Preference is enforced.
Parameter	Preference
Value	Value of Preference
Load Values	Allows you to upload values from other preferences
Choose	Allows you to select a value

Note

This section does not provide in-depth details of all available preferences. Some preferences have a pull-down list in the value field. Some preference values are defaults. Other values can be entered by the user by clicking in the value field and entering applicable text.

General Preferences

This section provides an overview of the General Preferences dialog box.

Parameter	Value	Description
Editor Command Line	<command line>	Editor for properties editing
Pager Command Line	<command line>	Editor for properties viewing
Open Window Command Line	<command line>	Command to open window
Open Window & Execute Command Line	<command line>	Command line to open window and execute command
Use Case Description Command Line	<command line>	Command line to open use-case description editor
Linked External File Type	<file extension>	Extension of linked External Files and use-case description files
Confirm On Exit	Always / Only on loss of information	Confirm exit when closing tool without save
Postscript Viewer Command Line	<command line>	Command line to open postscript editor
Dominant Variety of C Code	K&R C / ANSI C	Define selection order of subroutine implementation
Open Code Editor on Entering a Subroutine	Yes / No	Open code editor when opening subroutine properties
Initial Library List	<Component library list>	Initial Component libraries included (comma separated list)
Additional Library List	<Component library list>	Additional Component libraries included (comma separated list)
Truth Table Execution Mode	Upon Change / Every Step	Execute truth-tables every step or only when upon input change
Show long description of components	Yes / No	Display component long-description in Info view
Icon Editor	<command line>	Command line to open component icon editor
Produce WORD format for	Word7.0 / Word97	Word version to support in generated reports
Default Documentor Template	<documentor template>	Default documentor template

Default ASCII Device	<Device name>	Default device for ASCII printout
Default Postscript Device	<Device name>	Default device for Postscript printout
Plot Title and Header Font Name	<Font name>	Plot title and header font
Plot Title and Header Font Size	<Font size>	Plot title and header font Size
Plot Orientation	Landscape / Portrait	Plot Orientation
Plot Labels Presentation	Yes / No	Present labels in chart plots
Plot Names Presentation	Yes / No	Present names in chart plots
Plot Notes Presentation	Yes / No	Present notes in chart plots
Plot Headers Presentation	Top / Bottom / None	Header presentation in chart plots
Plot Chart Name and Date Presentation	Top / Bottom / None	Chart name and date presentation in plots
Configuration Management Tool	<CM tool name>	Default Configuration Management tool
External Tool Script	<command line>	Command line to invoke external tool script
Generate Template for External Tool Implementation	Yes / No	Generate template for External Tool implementation of Subroutine
Data Bank Default Directory	<directory path>	Default directory for new Databank
Workarea Default Directory	<directory path>	Default directory for new workarea
Export/Backup Default Directory	<directory path>	Default directory for Export/Backup operations
User Plugins Directory	<directory path>	Plugin files directory
Script to pack	<command line>	Script to pack exported files
Script to unpack	<command line>	Script to unpack file during import
Exact Case Mode	off / on	Support exact case for textual element names
Strict External-Activity Resolution	Yes / No	Resolve External Activities only to parent chart
Track Changes While Editing	Yes / No	Add track comment while modifying model
Add Change Description when Check-In Chart	Yes / No	Add track comment when checking-in charts
Open Last Wa Used	Yes / No	Open last used workarea on startup
Last Wa Used	<directory path>	Path of last used workarea (automatically saved)

Confirm before Open Last WA Used	Yes / No	Confirm before opening last used workarea on startup
Exiting Reactions Upon Activity Stop	Executed in Code only / Not Executed	exiting' triggered static-reactions execution upon 'stop' of parent Activity
View Info Using External Pager	Yes / No	View Info using external pager
Beautify Indent Size	<integer value>	Indentation size of 'Beautify' operation
Quick Edit Startup Mode	disable / off / on	State of Quick Edit mode upon startup
Allow Flow-line Loopbacks through Routers	Yes / No	Allows flowlines from a basic activity back to itself through a Router
DiffMerge tool Command Line	<command line>	Command to invoke DiffMerge tool
Display Data-flows in 'Where Referenced' results	Yes / No	Include data-flows in elements 'Where Referenced' results

Preferences Management

This section provides a brief overview of the Preferences Management dialog boxes.

Statechart Graphic Editor Preferences

Parameter	Value	Description
SGE / Procedural Statechart Color	<color>	Chart default color
And States / Or States	<color>	States default color
<XXX> Line Width	<integer value>	Line type default width
Names	<color>	Names default color
<XXX> Font	<Font name>	Element type default font
Transitions Style	Splines / Straight Lines	Transitions line style
Transitions / Default Transitions	<color>	Transitions / Default Transitions color
Labels	<color>	Labels default color
And Lines	<color>	And lines default color
<XXX> Connectors	<color>	Connector type default color
Connector / Diagram Connectors Text	<color>	Connectors / Diagram Connectors text default color
Notes	<color>	Notes default color
Combinational Assign.	<color>	Combinational Assign. default color
<XXX> Highlight Line	<color>	Element type Highlight Line default color
<XXX> Fill	<color>	Box type Fill default color
Filled Boxes	On / Off	Boxes filled by default
Auto Box Height	<Numerical value>	Auto box default Height
Auto Box Width	<Numerical value>	Auto box default width

Auto Box Name Prefix	<Alphanumerical value>	Auto box default name prefix

Activity-Chart Graphic Editor Preferences

Age Specific Preferences

Parameter	Value	Description
AGE Color	<color>	Chart default color
<XXX> Activities	<color>	Activity type's default color
<XXX> Line Width	<integer value>	Line type default width
Data Stores	<color>	Data Stores default color
Names	<color>	Names default color
<XXX> Font	<Font name>	Element type default font
Flow-Lines Style	Splines / Straight Lines	Flow-Lines style
Data / Control Flow-Lines	<color>	Flow-Lines color
Labels	<color>	Labels default color
<XXX> Connectors	<color>	Connector type's default color
Connector / Diagram Connectors Text	<color>	Connectors / Diagram Connectors text default color
Notes	<color>	Notes default color
Combinational Assign.	<color>	Combinational Assign. default color
Active / Suspended Activity Highlight Line	<color>	Activity Highlight Line default color
Active / Suspended Activity Highlight Fill	<color>	Activity Highlight fill default color
Filled Boxes	On / Off	Boxes filled by default
<XXX> Fill	<color>	Box type Fill default color
Auto Box Height	<Numerical value>	Auto box default Height
Auto Box Width	<Numerical value>	Auto box default width

Auto <XXX> Name Prefix	<Alphanumerical value>	Auto box type's default name prefix
Allow editing of Instance Component	Yes / No	Allow resize of component instance box
Routers / External Routers	<color>	Router type's default color
Routers / External Routers Fill Style	Solid / No Fill	Router type's fill style
Routers / External Routers <XXX> Line Style	Solid / Dashed	Router type's line style
Routers / External Routers Name Orientation	Vertical / Horizontal	Router type's default name orientation

Component Preferences

Parameter	Value	Description
Distance between stubs	<Numerical value>	Default distance between stubs
Minimal Component Height	<Numerical value>	Minimal component Height
Minimal Component Width	<Numerical value>	Minimal component width
Keep proportions of top level activity	Yes / No	Keep proportions of top level activity
Component	<color>	Component default color
Component Line Width	<integer value>	Component line default width
Name	<color>	Names default color
Name Font	<Font name>	Names default font
In/Out Stubs	<color>	Stubs default color
In/Out Stubs Font	<Font name>	Stubs default font
Information Flow In/Out Stubs	<color>	In/Out stubs information-flow color
Information Flow In/Out Stubs Font	<Font name>	In/Out stubs information-flow font
Notes	<color>	Notes default color
Notes Font	<Font name>	Notes default font

Module-chart Preferences

Parameter	Value	Description
MGE Color	<color>	Chart default color
Subsystem Modules	<color>	Subsystem Modules default color
<XXX> Line Width	<integer value>	Line type default width
Storage Modules	<color>	Storage Modules default color
Environment Modules	<color>	Environment Modules default color
Names	<color>	Names default color
<XXX> Font	<Font name>	Element type's default font
Flow-Lines Style	Rectilinear Lines, Straight Lines	Flow-Lines style
Flow-Lines	<color>	Flow-Lines default color
Labels	<color>	Labels default color
<XXX> Connectors	<color>	Connector type's default color
Connector / Diagram Connectors Text	<color>	Connector type's default text color
Notes	<color>	Notes default color
Filled Boxes	On / Off	Boxes filled by default
Level <n> Fill	<color>	Level n default fill color
Auto Box Height	<Numerical value>	Auto box default Height
Auto Box Width	<Numerical value>	Auto box default width
Auto Box Name Prefix	<Alphanumerical value>	Auto box default name prefix

Flowchart Graphic Editor Preferences

Parameter	Value	Description
LGE Color	<color>	Chart default color
<XXX> Boxes	<color>	Box type's default color
<XXX> Line Width	<integer value>	Line type default width
Actions	<color>	Actions default color
<XXX> Font	<Font name>	Element type's default font
Names	<color>	Names default color
Decisions	<color>	Decisions default color
Switch Expression	<color>	Switch Expression default color
Arrows Style	Splines / Straight Lines	Arrows line style
Arrows	<color>	Arrows default color
Labels	<color>	Labels default color
Start Arrows	<color>	Start Arrows default color
<XXX> Connectors	<color>	Connector type's default color
Connector / Diagram Connectors Text	<color>	Connectors text default color
Notes	<color>	Notes default color
<XXX> Highlight Line	<color>	Element type's highlight line default color
<XXX> Highlight Fill	<color>	Element type's highlight fill default color
Filled Boxes	On / Off	Boxes filled by default
Level <n> Fill	<color>	Level n default fill color
Auto Box Height	<Numerical value>	Auto box default Height
Auto Box Width	<Numerical value>	Auto box default width
Auto <XXX> Prefix	<Alphanumerical value>	Auto box default name prefix

Sequence Diagram Graphic Editor Preferences

Parameter	Value	Description
QGE Color	<color>	Chart default color
Names	<color>	Names default color
<XXX> Font	<Font name>	Element type's default font
<XXX> Line	<color>	Line type default color
<XXX> Line Width	<integer value>	Line type default width
Reference SD	<color>	Reference SD default color
Simple Message	<color>	Simple Message default color
Timing Constraint	<color>	Timing Constraint default color
Timing Constraint Text	<color>	Timing Constraint text default color
Labels	<color>	Labels default color
Notes	<color>	Notes default color
Pagination Overview	Yes / No	Display pagination lines on chart
Show Header Lines	Yes / No	Display Lifeline names on every page of a scrolled down Sequence diagram
Scenario Numbering Postfix	<Alphanumerical value>	Scenario numbering postfix
Scenario Numbering Style	A / a / 1	Scenario numbering style
Start Scenario Numbering From	<integer value>	Start scenario numbering from
Show Scenario Numbering	Yes / No	Show scenario numbering
Pagination Orientation	Landscape / Portrait	Pagination orientation
Pagination Overlap Ratio	<integer percentage>	Pagination overlap area to be printed twice

Use-Case Diagram Graphic Editor Preferences

Parameter	Value	Description
UGE Color	<color>	Chart default color
Names	<color>	Names default color
<XXX> Font	<Font name>	Element type's default font
Use-Case	<color>	Use-Case default color
<XXX> Line Width	<integer value>	Line type default width
Actor	<color>	Actor default color
Boundary	<color>	Boundary default color
Association	<color>	Association default color
Relation	<color>	Relation default color
Labels	<color>	Labels default color
Notes	<color>	Notes default color
Auto Use-Case Box Height	<Numerical value>	Auto Use-Case box default Height
Auto Use-Case Box Width	<Numerical value>	Auto Use-Case box default width
<XXX> Name Prefix	<Alphanumerical value>	Auto element default name prefix

Panel Graphic Editor Preferences

Parameter	Value	Description
PGE Color	<color>	Panel default color
Push Buttons	<color>	Push Buttons default color
<XXX> Fill	<color>	Element type's default fill color
<XXX> Line Width	<integer value>	Element type's line default width
Lamps	<color>	Lamps default color
Choice Buttons	<color>	Choice Buttons default color
Radio Buttons	<color>	Radio Buttons default color
Meters	<color>	Meters default color
<XXX> Text	<color>	Element type's text default color
<XXX> Localizer	<color>	Element type's Localizer default color
Knobs	<color>	Knobs default color
Slides	<color>	Slides default color
Displays	<color>	Displays default color
Labels	<color>	Labels default color
<XXX> Font	<Font name>	Element type's default font
Textual Notes	<color>	Textual Notes default color
Polygons	<color>	Polygons default color
Circles	<color>	Circles default color
Arcs	<color>	Arcs default color
Polylines	<color>	Polylines default color

Graphic Editors Preferences

Parameter	Value	Description
Grid Display	On / Off	Display grid
Grid Spacing	<Numerical value>	Spacing between grid lines
Grid Every Nth Point	<integer value>	Display every nth grid line
Grid Color	<color>	Grid color
Grid Alignment	Aligned on grid / Not aligned on grid	Align drawing to grid
Grid in Read-Only Chart	Yes / No	Display grid in read-only chart
Trap Radius	<integer value>	Cursor distance to select element
Display Names	Full / Hidden	Display or hide names
Display Labels	Full / Hidden	Display or hide labels
Carriage Return is New Line	Yes / No	Interpret carriage return as new-line
Allow Cut & Paste Between GEs	Yes / No	Support copy&paste between graphic editors
Single-Button Mouse	Yes / No	Using Single-Button mouse
Enable Scale Text	Yes / No	Scale font size when resizing chart
Enable Reshaping	Yes / No	Reshape drawings when resizing chart
Double Click Box Default Action	Properties / Info	Default action for double-clicking a box
Double Click Instance/Off-page Default Action	Open/Create Sub Chart / Properties / Info	Default action for double-clicking an instance box
Double Click Arrow Default Action	Properties / Info	Default action for double-clicking an arrow
Double Click Chart Default Action	Chart Properties / Chart Info / Open Parent	Default action for double-clicking chart background

Databank Browser Preferences

Parameter	Value	Description
Purge to Version	<integer value>	Number of revisions to save during purge operation
Group Permission	Read/Write / Read only / None	Group permissions in Databank
Others Permission	Read/Write / Read only / None	Others permissions in Databank
Save Unmodified Items	Yes / No	Force checkin of un-modified items
Save external file to Databank	Yes / No	Save external file to Databank
Automatic Databank Refresh	Yes / No	Automatically refresh displayed information when selecting databank tab
Check Out in single transaction	Yes / No	Perform check-out operation in single database transaction
Display Search filter in Databank Tab	Yes / No	Include the Search pane in Databank tab
Preserve elements Ids in chart-file	Yes / No	Save element Id's in checked-in charts

Simulation Preferences

Parameter	Value	Description
Scope View	Tree view / Creation rules view	View profile scope as tree or as list
Steps per Go	<integer value>	Limit to the number of steps taken during a long Go command
Infinite Loop	<integer value>	Limit for the number of interactions of For or While loops
Goback Limit	<integer value>	Maximum number of times a go back can be executed
Racing Read/Write	Yes / No	Report Read/Write racing during execution
Racing Write/Write	Yes / No	Report Write/Write racing during execution
Time Model	Asynchronous / Synchronous	Set the time model
K&R Compiler	<compiler command>	Compiler for K&R C user-code
K&R Flags	<compiler flags>	Flags for K&R C user-code compilation
K&R Dynamic Flags	<compiler flags>	Flags for K&R C user-code dll link
Ansi-C compiler	<compiler command>	Compiler for ANSI-C user-code
Ansi-C Flags	<compiler flags>	Flags for ANSI-C user-code compilation
Ansi-C Dynamic Flags	<compiler flags>	Flags for ANSI-C user-code dll link
Link Flags	<linker flags>	Flags for user-code link
Dynamic Link Flags	<linker flags>	Flags for user-code dll link
Libraries	<libraries>	Libraries to link user-code with
Leave Animated Truth Table on Screen	Yes / No	Leave animated truth-table on screen after execution
Have Access to Component Elements	Yes / No	Allow access to library component elements during Simulation
Test Files Extension	<file extension>	Extension of generated test-files

Trigger Evaluation	Upon Change / Every Step	Evaluation mode for Subroutines in action triggers
Max Visible Size Of Element Name	<integer value>	Maximum name length to display in Simulation monitors
Update Monitors During Long Go's	Yes / No	Update monitors every step during long Go's
Show Element Mode in Monitor	Yes / No	Show element mode (in/out/inout/local) in monitor

Properties Preferences

Parameter	Value	Description
Truth Tables, Number of Inputs	<integer value>	Number of input columns in new truth-table
Truth Tables, Number of Outputs	<integer value>	Number of output columns in new truth-table
Truth Tables, Color of 'Inputs' Section	<color>	Color of input columns in new truth-table
Truth Tables, Color of 'Outputs' Section	<color>	Color of output columns in new truth-table
Truth Tables, Color of 'Actions' Section	<color>	Color of Actions columns in new truth-table
Truth Tables, Color of Default Row	<color>	Color of default row in new truth-table
Truth Tables, With Default Row	Yes / No	Create default row in new truth-table
Truth Tables, With Action Section	Yes / No	Create Actions section in new truth-table
Open New Editor for References	Yes / No	Show elements of 'Open References' in new properties window
Append elements from GE to list	Yes / No	Display all Open-Properties from GE in single properties window
Long Description Editor Command Line	<command line>	Command line to open long-description editor
Action Language Editor Command Line	<command line>	Command line to open action-language editor
User Code Editor Command Line	<command line>	Command line to open user-code editor
Attribute Definitions Directory	<directory path>	Location of attribute definition files
Open References' based on 'Selected Implementation'	Yes / No	Open References' will regard only the Selected Implementation's references

Modify Elements Defined in a Read-Only Chart	Yes / No	Modify read-only chart elements without confirmation
Build Internal Data Structures On Popup	Yes / No	Build internal data-structures for future use when opening workarea
Use Internal Editor for Mini-Spec, Long-Description, Static Reaction, Action, Expression and Definition	Yes / No	Use the tool's internal editor for editing of Mini-Spec, Long-Description, Static Reaction, Action, Expression and Definition property fields
Auto-Binding of Generic Parameters	Formal Parameter Name / N/A	For auto-binding of generic parameters use the formal parameter name or N/A
Mass edit overwrite values	Yes / No	Overwrite previous property values when modifying via mass-edit
Mass edit ask on overwrite values	Yes / No	Confirm before overwriting previous property values via mass-edit
Update panel bindings upon element rename	Never / Always / Only unique	When to update references in panels upon element rename

Check Model Preferences

Parameter	Value	Description
Scope View	Tree view / Creation rules view	View profile scope as tree or as list
Require Box Attributes for Instance Boxes	Yes / No	Check attribute definitions for instance boxes
Report Errors/Warnings for Unresolved Elements	Yes / No	Apply all tests also to unresolved elements
Apply tests 4075 and 4077 to offpage instances	Yes / No	Apply tests 4075 and 4077 to offpage instances
Apply tests 4075 and 4077 to Activities with Selected-Implementation 'None'	Yes / No	Apply tests 4075 and 4077 to Activities with Selected-Implementation 'None'
Apply test 4061 to External Activities	Yes / No	Apply test 4061 to External Activities
Enhanced Show In Model In Results Tree	Yes / No	Enhance 'Show In Model' to all relevant elements
Statemate MicroC Compatibility Tests (for Statemate)	Yes / No	Statemate MicroC Compatibility Tests (for Statemate)
Design Tests	Yes / No	Execute design tests
Run Database Diagnostics before profile execution	Yes / No	Run Database-diagnostics before profile execution
Optimize report of used/ affected test results	Yes / No	Optimize report of used/affected test results

Prototype C Code Generator Preferences

Parameter	Value	Description
Scope View	Tree view / Creation rules view	View profile scope as tree or as list
Modularity Style	Balanced Mixture / In-line Oriented / Procedure Oriented	Policy for generating code in-line vs. as function
With Debugger	Yes / No	Generate code with debug support
Target Platform	<available target file>	Target platform for generated code
Time Model	Real Time / Simulated Asynchronous / Simulated Synchronous	Time Model of generated code
Graphical Back Animation	Yes / No	Generate code with GBA support
Infinite Loop Limit	<integer value>	Limit for the number of interactions of For or While loops
Push-Button delay time on Event (micro-seconds)	<integer value>	Time (in micro-seconds) that panel event push-buttons remain "pushed" before released
Memory Allocation for Double-Buffered Elements	Dynamic / Static	Memory allocation policy for double-buffered elements
Language	K&R C / Ada / ANSI C	Generated code selected language
K&R Compiler	<compiler command>	Compiler for K&R C generated-code
K&R Flags	<compiler flags>	Flags for K&R C generated-code compilation
Ansi-C compiler	<compiler command>	Compiler for ANSI-C generated-code
Ansi-C Flags	<compiler flags>	Flags for ANSI-C generated-code compilation
Link Flags	<linker flags>	Flags for generated-code link
Libraries	<libraries>	Libraries to link generated-code with
Ada File Spec Extension	<file extension>	Ada File Spec Extension

Ada File Body Extension	<file extension>	Ada File Body Extension
Ada Compiler	<compiler command>	Compiler for Ada generated-code
Ada Flags	<compiler flags>	Flags for Ada generated-code compilation
Ada Linker	<linker command>	Linker for Ada generated-code
Ada Link Flags	<linker flags>	Flags for Ada generated-code link
Ada Libraries	<libraries>	Libraries to link Ada generated-code with
Use 'in_sim' Expression in Code	Yes / No	Trigger expressions with 'in_sim' operator will have effect in generated code

MicroC Code Generator

Parameter	Value	Description
Scope View	Tree view / Creation rules view	View profile scope as tree or as list
Target Configuration	<name of CFG file>	CFG file with initial values for Code-generation options
Pop Up Warning File	Yes / No	Pop up warning file after code-generation
Time Expression Scale	Counters Ticks / Seconds / milli-Seconds	Interpret model time unit as
Enforce Immediate Entry to State Hierarchy	Yes / No	Execute all hierarchical default transitions in single step
Generate separate test expression for each Truth Table Row	Yes / No	Ignore joint cells in truth-tables
Install Timeout while in-State	Yes / No	Install static-reaction timeouts only in inside state
Stop Parent Activity When Descendants Stopped	Yes / No	Stop Parent Activity When all descendants Stopped
SAG Configuration File	<name of SAG configuration file>	CFG file with initial values for SAG options

Embedded Rapid Prototyper Preferences

Parameter	Value	Description
Scope View	Tree view / Creation rules view	View profile scope as tree or as list
Modularity Style	Balanced Mixture / In-line Oriented / Procedure Oriented	Policy for generating code in-line vs. as function
With Debugger	Yes / No	Generate code with debug support
With Remote Panel Server	Yes / No	Generate code with remote panel support
Target Platform	<available target file>	Target platform for generated code
Enable Trace	Yes / No	Generate code with trace support
Time Model	Real Time / Simulated Asynchronous / Simulated Synchronous	Time Model of generated code
Graphical Back Animation	Yes / No	Generate code with GBA support
Infinite Loop Limit	<integer value>	Limit for the number of interactions of For or While loops
Memory Allocation for Double-Buffered Elements	Dynamic / Static	Memory allocation policy for double-buffered elements
Language	K&R C / Ada / ANSI C	Generated code selected language

RT Interface Preferences

Parameter	Value	Description
DOORS I/F configuration file	<configuration file path>	Doors export configuration filename

Panel Builder Preferences

Specific Preferences (Auto Panel Specific Preferences)

Parameter	Value	Description
Used In Chart	Yes / No	Create panel interactors for elements used in chart
Defined In Chart	Yes / No	Create panel interactors for elements defined in chart
Top Level Inputs Outputs	Yes / No	Create panel interactors for to-level inputs/ outputs
With Descendants	Yes / No	Build panel for selected chart and its' descendants
With Generic Instances	Yes / No	Build panel for selected chart and instanti-ated generic charts
Merge To One Panel	Yes / No	Merge descendants and instantiated generic charts into selected charts' panel
Invoke PGE On Apply	Yes / No	Invoke PGE after build
Labels Reserved Height	<Numerical value>	Reserved Height for labels
Interactors Margin	<Numerical value>	Size of margin for interactors
Referenced In Info Flow	Yes / No	Create panel interactors for elements refer-enced in information-flows
Expand Arrays	Yes / No	Create panel interactors for array-indices
Expand Records	Yes / No	Create panel interactors for record-fields

Integer Preferences (Auto Panel Integer Preferences)

Parameter	Value	Description
Active Field	Yes / No	Create interactors for Integers
Bind Method	Input / Output / Input/ Output	Interactor bind method
Interactor Type	Vert. Choice / Horz. Choice / Display / Knob / Lamp / Meter / Push / Slider	Type of interactor
Interactor Width	<Numerical value>	Width of interactor
Interactor Height	<Numerical value>	Height of interactor
ForeGround Color	<color>	Interactor foreground color
Fill Color	<color>	Interactor fill color
Line Width	<integer value>	Interactor line width
Font	<Font name>	Interactor font
Text Color	<color>	Interactor text color
Localizer Color	<color>	Interactor localizer color
Number Of Buttons	<integer value>	Number of buttons in multi-choice interactor
Units	<units name>	Units name for Meter interactor
Display Format	String / Decimal / Hexa Decimal / Octal / Binary	Interactor value display format
Field Length	<integer value>	Interactor value field length
Maximum Scale	<Numerical value>	Maximum interactor scale value
Minimum Scale	<Numerical value>	Minimum interactor scale value
Primary Scale Step	<Numerical value>	Interactor scale step
Number Of Secondary Partitions	<integer value>	Number of partitions between primary scale steps

Real Preferences

Parameter	Value	Description
Active Field	Yes / No	Create interactors for Real elements
Bind Method	Input / Output / Input/ Output	Interactor bind method
Interactor Type	Vert. Choice / Horz. Choice / Display / Knob / Lamp / Meter / Push / Slider	Type of interactor
Interactor Width	<Numerical value>	Width of interactor
Interactor Height	<Numerical value>	Height of interactor
ForeGround Color	<color>	Interactor foreground color
Fill Color	<color>	Interactor fill color
Line Width	<integer value>	Interactor line width
Font	<Font name>	Interactor font
Text Color	<color>	Interactor text color
Localizer Color	<color>	Interactor localizer color
Units	<units name>	Units name for Meter interactor
Display Format	String / Decimal / Hexa Decimal / Octal / Binary	Interactor value display format
Field Length	<integer value>	Interactor value field length
Fraction Part	<integer value>	Number of characters to display after the decimal point
Maximum Scale	<Numerical value>	Maximum interactor scale value
Minimum Scale	<Numerical value>	Minimum interactor scale value
Primary Scale Step	<Numerical value>	Interactor scale step
Number Of Secondary Partitions	<integer value>	Number of partitions between primary scale steps

String Preferences (Auto Panel String Preferences)

Parameter	Value	Description
Active Field	Yes / No	Create interactors for String elements
Bind Method	Input / Output / Input/ Output	Interactor bind method
Interactor Type	Vert. Choice / Horz. Choice / Display / Knob / Lamp / Meter / Push / Slider	Type of interactor
Interactor Width	<Numerical value>	Width of interactor
Interactor Height	<Numerical value>	Height of interactor
ForeGround Color	<color>	Interactor foreground color
Fill Color	<color>	Interactor fill color
Line Width	<integer value>	Interactor line width
Font	<Font name>	Interactor font
Text Color	<color>	Interactor text color
Display Format	String / Decimal / Hexa Decimal / Octal / Binary	Interactor value display format
Field Length	<integer value>	Interactor value field length

Bit Preferences

Parameter	Value	Description
Active Field	Yes / No	Create interactors for Bit elements
Bind Method	Input / Output / Input/ Output	Interactor bind method
Interactor Type	Vert. Choice / Horz. Choice / Display / Knob / Lamp / Meter / Push / Slider	Type of interactor
Interactor Width	<Numerical value>	Width of interactor
Interactor Height	<Numerical value>	Height of interactor
ForeGround Color	<color>	Interactor foreground color
Fill Color	<color>	Interactor fill color
Line Width	<integer value>	Interactor line width
Font	<Font name>	Interactor font
Text Color	<color>	Interactor text color
Highlight Color	<color>	Interactor highlight color
Button Type	Flash / Toggle / Push Release	Push button type
Display Format	String / Decimal / Hexa Decimal / Octal / Binary	Interactor value display format
Field Length	<integer value>	Interactor value field length

Bit_Array Preferences (Auto Panel Bit-Array Preferences)

Parameter	Value	Description
Active Field	Yes / No	Create interactors for Bit-Array elements
Bind Method	Input / Output / Input/ Output	Interactor bind method
Interactor Type	Vert. Choice / Horz. Choice / Display / Knob / Lamp / Meter / Push / Slider	Type of interactor
Interactor Width	<Numerical value>	Width of interactor
Interactor Height	<Numerical value>	Height of interactor
ForeGround Color	<color>	Interactor foreground color
Fill Color	<color>	Interactor fill color
Line Width	<integer value>	Interactor line width
Font	<Font name>	Interactor font
Text Color	<color>	Interactor text color
Display Format	String / Decimal / Hexa Decimal / Octal / Binary	Interactor value display format
Field Length	<integer value>	Interactor value field length

Enum Preferences (Auto Panel Enum Preferences)

Parameter	Value	Description
Active Field	Yes / No	Create interactors for Enum-type elements
Bind Method	Input / Output / Input/ Output	Interactor bind method
Interactor Type	Vert. Choice / Horz. Choice / Display / Knob / Lamp / Meter / Push / Slider	Type of interactor
Interactor Width	<Numerical value>	Width of interactor
Interactor Height	<Numerical value>	Height of interactor
ForeGround Color	<color>	Interactor foreground color
Fill Color	<color>	Interactor fill color
Line Width	<integer value>	Interactor line width
Font	<Font name>	Interactor font
Text Color	<color>	Interactor text color
Number Of Buttons	<integer value>	Number of buttons in multi-choice interactor
Display Format	String / Decimal / Hexa Decimal / Octal / Binary	Interactor value display format
Field Length	<integer value>	Interactor value field length

Event Preferences (Auto Panel Event Preferences)

Parameter	Value	Description
Active Field	Yes / No	Create interactors for Events
Bind Method	Input / Output / Input/ Output	Interactor bind method
Interactor Type	Vert. Choice / Horz. Choice / Display / Knob / Lamp / Meter / Push / Slider	Type of interactor
Interactor Width	<Numerical value>	Width of interactor
Interactor Height	<Numerical value>	Height of interactor
ForeGround Color	<color>	Interactor foreground color
Fill Color	<color>	Interactor fill color
Line Width	<integer value>	Interactor line width
Highlight Color	<color>	Interactor highlight color
Button Type	Flash / Toggle / Push Release	Push button type

Condition Preferences (Auto Panel Condition Preferences)

Parameter	Value	Description
Active Field	Yes / No	Create interactors for Conditions
Bind Method	Input / Output / Input/ Output	Interactor bind method
Interactor Type	Vert. Choice / Horz. Choice / Display / Knob / Lamp / Meter / Push / Slider	Type of interactor
Interactor Width	<Numerical value>	Width of interactor
Interactor Height	<Numerical value>	Height of interactor
ForeGround Color	<color>	Interactor foreground color
Fill Color	<color>	Interactor fill color
Line Width	<integer value>	Interactor line width
Highlight Color	<color>	Interactor highlight color
Number Of Buttons	<integer value>	Number of buttons in multi-choice interactor
Button Type	Flash / Toggle / Push Release	Push button type

State Preferences (Auto Panel State Preferences)

Parameter	Value	Description
Active Field	Yes / No	Create interactors for States
Bind Method	Input / Output / Input/ Output	Interactor bind method
Interactor Type	Vert. Choice / Horz. Choice / Display / Knob / Lamp / Meter / Push / Slider	Type of interactor
Interactor Width	<Numerical value>	Width of interactor
Interactor Height	<Numerical value>	Height of interactor
ForeGround Color	<color>	Interactor foreground color
Fill Color	<color>	Interactor fill color
Line Width	<integer value>	Interactor line width
Highlight Color	<color>	Interactor highlight color
Number Of Buttons	<integer value>	Number of buttons in multi-choice interactor
Button Type	Flash / Toggle / Push Release	Push button type

Activity Preferences (Auto Panel Activity Preferences)

Parameter	Value	Description
Active Field	Yes / No	Create interactors for Activities
Bind Method	Input / Output / Input/ Output	Interactor bind method
Interactor Type	Vert. Choice / Horz. Choice / Display / Knob / Lamp / Meter / Push / Slider	Type of interactor
Interactor Width	<Numerical value>	Width of interactor
Interactor Height	<Numerical value>	Height of interactor
ForeGround Color	<color>	Interactor foreground color
Fill Color	<color>	Interactor fill color
Line Width	<integer value>	Interactor line width
Highlight Color	<color>	Interactor highlight color
Number Of Buttons	<integer value>	Number of buttons in multi-choice interactor
Button Type	Flash / Toggle / Push Release	Push button type

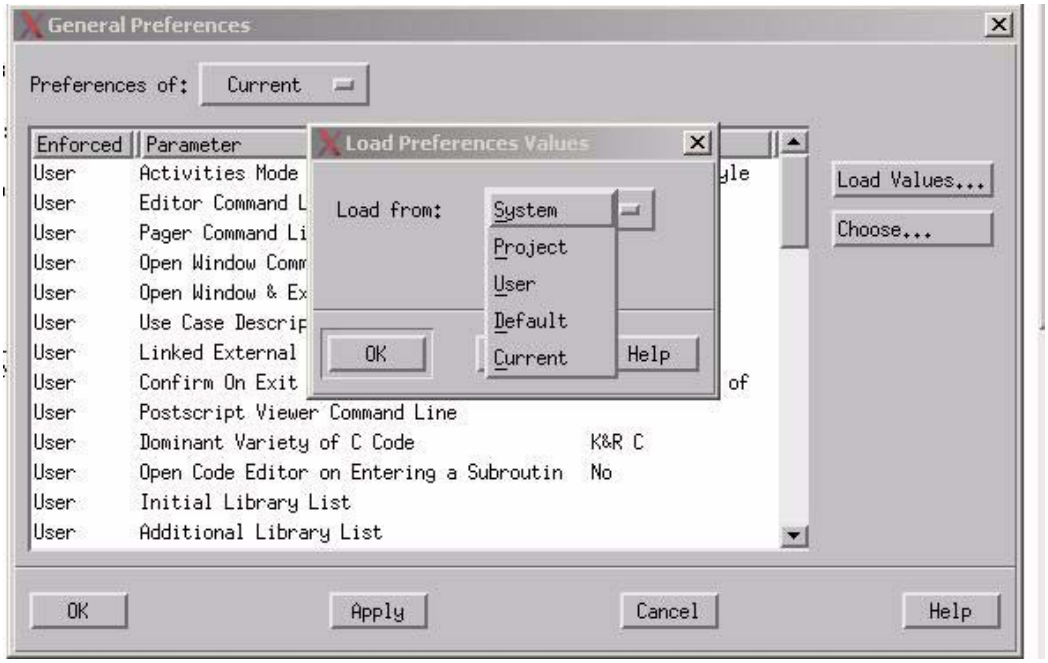
Activity Interface Browser and Reports Preferences

Parameter	Value	Description
Show charts in tree-view	Yes / No	Display charts in the AIF tree browser
View Interface Info Using External Pager	Yes / No	View Interface reports using external pager
Use full-path in Interface report	Yes / No	Use full-path names in Interface report
Interface Analysis Type	Graphical / Functional	Control whether interface analysis is based on data-flows, or usage of data
Show source/target in LCA chart	Yes / No	Display instance box in LCA chart as source/target activity (vs. basic activity)
Consider Environment Activity Flow-lines in Functional mode	Yes / No	

Loading Predefined Preferences

To load predefined preferences:

1. Click **Load Values**. The **Load Preferences Values** dialog box displays.



2. Select a preference type from the **Load from** pull-down menu. The supported types are **System**, **Project**, **User**, **Default**, and **Current**. For a description of these types, see **Specifying Where Preferences are Applied**.

3. If you select the preference type **Project** or **User**, a **Project** or **User Name** pull-down menu opens. Select the preferences to load from the choices in this pull-down menu.

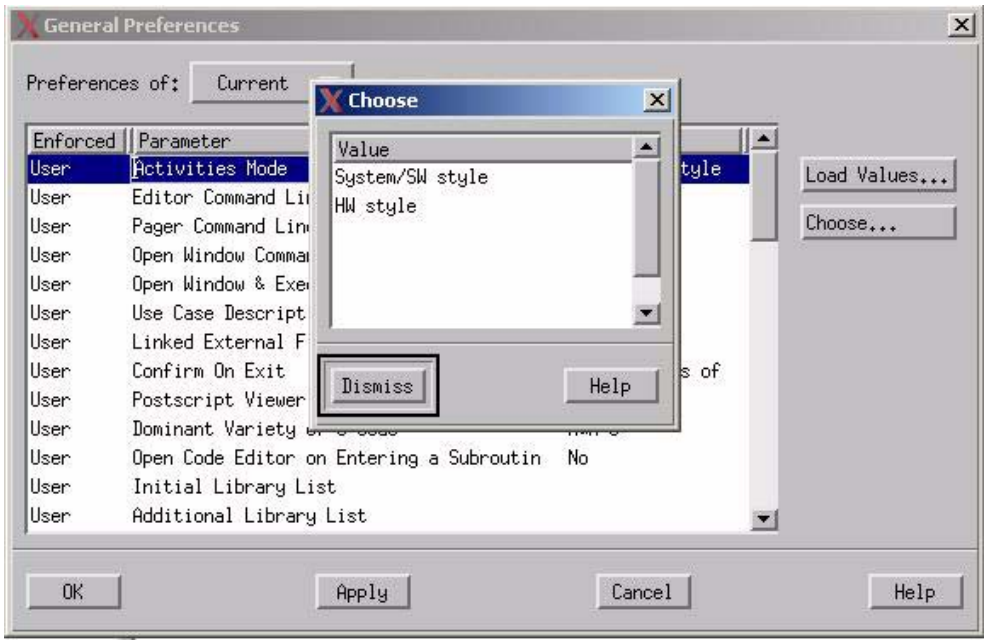
4. Click **OK** to exit the Load Preferences Values window.

Setting Parameter Preferences

To set parameter preferences:

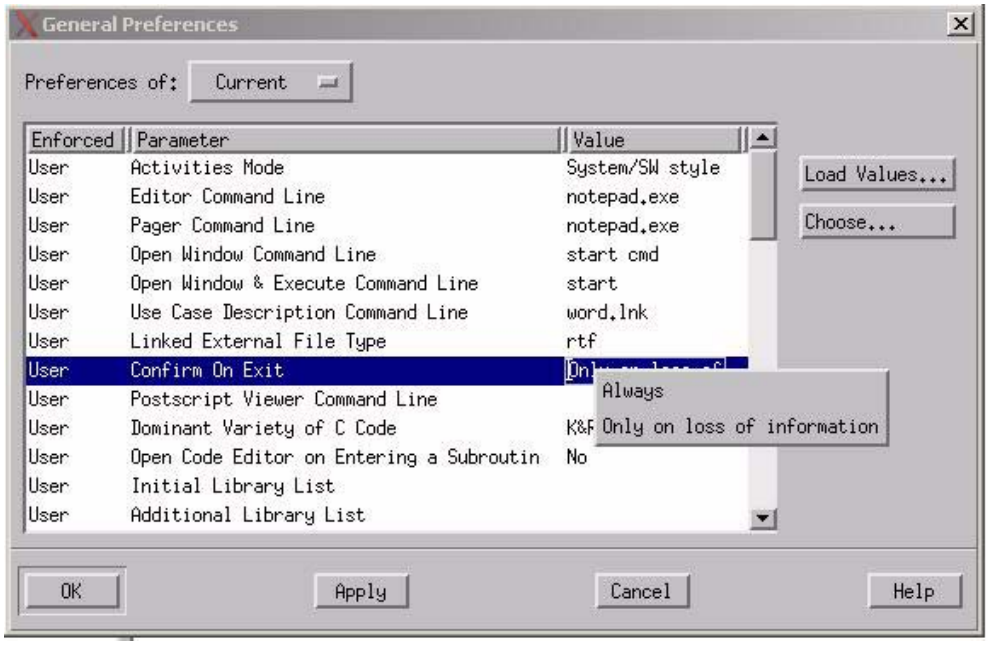
1. Doing one of the following:

   - Highlight a parameter, then click **Choose**. The **Choose** dialog box displays. Select one of the listed values to set the parameter.



   - Select the **Value** field of a parameter. The supported values are displayed. Select one of the choices or, if required, replace the text in the box (for example, to change the text editor you want to use).

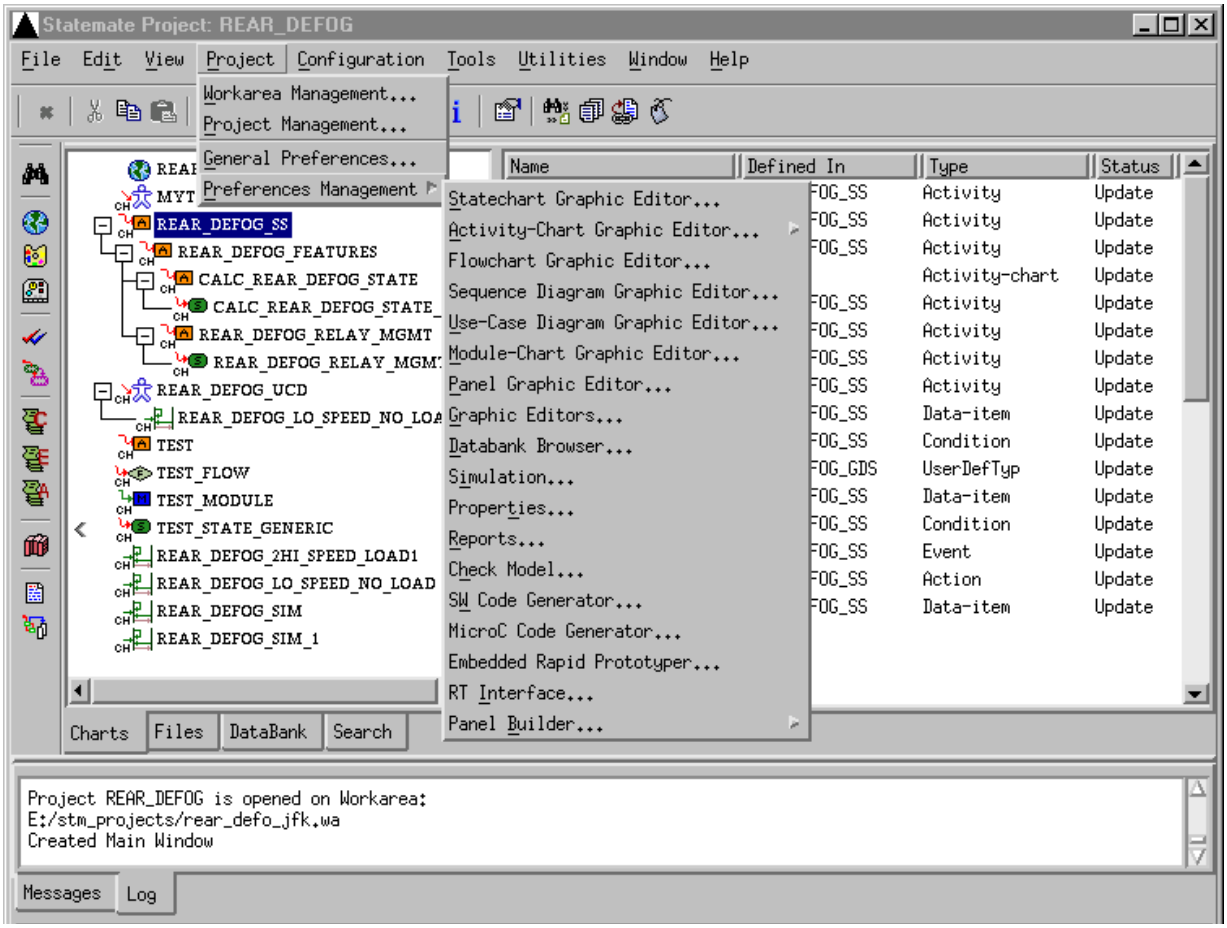
2. Click **Apply** to apply any changes you have made, then click **OK** to exit the **General Preferences** dialog box.


General Preferences
Preferences of:
Current
Enforced
Parameter
Value
User
Activities Mode
System/SW style
User
Editor Command Line
notepad.exe
User
Pager Command Line
notepad.exe
User
Open Window Command Line
start cmd
User
Open Window & Execute Command Line
start
User
Use Case Description Command Line
word.lnk
User
Linked External File Type
rtf
User
Confirm On Exit
User
Postscript Viewer Command Line
User
Dominant Variety of C Code
User
Open Code Editor on Entering a Subroutin
No
User
Initial Library List
User
Additional Library List
Always
Only on loss of information
Load Values...
Choose...
OK
Apply
Cancel
Help

Setting Preferences for Editors and Utilities

To set preferences for editors and utilities:

1. From the Rational Statemate main window, select **Project** > **Preferences Management**.



2. From the Preferences Management menu, select a specific editor or utility for which to set preferences.

> **Note:** You might not have access to all the preference options. For example, the system administrator can reserve the right to change the System preferences.

The steps for changing preferences for editors and utilities are the same as changing general preferences for Rational Statemate. For more information, see **Setting General Rational Statemate Preferences**.

Configuration Management

Configuration management may also be called *CM*, *change management*, *revision management*, and *source control*.

Each Rational Statemate project has a common repository called the databank. The databank is both centralized and permanent. That is, information in the databank belongs to the entire project and represents the current working design.

Databank

The *databank* contains several subdirectories that hold different types of Rational Statemate elements, for example, charts, panels or configuration files. Elements are stored in the databank as ASCII definition files.

A project databank can be placed in any directory in which you have read and write access. Many sites designate a special location for project data; check with your system administrator.

As you rework your charts, each modification is made to your workarea. When you want to permanently store a new version of your changes to enable others to share them, save the modified charts to the databank. Through a locking mechanism, you are ensured that your work will not conflict with the interests of the other project members.

While working on a project, you can perform configuration management operations such as checkins and checkouts using the toolbar from the Rational Statemate main window and the Databank Browser (the DataBank tab on the Rational Statemate main window). Rational Statemate provides a built-in CM tool similar to those found in products designed specifically for the task of configuration management. In addition, Rational Statemate enables you to transparently substitute a different CM Tool when you create a project.

Elements

Elements stored in the databank are called configuration items and stored as ASCII files. Multiple versions of the same configuration item can exist in the databank. All versions of a configuration item belong to the same owner and have the same access permissions.

Elements consist of the following:

- Charts (statecharts, activity charts, module charts, use-case diagrams, sequence diagrams, flowcharts, and global definition sets)
- Analysis profiles
- Simulation Control Language files
- Waveform profiles
- Status files
- Check Model profiles
- Code generation profiles (compilation profiles)
- Documentor (DGL) templates
- Include files
- Configuration files
- Panels
- Components (Every databank can contain components, but only databanks that are defined as libraries can “export” these components to other projects.)
- Targets and Cards

Note

Truth tables are not separate configuration items. A truth table belongs to the chart in which it was defined. When the chart containing the truth table is loaded into the workarea, the truth table is also loaded. It is also saved to the databank when the chart is saved.

Working with the Databank

To work with the databank:

1. From the Charts, Files, or DataBank tab of the Rational Statemate main window, select a chart.
2. Open the **Configuration** menu, by doing one of the following:

- Select the **Configuration** option.
- From the **DataBank** tab, right-click to display a menu allowing you to perform any of these tasks:
  - **Check Out** - copy the selected chart or file into your workarea from the databank. See **Checking Out Databank Items** for more information.
  - **Check Out With Descendants** - copy the chart, along with all its descendants, into your workarea from the databank.
  - **Lock** - Set a lock on the file in the databank and change the mode of the item in the workarea to update. See **Locking Databank Items** for more information.
  - **Release Lock** - Release the lock on the file in the databank and change the mode of the item in the workarea to read-only.
  - **Delete** - delete files and charts (that you own) from the databank.
  - **Purge** - delete older versions of files and charts (that you own) back to and including the version specified in the Databank Browser preferences.

  Note: Always use **Purge** to delete versions of configuration items. Do not use operating system commands for this purpose.

  - **Create Configuration** - create a new configuration. Selecting this option opens the **Create Configuration** dialog box. See **Creating a New Configuration** for more information.
  - **Execute Configuration** - Check out from the databank all files specified in the selected configuration file.
  - **Export** - Exports files and charts to another directory.
  - **Properties** - Enables you to see the databank properties for the selected elements.
  - **Properties in New Window** - The same as **Properties**, except that it opens in a new window if multiple windows are open.
  - **View Selected Version Changes** - Enables you to view any check-in or check-out comments for the selected version.
  - **View All Changes** - Enables you to view all check-in and check-out comments for the selected version.

Automatic Databank Refresh

You may set the **Automatic Databank Refresh** preference to control the behavior of the Databank refresh to one of these values:

- **Yes** - the Databank view always refreshes automatically
- **No** - the Databank view only refreshes when the **Name** and **Type** columns and the other columns are refreshed when the line is selected. In this mode, the Databank view can be fully refreshed upon request using the **View** > **Refresh** selections.

Checking Out Databank Items

You may check out databank items using either of these two DataBank tabs right-click menu options:

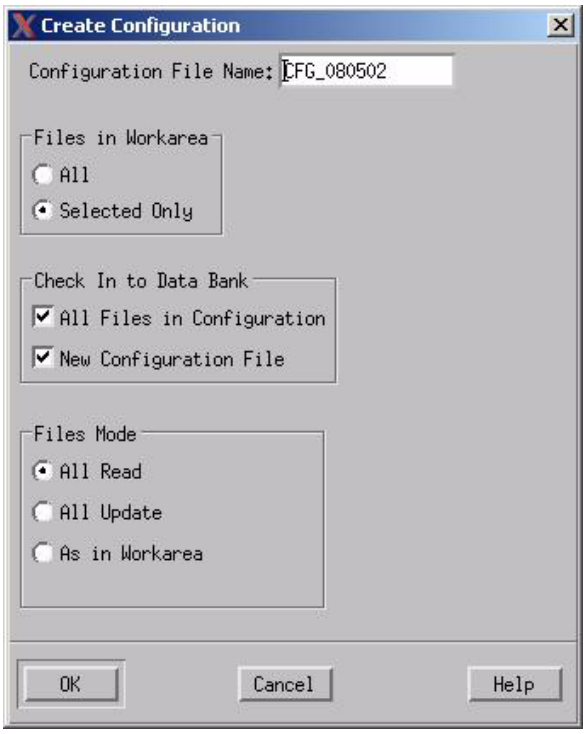
- **Check Out** - copy the selected chart or file into your workarea from the databank.
- **Check Out With Descendants** - copy the chart, along with all its descendants, into your workarea from the databank.

  You may also use the **Filtered Checkout** utility for activity charts and Global Definition Sets (GDS). You may select either of these filtered checkout modes:
- **Filtered Check Out Parent from Databank** - to checkout the parent chart from the Databank
- **Filtered** - loads the graphical elements and only the necessary textual elements of the parent chart into the workarea.

Creating a New Configuration

To create a new configuration:

1. From the **DataBank tab**, right-click and select the **Create Configuration** to display this dialog box.



2. Enter the name of the new configuration into the **Configuration File Name** or accept the displayed default.
3. Choose whether to use all the files in the workarea or just those selected.
4. Determine if you want all files in the configuration checked into the Databank, just the new configuration file, or both.
5. Choose whether the permissions for the configuration files are **All Read** (read only), **All Update** (all files can be updated), or just the permissions **As in Workarea**.
6. Click **OK** to save the new configuration.

Locking Databank Items

Locking a configuration item guarantees that other users cannot change the item in the databank while you are working on it. After your changes are complete, you can check it into the databank and either release or continue to hold the lock. To lock a configuration:

1. From the **DataBank** tab, select the file or files in the databank you want to lock.
2. Right-click and select **Lock**.

To remove the lock:

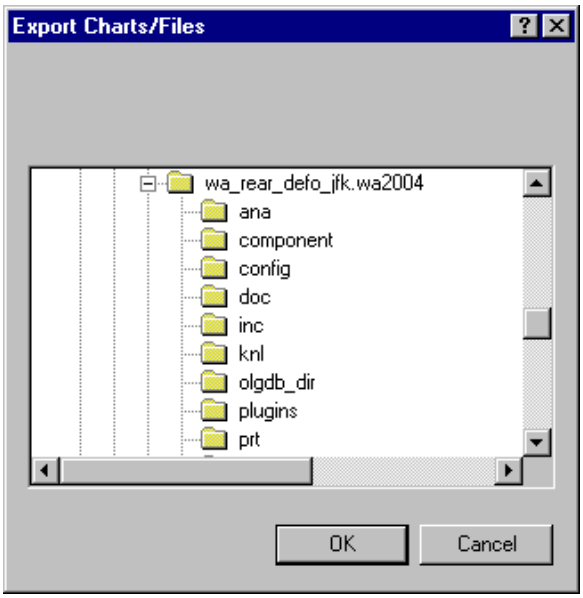
1. From the **DataBank** tab, select the file in the databank you want to release.
2. Select the **Release Lock** option from the right-click menu and change the mode of the item in the workarea to read-only.

No configuration item can be locked by two or more users simultaneously. When you create a new configuration item (a chart or file that you have created but not yet stored in the databank), it is automatically locked by you. You can, therefore, always save such a configuration item into the databank.

Exporting Charts and Files

To export charts and files from the databank:

1. From the **DataBank** tab, right-click and select the **Export Charts/Files** to display this dialog box.



2. Select the new directory to receive the exported charts or files.
3. Click **OK** to perform the export.

Error Handling when Loading Charts

Rational Statemate handles chart load faults during chart load operation as follows:

- When a name or an expression contains an illegal character in the upper character set (ASCII 128 to 255), the illegal characters is replaced with an underscore (“_”).
- When an illegal expression is found (not just an illegal character), the expression is converted to a comment
- When attempting to load an already existing chart to the workarea, Rational Statemate prompts you with suggestions for an alternative chart name.

Note

When either of the first two situations occurs, a change description is created and added to the chart change log.

Checking In and Out Elements

To improve efficiency in your configuration management work, charts can be checked out in read-only mode as a single transaction. If there is a mixture of read-only and update check outs, the read-only charts are checked out first in a single transaction, and the other charts are checked out next.

This feature is controlled by the Check Out in Single Transaction preference with these two options:

- **Yes** - Check out using single transaction
- **No** - Check out as a transaction per chart (Default)

Selecting “No” may result in many transactions if the project contains many charts. This would adversely affect the tool performance.

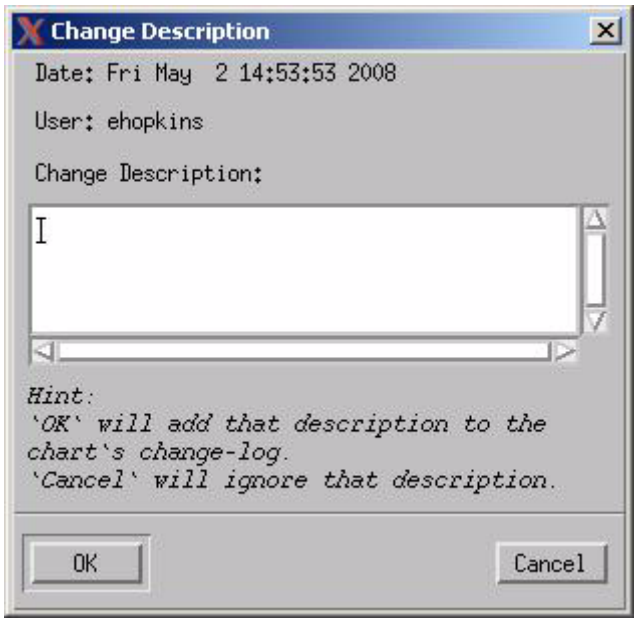
Note

An error in a single chart can cause the check-out operation to fail and would require a rollback of the changes in the databank. Therefore, the single transaction is performed only when checking out in read-only mode.

Tracking Changes

The Track Changes feature enables you to enter and log change descriptions, following modifications to charts in the Rational Statemate model. It also enables you to attach a description to a specific chart version when checking it into the Databank. You can add and view a change description from the workarea browser, graphic editor, and properties editor.

To add a change description manually, select **Tools > Track Changes > Add-Change Description**. The Change Description dialog displays, as below. Fill in the change description field and click **OK**.



To view the change description log for a chart from the workarea browser and the graphic editor select **Tools > Track Changes.**

Note

The Track Changes options vary slightly, depending on the selected tab.

Automatic Change Tracking

You can choose to have change descriptions entered automatically whenever Rational Statemate detects a change to the model. Any changes to the properties of graphical and textual elements are identified and prompt you with an editable change description.

To enable automatic mode, select **Tools > Track Changes > Track Changes While Editing**.

Note

Changing the automatic mode affects all the tools.

Track Changes Preferences

The following Track Change preferences are included in the General Preferences:

- **Track Changes While Editing** - sets the default value for the Track Changes automatic mode flag. This option is not set by default.
- **Add Change Description when Check-in Chart** - displays a change description window whenever you check in a chart to the databank. This option is set by default.

Types of Changes Tracked

The following changes in the properties editor entries of activities, states, and subroutines are tracked:

- Activity (selected Implementation, mini-spec, truth-table, subroutine binding, and termination type)
- State (static-reactions)
- Subroutine (selected implementation, K&R C code, Ada code, Rational Statemate language, truth-table, type, returned type, and parameter)

Note

When both the dimension and the content of a truth-table are modified during a single editing session, only the change in dimension is tracked.

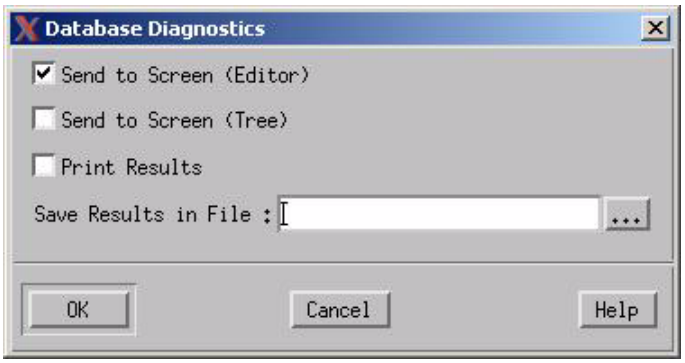
Track Changes Limitations

- Only the first line in each field of the Use Case Scenario list table is checked for changes.
- There is no change detection for changes in the Long description field.
- In the Databank Browser, there is no change detection during the Copy Chart and Rename Chart operations.

Database Diagnostics

The Database Diagnostics utility checks correctness of the workarea database (charts.data), and reports internal database errors, if exist (unlike the Check-model tool, which checks completeness and correctness of the Statemate user-model).

To examine database error information, use the **Utilities > Database Diagnostics** option. The database diagnostics results can be displayed as tree with the results ordered by error number or by error category.



Error Report

This utility supplies a report and allow quick resolution of the reported errors. The report results can now be displayed as a tree, suggesting various actions that can be performed on the erroneous elements shown in the report.

View and Resolve Errors

When viewing the results in the tree view, right-click to perform the following actions on the erroneous elements:

- Delete
- Show Properties
- Show In Model
- Rename (Can only be used on a single item (tree leaf))

Each action is associated with the relevant messages. When an error occurred with a reference to a chart, the “Show in model” option displays the Box from which the chart is referenced.

Note

Not all actions can be performed on every error message.

The default actions might or might not solve the problem depending on any ramifications of the problem and the change.

Plugins

Additional configuration management items can be added by using the Plugins capability.

This is done by adding textual file to the plugins directory in the STM_ROOT directory. The text files have the extension .txt. Each file describes the elements needed for a plugin entry. In addition a directory will be added under the plugins directory of the Workarea and Databank to hold the data.

The <plugin>.txt file format:

NAME	<name of the tool to be displayed in Rational Statemate Main Window>
ABBREV_NAME	<abbreviation of the tool name (for Workarea browser use)>
ICON	<path to icon (X-Icon) to be displayed in main window>
ICON_DIMMED	<path to icon to be displayed in main window when dimmed>
COMMAND	<path to command - See below>
FILE_TYPE	<file extension - See below >
SUBDIR	<sub directory in the plugins directory>
LICENSE	<String. Usage to be defined in future >

<path to command> - to be executed when clicking on the icon or when double clicking on a file with a matching extension in the Workarea browser. The command will be called with the following parameters:

1. <project name>
2. <user name>
3. <Databank path>
4. <Workarea path>
5. <STM ROOT>
6. [<The file name in the Workarea>]. Include the directory name, relevant only when executed from double clicking in the WAB.

<file extension> - of file related to the tool. This type will become a configuration item.

Only the NAME and COMMAND, above, entries are mandatory.
In the case of the ICON entry used, an icon will be displayed in the main window.
In the case of FILE_TYPE entry used, a configurable item will be added to Workarea browser list, and double clicking on the file will invoke the COMMAND with the file name.
All plugins files will be saved in a new directory in the Workarea, and in the Databank subdirectory "plugins" or in the case of a SUBDIR entry in the plugin text file, in
The specified sub directory.

Example of a <plugin>.txt file :
NAME ClearCaseExplorer
ICON C:\IBM\Rational\Statemate\4.6\etc\mmi\stm\CCexplorer.xpm
COMMAND C:\IBM\Rational\Statemate\4.6\bin\clear_case_explorer.bat

Using the Graphic Editors

This section describes the charts and diagrams that Rational Statemate supports and the graphic editors associated with them.

The topics are as follows:

- **Overview of the Rational Statemate Graphic Editors**
- **Working with Graphic Editors**
- **Working with Charts and Diagrams**

The graphic editors follow conventions similar to many other diagram drawing tools. For example, select the element or elements to be acted upon (copied, moved, deleted, etc.) and then select the action.

Use the graphic editors to:

- Create and edit charts and diagrams.
- Set chart properties, such as line width, font, size, etc.
- Enlarge or shrink the display for ease of viewing.
- Open other features such as the Properties window or the Check Model tool.

Overview of the Rational Statemate Graphic Editors

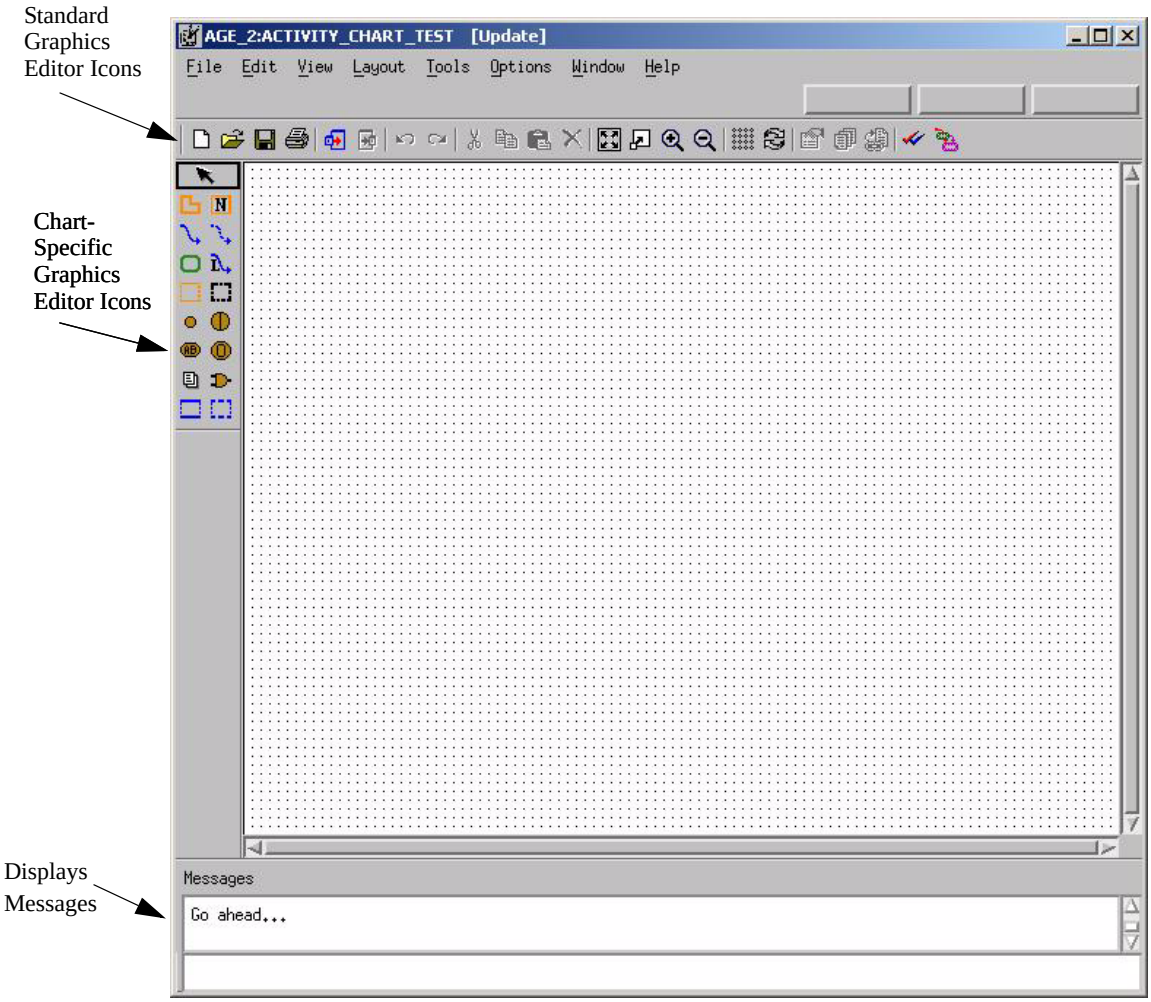
This sections provides an overview of the Graphic Editors

- Graphic Editor Icons
- Graphic Editor Menus

The following figure uses the Activity Charts graphics editor as an example. Each graphics editor is described in the detail in Working with Charts and Diagrams.

Note

The Grid feature is enabled.



Graphic Editor Icons

This section described the standard graphic editor icons that are used with most Rational Statemate graphic editors.

Note

Not all features are available with all graphic editors.

Icon	Description
	New - Opens a new diagram of the same type of graphics editor. For example, if you are working with the Use Case editor, it opens a new Use Case diagram.
	Open - Opens another existing diagram of the same type of graphics editor. For example, if you are working with the Use Case editor, it opens another existing Use Case diagram.
	Save - Saves any changes made to the diagram.
	Print - Prints the diagram.
	Open Parent/Open Related Chart
	Open Sub-Chart
	Undo - Undoes the last change.
	Redo - Repeats last change.
	Cut (to Paste Buffer) - deletes selected text/graphic and places in the paste buffer.
	Copy (to Paste Buffer) - copies selected text/graphic and places in the paste buffer.
	Paste - Pastes selected text/graphic.
	Delete - deletes selected text/graphic.
	Full View - displays the full chart.

Icon	Description
	Zoom Area - Allows you to select an area of the chart to zoom into.
	Zoom In, Zoom Out - Allows you to zoom in and out of a selected area of the chart.
	Grid - displays/hides the grid.
	Refresh - Refreshes the diagram.
	Properties - Opens the Properties dialog box.
	Long Description - Opens the Properties dialog box. See
	Link to External File - Opens the Properties dialog box.
	Invoke Check Model - Opens the Check Model tool. See the *Check Model Reference Manual* for more information.
	Invoke Simulation - Opens the Simulation tool. Reference to the *Simulation Reference Manual* for more information.

Graphic Editor Menus

This section lists the menus supported by the graphic editors, and describes the menu options for each menu.

The following table lists the menus that appear in the menu bar for each graphic editor.

Editor	Menu						
	File	Edit	View	Layout	Tools	Options	Help
Activity Chart	X	X	X	X	X	X	X
Module-chart	X	X	X	X	X	X	X
Statechart	X	X	X	X	X	X	X
Sequence Diagram	X	X	X	X	X	X	X
Use-case Diagram	X		X		X	X	X
Flowchart	X	X	X	X	X	X	X

Note

The use case diagram editor has special Copy and Select menus that take the place of the Edit menu in the other graphic editors.

The following sections describe the main graphic-editor menus (with the exception of the Help menu). Each section lists the menu items available for each editor and describes the menu options.

File Menu

The following table lists the **File** menu items and which items each graphic editor supports.

Menu Item	Activity	Module	State	Sequence	Use Case	Flow
New	X	X	X	X	X	X
Open	X	X	X	X	X	X
Close	X	X	X	X	X	X
Save	X	X	X	X	X	X
Open Parent	X	X	X			X
Open Sub-chart	X	X	X	X		X
Open Related charts					X	
Insert chart	X	X	X			X
Create sub-chart	X	X	X			X
Create Sequence Diagram	X					
Preview Component	X					
Print	X	X	X	X	X	X
Exit	X	X	X	X	X	X

The following table describes the **File** menu items.

Menu Item	Description
New	Creates a new chart.
Open	Opens an existing chart.
Close	Closes the current chart. If the chart includes graphical or textual changes not yet saved, you are prompted to save these before the chart is exited. The graphic editor remains open and is now labeled Empty Editor. (You are prompted to name the chart when you exit.)
Save	Saves the current chart.
Open Parent	Opens a new graphic editor for the chart immediately above the current chart in the hierarchy.
Open Sub-chart	Opens a new graphic editor for an existing sub-chart (generic or off-page). **Note:** This operation is valid only if the selected box name contains an off-page (@NAME) reference or a generic chart instantiation (I<NAME) and the chart exists in the workarea.

Menu Item	Description
Open Related charts	Opens a chart that a use case diagram element is based on.
Insert chart	Inserts the contents of another chart into the current graphic editor window using the Insert window. From the Insert window you can do the following: • In the From Chart textbox, enter or select the chart to be inserted. • In addition to being able to insert an entire chart, you can insert a single activity (state or module) for the specified chart. Click the down arrow to view a list of possible activities (states or modules) to be inserted. If the selected chart has no sub-activities, the list is blank. • Optionally, deselect **Copy Element Definitions**, if you do not want textual elements included.
Create sub-chart	Creates a sub-chart.
Create Sequence Diagram	Creates a sequence diagram from an activity chart.
Preview Component	Previews a selected component in an activity chart.
Print	Prints the current chart.
Exit	Exits from the graphic editor and closes any open charts. If the chart includes graphical or textual changes not yet saved, you are prompted to save these before the chart is exited. The graphic editor remains open and is now labeled Empty Editor. (You are prompted to name the chart when you exit.)

Edit Menu

The following table lists the **Edit** menu items and which items each graphic editor supports.

Menu Item	Activity	Module	State	Sequence	Use Case	Flow
Undo	X	X	X	X	X	X
Redo	X	X	X	X	X	X
Cut	X	X	X	X	X	X
Copy	X	X	X	X	X	X
Paste	X	X	X	X	X	X
Insert Component	X					
Copy Component	X					
Move	X	X	X	X	X	X
Stretch	X	X	X	X	X	X
Edit Text	X	X	X	X	X	X
Delete	X	X	X	X	X	X
Properties	X	X	X	X	X	X
Select	X	X	X	X	X	X

The following table describes the **Edit** menu items.

Menu Item	Description
Undo	Undoes the last operation.
Redo	Redoes the last operation.
Cut	Deletes the selected text or elements and places them in the paste buffer.
Copy	Copies selected text or elements into the paste buffer.
Paste	Pastes selected text or elements from the paste buffer.
Insert Component	Inserts a component.
Copy Component	Copies a component.
Move	Moves selected text or elements.
Stretch	Stretches selected text or elements.
Edit Text	Enables text-edit mode.
Delete	Deletes selected elements.

Menu Item	Description
Properties	Modifies the following display properties: • Color • Font • Line Width
Select	Selects text or elements.

View Menu

The following table lists the **View** menu items and which items each graphic editor supports.

Menu Item	Activity	Module	State	Sequence	Use Case	Flow
Full View	X	X	X	X	X	X
Zoom Area	X	X	X	X	X	X
Zoom In	X	X	X	X	X	X
Zoom Out	X	X	X	X	X	X
Refresh	X	X	X	X	X	X
Hide	X	X	X	X	X	X
Reveal	X	X	X	X	X	X
Grid	X	X	X	X	X	X
Grid Setting	X	X	X	X	X	X
Messages	X	X	X	X	X	X
Drawing Icons	X	X	X	X	X	X
Command Bar	X	X	X	X	X	X
Toolbar	X	X	X	X	X	X
Pagination Preview				X		
Floating Lifeline Names				X		
Scenario Auto Numbering				X		

The following table describes the **View** menu items.

Menu Item	Description
Full View	Expands or reduces the drawing frame to fit into the graphic canvas. This is the default when a graphic editor is first opened.
Zoom Area **Zoom In** **Zoom Out**	Enlarges or shrinks the drawing on your screen, without changing the window dimensions. Choose one of the following: **Note:** By default, zoom operations do not affect the text. Select **Options > Enable Scale Text** to change the size of both the text and lines when you zoom into and out of the canvas. • To zoom in to a particular area of a drawing, select **Zoom Area**. Drag a rectangle around the area of the drawing that you want to zoom into. The area you selected expands to fill the canvas. • To zoom into a drawing, select **Zoom In.** The drawing expands approximately 30%. Use the scroll bars to pan the window. To further enlarge the drawing, you can select **Zoom In** several times. • To zoom out of a drawing, select **Zoom Out**. The drawing contracts approximately 30%. Use the scroll bars to pan the window. • Select **Full View** to return to the original size.
Refresh	Redisplays the graphics on the canvas. Occasionally erroneous graphics can remain on the screen during editing. Use this operation to clear them from the screen display.
Hide	Hides the detail underneath a box. The box is shaded to show that there is missing detail underneath. You can use this operation to hide details during printing, for example.
Reveal	Redisplays previously hidden elements.
Grid	Displays the editing grid. By default, when you open a graphic editor, the grid is not displayed. Select **Grid** again to turn the grid display off.

Menu Item	Description
Grid Setting	Displays the Grid Setting dialog box, which allows you to change the grid settings. The grid consists of a matrix of points the pointer/drawing elements snap to. The Grid Setting dialog box shows the width and height of the drawing frame, but this information is only for reference. You cannot change these values. To expand the drawing area to fill the window, use **Layout > Reframe**. To make the grid • Active and visible, select **Active**. • Active and invisible, select **Snap to grid**. To specify the spacing of the grid points, set the **Grid Spacing** number. The up and down arrows provide a quick way to double or halve the spacing between grid points. To specify how many of the grid points to show, use the **Every nth grid point** field. (If this value is 1, all the grid points are shown. If this value is 2, only every 2nd grid point is shown, etc. The other grid points exist; however, you cannot see them.) Click **Apply** to apply your changes and continue, and click **OK** when you are finished.
Messages	Toggles between showing and hiding the Message and Log tabs.
Drawing Icons	Toggles between showing and hiding the drawing toolbar.
Command Bar	Toggles between showing and hiding the command toolbar.
Toolbar	Toggles between showing and hiding the Rational Statemate toolbar and various parts of the standard toolbar. You can choose whether to display the Rational Statemate toolbar vertically along the left side of the window or horizontally across the top of the window. In the Rational Statemate main window, select **View > Toolbars > Tools (Horiz.) or Tools (Vert.)**.
Pagination Preview	Allows you to preview the page.
Floating Lifeline Names	Toggles between showing and hiding names of floating lifelines.
Scenario Auto Numbering	Toggles between showing and hiding scenario auto numbering.

Menu Item	Description
View Filter	Displays the View Filter dialog box, which allows you to control the information displayed on the various displays. The View Filter dialog box has four tabs: • Main Tree Browser -- controls what displays on the left pane of the Rational Statemate Main Window • Elements Matrix -- controls what displays on the right pane of the Rational Statemate Main Window • Files -- controls what displays on the Files tab • DataBank -- controls what displays on the Files tab Each tab contains a set of items that can be either displayed or suppressed by the View Filter. Select individual items to display or suppress by clicking the item button. Click **Apply** to apply your changes and continue, and click **OK** when you are finished.

Layout Menu

The following table lists the **Layout** menu items and which items each graphic editor supports.

Menu Item	Activity	Module	State	Sequence	Use Case	Flow
Replicate	X	X		X	X	X
Arrange	X	X	X	X	X	X
Align to Grid	X	X	X	X	X	X
Reframe	X	X		X	X	X
Vertical Spacing	X		X			
Horizontal Spacing	X		X			

The following table describes the **Layout** menu items.

Menu Item	Description
Replicate	Quickly creates a grid of drawing elements. • Select the elements to copy. • Position the selected elements in the corner of the grid of elements to be created. Select **Replicate**. The Replicate dialog box opens. • Fill in the **Number of Rows** and **Number of Columns** text boxes to set the number of rows and columns to be copied. • Click **OK**. • As you move the pointer in the graphic canvas, you see shadow images of the drawing elements you are replicating arranged in a grid. Click when the shadow images are in the correct position.
Arrange	Aligns selected elements vertically and horizontally, as well as arranging the spacing between them. You can do the following: • Rearrange elements in relation to each other. This option has no effect, if only one element is selected. **Note:** Be careful not to select both the names and the boxes when arranging charts, because you can move a name out of its box using the arrange operations. If you only select the box, the name moves with it. • Align elements around a vertical line, based on the system calculation of the average location of the elements. You can align elements by their: * Left edge * Center * Right edge • Align elements around a horizontal line, based on the system calculation of the average location of the elements. You can align elements by their: * Top edge * Center * Bottom edge

Menu Item	Description
Arrange (continued0	• Move elements so that they are evenly spaced vertically, based on the system calculation of the average location of the elements. You can space elements evenly by the: * Bottom edge * Center * Top edge * Space between elements • Move elements so they are evenly spaced horizontally, based on the system calculation of the average location of the elements. You can space elements evenly by the: * Left edge * Center * Right edge * Space between elements
Align to Grid	Better aligns text and elements to the grid.
Reframe	Changes the size and shape of the drawing frame (or page). • Select **View > Full View** before this operation. • Use the window frame resize bars to change the size or shape of the entire graphic editor window. • Click **Reframe**. The drawing frame is changed so that it fits the new size and shape of the graphic canvas exactly. **Caution:** This operation can change the origin of the grid. If you are using the grid, be careful in using this operation, because it may be difficult to make your drawing line up correctly afterwards.
Vertical Spacing	Adds vertical spacing for selected elements.
Horizontal Spacing	Adds horizontal spacing for selected elements.

Tools Menu

The following table lists the **Tools** menu items and which items each graphic editor supports.

Menu Item	Activity	Module	State	Sequence	Use Case	Flow
Track Changes	X	X	X	X	X	X
Properties	X	X	X	X	X	X
Info	X	X	X	X	X	X

Menu Item	Activity	Module	State	Sequence	Use Case	Flow
Local Interface Info	X					
Global Interface Info (Activities)	X					
Global Interface Info (Elements)	X					
Chart Properties	X	X	X	X	X	X
Chart Info	X	X	X	X	X	X
Subroutine Properties	X					
Simulation	X		X			
Check Model	X	X	X	X		X

The following table describes the Tools menu items.

Menu Item	Description
Track Changes	Maintains or reports change history for an element.
Properties	Modifies the following display properties: • Color • Font • Line Width
Info	Displays a summary of the properties for selected elements.
Local Interface Info	Displays a summary of direct inputs and outputs for the element.
Global Interface Info (Activities)	Displays a summary of the most distant activities the input and output signals flow to or from, sorted by source/sink activity.
Global Interface Info (Elements)	Displays a summary of the most distant activities the input and output signals flow to or from, sorted by the name of the input or output.
Chart Properties	Opens the Properties window for the chart.
Chart Info	Displays a summary of the properties for the chart.
Subroutine Properties	**Note:** This selection is only available for the activity chart editor. For an activity that is bound to a subroutine, opens the Properties window and the dialog box with the subroutine implementation (body). If there are multiple implementations currently defined for the subroutine, the implementation that is identified in the Select Implementation field is opened. (Select **Project > General Preferences > Open Code Editor on Entering a subroutine** to go directly to the subroutine template code editor.)
Simulation	Opens a simulation execution window directly from your open chart. **Note:** This selection is not available for the module-chart editor.
Check Model	Opens the Check Model tool. For more information on Check Model, see the *Check Model User's Guide.*

Options Menu

The following table lists the **Options** menu items and which items each graphic editor supports.

Menu Item	Activity	Module	State	Sequence	Use Case	Flow
Gravity Setting	X	X	X	X	X	X
PC Mouse (2 Button)	X	X	X	X	X	X
Preserve Selection	X	X	X	X	X	X
Simulation Highlight	X		X			
Enable Scale Text	X	X	X	X	X	X
Advanced: Filled Boxes	X	X	X			X
Advanced: Fill Colors	X	X	X			X
Advanced: Carriage Return is New Line	X	X	X	X	X	X
Advanced: Enabled Reshaping	X	X	X		X	X
Preferences Management	X	X	X	X	X	X

The following table describes the **Options** menu items.

Menu Item	Description
Gravity Setting	Changes the gravity setting. Gravity is the distance you must get within to attach arrow heads to boxes or to select elements. To change the gravity setting: • In the Gravity Setting dialog box, specify the **Gravity Distance** in pixels. • Click **Apply** to apply your selection and continue, and click **OK** when you are finished.
PC Mouse (2 Button)	By default, you move a drawing element by dragging it with the middle mouse button and selecting it using the left mouse button. When you select PC Mouse, you can drag a drawing element with the left mouse button, as well as select with it. This option is available for users who are accustomed to applications running under Microsoft Windows or on an Apple Macintosh. When PC Mouse is set, the mouse conventions for editing are similar to the conventions on these machines. **Note:** You can also set the single-button mouse option as a general graphic editors preference.

Menu Item	Description
Preserve Selection	By default, when you create or move an element, the element becomes selected. This option changes to the opposite state. **Note:** The Preserve Selection option is not saved in the chart.
Simulation Highlight	Allows you to specify color and style of highlighting states and activities during simulation. To make any changes desired to either the transition or state highlighting defaults, select the [**...**] next to the appropriate option. Click **OK.** **Note:** The Simulation Highlight options are not saved in the chart.
Enable Scale Text	Causes both the text and lines in a diagram to change size when you zoom into and out of the canvas. By default, zoom operations do not affect the text. **Note:** The Enable Scale Text option is not saved in the chart.
Advanced: Filled Boxes	Temporarily displays boxes filled with five colors to make it easier to see the box hierarchy in complex charts. (The system automatically selects the five colors, based on your chart hierarchy. If your chart has more than five levels, the system repeats this process.) **Note:** The Filled Boxes option is not saved in the chart. This option can cause slower system performance.
Advanced: Fill Colors	Allows you to select colors to fill boxes. **Note:** The Fill Colors option is not saved in the chart.
Advanced: Carriage Return is New Line	By default, when you are editing text, each time you press Return, a new line displays. Pressing Ctrl + Return ends text editing. This option reverses these settings (that is, you press Ctrl + Return for a new line and Return to end the edit). **Note:** The Carriage Return is New Line option is not saved in the chart.
Advanced: Enabled Reshaping	Provides more precise control when stretching polygons. • When enabled, only particular areas of the polygon are stretched. • When disabled, the entire polygon is stretched, which may not give the desired result. **Note:** The Enabled Reshaping option is not saved in the chart.
Preferences Management	Allows you to modify general preferences for all graphic editors or for only the current graphic editor.

Working with Graphic Editors

The following sections explain how to perform general tasks in all graphic editors:

- Starting a Graphic Editor
- Creating a New Chart or Diagram
- Drawing Operations in Graphic Editors
- General Operations in Graphic Editors

Starting a Graphic Editor

To start a graphic editor:

- From the **Charts** tab of the Rational Statemate main window, double-click on a chart to open it.
- From the Rational Statemate main window, click the **Graphic Editors** icon . If you have selected a chart, that chart is opened in the appropriate graphic editor. Otherwise, the Open Chart window opens (see Creating a New Chart or Diagram from the Open Chart Window).
- From another graphic editor, use the **File > Open, File > Open Sub-chart**, or **File > Open Parent** menu options.

Creating a New Chart or Diagram

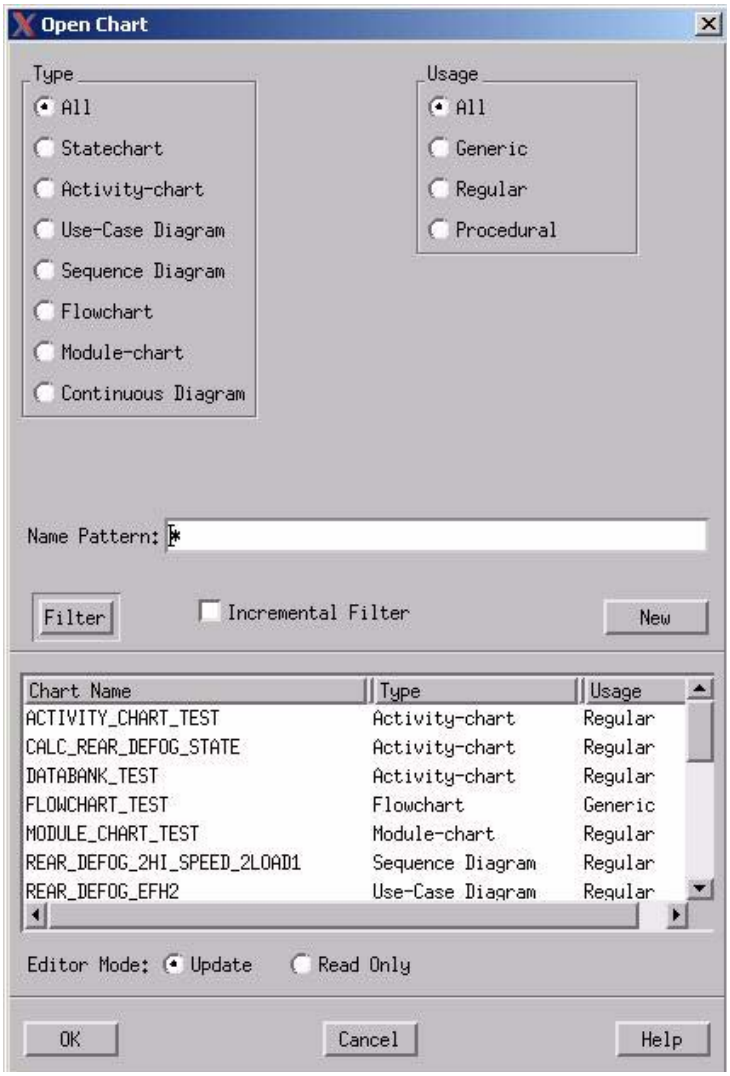
A new chart or diagram is created:

- From within a graphic editor.
- From the **Open Chart** window.

Creating a New Chart or Diagram from the Open Chart Window

To create a new chart or diagram:

1. Click the **Graphic Editors** icon in the main Rational Statemate main window. The **Open Chart** window opens.



2. From the **Type** section, select the type of chart you want to create. The possible values are as follows:

   - **All** - Used to open existing charts; specifies that all types of existing charts are to be displayed.
   - **Statechart** - For information on statecharts, see **Statecharts**.
   - **Activity Chart** - For information on activity charts, see **Activity Charts**.
   - **Use-Case Diagram** - For information on use case diagrams, see **Use Case Diagrams**.
   - **Sequence Diagram** - For information on Sequence Diagrams, see **Sequence Diagrams**.
   - **Flowchart** - For information on flowcharts, see **Flowcharts**.
   - **Module-chart** - For information on module charts, see **Module Charts**.

3. In the **Usage** area, select a particular usage. The possible values are as follows:

   - **All** - Used to open existing charts; specifies that all usages of existing charts are to be displayed.
   - **Generic** - Generic charts enable reuse of parts of a specification. A generic chart makes it possible to represent common portions of the model as a single chart that can be instantiated in many places, and in this it is similar to a procedure in a conventional programming language.

     Generic charts are linked to the rest of the model via parameters; no other elements (besides the definitions in global definition sets) are recognized by both generic charts and other portions of the model.

- **Regular** - A non-generic chart.
- **Procedural** - A specialized derivative of a statechart. Procedural Statecharts:
  - Are executed entirely in one step.
  - Must contain a termination connector.
  - When called, run from the default to the termination connector (including any loops) within a single step.

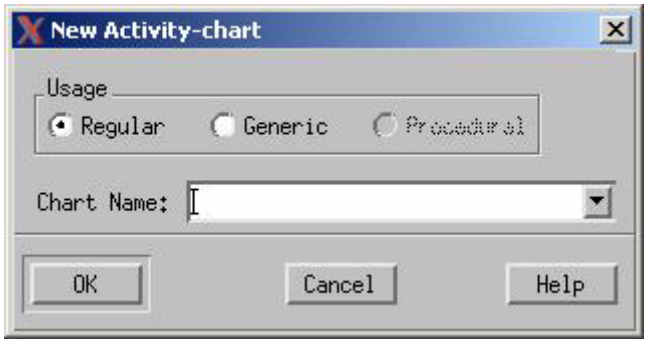
4. In the **Name Pattern** box, enter a name for the chart or diagram you are creating.
5. When done, click **New**. The appropriate graphic editor opens.

Creating a New Chart or Diagram with a Graphic Editor

To create a new chart using the current graphic editor:

1. From within the graphic editor, select **File > New**.

   The **New Chart** window opens with the name of the chart editor you are currently in.



2. Specify **Regular**, **Generic**, or **Procedural** from the Usage area. For more information on these choices, see Creating a New Chart or Diagram from the Open Chart Window.
3. In the **Chart Name** textbox, enter the name or use the selection box to select a name. (The selection list is of the unresolved charts in the workarea.)
4. Select **OK** to confirm your choices.

   The current chart is closed (you are prompted to save changes first) and a new graphic editor opens.

Drawing Operations in Graphic Editors

When drawing in a graphic editor, place the cursor at the desired location for the upper, left-hand corner of the activity and click and drag to the desired location of the lower, right-hand corner of the activity. A ghost image shows the activity outline.

Graphic editors support the following drawing operations:

- **Drawing Boxes**
- **Drawing Lines**
- **Drawing Connectors**
- **Editing Text**
- **Selecting Elements**
- **Labeling Elements**
- **Moving Elements**
- **Copying Elements**
- **Resizing Elements**
- **Deleting Elements**
- **Constraining Graphic Operations**

Note

Not all graphic editors support all of the various drawing operations.

Drawing Boxes

To draw a box, select the **Box** icon.

Use one of the following procedures to draw boxes:

- Double-click to draw a default size box which is auto-named.
- Click and drag to draw a rectangle. When you hold down **Shift**, the resulting box is square.
- Click to position three corners of the box, then double-click (or use the middle mouse button) on the first corner to complete the box.
- Click to position one corner, and then double-click (or use the middle mouse button) to position the opposite corner. A rectangular box is created.

Drawing Lines

To draw a line, select a line icon (**Data Flow** or **Control Flow**).

Use one of the following procedures to draw splines and straight-line segments:

- Click and drag the cursor to draw the line. When you hold down the Shift key, the resulting line is horizontal or vertical.
- Click to position the points on the line, then double-click (or use the middle mouse button) to complete the line.
- To end a line on an invalid end point, press the middle mouse button. An invalid line is denoted as such with a line at the end of the arrow head.

Drawing Connectors

To draw a connector:

1. Select a connector icon.
2. Click to place the connector.

Editing Text

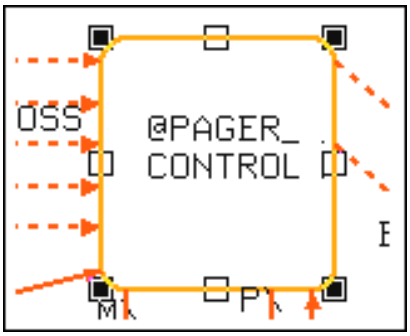
To edit text:

1. Select the text to be edited. The text is highlighted.
2. Click on the selected text a second time. Now the text is displayed in a reverse video box and the pointer changes to an I-beam.
3. Edit the selection.
4. Move the pointer away from the text and click to end the edit.

If the text in a text box is difficult to read, click the **Beautify** button to reformat the text to improve readability. You may also adjust the indention of the text with the setting in the **Beautify Indent Size** preference.

Selecting Elements

When you select one or more graphic elements, the current selection is marked with filled squares, called selection handles. If the graphic editor is in selection mode (you are not drawing something), the current selection is also marked with hollow squares, called stretch handles.



Different kinds of drawing elements are marked with selection handles in the following ways:

- **Boxes** - Have a marker placed on each corner.
- **Arrows and lines** - Have a marker placed on each end of the line, and on each control point on the line. Selecting an arrow automatically selects its label. A label cannot become disconnected from its arrow, even when moved.
- **Connectors** - Have a marker placed in the center of the connector. In this case, the marker color is different than the connector color, so that you can see it more easily.
- **Circles** - Have four markers, one each side, and one at the top and bottom on the edge of the circle.
- **Text** - Has a dotted box surrounding the words.

Note

There are usually eight stretch handles at the corners and edges of an imaginary rectangle that contains all the selected elements. If the markers are too close to each other, the number of stretch handles is reduced, which can make it impossible to resize an element without zooming in to that area of the diagram.

Labeling Elements

To label elements, do **one** of the following:

- Immediately after you draw an element, type the label and click to move it to the desired location. The cross-hair pointer must intersect a line or be inside a box.
- Click the label icon and type the label, use the mouse to move it to the desired location, and click to position it. The tip of the pen pointer must be on the line.

  Note: Do not use the Caps Lock option because it might prevent you from making further selections.

Moving Elements

To move elements:

1. Select one or more elements to move.
2. Position the pointer over the selected elements. To move text, position the pointer over the square in the lower left-hand corner of the dashed box that delimits the text element (as shown). The pointer changes to a four-way arrow.
3. Click and drag to position the element where you want it to appear.

Note: If an element becomes erroneous during a move operation, such as overlapping boxes, the erroneous elements are highlighted (with “Xs) and they will not be part of the model until they are corrected.

Copying Elements

To copy elements:

1. Select one or more elements to copy. It changes to a four-way arrow.
2. **Press Ctrl** and click and drag the pointer to position the second image where you want the copy to appear.

Note: If an element becomes erroneous during a copy operation, such as overlapping boxes, the erroneous elements are highlighted (with “Xs), and they will not be part of the model until they are corrected.

Resizing Elements

To resize elements:

1. Select one or more elements to resize. Resize handles surround the element.
2. To resize the element in one dimension, position the pointer on the closest midpoint (so it displays as a bar). To resize the element in two dimensions at the same time, position the pointer on the closest corner (so it displays as an angle bracket).
3. Click and drag to resize the element.

Note

If an element becomes erroneous during a stretch operation, such as overlapping boxes, the erroneous elements are highlighted (with “Xs) and they will not be part of the model until they are corrected.

Deleting Elements

To delete elements:

1. Select the element to be deleted.
2. Press **Delete**.

If you delete an element by mistake, select **Edit > Undo**.

Note

If **Delete** does not work on your system, try **Backspace**.

Constraining Graphic Operations

Pressing **Shift** while drawing, stretching, or moving elements can greatly facilitate the process of creating professional-looking charts.

Copying, moving, and one-dimensional drawing and stretching operations become constrained to a horizontal or vertical direction only. For example, to draw perfectly horizontal lines, hold down **Shift**.

Two-dimensional drawing and stretching operations become constrained to a diagonal direction only. For example, to draw a perfectly square box or to stretch an box without changing its shape, hold down **Shift**.

General Operations in Graphic Editors

The following sections describe the more common operations performed in graphic editors. For descriptions of all the menu options for each editor, see **Graphic Editor Menus**.

Opening the Properties Window for Elements

To open a Properties window for selected elements:

1. Select the elements. For instructions on how to select elements, see **Selecting Elements**.
2. Select **Tools > Properties** or, if the element is a basic element (not a sub-chart), double-click on the element.

Displaying Element Properties

To display a summary of the properties for selected elements:

1. Select the elements. For instructions on how to select elements, see **Selecting Elements**.
2. Select **Tools > Info**. An Info window opens. Use the scroll bars to scan the window.

Opening the Properties Window for an Entire Chart

To open the Properties window for an entire chart, select **Tools > Chart Properties**.

Note

This operation is particularly useful for generic charts, because properties are used to define the formal parameters for the chart.

Displaying Chart Properties

To display a summary of the properties for a chart, select **Tools > Chart Info**. A Chart Info window opens. Use the scroll bars to navigate the window.

Displaying Subroutine Properties

Note

This selection is only available for the activity chart editor.

When an activity that was previously bound to a subroutine is selected, use this option to open the **Properties** window and the dialog box with the subroutine implementation (body). If there are multiple implementations currently defined for the subroutine, the implementation that is identified in the Select Implementation field is opened.

(Set the General preference **Open Code Editor on Entering a subroutine** to go directly to the subroutine template code editor.)

This option is also available from the right-click popup menu, when the activity is selected.

Opening a Simulation Execution Window

Note

This selection is not available for the module-chart editor.

To open a simulation execution window directly from your open chart, select **Tools > Simulation**. A Simulation Execution window opens.

For more information on simulation tool, see the *Rational Statemate Simulation Reference Manual*.

Invoking the Check Model Tool

To start the Check Model tool to perform a correctness and completeness checks on an open chart, select **Tools > Check Model**. The tests are performed and the report on them is displayed on your screen.

For more information on the Check Model Tool, see the *Check Model User's Guide*.

Invoking the RT Interface

To start the Rational Statemate-to-DOORS RT Interface, select **Tools > RT Interface-Accessing Requirements Tracing Tools**.

For more information on the RT Interface, see **Rational DOORS RT Interface**s.

Closing a Chart

To close a chart, select **File > Close**.

If the chart includes graphic or textual changes not yet saved, you are prompted to save these before the chart is exited. The editor remains open and is now labeled Empty Editor. (You will be prompted to name the chart when you exit.)

Saving a Chart

To save the open chart, select **File** > **Save**.

Opening a Parent Chart

To open a new graphic editor for the chart immediately above the current chart hierarchy, select **File** > **Open Parent**.

Opening a Sub-Chart

To open a new graphic editor for an existing sub-chart (generic or off-page):

1. Select the box.
2. Select **File** > **Open Sub-Chart**.

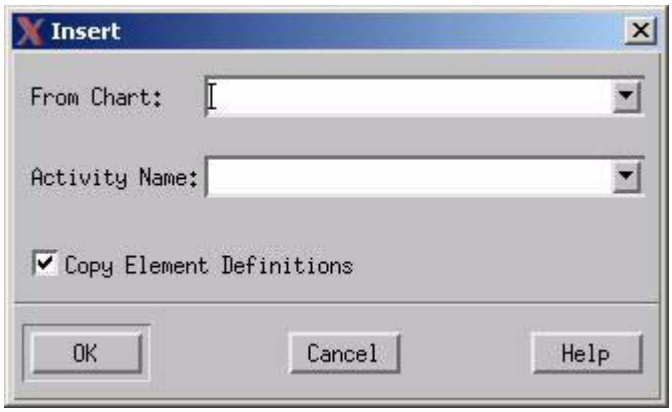
Note

This operation is only valid if the selected box name contains an off-page (@NAME) reference or a generic chart instantiation (I<NAME) and the chart exists in the workarea.

Inserting a Chart

To insert the contents of another chart into the current graphic editor window:

1. Specify **File** > **Insert Chart**. The **Insert** dialog box displays.



2. In the **From Chart** box, enter or select the chart to be inserted.
3. In addition to being able to insert an entire chart, you can insert a single activity (state or module) for the specified chart. In the **Type Name** text box, click the down arrow to see a list of possible activities (states or modules) to be inserted.

   Note: If the selected chart has no sub-activities, the list is blank.

4. Optionally, deselect **Copy Element Definitions** if you do not want textual elements included.

Creating a Sub-Chart

To create a new chart from the currently selected box:

1. Select the box.
2. Select **File** > **Create Sub-Chart**.

Exiting the Graphic Editor

To close an open chart and exit from the graphic editor, select **File** > **Exit**.

Working with Charts and Diagrams

The following sections explain in general terms how to work with charts and diagrams:

- **Activity Charts**
- **Module Charts**
- **Statecharts**
- **Use Case Diagrams**
- **Sequence Diagrams**
- **Flowcharts**

Activity Charts

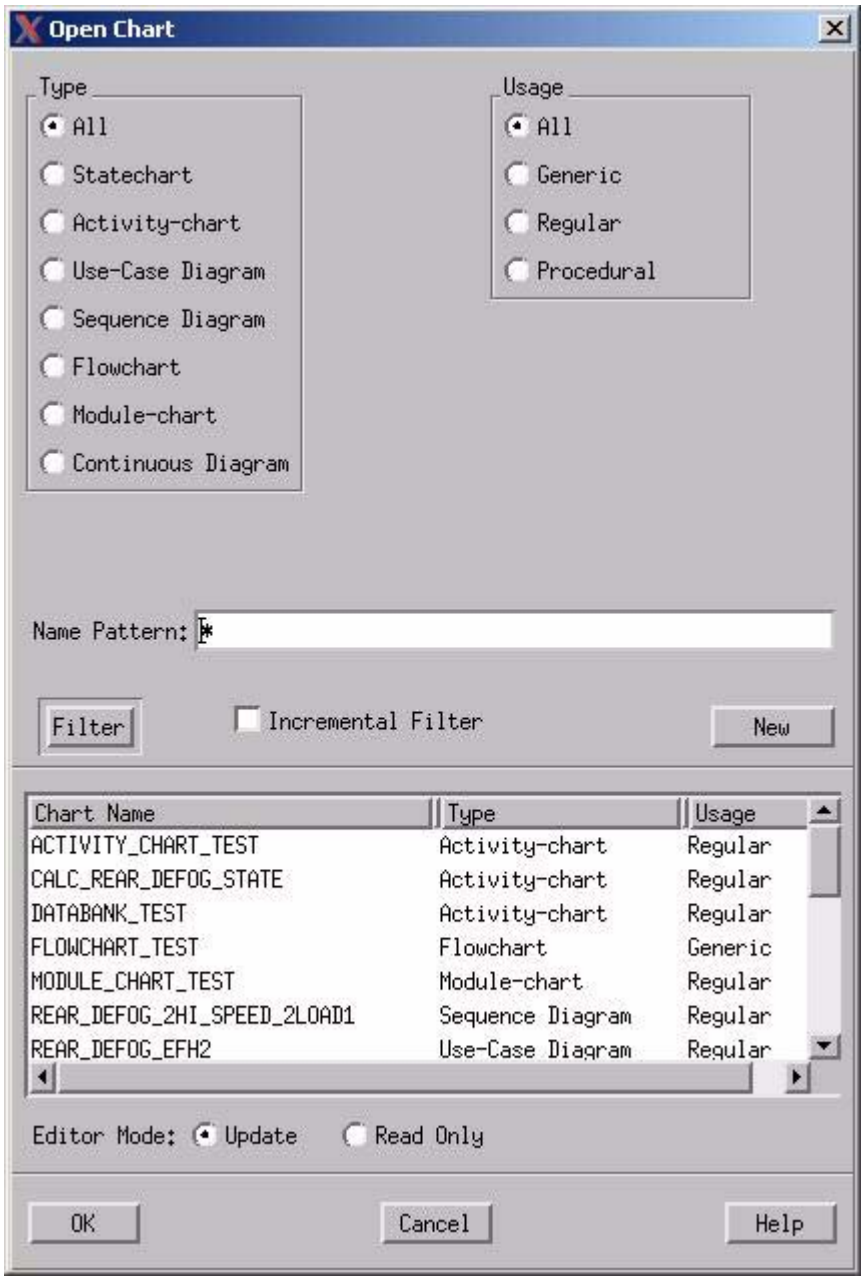
An activity chart describes the functional view of the system using activities as the primary building block. This is sometimes referred to as the process-oriented view. A system description can contain one or more activity charts. Activity charts, which can be connected to module charts, describe the functionality of individual modules.

Activity charts can also be connected to statecharts, which either define the behavior of individual activities or control groups of activities as a control-activity.

The following sections describe in more detail how to use activity charts.

Accessing an Activity Chart

To access an activity chart, see **Starting a Graphic Editor**. The activity chart editor opens, as illustrated in the following figure.


Open Chart
Type
All
Statechart
Activity-chart
Use-Case Diagram
Sequence Diagram
Flowchart
Module-chart
Continuous Diagram
Usage
All
Generic
Regular
Procedural
Name Pattern: *
Filter
Incremental Filter
New
Chart Name
Type
Usage
ACTIVITY_CHART_TEST
Activity-chart
Regular
CALC_REAR_DEFOG_STATE
Activity-chart
Regular
DATABANK_TEST
Activity-chart
Regular
FLOWCHART_TEST
Flowchart
Generic
MODULE_CHART_TEST
Module-chart
Regular
REAR_DEFOG_2HI_SPEED_2LOAD1
Sequence Diagram
Regular
REAR_DEFOG_EFH2
Use-Case Diagram
Regular
Editor Mode: Update
Read Only
OK
Cancel
Help

Activity Chart Icons

The following icons support drawing and naming operations in an activity chart. For general information on drawing and naming operations in Rational Statemate, see **Drawing Operations in Graphic Editors** and **General Operations in Graphic Editors**.

General Activity Icons

An activity is the primary graphic element in activity charts that represent a function in the functional view of the system. An activity represents something that transforms inputs into outputs.

There are three types of activities:

- Internal activities (solid rectangle)
- External activities (dashed rectangle)
- Control activities (rounded rectangle)

Activities can be allocated to modules (structure) and can contain statecharts. You can specify the behavior of an activity by connecting it to a subroutine.

- Procedure-like activities can be connected to procedures in any of the languages supported.
- Internal primitive activities (reactive-controlled and reactive-self) can be connected to tasks (no mini-specs or decomposition is allowed).
- External activities can only be connected to tasks.

	Create Internal Activity - creates an internal activity.
	Name Existing Activity - Names an existing activity.
	Create Control Activity - creates a control activity. The control activity senses and controls the status of sibling activities. If an activity does not contain a control activity, the children of the activity are active when the parent is active. Control activities cannot be hierarchically decomposed. Only one control activity is allowed per activity hierarchical level. A control activity cannot have any subactivities and is specified by an off-page statechart or flowchart. An @ symbol precedes the title of control activities.

	Create External Activity - creates an external activity, which is an activity outside the scope of the topmost activity in a particular activity chart. Because activity charts are hierarchical, an external activity is usually resolved to a box in a chart higher in the chart hierarchy. However, an external activity can be resolved to a box that is an internal activity at a higher level. In this case it remains simply an external activity when referenced in the lower chart. External activities cannot be hierarchically decomposed. There can be more than one occurrence of the same external activity.

Data Flow Icons

Two types of flow lines are allowed in activity charts: data flow lines, drawn as solid arrows, and control flow lines, drawn as dashed arrows.

	Create Data-Flow - Establishes a data flow between two activities. Data flow lines carry information that is used in computations and data-processing operations
	Create Control Flow - Establishes a control flow between two activities. Control flow lines carry information or signals that are used in making control decisions, for example, commands or synchronization messages.
	Label Existing Flow Line - Enables you to name a flow line that denotes either a single information element that flows along the line or a group of such elements. Flow labels in activity charts, module charts, and information-flow elements can be any primitive (variable) data element (event, condition, data-item) or information flow. In addition they can be elements on any level of a primitive data element (array element, array slice, and record/union field). Array elements can use only literal constants.

Connector Icons

Connector icons are circular or oval graphic elements used in charts to join and divide arrows or to enable an arrow to exist on multiple pages. Their purpose is to clarify a specification by reducing the number of arrows.

	Create Junction Connector - creates a junction connector, which reduces the number of lengthy flow lines by connecting different elements together. These elements then form a single flow line that emanates from or enters a common box or connector.
	Create Composition Connector - creates a composition connector, which can only be used with a record data-item. The composition connector directs the components of a record to two different target activities. For example, a data-store called RANGE has a record type of data-item with two components called LOW_LIMIT and HIGH_LIMIT. If the flow goes out from the data-store, the composition connector splits the record into the two components. The composition connector can also go in the other direction, where multiple flow lines labelled with the record components enter the connector and the single flow line denoting the record flow emanates from it.
	Create Diagram Connector - creates a diagram connector, which connects a target and a source that are far from each other. Using this type of connector eliminates the need for long arrows.
	Create To-Control Connector - creates a to-control connector, which connects only to the control activity. Using this type of connector eliminates the need for long arrows.

Router Icons

Router icons enable you to create routers, which conceal multiple flow lines to make an activity chart more readable. For more information on using routers, see **The Router Element**.

	Create Router - creates a router block used within the scope of the top-most activity in a particular activity chart.
	Create External Router - creates router blocks used outside the scope of the top-most activity in a particular activity chart. Because activity charts are hierarchical, an external router is always resolved to an internal router in a chart higher in the chart hierarchy.

Miscellaneous Icons

	Select Mode - places the editor in Select mode, so you can access the editing options the editor supports.
	Create Data-Store - creates a data-store, which contains information on activities for later use. Data-stores can also be used to total large volumes of data, continuously accumulating over time.
	Create Textual Note - creates free text that you can use to annotate the chart.

	Create Combinational Assignment - creates a combinational assignment, which is the expression used to assign a value to a combinational element, with syntax like the following:
	`X := Y1 when C1 else`
	`Y2 when C2 else`
	`...`
	`Yn`
	where X is a variable condition or data-item, Y1 to Yn are expressions, and C1 to Cn are condition expressions.
	Combinational elements represent asynchronous behavior; they are elements whose value is assigned continuously (rather than evaluated once each step).

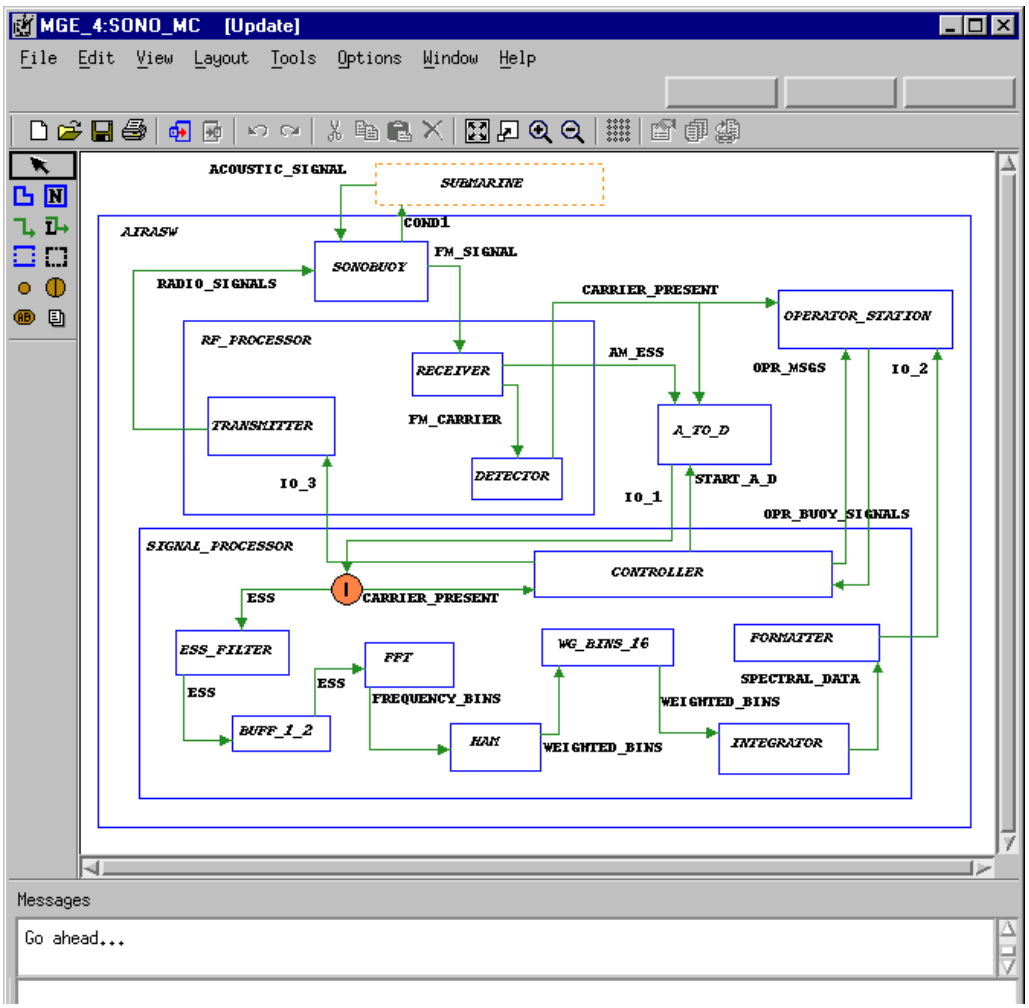
Module Charts

A module chart describes the structural view of the system using modules as the primary building block. A system description can contain one or more module charts. They are located at the top of the chart hierarchy in a system model. Module charts can be connected to activity charts, which describe the functionality of individual modules.

The following sections describe in more detail how to use module charts.

Accessing a Module Chart

To access a module chart, see Starting a Graphic Editor. The module-chart editor displays.



Module Chart Icons

The following icons support drawing and naming operations in a module chart. For general information on drawing and naming operations in Rational Statemate, see **Drawing Operations in Graphic Editors** and **General Operations in Graphic Editors**.

General Module Icons

Modules are the primary graphic element used in module charts. Modules are used to represent the structure of the system. There are two types of modules: internal (solid rectangle) and external (dashed rectangle). The functionality of a module is shown by an activity chart.

	Create Internal Module - creates an internal module.
	Name Existing Module - names an existing module.
	Create Environment Module - creates an environmental module, which is a module outside the scope of the topmost module in a particular module chart. Because module charts are hierarchical, an environmental module is usually resolved to a box in a chart higher in the chart hierarchy. However, an environmental module can be resolved to a box that is an internal module at a higher level. In this case it remains simply an environmental module when referenced in the lower chart.

Data Flow Icons

Two types of flow lines are allowed in activity charts: data flow lines, drawn as solid arrows, and control flow lines, drawn as dashed arrows.

	Create Flow Line - establishes a data flow between two activities. Typically, data flow lines carry information that is used in computations and data-processing operations
	Label Existing Flow Line - enables you to name a flow line that denotes either a single information element that flows along the line or a group of such elements. Flow labels in activity charts, module charts, and information-flow elements can be any primitive (variable) data element (event, condition, data-item) or information flow. In addition they can be elements on any level of a primitive data element (array element, array slice, and record/union field). Array elements can use only literal constants.

Connector Icons

Connector icons are circular or oval graphic elements used in charts to join and divide arrows or to enable an arrow to exist on multiple pages. Their purpose is to clarify a specification by reducing the number of arrows.

	Create Junction Connector - creates a junction connector which reduces the number of lengthy flow lines by connecting different elements together. These elements then form a single flow line that emanates from or enters a common box or connector.
	Create Composition Connector - creates a composition connector, which can only be used with a record data-item. The composition connector directs the components of a record to two different target activities. For example, a data-store called RANGE has a record type of data-item with two components called LOW_LIMIT and HIGH_LIMIT. If the flow goes out from the data-store, the composition connector splits the record into the two components. The composition connector can also go in the other direction, where multiple flow lines labelled with the record's components enter the connector and the single flow line denoting the record flow emanates from it.
	Create Diagram Connector - creates a diagram connector, which connects a target and a source that are far from each other. Using this type of connector eliminates the need for long arrows.

Miscellaneous Icons

	Select Mode - places the editor in Select mode, so you can access the editing options the editor supports.
	Create Storage Module - creates a data-store, which contains information on module for later use. Data-stores can also be used to total large volumes of data, continuously accumulating over time. Data-stores are always basic; they cannot contain other data-stores or modules. Sibling data-stores must have unique names.
	Create Textual Note - creates free text that you can use to annotate the chart.

Statecharts

Statecharts describe the system's behavior over time, including the dynamics of activities, their control and timing behavior, the states and modes of the system, and the conditions and events that cause modes to change and other occurrences to take place. It thus also provides answers to questions about causality, concurrency, and synchronization.

Statecharts constitute an extensive generalization of state-transition diagrams. They allow for multi-level states, decomposed in an and/or fashion, and thus support economical specification of concurrency and encapsulation. They incorporate a broadcast communication mechanism, timeout, and delay operators for specifying synchronization and timing information, and a means for specifying transitions that depend on the history of the system's behavior.

You can set transition priority by adding a priority value (positive integer number) and assigning it to a transition. This creates a graphical notion of the transition priority.

Each element in the statechart has properties, which can contain additional information. For example, an event element can be used to define a compound event by an expression involving other events and conditions.

Note

Rational Statemate supports a special type of statechart known as a Procedural Statechart. A Procedural Statechart is a specialized derivative of a statechart that does the following:

– Is executed entirely in one step.

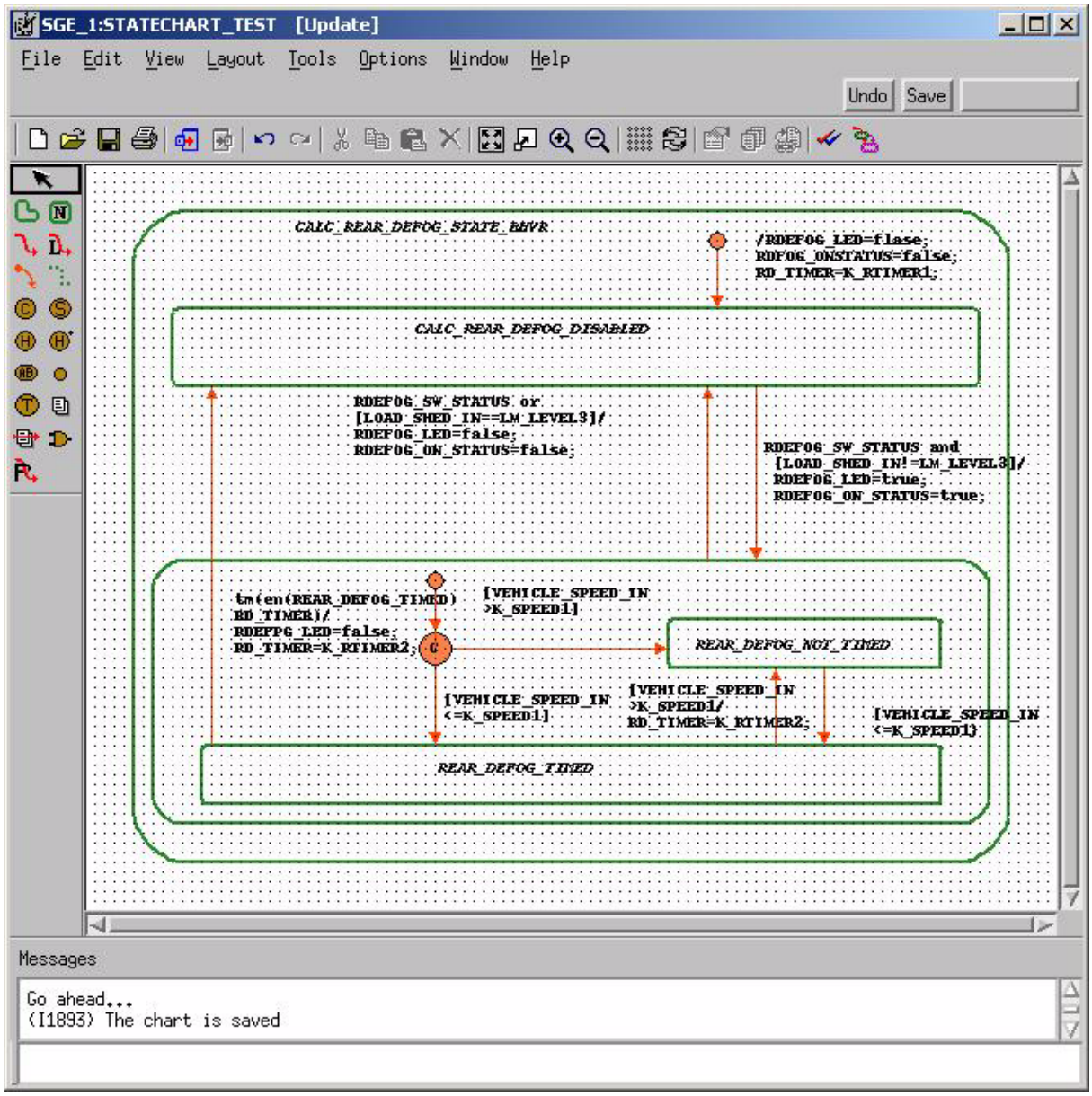
– Must contain a termination connector.

– When called, runs from the default to the termination connector (including any loops) within a single step.

The following sections describe in more detail how to use statecharts.

Accessing a Statechart

To access a statechart, see **Starting a Graphic Editor**. The statechart editor displays.



Statechart Icons

The following icons support drawing and naming operations in a statechart. For general information on drawing and naming operations in Rational Statemate, see **Drawing Operations in Graphic Editors** and **General Operations in Graphic Editors**.

General State Icons

	Create State - creates an OR state.
	Name Existing State - names a state.

Data Flow Icons

Two types of flow lines are allowed in statecharts: data flow lines, drawn as solid arrows, and control flow lines, drawn as dashed arrows. Typically, data flow lines carry information that is used in computations and data-processing operations.

Note

Default transitions do not have a source state and do not support triggers.

	Create Transition - creates an event that makes the model leave one state and enter another. You label each transition with the trigger that causes it to be taken and, optionally, with an action. Separate the trigger from the action with a slash.
	Label Existing Transition - allows you to name a flow line that denotes either a single information element that flows along the line or a group of such elements.
	Create Default Transition - establishes a control flow between two activities. Control flow lines carry information or signals that are used in making control decisions, for example, commands or synchronization messages.
	Create And-Line - divides a state into orthogonal (or parallel) components.

Connector Icons

Connector icons are circular or oval graphic elements used in charts to join and divide arrows or to enable an arrow to exist on multiple pages. Their purpose is to clarify a specification by reducing the number of arrows.

	Create Condition Connector - creates a condition connector used to create a branch in a transition, typically labeled with conditions. A condition connector emphasizes a choice based on a condition (e.g., x>1, x<1, x=1). The triggers must be mutually exclusive.
	Create Switch Connector - creates a switch detector used to create a branch in a transition, typically labeled with data items or events. A switch connector emphasizes a choice based on events (e.g., input_1, input_2, input_3). The triggers must be mutually exclusive.
	Create History Connector - creates a history connector used to return to the most recently visited state on the same level of hierarchy.
	Create Deep History Connector - creates a deep history connector, used to return to the most recently visited state at all levels of hierarchy. A deep history connector has an asterisk after it (H*).
	Create Diagram Connector - creates a diagram connector, which connects a target and a source that are far from each other. Using this type of connector eliminates the need for long arrows.
	Create Junction Connector - creates a junction connector, which reduces the number of lengthy flow lines by connecting different elements together. These elements then form a single flow line that emanates from or enters a common box or connector.
	Create Termination Connector - creates a termination connector that terminates the statechart's behavior.

Miscellaneous Icons

	Select Mode - places the editor in Select mode, so you can access the editing options the editor supports.
	Create Combinational Assignment - creates a combinational assignment, which is the expression used to assign a value to a combinational element, with syntax like the following: X := Y1 when C1 else Y2 when C2 else ... Yn where X is a variable condition or data-item, Y1 to Yn are expressions, and C1 to Cn are condition expressions. Combinational elements represent asynchronous behavior; they are elements whose value is assigned continuously (rather than evaluated once each step).
	Create Transition Note - creates free text that you can use to annotate the chart.
	Create External Router - creates free text that you can use to annotate the chart.
	Set Transitions Priority - creates free text that allows you to write a priority value (positive integer number) and assign it to a transition.

Associating a Statechart with an Activity

Statecharts describe the behavior of an activity that is defined in an activity chart, and the two can be associated within the scope of the project itself.

To associate a statechart with a corresponding activity:

1. Open the corresponding activity chart.
2. Using the **Create Control Activity** icon, draw a control activity and label it with the name of the corresponding statechart preceded by an at (@) sign. You might, for example, label it like the following:

   ```
   @CALC_REAR_DEFOG_STATE_BHVR
   ```

3. Select **File > Save**, then **File > Exit** to write the association and quit the chart.

"Only Once" Test Benches

Two types of test benches are available in statecharts:

- Run Every Step
- Run Only Once

The Every Step Testbenches run the task `Testbench`. This task executes every step. The Only Once Testbenches run the task `TestBench_run_once`. The `TestBench_run_once` task executes only once on startup. In the OSI's (`mainloop_sc` and `mainloop_sc_ext`), the call to the generated function:

- `TestBench_run_once` is (by default) put in the file `user_code.c` (<profile>.c), after the call to the call to `TASKINIT()`.
- The call to the function `TestBench_run_once` is wrapped with the preprocessor flag `TB_ONLY_ONCE`.
- The preprocessor flag is generated in the file `cmp_flg.h` if there are such Only Once testbenches.
- Add a new Design-Attribute to Statechart:
  - Name: "When used as Testbench, run"
  - Possible Values: "Every Step" - runs under the Task TestBench, which runs every step, used as the default value.
  - "Only Once" - runs under the Task "TestBench_run_once," which runs only once on system startup.
  - The Design-Attributes are available for update from the OSI's:

    ```
    MAINLOOP_SC
    MAINLOOP_SC_EXT
    DEFAULT
    ```

Use Case Diagrams

Use case diagrams illustrate, at a very high level, the relationship between “actors” (whoever or whatever interacts with the system being designed) and the system itself. Each use case describes the sequence of events of an actor (human or system) using the system to complete some process. They provide a natural high-level view of the intended external functionality of the system that is understandable by engineers and non-engineers alike.

A use case is a generic description of an entire transaction involving several elements. It can also describe the behavior of a set of elements, such as an organization. A use-case model thus presents a collection of use cases and is typically used to specify or characterize the behavior of a whole system or a part of a system together with one or more external actors that interact with that system. An individual use case can have a name (although it is typically not a simple name). Its meaning is often written as an informal text description of the external actors and the sequences of events between elements that make up the transaction.

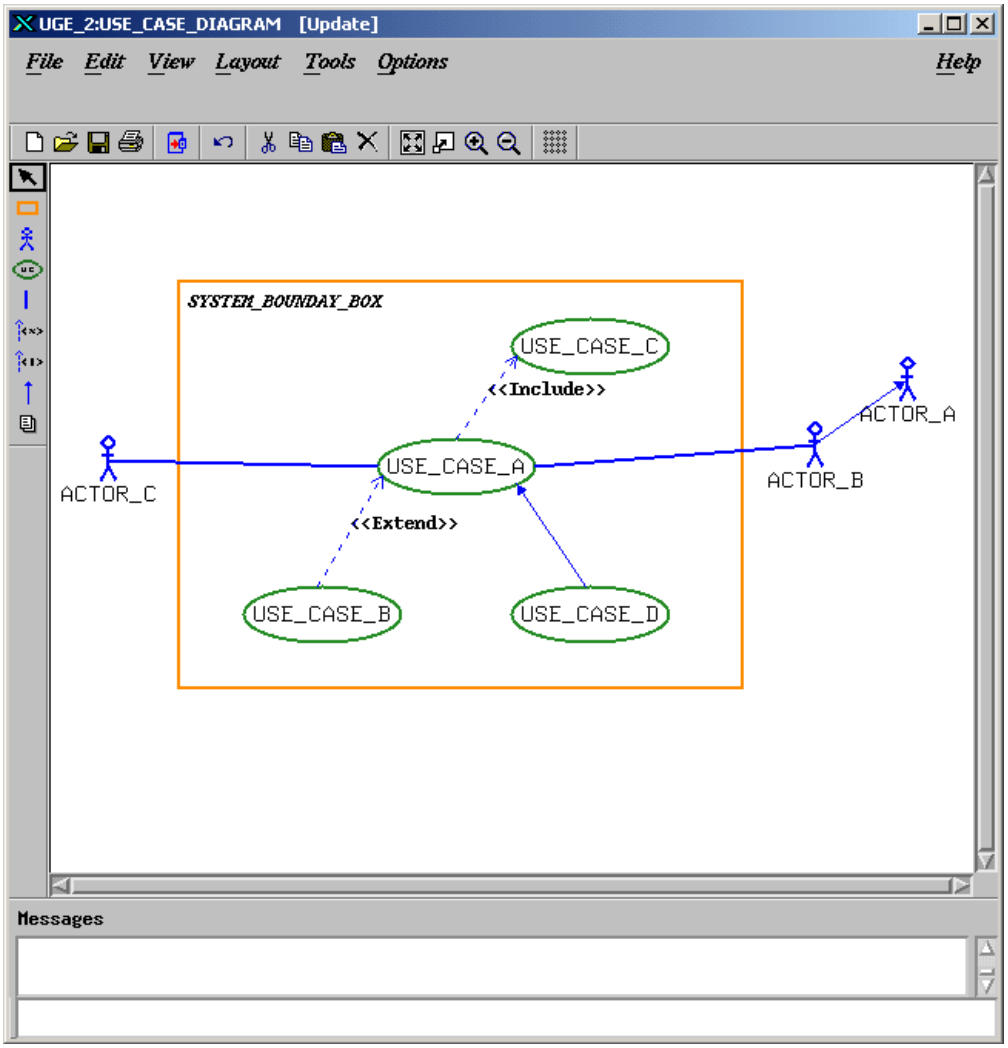
Note that use case diagrams treat the system itself as transparent, because the internals of the system are not of interest in this type of diagram. Use case diagrams capture key uses of the system and are therefore an excellent means of illustrating and explaining project requirements at a very high, non-technical level.

The following sections describe in more detail how to use case diagrams.

Accessing a Use Case Diagram

To access a use case diagram, see **Starting a Graphic Editor**. The use-case graphic editor opens, as illustrated in the following figure.

Three actors, represented as stick figures, are shown interacting with one another and with a particular subsystem of the system. The diagram also depicts possible interactions between individual subsystems.


UGE_2:USE_CASE_DIAGRAM [Update]
File Edit View Layout Tools Options Help
SYSTEM_BOUNDAY_BOX
USE_CASE_C
<<Include>>
ACTOR_A
ACTOR_C
USE_CASE_A
ACTOR_B
<<Extend>>
USE_CASE_B
USE_CASE_D
Messages

Use Case Diagram Icons

The following icons support drawing and naming operations in a use case diagram. For general information on drawing and naming operations in Rational Statemate, see **Drawing Operations in Graphic Editors** and **General Operations in Graphic Editors**.

General Use-Case Icons

	Create Boundary Box - creates a boundary box. A boundary box depicts the limits of the system and is shown as a rectangle spanning all the use cases in the system.
	Create Actor - creates an actor. An actor is any entity (person or system) that performs certain roles in the system defined by a boundary box. An actor is depicted as a stick figure and only interacts with a use case.
	Create Use Case - creates a use case. A use case is a distinct piece of functionality within a system and is shown as an oval.

Association and Relational Icons

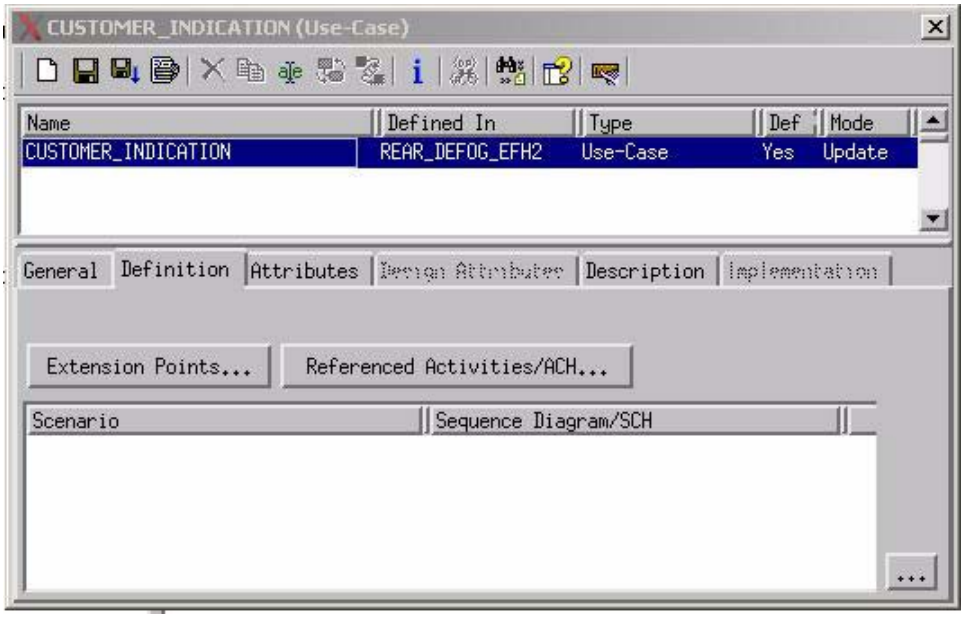
	Create Association - creates an association. Associations depict the interaction between an actor and a use case. This is the only relationship between actors and use cases. The figure shows all the actors as having an association with use case A.
	Create Extend Relation - creates an extended relationship. An extended relationship indicates that a “child” use case adds to the existing functionality and characteristics of the “parent” use case. An extended relationship is depicted with a directed arrow having a dotted shaft, similar to the include relationship. The tip of the arrowhead points to the parent use case and the child use case is connected at the base of the arrow. The figure shows an extended relationship between use case B and use case A.
	Create Include Relation - creates an include relationship. An include relationship indicates that one use case uses the functionality of another use case as a part of its process flow. An include relationship is depicted with a directed arrow having a dotted shaft. The figure shows an include relationship between use case A and use case C.
	Create Generalization Relation - creates a generalization relationship. A generalization relationship is a parent-child relationship between use cases in which the child use case performs the same task as the parent but in a different way. A generalization is shown as a directed arrow with a triangle arrowhead. The child use case is connected at the base of the arrow. The tip of the arrow is connected to the parent use case. The figure shows a generalization relationship from use case D to use case A.

Miscellaneous Icons

	Put GE into Select Mode - places the editor in Select mode, so you can access the editing options the editor supports.
	Create Textual Note - creates free text that you can use to annotate the diagram.

Use Case Diagram Properties

The use cases in a use case diagram can be described and linked with a Rational Statemate model through the use-case properties.



A use case has the following special properties:

- **Use-Case External Description** - describes the use case, using a customizable template.
- **Scenario List** - list of scenarios associated with the use case. Each scenario can be linked to a sequence diagram, and a set of attributes describing it.

Multiple instances of both actors and use cases are allowed in the same use case diagram. That means that in the figure, there could be another ellipse (use case) named “USE_CASE_A” that models the same use case.

An actor and a use case cannot have the same name in the same use case diagram.

You can use full path names for actors and use cases. For example, you can see (and thus reuse) USE_CASE_A defined in the figure from another use case diagram by using its full-path name USE_CASE_DIAGRAM:USE_CASE_A.

Attribute fields are available to use case diagram elements, with customizable attribute template-files.

Note

The template of the use-case description file is found in `STM_ROOT/etc/knl`, named "use_case_description_template" and is copied to the workarea, to the directory "config," with the addition of an extension taken from the "General" preference: "Use Case Description File Type," default to "RTF" on Windows, and to "txt" on Solaris.

The command to open the editor is taken from the "General" preference: "Use Case Description Command Line", default to "winword" on Windows, and to "xedit" on Solaris.

The following table lists attributes template files for use case diagram elements.

Element	Attributes Template File
Use-case diagram (chart)	uch_attributes.def
Use case	uc_attributes.def
Scenario, in the Use-Case Scenario List	scen_attributes.def
Actor	at_attributes.def
Association	Not supported
Dependency (Extend, Include)	Not supported
Generalization relation	Not supported
Boundary box	bn_attributes.def
Textual note	Not supported

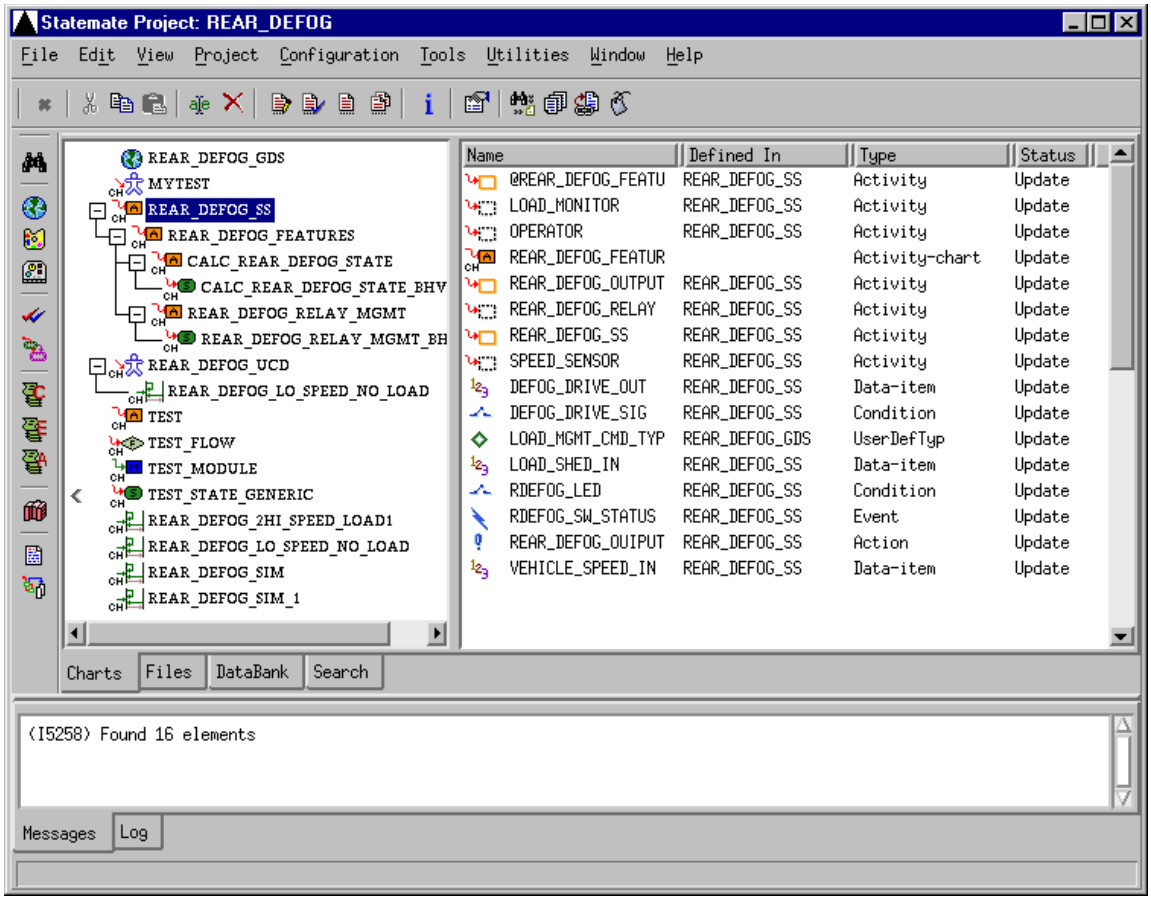
Linking Use Cases to Scenarios

Rational Statemate enables you to link use cases to scenarios that are defined in sequence diagrams. For more information on sequence diagrams, see **Sequence Diagrams**.

To link a use case to a scenario defined in a sequence diagram:

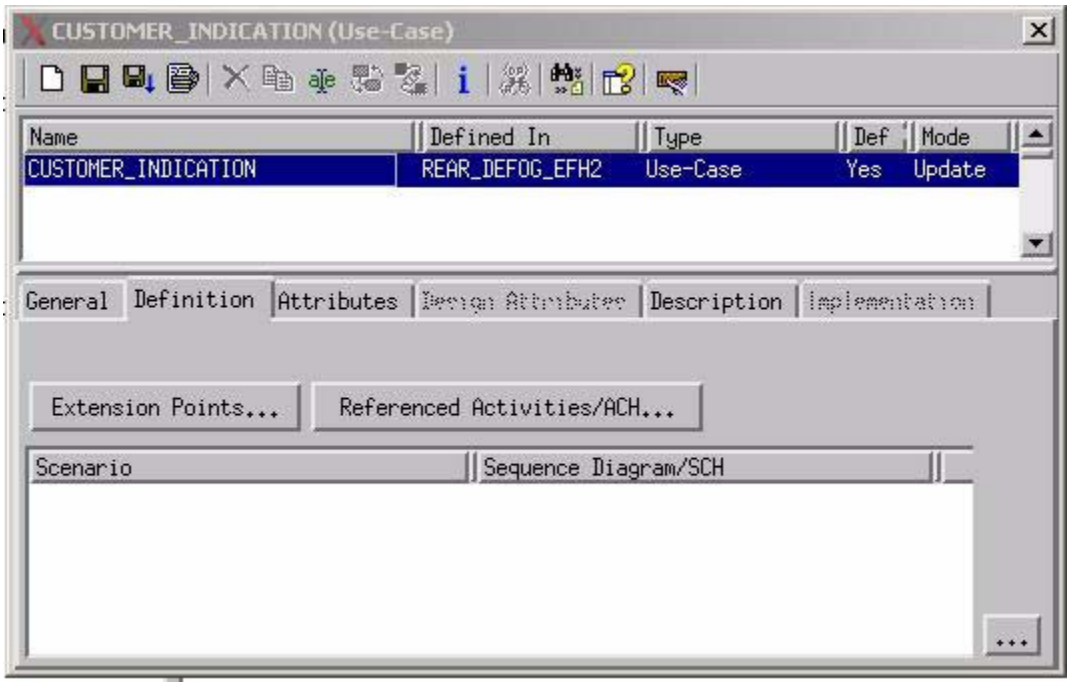
1. Create a sequence diagram that is based on the same system described in the use case diagram you have created. For more information, see **Creating a New Chart or Diagram**.

2. Select the **Charts** tab of the Rational Statemate main window, as shown in the following figure.



3. Click on the use case diagram that you want to link. The use case diagram opens.

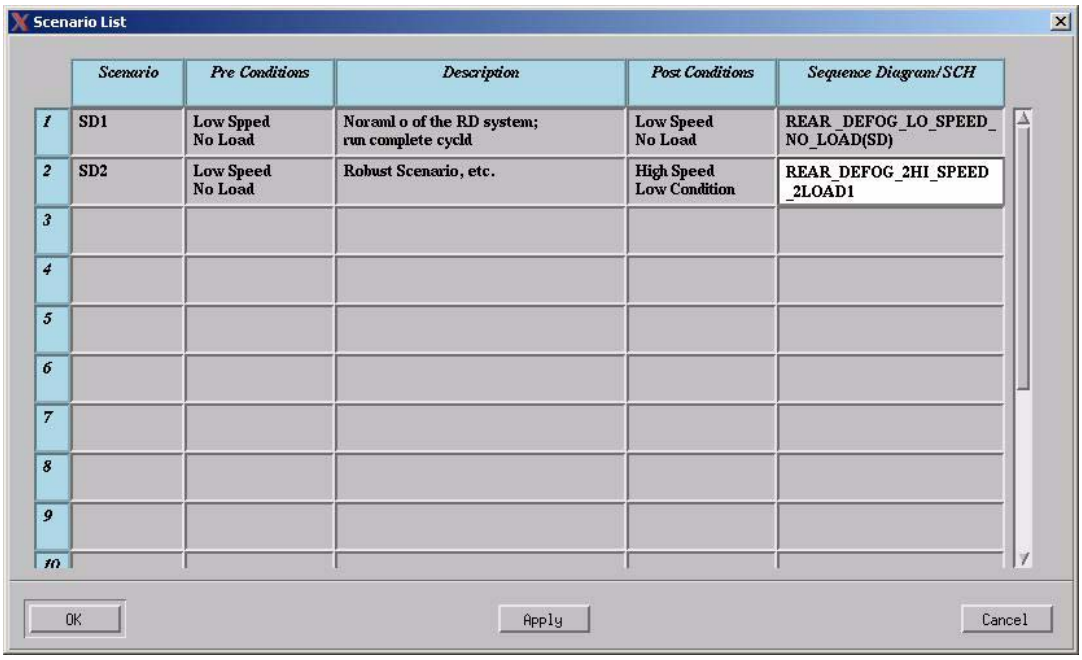
4. Select **Tools** > **Properties**. The Properties window opens, as shown in the following figure.



5. In the **Use-Case Description** field, enter text that explains the use case, for example:

   You can request a change in state of the rear defog control system. Once the system has been activated, it automatically times out and turns itself off without any user interaction. The timing function can be modified by vehicle speed and load management.

6. To create the mapping between a use case and one or more scenarios, select "**...**" next to the Scenario area**.** The Scenario List window displays.



Sequence Diagrams

A sequence diagram depicts the sequence of actions that occur in a system, visually capturing the dynamic behavior of a system.

A sequence diagram is two-dimensional. On the horizontal axis, it shows the life of the element that it represents, while on the vertical axis, it shows the sequence of the creation or invocation of these elements.

Because it uses class name and element name references, the sequence diagram is very useful in elaborating and detailing the dynamic design and the sequence and origin of invocation of elements. Hence, the sequence diagram is one of the most widely used dynamic diagrams in UML.

Sequence diagrams are used to:

- **Requirement capturing** - You can create a sequence diagram using the sequence-diagram editor, without an existing activity chart. As such, the sequence diagram is a powerful, yet intuitive tool for recording project requirements.

  The sequence diagram provides a clear expression of the elements and their interactions, and helps to avoid the ambiguities, difficulties, and errors of a text-only specification.
- **Logic checking** - An sequence diagram can expedite the confirmation of the existence of all elements. It also enables you to examine how the elements interact within a time flow.
- **Model checking** - Safety and liveliness checks can be performed for a model represented by an sequence diagram. Safety checks confirm that forbidden events are excluded by the logic, for example, that an elevator door is never opened when the elevator is between floors. Liveliness checks confirm that required event constraints are specified, for example, that an elevator door closure is commanded within two seconds of the floor selection.

A sequence diagram provides a view of a system or subsystem model that graphically displays the following:

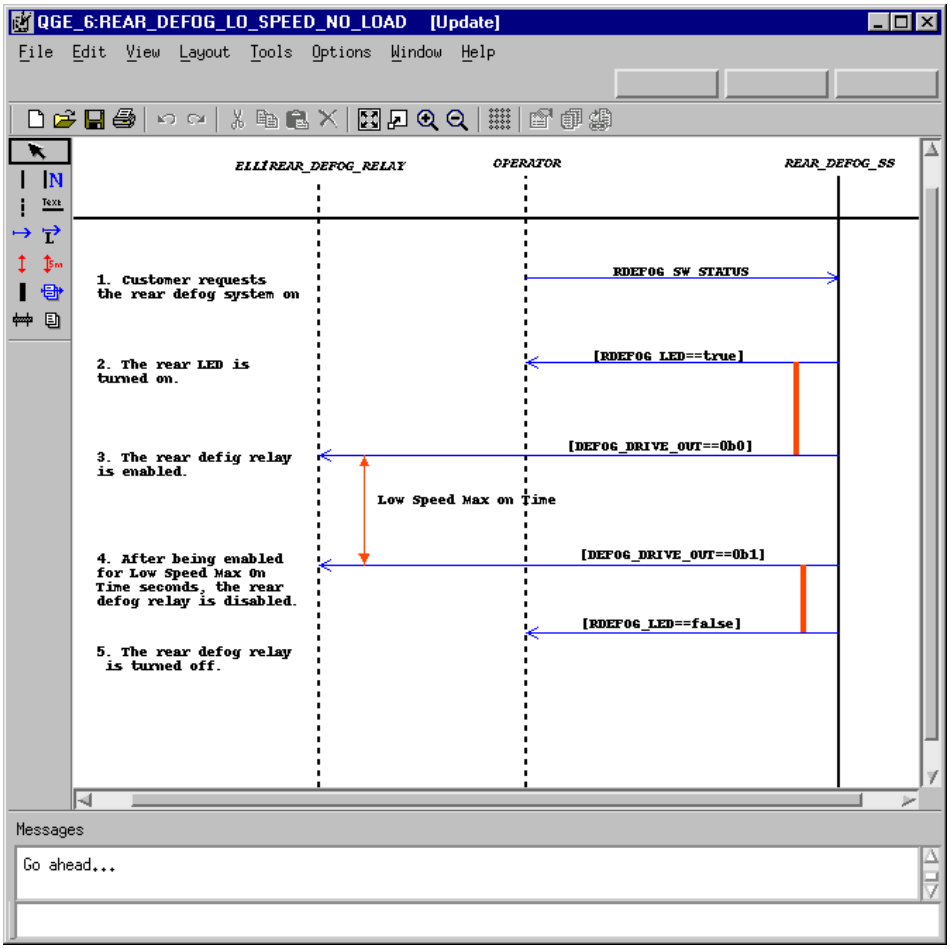
- Message flows between activities of a model, in order of occurrence.
- Source and target activities of each message.
- Time constraints between messages that must occur within specified time limits.
- Descriptive comments.

Because an sequence diagram represents message flow between activities, the dominant symbols are those representing activities and messages.

- Activities in a sequence diagram are represented by parallel vertical lines (lifelines), with time flow being from the top down. Lifelines can be as long as necessary to encompass all the modeled events.
- Messages are displayed as horizontal arrows connecting the source activity lifeline to the target activity lifeline. Each activity lifeline is labeled with the names of the corresponding elements in the activity chart.

Accessing a Sequence Diagram

For information on how to access a sequence diagram, see **Starting a Graphic Editor**. The sequence diagram editor opens, as illustrated in the following figure.



Sequence Diagram Icons

The following icons support drawing and naming operations in a sequence diagram. For general information on drawing and naming operations in Rational Statemate, see **Drawing Operations in Graphic Editors** and **General Operations in Graphic Editors**.

General Sequence Diagram Icons

	Create a Life Line - A vertical line representing an activity in the model. The activity label displays at the top.
	Name a Life Line - Names an existing lifeline.
	Create External Life Line - A dashed vertical line, representing an external activity that may or may not be in the model.
	Create a Partition Line and Text - A horizontal line, distinguishing between scenarios.
	Create Simple Message - An arrow from one activity (source) to another (target).
	Label a Message - A horizontal arrow, representing interaction between activities. A message label consists of an event expression.
	Create Timing Constraint - A vertical line connecting two messages, representing a timing constraint specification between the two messages. If one of the messages is moved, the timing constraint line is stretched or compressed accordingly.
	Create Timing Constraint Text - A free text note that is attached to a specific timing constraint.
	Create Order Insignificant Line - A vertical line connecting two messages, signifying that the designated messages can occur in any order.
	Create Message Note - A free text note that is associated with a specific timing constraint. By default the note is displayed on the left side of the sequence diagram, horizontally aligned with the timing constraint. The name assigned to a sequence diagram graphic element can be a path name, designating the exact location in the model represented by the sequence diagram element. A path name has the following format: `Chart Name:Activity[ .Activity]*`
	Create a Reference SD - A thick horizontal, shaded line extending the full width across all lifelines, representing a separate sequence diagram. Execution of activities below the referenced sequenced diagram line continues only after all the activities above the line are completed.

Miscellaneous Icons

	Put GE into Select Mode - places the editor in Select mode, so you can access the editing options the editor supports.
	Create Textual Note - creates free text that you can use to annotate the diagram.

Sequence Diagram Drawing Notes

The following notes apply to drawing in a sequence diagram:

- To draw a partition line, place the cursor at the desired location (off of a lifeline) and click and drag slightly to the right. The partition line extends infinitely.
- To draw a lifeline, place the cursor at the desired location and click. A lifeline extends infinitely.
- To draw a simple message, locate the starting lifeline, place the cursor on the lifeline and either:
  - Click on the starting lifeline to set the tail of the message line. Then click on the destination lifeline to set the head (arrow) of the message line.
  - Drag the cursor from the starting lifeline to the destination lifeline.
- Make sure that you enter the tables correctly because they contain code that Rational Statemate uses when executing the diagram. The syntax is checked when you finish typing and any errors are displayed in the message box at the bottom of the editor.
- Message notes are free text and are not associated with any lines. You attach a message note to a line by clicking on the line.

Lifeline Decomposition

Sequence diagrams can include an off-page (decomposed) lifeline, named <name>@<Sequence diagram name>, representing another sequence diagram in the workarea.

To create an off-page lifeline, create a new lifeline and prefix its name with “@”.

Note

It is illegal to define circular referencing.

Integrating Sequence Diagrams with the Rational Statemate Model

Integration of a sequence diagram with the Rational Statemate model consists of element name linkages. The sequence diagram contains no actual functionality of the model. When an sequence diagram template is generated from an activity chart, lifelines are automatically assigned the corresponding names in the activity chart.

The message elements are linked to the model by means of name resolution rules.

Other notations, such as time constraints, provide essential details of the requirements.

Generating a Sequence Diagram from an Activity Chart

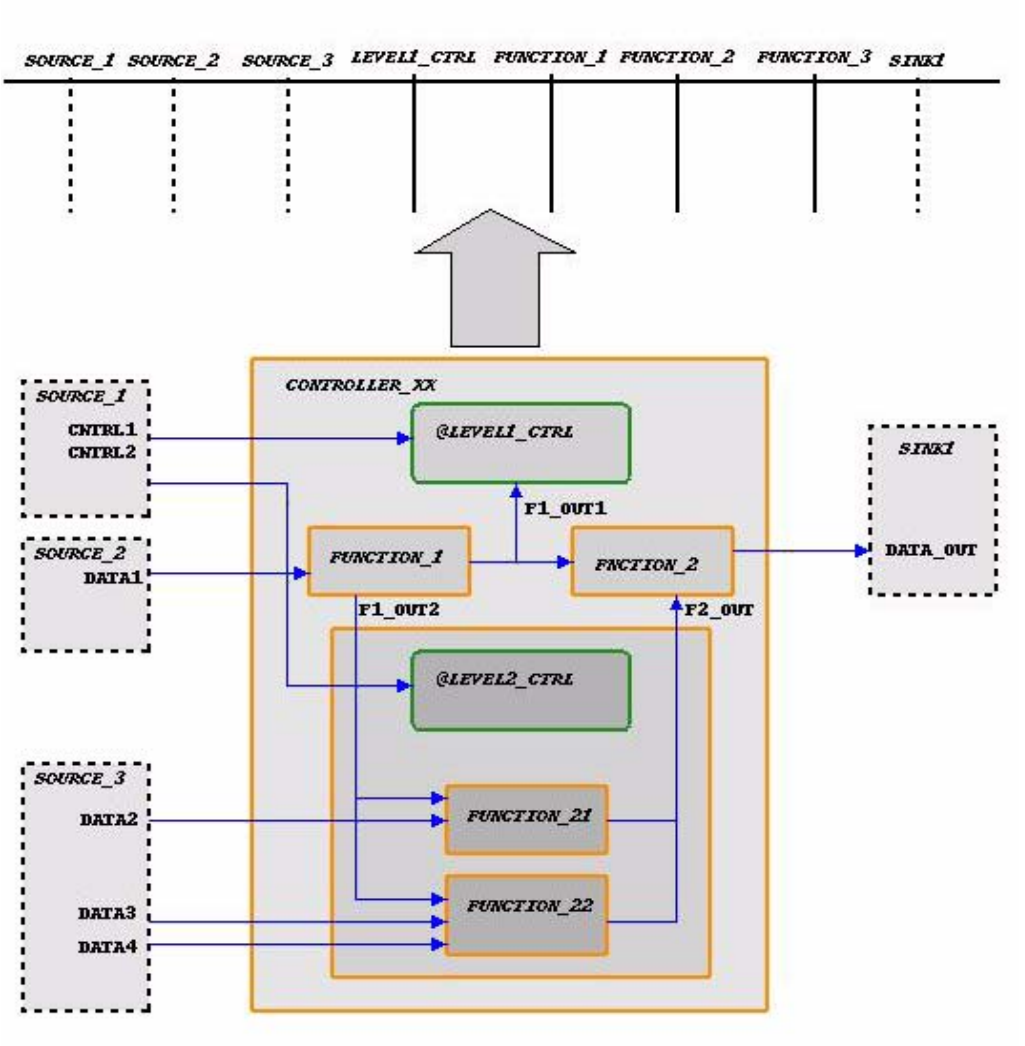
A sequence diagram can be created manually before an activity chart has been created, or automatically from an activity chart. You can save time by generating a sequence diagram template from the activity chart. A sequence diagram template generated in this manner contains all the lifelines for each activity. However, you must enter and label the message lines and other notations manually.

Note

Generating a sequence diagram from an activity chart creates only an initial template; subsequent changes in the activity chart are not automatically updated in the sequence diagram.

To generate a sequence diagram template from an activity chart:

1. Open the activity chart for which you want to create a sequence diagram.
2. Select the main activity. If a subactivity is selected, the generated sequence diagram contains only activities within the selected sub-activity chart. External lifelines are created for activities that are connected to (or into) the selected activity.
3. Select **File > Create Sequence Diagram**.
4. The sequence diagram opens, as shown in the following figure, having all the activities and external activities of the selected activity chart. The messages, however are not created.


SOURCE_1
SOURCE_2
SOURCE_3
LEVEL1_CTRL
FUNCTION_1
FUNCTION_2
FUNCTION_3
SINK1
CONTROLLER_XX
SOURCE_1
CNTRL1
CNTRL2
@LEVEL1_CTRL
F1_OUT1
SINK1
SOURCE_2
DATA1
FUNCTION_1
FNCTION_2
DATA_OUT
F1_OUT2
F2_OUT
@LEVEL2_CTRL
SOURCE_3
DATA2
FUNCTION_21
DATA3
FUNCTION_22
DATA4

Using Properties with Sequence Diagrams

You can open the Properties window in a sequence diagram as in the other Rational Statemate editors. Either right-click any element in the sequence diagram and choose **Properties** from the popup menu, or select an element and choose **Properties** from the sequence diagram editor Tools menu.

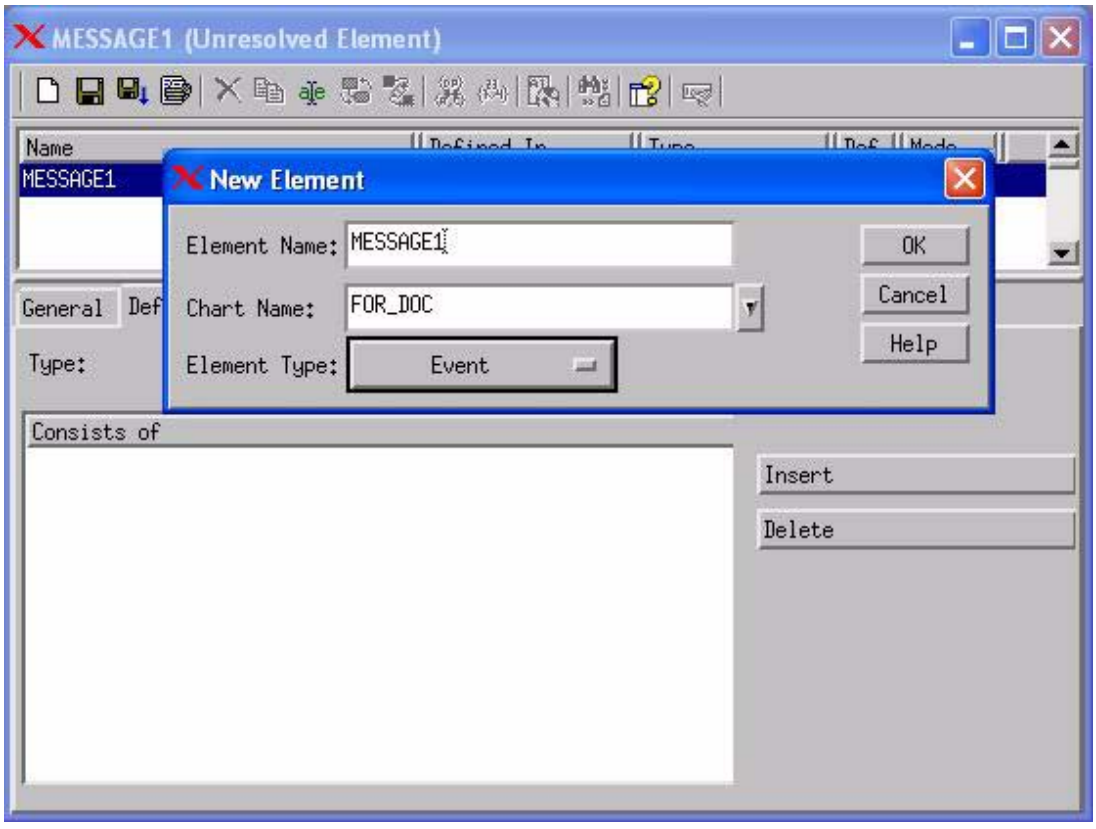
When using the properties for a sequence diagram, you can specify the scope either for a specific activity chart or for the whole model.

Opening the Properties window on an element enables you to resolve the element, that is, to find where it occurs in an activity chart.

To resolve the definition of a sequence diagram element:

1. Right-click on the element to be resolved, and choose **Properties** from the popup menu. The Properties window opens.

2. If the element is unresolved, the message *Unresolved Element* displays in the editor window. Choose **File > New** or click **Save**. The New Element popup dialog box is displayed, as shown in the following figure, that enables manual definition of the element.



Auto-Numbering in Sequence Diagrams

The Sequence Diagram Editor can perform two types of autonumbering:

- Auto-Numbering of Scenarios
- Auto-Numbering of Message-Notes

Auto-Numbering of Scenarios

The Scenario Auto-Numbering view mode in the Sequence Diagram Graphic Editor causes Sequence Diagram scenarios (partition lines) to be sequentially numbered according to the format defined by the three Sequence Diagram Graphic Editor preferences:

- Scenario Numbering Postfix
- Scenario Numbering Style
- Start Scenario Numbering From

The numbering is visible for those Sequence Diagram scenarios (partition lines) that have partition line text associated with them. When the scenario ordering is changed, the numbering is automatically refreshed. The mode is enabled and disabled by the Sequence Diagram Graphic Editor preference “Show Scenario Numbering.” Changing the numbering method defined by the specified preferences does not cause automatic numbering refresh. To forced a refresh operation, toggle the preference Show Scenario Numbering.

The scenario numbering does not affect the uniqueness of scenario names (required for sequence diagram animation).

Auto-Numbering of Message-Notes

The Scenario Auto-Numbering feature, when enabled, also autonumbers message-notes in addition to the partition line text.

Print Pagination

The Sequence Diagram Editor has a Pagination Preview mode which shows the subdivision of the sequence diagram into multiple printed pages and also allows you to set the print page orientation and page overlap.

In a sequence diagram, the Pagination Preview mode is controlled from **View > Pagination Preview**.

In the main display, you can set the Pagination Preview mode as well as set page orientation and page overlap by the following preferences, accessed from the **Project > Preferences Management>Sequence Diagram Graphic Editor:**

- **Pagination Preview (Yes/No)** - Enables Pagination Preview mode
- **Pagination Orientation (Portrait/Landscape)** - Defines the page orientation to be used in sequence diagram pagination
- **Pagination Overlap Ratio ([%])** - Defines the desired overlapping area, which will be printed twice

Note

These preferences also affect the Documentor. In order for changes in Preferences values to take effect in the Documentor, the Documentor should be closed and re-opened.

Flowcharts

A flowchart represents a process graphically. It includes the entire process from start to finish, showing inputs, pathways and circuits, and action or decision points.

The following sections describe in more detail how to use flowcharts.

Accessing a Flowchart

To access a flowchart, see **Starting a Graphic Editor**.

Flowchart Icons

The following icons support drawing and naming operations in an flowchart. For general information on drawing and naming operations in Rational Statemate, see **Drawing Operations in Graphic Editors** and **General Operations in Graphic Editors**.

General Flow Icons

An activity is the primary graphic element in flowcharts that represent a function in the functional view of the system. An activity represents something that transforms inputs into outputs.

	Create Box - creates a box representing a stage in a flow.
	Write Action in Box - creates an action that describes the meaning of a flowchart box.
	Create Instance Box - creates an instance box.
	Name Instance Box - creates a name for an instance box.
	Create Decision - creates a decision box, which directs a flow depending on the result of a decision expression.
	Write Decision Expression - creates an expression that gives a Yes or No result.
	Create Switch - creates a witch box, which directs a flow depending on the result of a switch expression.
	Write Switch Expression - creates an expression whose value is used to control a switch.
	Create Arrow - creates an connection between elements of a flowchart.
	Label Existing Arrow - creates a label for a flowchart arrow.
	Create Start Arrow - creates the starting point for a connection to a distant flowchart element.
	Create End Connector - creates the end point of a connection from a distant flowchart element.

Connector Icons

Connector icons are circular or oval graphic elements used in charts to join and divide arrows or to enable an arrow to exist on multiple pages. Their purpose is to clarify a specification by reducing the number of arrows.

	Create Diagram Connector - creates a diagram connector, which connects a target and a source that are far from each other. Using this type of connector eliminates the need for long arrows.

Miscellaneous Icons

	Select Mode - places the editor in Select mode, so you can access the editing options the editor supports.
	Create Textual Note - creates free text that you can use to annotate the diagram.

Flowchart as Subroutine Implementation

Flowcharts may be implemented as subroutines in the Rational Statemate model. The following are the rules for this flowchart implementation:

- To open/create a procedural flowchart automatically, select the “flowchart” implementation in a subroutine's properties form and click **Edit**.
- A procedural flowchart must have the name of the subroutine to which it is connected.
- Visually, a procedural flowchart will look exactly the same as regular or generic flowcharts, except for three toolbar icons which are disabled:
  - Create Instance Box
  - Name Instance Box
  - Create Diagram Connector
- Procedural flowcharts are not displayed in the tree view of the Workarea.
- Icons of procedural flowcharts do not display in the element’s view of the definition chart of the connected subroutine (as procedural Statecharts are).
- During check-in/checkout of a subroutine with flowchart implementation, the connected flowchart (if it exists) is also checked-in/out.
- Generic and off page instances are not allowed inside a procedural flowchart.
- All relevant Rational Statemate tools support procedural flowcharts: Check Model, Simulation, Code-Generators, Documentor, Component libraries and Search tool.

- Flowcharts as Subroutine implementation are presented into the anaport-like procedural Statecharts.
- Each Flowchart creates a separate scope that contains all the boxes and arrows that are defined in the Flowchart.
- The type of the chart in this scope is: stmm_ch_flowchart.
- Flowchart boxes are handled with the States APIs.
- Flowchart arrows are handled with Transitions APIs.
- The enum type: stmm_st_type (in api_types.h) was extended to include additional types which represents Flowchart boxes types:

```
typedef enum {
stmm_st_or, /* Or State */
stmm_st_and, /* And State */
stmm_st_reference,
stmm_st_diagram,
stmm_st_action_box,
stmm_st_decision_box,
stmm_st_switch_box
} stmm_st_type;
```

Panels

This section describes panels and how to use them. The following topics are covered:

- **Using the Panel Editor**
- **Binding Interactors**
- **Using the Panel Builder**

Panels provide a visual interface to the simulated model or generated code for debugging and prototyping purposes. A panel is built using predefined interactors and user-defined shapes. The dynamic behavior of these interactors and shapes is defined by binding them to elements in the model.

Typically, panels represent the user interface to a system. However, it is also quite common for panels to be a logical representation of a system, for example, showing the routing of packets through a communications network, or the failure states of valves and pumps in an aircraft fuel system.

There are two methods for creating a panel:

- **Panel Editor** creates a panel using graphical elements such as lines, circles, and polygons. Predefined elements (interactors) with built-in animation properties are also available to represent the inputs and outputs of the model.
- **Panel Builder** creates a panel from a chart and automatically bind the interactors. The **Panel Editor** is then used to edit the panel and refine it.

Unlike charts, there are no formal semantics to the hierarchy in a panel, so operations such as “Open Parent” and “Edit Sub-chart” are not supported. Also, there is no checking for semantic correctness of the picture while editing. Therefore, it is important to understand the relationship between a panel and the rest of the model.

The recommended approach is to do the following:

1. Create panels using the **Panel Editor** (see **Using the Panel Editor**) or **Panel Builder** (see **Using the Panel Builder**). The default interactor for each element type can be set in the model.

2. Using the **Panel Editor,** edit the panels and modify the appearance and operation of the interactors. To improve the representation of a system, add groups of ordinary drawing elements. The animation of interactors and groups of drawing elements is controlled by the bindings to the model.

3. Check the bindings of the graphics to their corresponding elements.

4. Execute the model using simulation and/or code generation.

The following sections explain how to work with the panel editor and Panel Builder. For information on graphic editors in general (drawing, naming, and so forth), see **Working with Graphic Editors**. For descriptions of all the menu options in the panel editor, see **Panel Editor Menus**.

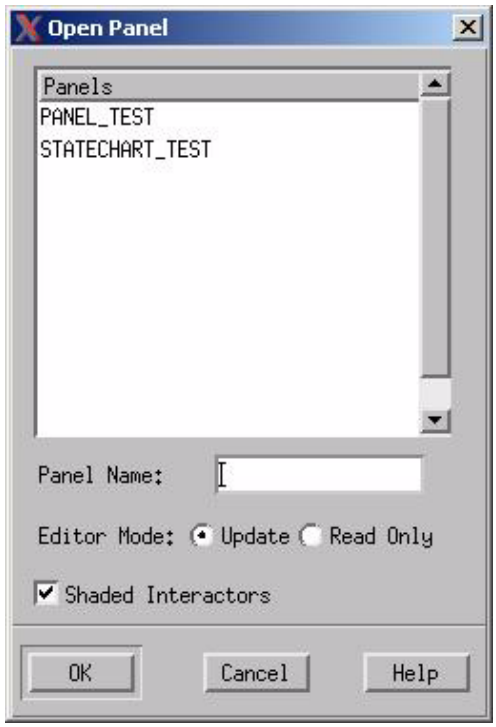
Using the Panel Editor

The **Panel Editor** creates a realistic mock-up that represents the system being designed. The panel is associated with a model and is animated during model execution. Thus, the graphic panels are used to demonstrate that the system behavior is correct (by using them to enter inputs and monitor outputs).

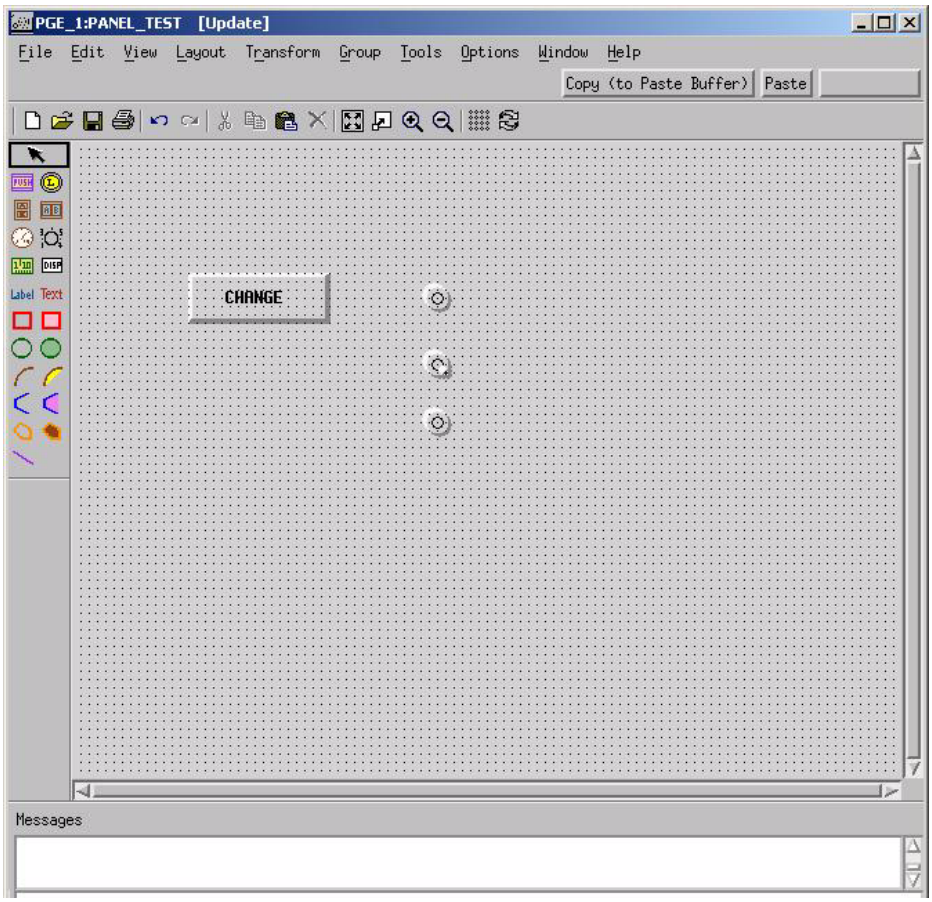
Accessing the Panel Editor

To access the **Panel Editor:**

1. From the Rational Statemate main window, click the **Panel Editor** icon . The **Open Panel** dialog box displays.



2. Under **Panels**, click a panel name and then click **OK**. The **Panel Editor** displays



Panel Editor Menus

This section lists the menus supported by the panel editor, and describes the menu items for each menu.

The following menus are supported by the panel editor:

- File Menu
- Edit Menu
- View Menu
- Layout Menu
- Transform Menu
- Group Menu
- Tools Menu
- Options Menu
- Help

The following sections describe the menu items available for each of the panel editor menus (with the exception of the Help menu).

File Menu

The following table describes the File menu items.

Menu Item	Description
New	Creates a new panel.
Open	Opens an existing panel.
Close	Closes the current panel. If the panel includes graphical or textual changes not yet saved, you are prompted to save these before the panel is exited. The panel editor remains open and is now labeled Empty Editor. (You are prompted to name the panel when you exit.)
Save	Saves the current panel.

Menu Item	Description
Insert panel	Inserts the contents of another panel into the current Panel Editor window using the Insert window. From the Insert window, you can do the following: • In the From Panel textbox, enter or select the panel to be inserted. • In addition to inserting an entire panel, you can insert a single activity (state or module) for the specified panel. Click the down arrow to view a list of possible activities (states or modules) to be inserted. If the selected panel has no sub-activities, the list is blank. • Optionally, deselect **Copy Element Definitions**, if you do not want textual elements included.
Print	Prints the current panel.
Exit	Exits from the panel editor and closes any open panels. If the panel includes graphical or textual changes not yet saved, you are prompted to save these before the panel is exited. The panel editor remains open and is now labeled Empty Editor. (You are prompted to name the panel when you exit.)

Edit Menu

The following table describes the Edit menu items.

Menu Item	Description
Undo	Undoes the last operation.
Redo	Redoes the last operation.
Cut (to Paste Buffer)	Deletes the selected text or elements and places them in the paste buffer.
Copy (to Paste Buffer)	Copies selected text or elements into the paste buffer.
Paste	Pastes selected text or elements from the paste buffer.
Move	Moves selected text or elements.
Stretch	Stretches selected text or elements.
Edit Text	Enables text-edit mode.
Delete	Deletes selected elements.
Properties	Modifies the following display properties: • Color • Fill Color • Font • Line Width
Select	Selects text or elements.

View Menu

The following table describes the View menu items.

Menu Item	Description
Full View	Expands or reduces the drawing frame to fit into the graphic canvas. This is the default when a graphic editor is first opened.
Zoom Area **Zoom In** **Zoom Out**	Enlarges or shrinks the drawing on your screen, without changing the window dimensions. Choose one of the following: **Note:** By default, zoom operations do not affect the text. Select **Options > Enable Scale Text** to change the size of both the text and lines when you zoom into and out of the canvas. • To zoom in to a particular area of a drawing, select **Zoom Area**. Drag a rectangle around the area of the drawing that you want to zoom into. The area you selected expands to fill the canvas. • To zoom into a drawing, select **Zoom In**. The drawing expands approximately 30%. Use the scroll bars to pan the window. To further enlarge the drawing, you can select **Zoom In** several times. • To zoom out of a drawing, select **Zoom Out**. The drawing contracts approximately 30%. Use the scroll bars to pan the window. • Select **Full View** to return to the original size.
Refresh	Redisplays the graphics on the canvas. Occasionally erroneous graphics can remain on the screen during editing. Use this operation to clear them from the screen display.
Hide	Hides the detail underneath a box. The box is shaded to show that there is missing detail underneath. You can use this operation to hide details during printing, for example.
Reveal	Redisplays previously hidden elements.
Grid	Displays the editing grid. By default, when you open a graphic editor, the grid is not displayed. Select **Grid** again to turn the grid display off.

Menu Item	Description
Grid Setting	Displays the Grid Setting dialog box, which allows you to change the grid settings. The grid consists of a matrix of points the pointer/drawing elements snap to. The Grid Setting dialog box shows the width and height of the drawing frame, but this information is only for reference. You cannot change these values. To expand the drawing area to fill the window, use **Layout > Reframe**. To make the grid • Active and visible, select **Active**. • Active and invisible, select **Snap to grid**. To specify the spacing of the grid points, set the **Grid Spacing** number. The up and down arrows provide a quick way to double or halve the spacing between grid points. To specify how many of the grid points to show, use the **Every nth grid point** field. (If this value is 1, all the grid points are shown. If this value is 2, only every 2nd grid point is shown, etc. The other grid points exist; however, you cannot see them.) Click **Apply** to apply your changes and continue, and click **OK** when you are finished.
Messages	Toggles between showing and hiding the Message and Log tabs.
Drawing Icons	Toggles between showing and hiding the drawing toolbar.
Command Bar	Toggles between showing and hiding the command toolbar.
Toolbar	Toggles between showing and hiding the Rational Statemate toolbar and various parts of the standard toolbar. You can choose whether to display the Rational Statemate toolbar vertically along the left side of the window or horizontally across the top of the window. In the Rational Statemate main window, select **View > Toolbars > Tools (Horiz.)** or **Tools (Vert.)**.

Layout Menu

The following table describes the Layout menu items.

Menu Item	Description
Replicate	Quickly creates a grid of drawing elements. • Select the elements to copy. • Position the selected elements in the corner of the grid of elements to be created. Select **Replicate**. The Replicate dialog box opens. • Fill in the **Number of Rows** and **Number of Columns** text boxes to set the number of rows and columns to be copied. • Click **OK**. • As you move the pointer in the graphic canvas, you see shadow images of the drawing elements you are replicating arranged in a grid. Left-click when the shadow images are in the correct position.
Arrange	Aligns selected elements vertically and horizontally, as well as arranging the spacing between them. You can do the following: • Rearrange elements in relation to each other. This option has no effect, if only one element is selected. **Note:** Be careful not to select both the names and the boxes when arranging panels, because you can move a name out of its box using the arrange operations. If you only select the box, the name moves with it. • Align elements around a vertical line, based on the system's calculation of the average location of the elements. You can align elements by their: * Left edge * Center * Right edge • Align elements around a horizontal line, based on the system's calculation of the average location of the elements. You can align elements by their: * Top edge * Center * Bottom edge

Menu Item	Description
Arrange (Continued)	• Move elements so that they are evenly spaced vertically, based on the system's calculation of the average location of the elements. You can space elements evenly by the: * Bottom edge * Center * Top edge * Space between elements • Move elements so they are evenly spaced horizontally, based on the system's calculation of the average location of the elements. You can space elements evenly by the: * Left edge * Center * Right edge * Space between elements
Align to Grid	Aligns text and elements to the grid.
Reframe	Changes the size and shape of the drawing frame (or page). • Select **View > Full View** before this operation. • Use the window frame resize bars to change the size or shape of the entire panel editor window. • Click **Reframe**. The drawing frame is changed so that it fits the new size and shape of the graphic canvas exactly. **Caution:** This operation can change the origin of the grid. If you are using the grid, be careful in using this operation, because it may be difficult to make your drawing line up correctly afterwards.
Back	Sets the display priority of elements to be in back of other elements. • Select the elements or groups that are to be in the back. • Select **Back**.
Front	Sets the display priority of elements to be in front of other elements. • Select the elements or groups that are to be in the front. • Select **Front**.

Transform Menu

The following table describes the Transform menu items.

Menu Item	Description
Scale	Scales the selected elements around the pivot point (that is, stretches them, while preserving their perspective).
Rotate	Rotates the selected elements around the pivot point. Note that textual elements cannot be rotated. Hold down the Shift key for a constrained rotation (one that will only rotate the element by multiples of 45 degrees). **Note:** To move the pivot point, select it with the middle mouse button (on a two-button mouse, press both buttons) and drag it to a new location.
Set Pivot	Defines a new pivot point.The default pivot point is located at the center of a graphical element. You can move the pivot point by selecting and dragging it to a new location with the middle mouse button (on a two-button mouse, press both buttons).
Reset Pivot	Restores the default pivot point. The default pivot point is located at the center of a graphic element.
Mirror X Axis	Redraws the selected elements as if they are reflected around a vertical line passing through the pivot point.
Mirror Y Axis	Redraws the selected elements as if they are reflected around a horizontal line passing through the pivot point.
Copy Mirror X Axis	Copies the selected elements as if they are reflected around a vertical line passing through the pivot point.
Copy Mirror Y Axis	Copies the selected elements as if they are reflected around a horizontal line passing through the pivot point.

Group Menu

The following table describes the Group menu items.

Menu Item	Description
Make Group	Establishes a collection of elements as a group. You can control the appearance of all the elements in the group at the same time. To edit a group after it has been created, use the **Dive Group** operation.
Un-Group	Dismantles a top level group or sub-group of elements. Un-group discards all the binding information associated with the group.
Dive Group	Separates a group so you can edit the group members. The members of the group can be individually selected; elements outside the group are unselectable. To remove an element from a group, dive into the group, cut the element, surface from the group, then paste the element.
Surface Group	Regroups the selected elements after a **Dive Group** operation.
Top Surface Group	Regroups all the groups at all levels of the group structure. Because you can have groups within groups, you can dive several layers into the group structure. **Top Group** is a short-cut for selecting **Surface Group** several times.

Tools Menu

The following table describes the Tools menu items.

Menu Item	Description
Bind	Binds selected interactors or groups to model elements.
Properties	Allows you to set or modify properties for selected interactors or groups.
Binding Report	Lists the binding parameters for all or selected interactors or groups.
Check Bindings	Checks the binding parameters for selected interactors or groups. The bindings are checked and errors are reported.

Options Menu

The following table describes the Options menu items.

Menu Item	Description
Gravity Setting	Changes the gravity setting. Gravity is the distance you must get within to attach arrow heads to boxes or to select elements. To change the gravity setting, do the following: • In the Gravity Setting dialog box, specify the **Gravity Distance** in pixels. • Click **Apply** to apply your selection and continue, and click **OK** when you are finished.
PC Mouse (2 Button)	By default, you move an element by dragging it with the middle mouse button and selecting it using the left mouse button. When you select PC Mouse, you can drag an element with the left mouse button, as well as select with it. This option is available for users who are accustomed to applications running under Microsoft Windows or on an Apple Macintosh. When PC Mouse is set, the mouse conventions for editing are similar to the conventions on these machines. **Note:** You can also set the single-button mouse option as a general graphic editors preference.
Preserve Selection	By default, when you create or move an element, the element becomes selected. This option changes to the opposite state. **Note:** The Preserve Selection option is not saved in the chart.
Enable Scale Text	Causes both the text and lines in a diagram to change size when you zoom into and out of the canvas. By default, zoom operations do not affect the text. **Note:** The Enable Scale Text option is not saved in the chart.
Preferences Management	Allows you to modify general preferences for the panel editor: • General GEs Specific Preference • PGE Specific Preferences

Panel Editor Icons

The following icons support interactors and drawing and naming operations in a panel. For general information on drawing and naming operations in Rational Statemate, see **Drawing Operations in Graphic Editors** and **General Operations in Graphic Editors**.

Interactor Icons

Interactor icons are graphical representations of input/output devices in panels that have a predefined behavior.

	Create Push Button Interactor - When in, makes a condition true/false or generates an event. When out, displays the value of a condition, event, or state.
	Create Lamp Interactor - displays the value of a condition, event, or state**.**
	Create Vertical Multi-Choice Interactor - When in, sets the controlling data-item or condition to the value of the button. When out, monitors the value of a data-item.
	Create Horizontal Multi-Choice Interactor - When in, sets the controlling data-item or condition to the value of the button. When out, displays the value of a condition or data-item.
	Create Meter Interactor - When in, sets the value of a data-item (integer or real). When out, displays the value of a data-item**.**
	Create Knob Interactor - When in, sets the value of a data-item (integer or real). When out, displays the value of a data-item**.**
	Create Slider Interactor - When in, sets the value of a data-item (integer or real). When out, displays the value of a data-item**.**
	Create Textual Display Interactor - When in, sets the value of a data-item through typing a value. When out, displays the value of a data-item (integer, real, bit-array).

Drawing and Naming Icons

	Label Existing Interactor. Enters a label. After the label is entered, it is associated with an interactor or a group**.**
	Create Free Text. Enters free text anywhere on the drawing. This text is not associated with any element on the drawing canvas.
	Create Box Graphic. Draws a box.
	Create Filled Box Graphic. Draws a filled box.
	Create Circle Graphic. Draws a circle.
	Create Filled Circle Graphic. Draws a filled circle.
	Create Arc Graphic. Draws an arc.
	Create Filled Arc Graphic. Draws a filled arc.
	Create Multi-Line Graphic. Draws line segments. Click each vertex of the line. Double-click, or press the middle mouse button to create the end point.
	Create Filled Multi-Line Graphic. Draws filled line segments. Click each vertex of the line. Double-click, or press the middle mouse button to create the end point.
	Create Polygon Graphic. Draws a polygon. Click each vertex of the polygon. Double-click, or press the middle mouse button to create the final vertex.
	Create Filled Polygon Graphic. Draws a filled polygon. Click each vertex of the polygon. Double-click, or press the middle mouse button to create the final vertex.
	Create Line Graphic. Draws a line.

Miscellaneous Icon

	Put GE into Select Mode. Places the editor in **Select** mode to access the editing options the editor supports.

Binding Interactors

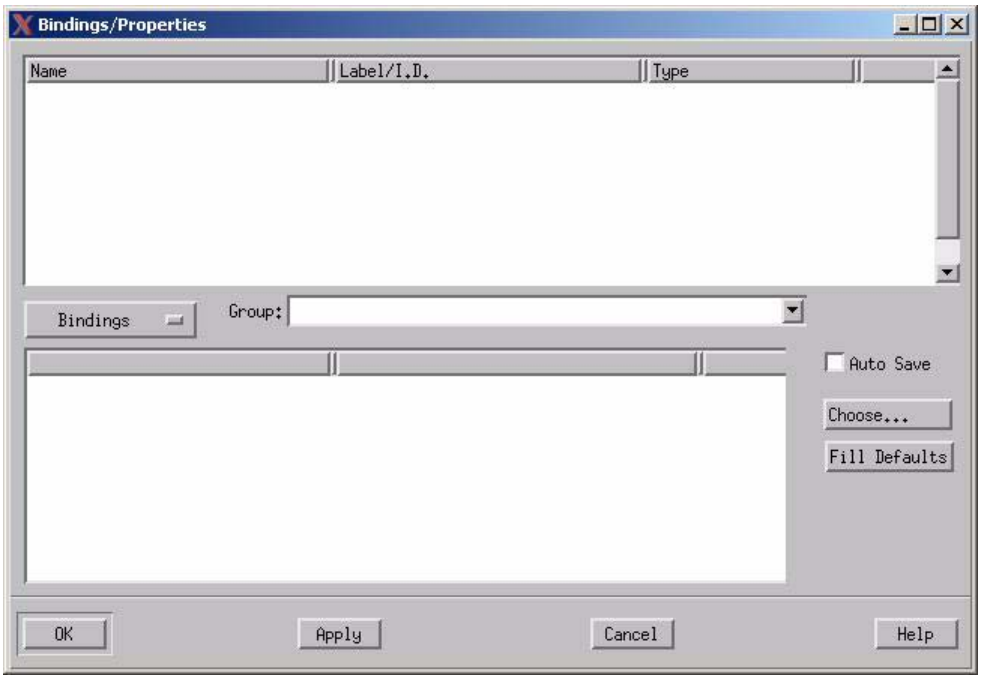
In the **Panel Editor**, interactors are bound one of three ways:

- Accepting input
- Providing output
- Accepting input and providing output

The way an interactor is bound indicates whether the bound element affects, or is affected by, the element that it is bound to.

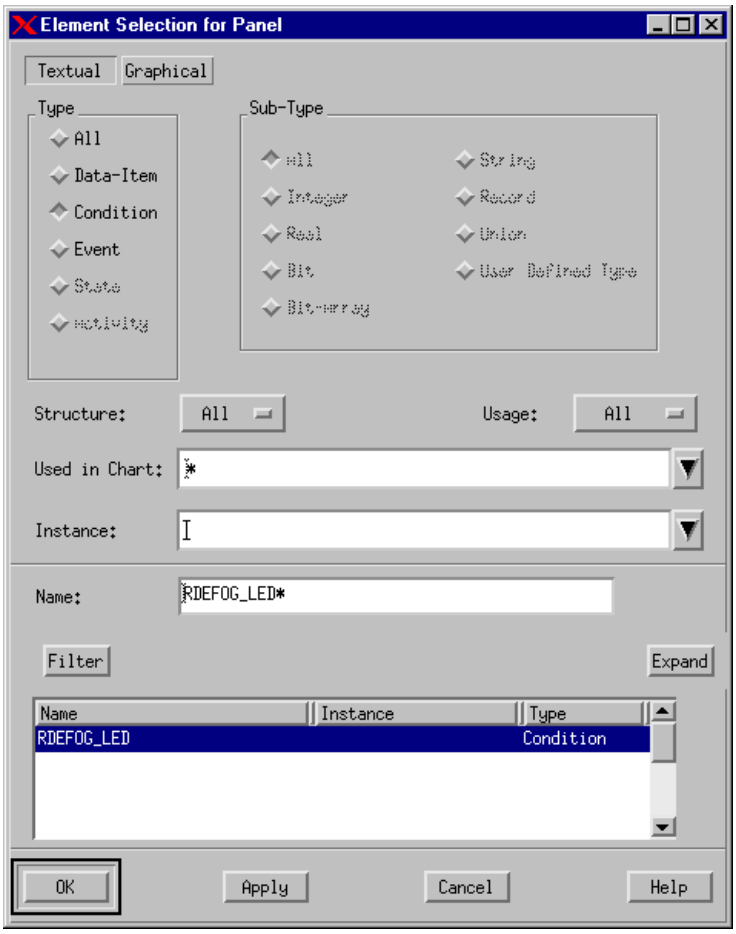
To connect interactors (or groups of interactors) to elements in your model:

1. Select the elements to bind.
2. On the **Tools** menu, click **Bind**. The **Bindings/Properties** dialog box displays.



From the **Bindings/Properties** dialog box, the following properties and actions are supported:

- **Name -** displays the name of the interactor.
- **Label/I.D**. **-** displays the label associated with the interactor.
- **Type -** displays the type of interactor.
- **Bindings/Properties -** a button that toggles between the binding information and the properties information for the currently selected element.
- **Group** -
- **Auto Save -** When the check box is selected, each edit is automatically applied as it is inputted.
- **Bind Method -** Choose from **Input, Output,** or **Input/Output**.
- **Controlled by** -
- **Element type** - Choose from **Bit**, **Condition**, **Event**, **State**, or **Activity**.
- **Button type -** Choose from **Flash**, **Toggle**, or **Push Release**.
- **Choose -** Defines an element. Select the **Controlled By** field from the editing area by clicking on the field name and then click **Choose**. The **Element Selection for Panel** dialog box displays with the selections for the field.



- From the Element Selection for Panel window, you can do the following:
  - Create a list of elements of the needed type and subtype. If necessary, specify a chart name and/or a pattern for the name of elements to be filtered.
  - Click **Filter**. A list of elements displays.
  - Select the elements you want from the list.
  - To confirm your choices, click **OK** or **Apply**.
- The **Fill Defaults** option enables you to fill in the enumerated values automatically when you are binding an enumerated type to an interactor or group that can be associated with an enumerated value, for example, the vertical and horizontal choice buttons.

The following sections describe the interactor bindings supported by the panel editor.

Individual Bindings

The following bindings are supported for individual interactors.

- **Button Type** - This binding is supported by the Push Button interactor, which enables you to choose from Flash, Toggle, and Push Release.
- **Button #** - Enables you to associate each button with a unique value.
- **Format** - Specifies the display format (binary, octal, hexadecimal, or decimal).
- **Field Length** - Specifies the number of characters to display.
- **Fractional Part** - Specifies the number of characters after the decimal point to display.

Group Bindings

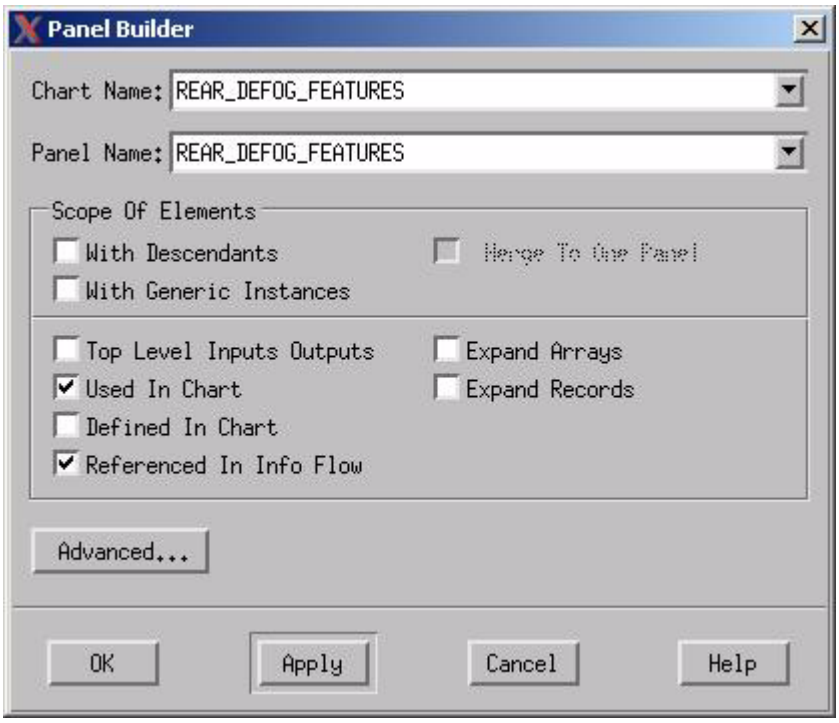
The following bindings are supported for grouped interactors.

- **Group Color** - Changes the fill color of the element according to the value of the controlling element. The choice of colors and values is made using the Add Color button.
- **Group Flip** - Causes the group to be mirrored according to the value of the controlling element. The effect is similar to performing a Transform>Mirror X/Y- Axis operation on the group.
- **Group Move** - Controls the position of the group within a frame. By default the frame is the whole panel window. You can make a frame by drawing and naming a rectangular box.
- **Group Rotate** - The angle of rotation of the group is controlled by the value of the controlling element. Rotation is performed around the rotation point of the group.
- **Group Scale** - Causes the group to change size according to the value of the controlling element. The scaling is always done evenly in the x- and y-axis.
- **Group Visibility** - Controls the visibility of the whole group.When the controlling element is active/true, the group becomes visible. When the controlling element is inactive / false, the group becomes invisible.
- **Sub-element Visibility** - Controls the visibility of the individual elements that are contained in the group. Each of the sub-elements becomes visible when the controlling model element takes on the value that is associated with the sub-element.

Using the Panel Builder

The Panel Builder automatically generates panels from charts and binds the interactors. These panels can be edited with the panel editor.

To start the Panel Builder from the Charts tab of the Rational Statemate main window, highlight a chart and select **Tools > Panel Builder**. The **Panel Builder** dialog box displays.



The Panel Builder window contains the following fields:

- **Chart Name** - Specifies the name for a chart you want to use as the basis for a panel. Alternatively, you can browse for a chart.
- **Panel Name** - Specifies the name for the panel you are creating. Alternatively, you can browse for a panel name.
- **Scope of Elements** - Specifies which elements are used in the panel.

  The buttons above the horizontal line let you select the charts to include in the scope of the panel:

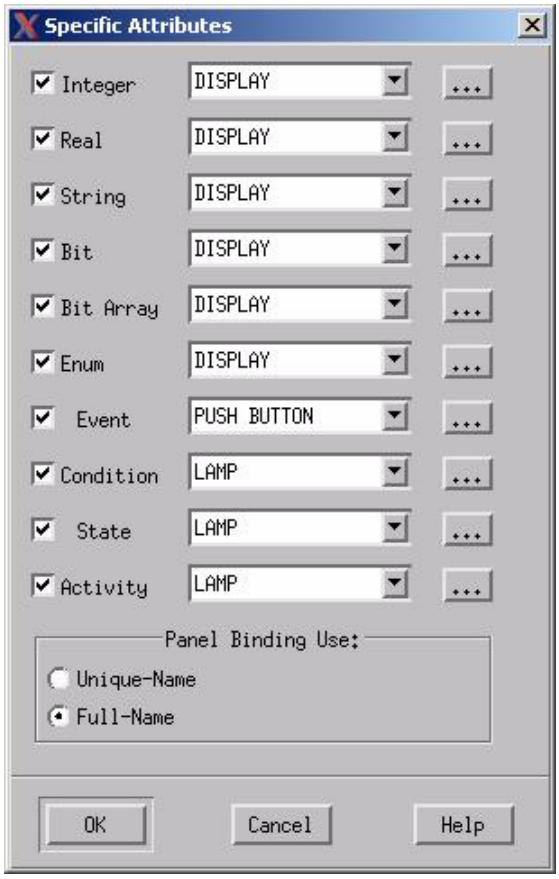
  - **With Descendant** - Enables you to include descendant charts. If you select this button, you can merge the descendant charts into one panel by selecting **Merge to One Panel**.
  - **With Generic Instances** - Enables you to include elements inside Generic Instances. If you select this button, you can merge the generic instances into one panel by selecting the button **Merge to One Panel**.
  - **Merge to One Panel -** When this button is pushed all controls from all scopes are merged into a single panel. When this button is not pushed, the tool will create a separate panel for each of the descendant charts or generic instances.

    The **Merge to One Panel** flag is enabled when either the **With Descendants** or the **With Generic Instances** buttons are pushed. Otherwise it is disabled.

  The buttons below the horizontal line let you select the elements that the panel will include, for each of the charts defined in the scope:

  - **Top Level Inputs Outputs -** Use only elements with top-level inputs or outputs.
  - **Used In Chart -** Use all the elements in the chart.
  - **Defined in Chart** - Use only elements defined in the chart.
  - **Referenced In Info Flow -** Uses elements that are part of the information flow.
  - **Expand Arrays -** Use all elements of an array.
  - **Expand Records -** Use all fields in a record.

- **Advanced** - This optional field opens the Specific Attributes window, as shown in the following figure. This window enables you to set the default interactor for each element type in the model.



Element Properties

This section describes elements and element properties. The topics are as follows:

- Understanding Elements
- Creating and Modifying Elements
- Searching for Elements

Every element in a model has a corresponding set of properties, that is, additional information about the element, such as descriptions, attributes, and relationships with other elements. Such additional information can be formal (that is, possessing some semantics that is relevant to the model and its behavior) or informal.

Some properties are relevant to all types of elements, while others apply only to certain types. The following table lists the properties and the element types to which they apply.

Property	Element Type		
	All	Graphical	Textual
Short description (one line)	X	X	X
Description (unlimited text)	X	X	X
Attributes	X	X	X
Link to external file	X	X	X
Behavioral description		X	X
Design attributes (for Micro C code generation)		X	X
Formally defined (e.g., data type, variable)			X

You can create or modify element properties by using the Properties window.

The following sections discuss properties and the Properties window in more detail.

Understanding Elements

A model is made up of *elements* whose definitions are stored as properties. In the properties dialog box, you create or modify element properties and search for elements using the **Search** tool. see **Creating and Modifying Elements** for more information.

Rational Statemate supports the following elements:

- **Textual Elements**
- **Graphical Elements**
- **Chart Elements**

Each kind of element supports a variety of types, sub-types, and structures.

The following sections explain in more detail the kinds of elements that Rational Statemate supports and the types, sub-types, and structures associated with them.

Textual Elements

Rational Statemate supports the textual elements (types, sub-types, and structures) described in the following table.

Element	Description
Types	
Data-Item	A unit of information that can be one of the following: • Numeric - Integer or Real • String • Bit or Bit Array • User-Defined Type • Record • Union Aliases are supported with the generated code the same as for compounds, a macro with the alias name and definition.
UserDef Type	A data-type that consists of several fields of possibly different types. A user-defined type is analogous to the `typedef` statement in C or the `type is` statement in Ada. User-defined types are often required to be visible throughout the entire model, so they are usually defined in a global definition set.
Condition	A persistent signal whose value is either TRUE or FALSE. Something occurring over a span of time, for example, the light is on. All conditions are enclosed in square brackets (for example, [C]). In general, the action `tr! (C)` has the effect of setting the truth value of condition C to TRUE, and the corresponding action `make_false (C)` (abbreviated `fs! (C)` sets it to FALSE. (Contrast with Event.)

Element	Description
Event	An instantaneous signal used for synchronization purposes. An event indicates that something has happened. Events occur at a precise instant in time, and if not immediately sensed they are lost. Events “live” only for the duration of the step immediately following their occurrence. (Contrast with Condition.)
Action	Any “work” done as a result of one of the following: • Making a transition in a statechart • Executing a static reaction within a state • Executing a mini-spec within an activity A single action can consist of making an assignment, generating an event, invoking a defined (named) action, or several special types of expressions (starting/stopping/ suspending activities, clearing history, etc.).
Info-Flow	Information flows are containers for other elements (such as conditions, events, data items, and other information flows). They reduce the number of flow lines, which makes the chart more readable and helps the viewer to better comprehend the specification.
Field	In addition to basic types, a data item can be a composite of named components, referred to as fields, each of which can be a data item of any type or a condition. Rational Statemate supports two kinds of composites: records and unions. The entire construct is referenced by its name (e.g., on a flow line), while a particular field is referenced using the dot notation: `<record/union reference>.<field reference>`
Subroutine	You can define function, procedure, and task subroutines using: • K&R C • ANSI C • Ada • Rational Statemate Action Language • Procedural Statecharts (for procedures only) • Flowchart You can use subroutines: • Within a model as part of triggers and actions • Connected to activities/blocks to describe their implementation • Connected to Rational Statemate elements as callbacks In addition, any C code that has been used to describe subroutines within a model can automatically be included within the generated code. Subroutines have textual information like any other Rational Statemate element (for example, short description, long description, and attributes).
Sub-Types	
Integer	One of five primitive data types in the Rational Statemate language. (The other four are: real, bit, bit array, and string.) The maximum value of integers allowed depends on the architecture of the machine on which Rational Statemate is running. On a 32-bit machine, it is (2**31)-1.

Element	Description
Real	One of five primitive data types in the Rational Statemate language. (The other four are: integer, bit, bit array, and string.) A real is also sometimes called a floating point number. In Rational Statemate, values can be referred to in either 'nnn.mmm' form or 'n.mmm E+ee' form. The values allowed depend on the architecture of the machine on which Rational Statemate is running. This is usually in the range -1.0 E+38 to 1.0E +38.
Bit	One of five primitive data types in the Rational Statemate language. (The other four are: integer, real, bit array, and string.) A bit can hold a single binary value. Its literals are 0b1 and 0b0.
Bit-Array	One of five primitive data types in the Rational Statemate language. (The other four are: integer, real, bit, and string.) A bit array is an array of bits. The length and indices are specified when the element is defined. The defaults values are: length - 32 bits, right index (most significant bit) - 31, and left index (least significant bit - 0. Bit arrays are treated as unsigned numbers by Rational Statemate. The software supports both implicit and explicit conversion to other Rational Statemate types.
String	One of five primitive data types in the Rational Statemate language. (The other four are: integer, real, bit, and bit array.) A string can hold any number of characters up to its defined length. The default length for a string is 80 characters.
Record	A data type that consists of several fields of possibly different predefined types or user-defined types. When a data item is declared to be a record, it is defined to contain all of its fields. A record is analogous to a structure in C and a record in Ada. You access a field in a record by using both the record name and field name separated by a '.' (for example, RECORD_NAME.FIELD_NAME). Aliases are supported with the generated code the same as for compounds, a macro with the alias name and definition.
Union	A data type that consists of several fields of possibly different predefined types or user-defined types. When a data item is declared to be a union, it is not defined to contain all of its fields. Rather, it is defined to contain one of its possible fields at any point in time. A union is analogous to a union in C and a variant record in Ada. You access a field in a union by using both the union name and field name separated by a '.' (for example, UNION_NAME.FIELD_NAME).
User-Type	A data type that consists of several fields of possibly different types. A user-defined type is analogous to the `typedef` statement in C or the `type is` statement in Ada. User-defined types are often required to be visible throughout the entire model, so they are usually defined in a global definition set.
Enum-Type	Enumerated values are constants. In Rational Statemate you can define a User-Defined Type (UDT) to be an enumerated type, and then define enumerated values for this type. For example, you can define a UDT as enumerated type COLOR, and then define COLOR as {RED, GREEN, BLUE, YELLOW}. These colors are enumerated values that you can use in expressions (such as `/X:=RED;`).

Element	Description
Structures	
Single	A data item that is neither an array nor a queue.
Array	A one-dimensional (vector) grouping of data items, events, and conditions under a single name whose individual elements are addressed through a reference index.
Queue	An ordered, unlimited collection of data items, all of the same data type. This data type can be any predefined or user-defined type. A queue is essentially a single structure that holds many elements. These elements can be put in at either the front or the back, but can only be retrieved at the front. • `put!` adds the value of the expression to the queue's tail. • `uput!` adds the value of the expression to the queue's head. **Note:** Put actions are accumulated and performed at the end of the step. This scheme reduces the chances of race conditions. • `get!` actions are performed when they are encountered. • `peek!` copies the queue's head element without removing it. • `fl!` totally clears the queue. • `q_length` returns the length of the queue.

Textual Types, Sub-types, and Structures

The relationships among textual types, sub-types, and structures are shown in the following table.

Type	Subtype	Structure
Data-Item	• Integer • Real • Bit • Bit-Array • String • Record • Union • User-Type	• Single • Array • Queue
UserDef Type	• Integer • Real • Bit • Bit-Array • String • Record • Union • User-Type • Enum-Type	• Single • Array • Queue
Condition	N/A	• Single • Array
Event	N/A	• Single • Array
Action	N/A	N/A
Info-Flow	N/A	N/A
Field	• Integer • Real • Bit • Bit-Array • String • Record • User-Type	• Single • Array • Queue
Subroutine	N/A	N/A

Note

When using structured elements (Arrays, Records, Unions) in a Rational Statemate model, it is recommended to create a User-defined Type of that structure, and define the data-items as of this type.

Default Values for Textual Elements

Textual elements include a default value field. The default value is either a literal value or const. This default value can be reset at runtime. For more information about resetting the default, see **Resetting Default Values for Elements**.

The **Default Value** field is available in the following textual elements:

- Data Items of basic types
- Conditions
- Record Fields
- User Defined Types of basic types and enumerated type
- User Defined Types of Enums

In the case of Data Items and User Defined Type, the default is available for the following data types:

- Integer
- Real
- String
- Bit
- Bit array
- Enum-type User Types

Limitation

Default value is NOT available for the following elements:

- Generic parameters
- Subroutine parameters
- Elements with usage alias, compound, and constant
- Unions Fields
- Queues

For the record's fields, all of the rules of default values apply as for single elements. For unions, there is no default value for each field. (In runtime, fields are set to the default value according to the type.) For arrays of basic types, the default value is a single value that is applied to all elements in the array.

Graphical Elements

Rational Statemate supports the graphical elements (types, sub-types, and structures) described in the following table.

Element	Description
Types	
State	The primary graphical element used in statecharts. States represent behavior of the system, or part of the system. States in a statechart differ from states shown in more traditional state diagrams, or finite state machines (FSM), in two ways: they can be divided into sub-states hierarchically and they represent parallel state behavior. There are two types of states: • Or-states are similar to states in traditional FSM. The statechart can be in only one Or-state at the same level of hierarchy at one time. Or-states are represented with a rounded rectangle. • And-states are shown by dividing an Or-state into sub-states with a dashed line. And-states show concurrent, or parallel, behavior.
Activity	The primary graphical element used in activity charts. Activities represent functions in the functional view of the system. An activity represents something that transforms inputs into outputs. There are three types of activities: • Internal activities (solid rectangle) • External activities (dashed rectangle) • Control activities (rounded rectangle) Activities can be allocated to modules (structure) and can contain statecharts. You can specify the behavior of an activity by connecting it to a subroutine. • Procedure-like activities can be connected to procedures within any of the languages supported. • Internal primitive activities (reactive-controlled and reactive-self) can be connected to tasks (no mini-specs or decomposition is allowed). • External activities can only be connected to tasks.
Module	The primary graphical element used in module charts. Modules are used to represent the structure of the system. There are two types of modules: internal (solid rectangle) and external (dashed rectangle). The functionality of a module is shown by describing it by an activity chart.
Actor	Any entity (person or system) that performs certain roles in the system defined by a boundary box. An actor is depicted as a stick figure and only interacts with a use case
Use-Case	A distinct piece of functionality within a system.
Boundary-Box	Depicts the limits of the system and is shown as a rectangle spanning all the use cases in the system

Element	Description
Sub-Types	
Or	One of two types of states that can be used in a statechart. The Or-state enables a user to represent sequential behavior. The Or-state is similar to the states used in traditional state diagrams or finite state machines. The statechart can be in one, and only one, Or-state at any one time (at a particular level of the state hierarchy). Or-states can be subdivided into smaller states (decomposed).
And	And-states (sometimes called concurrent or orthogonal states) enable a user to represent parallel behavior. When a statechart enters one And-state, it simultaneously enters all other And-states at that level of the state hierarchy. And-states can be subdivided into smaller states (decomposed).
Internal	Any activity within the scope of the topmost activity in a particular activity chart.
External	Any activity outside the scope of the topmost activity in a particular activity chart. Because activity charts are hierarchical, an external activity is usually resolved to a box in a chart higher in the chart hierarchy. However, an external activity can be resolved to a box that is an internal activity at a higher level. In this case it remains simply an external activity when referenced in the lower chart. The General Preference “Strict External-Activity Resolution” allows more flexibility while resolving External Activity to its definition Activity. Set the preference: “General Preferences...” -> “Strict External-Activity Resolution” to “No” to allow more flexibility and to “Yes” to keep the previous behavior.
Control	Describes the behavior of the activity in which it resides, and controls activities on the same hierarchical level. An activity can have only one control activity. An @ symbol precedes the title of a control activity.
Data-Store	Stores information on activities for later use. Data stores can also be used to total large volumes of data, continuously accumulating over time. Data stores are always basic; they cannot contain other data stores or activities.
External Router	See **Working with the Router** (External Router).
Router	See **Working with the Router** (Internal Router).
Execution	Any module within the scope of the topmost module in a particular module chart.
Storage	Analogous to a data store in an activity, except it is used in a module. Stores information on modules for later use. Data stores can also be used to total large volumes of data, continuously accumulating over time. Data stores are always basic; they cannot contain other data stores or modules.
Structures	
Basic	Specifies charts that have no descendants.
Non-Basic	Specifies charts that have descendants.

The relationships between graphical types, sub-types, and structures are shown in following table.

Type	Subtype	Structure
State	• Or • And	• Basic • Non-Basic
Activity	• Internal • External • Control • Data-Store • External Router • Router	• Basic • Non-Basic
Module	• Execution • External • Storage	• Basic • Non-Basic
Actor	• N/A	• N/A
Use-Case	• N/A	• N/A
Boundary-Box	• N/A	• N/A

Chart Elements

Rational Statemate supports the chart elements described in the following table:

Element	Description
Statechart	Describes the system's behavior over time, including the dynamics of activities, their control and timing behavior, the states and modes of the system, and the conditions and events that cause modes to change and other occurrences to take place. It thus also provides answers to questions about causality, concurrency and synchronization. Statecharts constitute an extensive generalization of state-transition diagrams. They allow for multi-level states, decomposed in an and/or fashion, and thus support economical specification of concurrency and encapsulation. They incorporate a broadcast communication mechanism, timeout and delay operators for specifying synchronization and timing information, and a means for specifying transitions that depend on the history of the system's behavior. Each element in the statechart has properties, which can contain additional information. For example, an event element can be used to define a compound event by an expression involving other events and conditions.
Activity Chart	Describes the functional view of the system using activities as the primary building block. This is sometimes referred to as the process-oriented view. A system description can contain one or more activity charts. Activity charts, which can be connected to module charts, describe the functionality of individual modules. Activity charts can be connected to statecharts. Statecharts either define the behavior of individual activities or control groups of activities as a control activity.

Use-Case Diagram	Illustrates – at a very high level – the relationship between “actors” (whoever or whatever interacts with the system being designed) and the system itself. They provide a natural high-level view of the intended external functionality of the system that is understandable by engineers and non-engineers alike.
Sequence Diagram	Depicts the sequence of actions that occur in a system, visually capturing the dynamic behavior of a system.
Flowchart	Represents a process graphically. A flowchart represents the entire process from start to finish, showing inputs, pathways and circuits, and action or decision points.
Module Chart	Describes the structural view of the system using modules as the primary building block. A system description can contain one or more module charts. Module charts are at the top of the chart hierarchy in a system model. Module charts can be connected to activity charts. Activity charts describe the functionality of individual modules.
Global Definition Set (GDS)	Contains definitions of user-defined types as well as constant data items and conditions. The elements that appear in a GDS are visible in the entire model. Data types defined in a chart or inherited from a parent chart take precedence over data types defined in a GDS. A GDS is similar to a chart in that both are configuration items of the model. That is, both charts and GDSs contain parts of the model and can be saved and loaded separately from other parts. A GDS cannot contain any other graphical or non-graphical information. There can be several GDS’s in one model, but there is no hierarchical relationship between them, or between them and the charts in the model. The GDS editor enables you to create a new GDS or access an existing GDS and enter the Properties window for the specified GDS. The GDS Visibility feature enables you to control the usage of definitions in a Global Definition Set (GDS) and to restrict their visibility. It allows the integration of some independent developed models, without having to worry about collisions between “global” variables and types, defined in each one of the different subsystems

The relationships between each supported chart and possible chart uses (generic/regular/procedural) are shown in following table.

Chart Type	Chart Use
Statechart	• **Generic** - Generic charts enable reuse of parts of a specification. A generic chart makes it possible to represent common portions of the model as a single chart that can be instantiated in many places, and in this it is similar to a procedure in a conventional programming language. • **Regular** - A non-generic chart. • **Procedural** - A Procedural Statechart is a specialized derivative of a statechart.
Activity Chart	• **Generic** - Generic charts enable reuse of parts of a specification. A generic chart makes it possible to represent common portions of the model as a single chart that can be instantiated in many places, and in this it is similar to a procedure in a conventional programming language. • **Regular** - A non-generic chart.
Use-Case Diagram	• **Regular** - A non-generic chart.
Sequence Diagram	• **Regular** - A non-generic chart.
Flowchart	• **Generic** - Generic charts enable reuse of parts of a specification. A generic chart makes it possible to represent common portions of the model as a single chart that can be instantiated in many places, and in this it is similar to a procedure in a conventional programming language. • **Regular** - A non-generic chart. • **Procedural** - A Procedural Flowchart is a specialized derivative of a Flowchart, which is used as a possible implementation of a Subroutine.
Module Chart	• **Generic** - Generic charts enable reuse of parts of a specification. A generic chart makes it possible to represent common portions of the model as a single chart that can be instantiated in many places, and in this it is similar to a procedure in a conventional programming language. • **Regular** - A non-generic chart.
Global Definition Set	N/A

Creating and Modifying Elements

You create and modify elements using the **Properties** dialog box and the Quick-Edit mode.

Quick-Edit Mode

This mode is mainly intended for use when quick intensive editing is required. You may use the quick-edit mode to change elements when it is not necessary to maintain all of its internal data-structure caches and thus their associated calculation overhead. Analysis tools, such as Check-Model, Simulation, Code-Generators, Documentor, always run in regular mode and perform all required calculations. In the Main window toolbar use the Quick-Edit mode toggle to switch the mode on and off.

- **Pushed** - Quick-Edit is “On”
- **Not pushed** - Quick-Edit is “Off”

The Quick-Edit Startup Mode preference can be manually set to Disable, On, or Off (Default):

Elements in the Chart Hierarchy

When working in Quick-Edit mode, all elements that meet these criteria appear in the Elements-View (and in their Properties form) as “Textual” (i.e., unresolved):

- Referenced in the chart-hierarchy
- Normally resolved to a defined element higher in the tree

Quick-Edit Mode Limitations

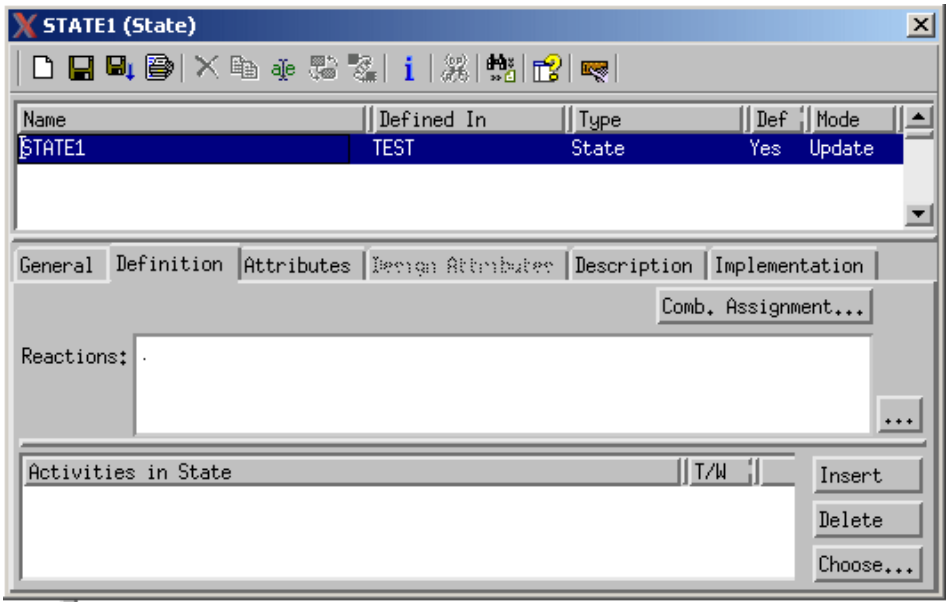
The following operations are not available while in Quick-Edit mode:

- Interface report
- Info
- Filtered check out
- Where referenced
- Advance Query
- Activity Interface Browser

Invoking the Properties Dialog Box

To open the Properties dialog box, use one of the following methods:

- In the Rational Statemate main window, select **Edit > Properties** (if you have selected charts).
- Select an element in the element matrix and click the **Properties** icon
- From within the element matrix right pane, double-click on an element (The result of this method is shown in illustration below.)
- Select one or more elements in the element matrix. Select **Edit > Properties**.



Note

Matrices, that support graphical sorting, display a sorted-by column and direction following a graphical sort operation (clicking the column header).

You may also display the Properties dialog box from the graphic editors (except the **Panel Editor**) using this method:

1. Select **Tools** > **Chart Properties** or within the graphic canvas, right-click to display a popup menu.
2. Click **Properties**.

Subroutine Properties

In the Properties dialog box, the Implementation list toolbar contains the **Select Implementation** icon to set a subroutine as the selected implementation. Highlight the implementation and click the **Select Implementation** icon to make this change.

To assist you, the “Selected Implementation” combination box in the **Definition** tab also has an asterisk (“*”) before any non-empty implementation.

Editing Multiple Elements

The properties dialog allows editing of multiple elements in a single operation, when the elements have the same properties dialog (e.g., all elements are single-variable-integer-data-items).

Record/Union Field Properties

Record/Union field properties can be accessed directly from the **Search** tab without opening the Properties of the containing Record/Union. The multiple elements editing feature is also available for Record/Union fields, when their properties were opened from the **Search** tab.

Cut, Copy and Paste Operations on Record/Union Fields

The Record/Union field matrix in the Properties dialog can Cut, Copy and Paste using these methods:

- A pop-menu with the three operations on the matrix
- The copied information from the matrix can be pasted into another Record/Union field matrix. as well as into a text editor as text
- Text in the format of the matrix (<name> <type>) can be copied from a text editor and pasted into the matrix (example: “MY_FIELD Integer”)

Properties Preference “Mass edit overwrite values”

The “Mass edit overwrite values” preference controls mass changes when editing elements. When the preference is set to **Yes,** the **Save** operation overrides existing values of edited property fields with the new entered values in all affected elements. When the preference is set to **No** (the default setting), the **Save** operation does not override existing values of edited property fields. Only property fields with no previous value are assigned with the new entered value.

Properties Window

This section describes these Properties dialog box features:

- Individual Property Fields Display
- Toolbar Operations
- Property Information Displayed in Tabs
- Creating and Modifying Elements
- Resetting Default Values for Elements

Individual Property Fields Display

Individual element properties are displayed as rows in the upper pane of the Properties dialog box. For each property, the following fields are displayed:

- **Name Field -** This is a read-only field that displays the name of the element(s). To rename an element, select **Edit** > **Rename**.
- **Defined In Field** - displays the chart or GDS in which the element is resolved, if applicable.
- **Type Field** - displays the data type of the element.
- **Def Field -** displays Yes if the element is defined or No if it is not.
- **Mode -** Indicates if the element can be updated or is read-only.

Toolbar Operations

The Properties window has the following toolbar operations:

- **Open References** - displays the references to/for the selected element. The referenced elements are appended to the listing in the current Properties dialog box unless you specify **Options > Open New Editor For References**.
- **Save** buttons - Enable you to save changes made to elements and move to the previous or next element.

Searching Charts

The Search dialog box for Charts includes two check boxes:

- **Defined** - Controls search of real charts
- **Undefined** - Controls search of referenced charts (which do not exist in the workarea, but are referenced by instance boxes)

Property Information Displayed in Tabs

The lower pane of the **Properties** dialog box contains tabs that provide additional information for the property you select in the upper pane and is based on the type of element you are modifying.

Note

The fields in the dialog box are resized together when the dialog box is resized.

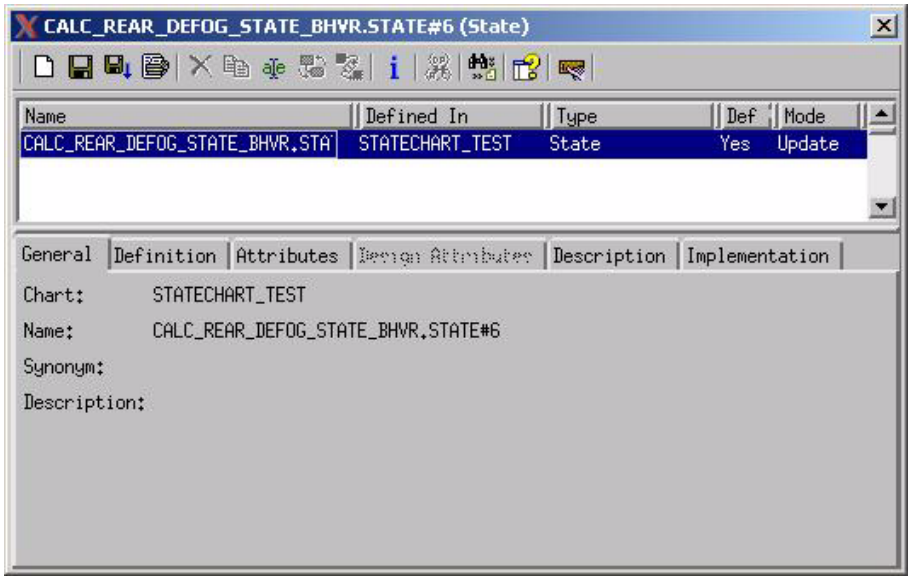
In this section, the tabs that do not change for each textual element type are described, including:

- **General Tab**
- **Attributes Tab**
- **Design Attributes Tab**
- **Description Tab**

The tabs that change with each textual element type are shown in the appropriate sections. They include:

- Description tab
- Implementation tab

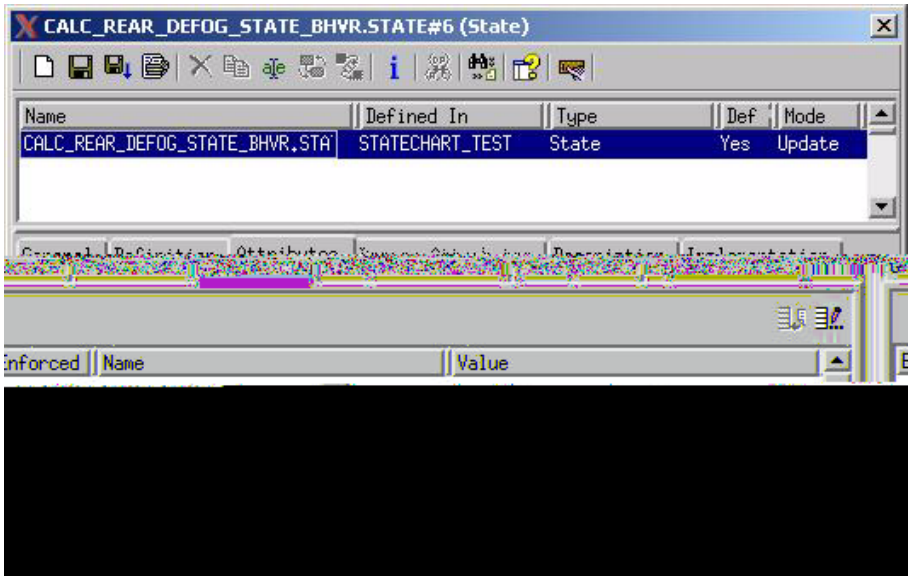
General Tab



The **General** tab displays basic information about the element:

- **Chart** - A text box that displays the name of the chart the element is associated with.
- **Name - A** text box displays the name of the element.
- **Synonym** - A text box for entering an alternate name for the element. This name may be used in other expressions to reference the element.
- **Description** - A text box for adding a brief description of the element (up to 80 characters).

Attributes Tab



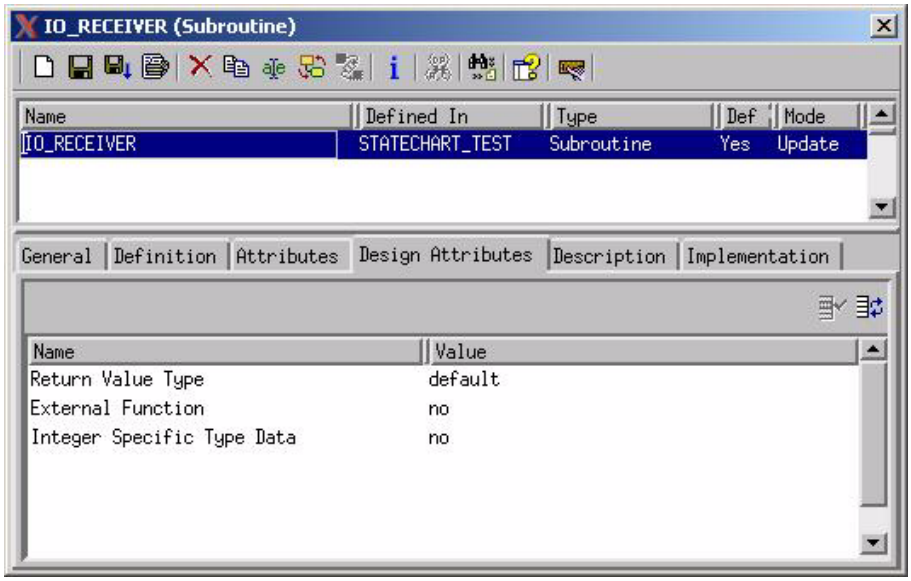
In the **Attributes** tab, the value of an existing attribute in the matrix can be changed.

To add, remove, or rename attributes, click **Edit Attributes** . The **Attributes** dialog box displays.

Mapping of Statemate element types to attribute definition files :

Element type	Definition file
Module-charts	mch_attributes.def
Activity-charts	ach_attributes.def
Statecharts	sch_attributes.def
Flowcharts	fch_attributes.def
Sequence-diagrams	qch_attributes.def
Use-case diagrams	uch_attributes.def
GDS's	gds_attributes.def
Modules	md_attributes.def
Activities	ac_attributes.def
Data-stores	ds_attributes.def
Routers	rt_attributes.def
States	st_attributes.def
Use-cases	uc_attributes.def
Actors	at_attributes.def
Boubdary-boxes	bn_attributes.def
Components	component_attributes.def
Actions	an_attributes.def
Conditions	co_attributes.def
Data-items	di_attributes.def
Data-types	dt_attributes.def
Events	ev_attributes.def
Fields	fd_attributes.def
Information-flows	if_attributes.def
Subroutines	sb_attributes.def

Design Attributes Tab

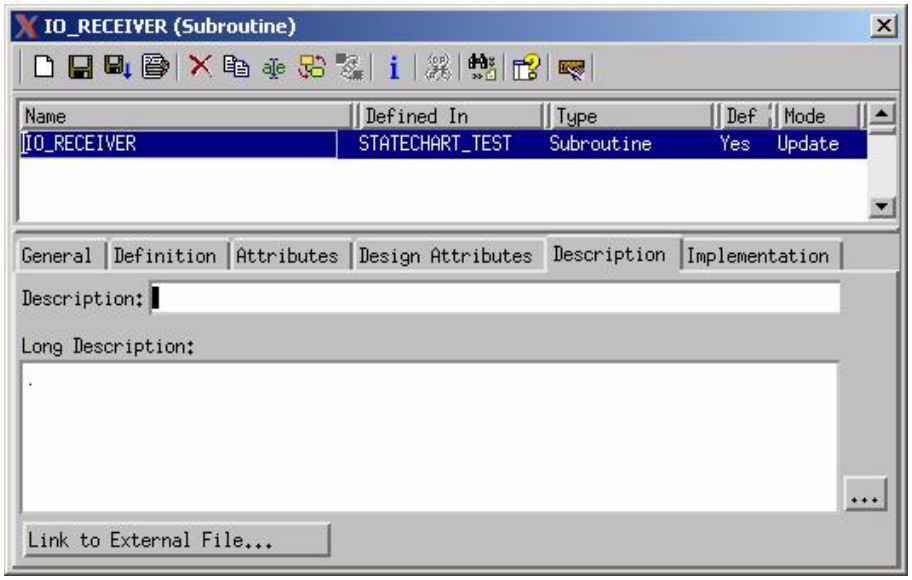


The **Design Attributes** tab lets you edit the design attributes for the element.

Note

This tab is available only in the non-classic mode.

Description Tab



The description tab lets you enter a description for each of the element.

Creating and Modifying Elements

The following sections explain how to use the **Properties** dialog box to create or modify the following elements and their various types:

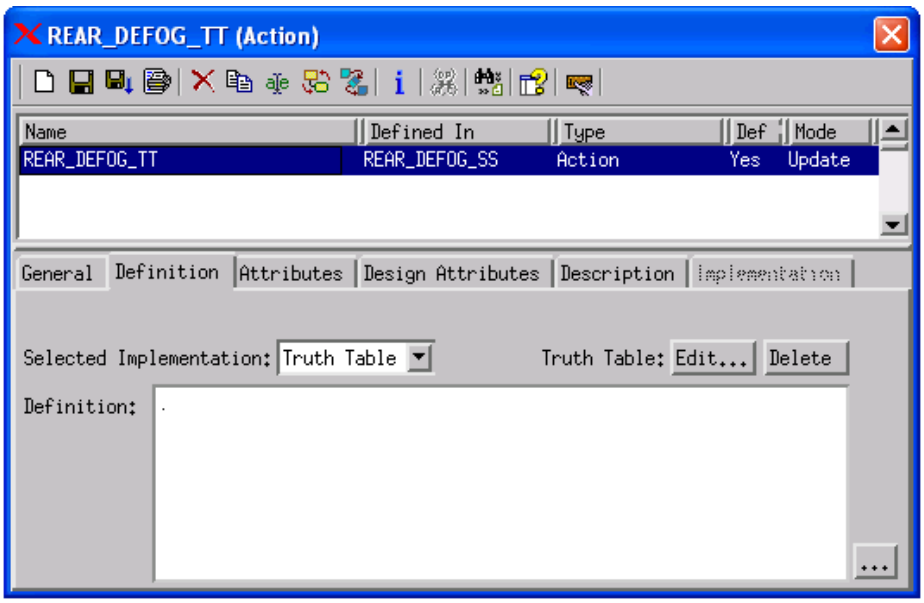
- **Textual Elements**
- **Graphical Elements**

Textual Elements

The following textual element types can be created or modified in the **Properties** dialog box:

- **Action**
- **Data-Item**
- **User Defined Type**
- **Condition and Event**
- **Fields**
- **Information-Flow**
- **Subroutine**

Action



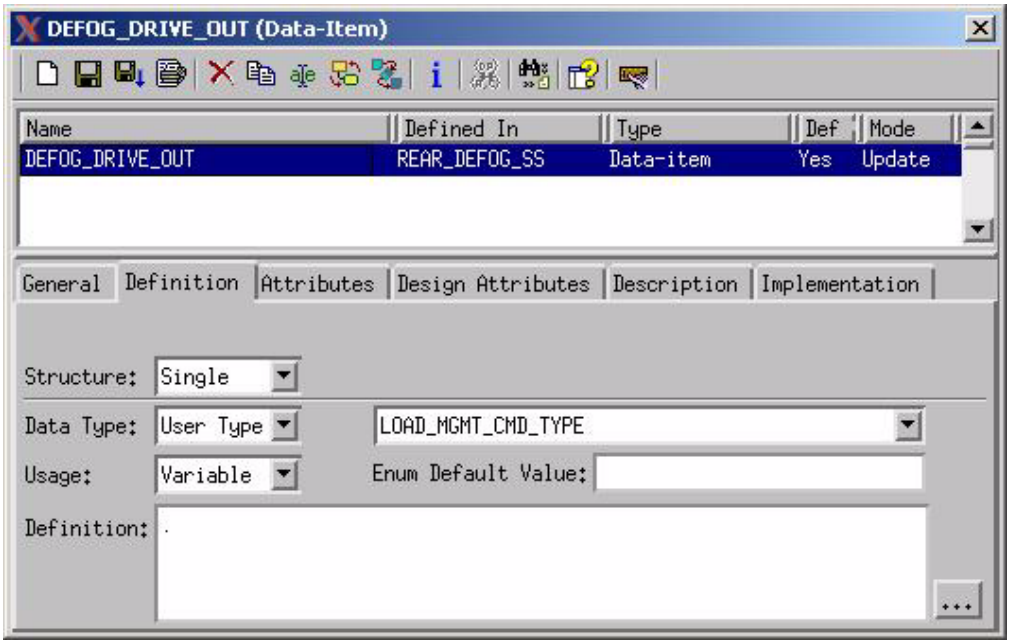
The **Action** properties dialog box contains the following fields:

- **Selected Implementation -** A pull-down menu with the following options:
  - **Definition -** Specifies the action taken is defined in the Description text box.
  - **Truth Table -** Specifies the action taken is defined in a truth table. Actions defined in a truth table are considered compound actions and follow the same scoping rules as other textual elements in the model.

    For more information on truth tables and for instructions on how to define a truth table in Rational Statemate, see **Truth Tables**.
  - **Best Match -** Directs Rational Statemate to select the implementation to use if more than one has been specified. The order in which the analysis tool selects the implementation is based on the ordering in this pull-down menu.
  - **None -** No implementation is specified.
- **Definition -** A text box for the definition, if **Definition** is chosen as the **Selected Implementation**.
- **Truth Table Edit/Delete** buttons - Used to set up or delete a truth table, if **Truth Table** is chosen as the **Selected Implementation**.

  For more information on truth tables and for instructions on how to define a truth table in Rational Statemate, see **Lookup Tables**.

Data-Item



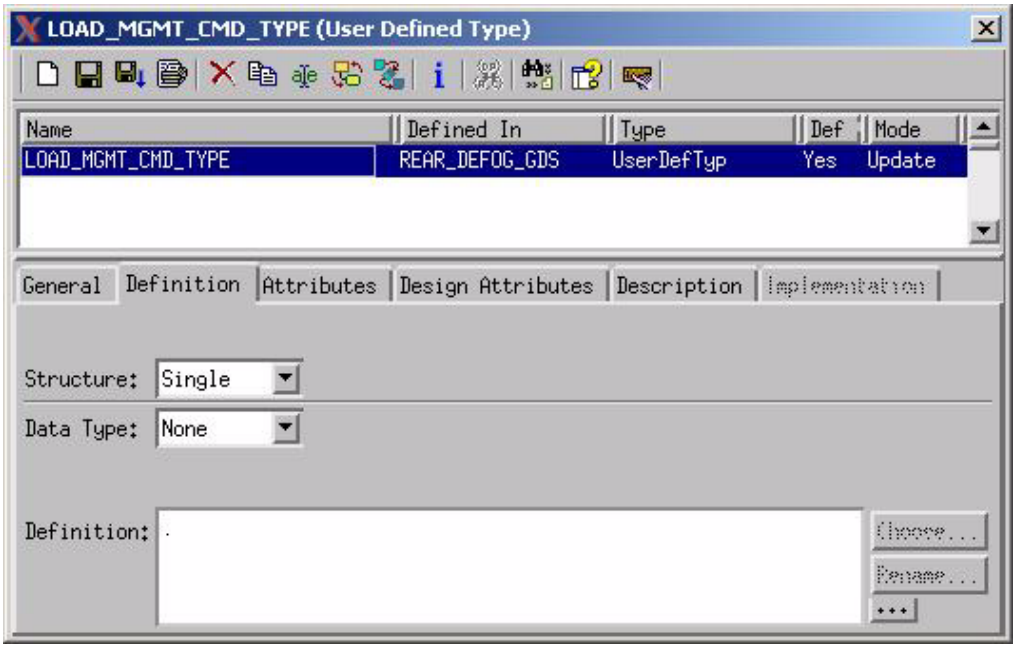
The **Data-Item** properties dialog box contains the following items:

- **Structure** - A pull-down menu to select the type of structure associated with the element.
- **Data Type** - A pull-down menu to select a data type for the element: integer, real, string, bit, bit-array, user type, record, or union. For more information on sub-types, see **Sub-Types**.

  Additional values can be set, depending on the data type selected. For example, for an integer variable, the following values can be set:

  - **#Bits** - Number of bits.
  - **Min** - Minimum value.
  - **Max** - Maximum value.
  - **Default Value** - Default value for variable.
- **Usage** - A pull-down menu to select the type of usage. A Rational Statemate information element may have one of four usages: variable, constant, alias, and compound. Certain usages are restricted based on how the element is referred to in the model and on the type of the element.
- **Definition** - A text box for a definition of the element logic.

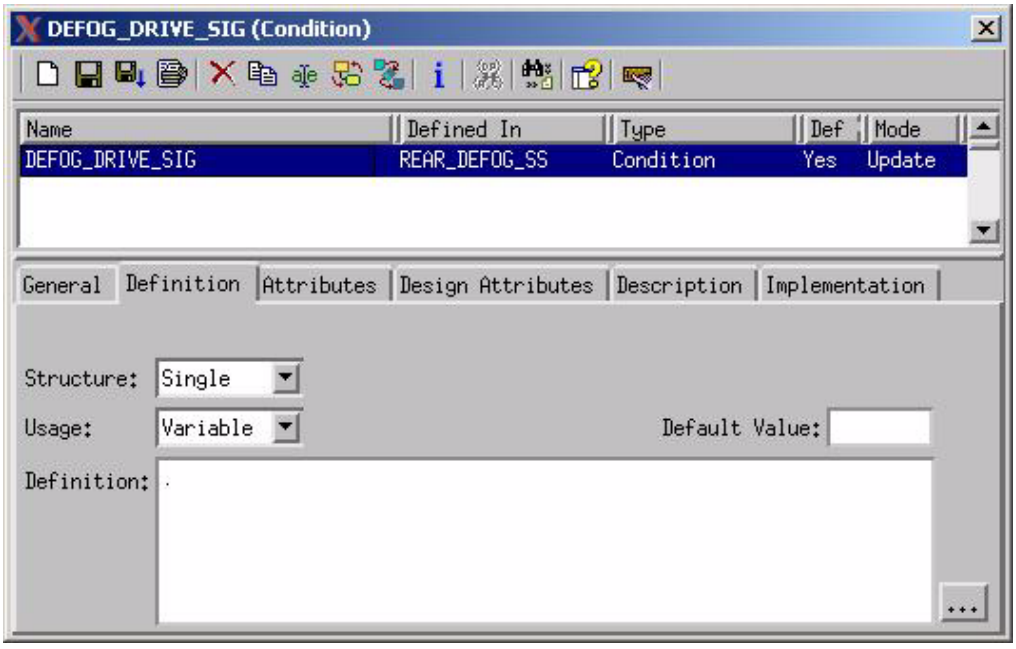
User Defined Type



The **User Defined Type** properties dialog box contains the following items:

- **Structure -** A pull-down menu to select the type of structure associated with the element.
- **Data Type -** A pull-down menu to select a data type for the element: integer, real, string, bit, bit-array, user type, record, or union. For more information on sub-types, see **Sub-Types**.
- **Usage -** A pull-down menu to select the type of usage. A Rational Statemate information element may have one of four usages: variable, constant, alias, and compound. Certain usages are restricted based on how the element is referred to in the model and on the type of the element.
- **Definition -** A text box for a definition of the element logic.

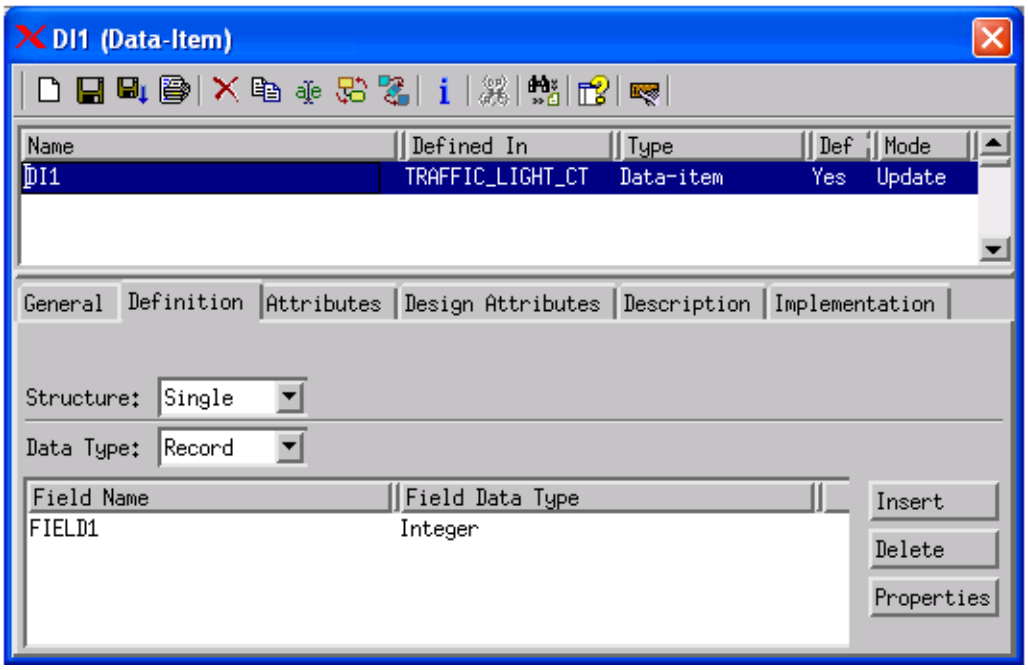
Condition and Event



The **Condition and Event** properties dialog box contains the following items:

- **Structure** - A pull-down menu to select the type of structure associated with the element.
- **Usage** - A pull-down menu to select from one of four usages: variable, constant, alias, and compound. Certain usages are restricted based on how the element is referred to in the model and on the type of the element.
- **Definition** - A text box for the definition of the conditions applicable to this element.

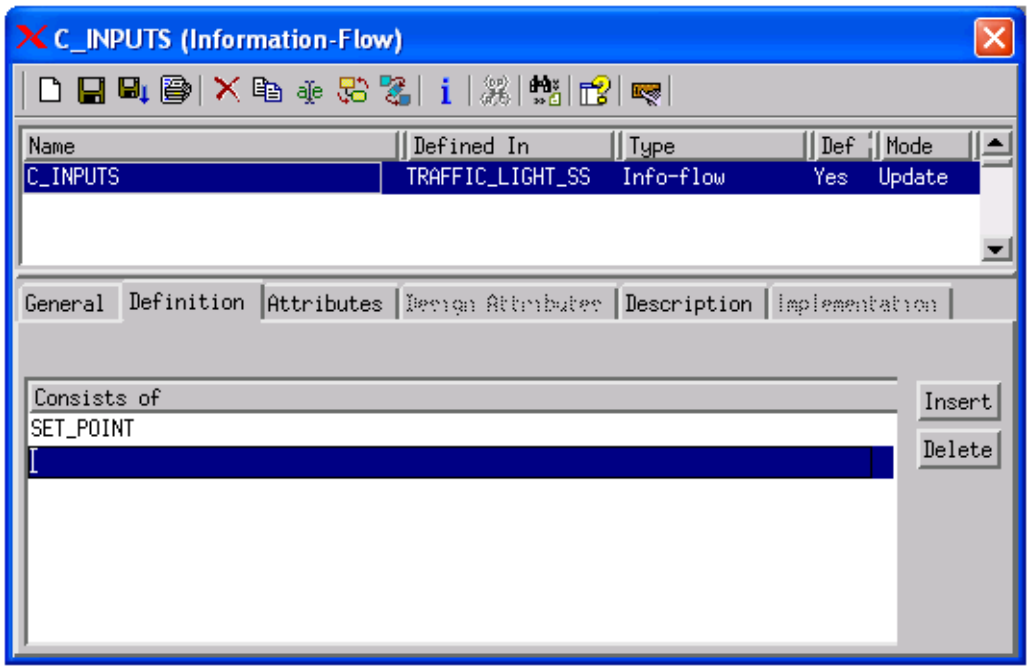
Fields



The **Fields** properties dialog box contains the following items:

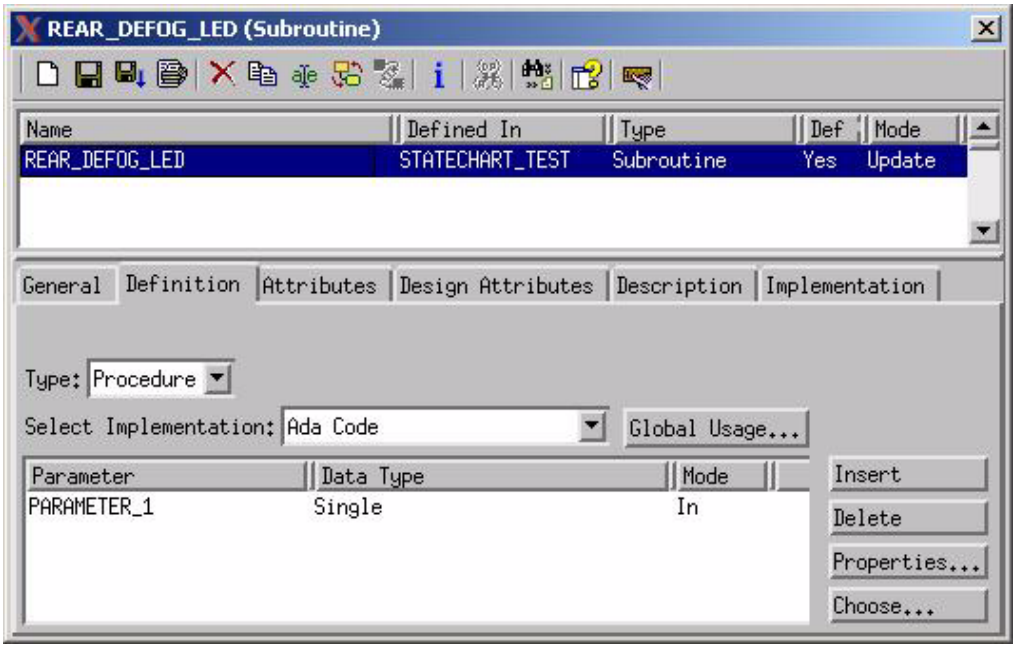
- **Structure -** A pull-down menu to select the type of structure associated with the element: single or array.
- **Data Type -** A pull-down menu to select a data type for the element: integer, real, string, bit, bit-array, user type, record, or union. For more information on sub-types, see **Sub-Types**..
- The **Field Name** and **Field Data Type** columns contain the names of the fields and their associated data types.
- **Insert** button - inserts the selected field.
- **Delete** button - deletes the selected field.
- **Properties** button
- **Info** button

Information-Flow



The **Information-Flow** properties dialog box contains one field: **Consists of.** It displays a matrix of elements that make up the information flow. Elements can be added or deleted by using the **Insert** and **Delete** buttons. A popup menu contains the cut, copy, and paste functions for adding or deleting information flow items from and into the list.
Double-click on an item in the "Consist of" list navigates to the properties of that item. The Properties preference “Open New Editor for References" determines whether this opens in a new properties dialog or the current one.

Subroutine



The **Subroutine** properties dialog box contains the following items:

- **Type -** A pull-down menu to select from the following types:
  - **Procedure -** A subroutine that has no return value but can have multiple parameters. Each parameter can be INPUT, OUTPUT, or INPUT/OUTPUT.
  - **Function -** A subroutine that returns a value and can have multiple parameters. All function parameters are inputs.
  - **Task -** A special form of procedure connected to activities for C and Ada only. Task parameters can be INPUTs, OUTPUTs, or INPUT/OUTPUTs.
- **Selected Implementation - A** pull-down menu to select the implementation to be used for the subroutine. The following implementations are supported:
  - **K&R C Code**
  - **ANSI C Code**
  - **ADA Code**
  - **Procedural Statechart -** A specialized derivative of a statechart and does the following:
    * Is executed entirely in one step.
    * Must contain a termination connector.
    * When called, runs from the default to the termination connector (including any loops) within a single step.

- **Flowchart -** A specialized derivative of a flowchart and does the following:
  * Is executed entirely in one step.
  * When called, runs from the start to the end (including any loops) within a single step.
  * May return a value, if used with subroutine of type “Function.”
  * Generic and offpage instances are allowed.
- **Rational Statemate Action Language -** A subroutine written using the standard Rational Statemate action statements. Normally, action statements are executed within the context of Rational Statemate semantics (for multiple action statements within a single step, all of the assignments occur at the same time). Within an action language subroutine, all of the assignments occur in the order the statements were written.
- **Truth Table -** Specifies that the action to be taken is to be defined in a truth table. Actions defined in a truth table are considered compound actions and follow the same scoping rules as other textual elements in the model.

  For more information on truth tables and for instructions on how to define a truth table in Rational Statemate, see **Truth Tables**.
- **Lookup Table -** Specifies that the action to be taken is to be defined in a lookup table. Actions defined in a lookup table are considered compound actions and follow the same scoping rules as other textual elements in the model.

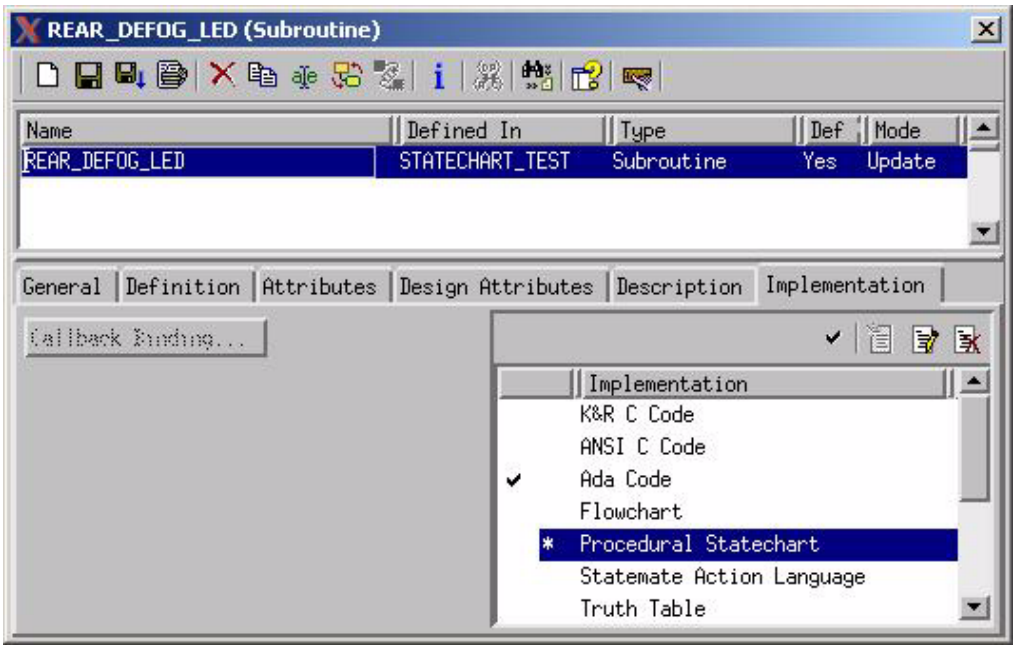
  For more information on lookup tables and for instructions on how to define a lookup table in Rational Statemate, see **Truth Tables**.
- **External Tool -** Enables invoking formatting script on ANSI-C code.
- **Best Match -** Used when more than one implementation/language has been specified for the subroutine. Best Match indicates that each analysis tool (Simulation or Code Generation) uses the first language (for which you have specified an implementation) that can be used.

  For example, Ada code generation does not use a C code definition. The order in which the analysis tools selects a language implementation is the order in which the languages appear in the menu listing.

  You can override the Best Match by explicitly selecting an implementation/ language to be used. If the explicitly selected language has not been previously specified or is not appropriate, the analysis tool uses the Best Match selection.
- **None -** No implementation is used.
- **External Code/None -** Allows the user to define the interface of the subroutine. Complete implementation of the subroutine is not provided.

Note: This mode is only available in the Rational Statemate non-classic mode.

Implementation Tab



The **Implementation** tab in the **Subroutine Properties** dialog box lets you access and edit subroutine implementations. A subroutine can have more than one implementation. The implementation that is actually used is the one chosen by the **Selected Implementation** drop-down menu. The possible implementations are listed in the figure:

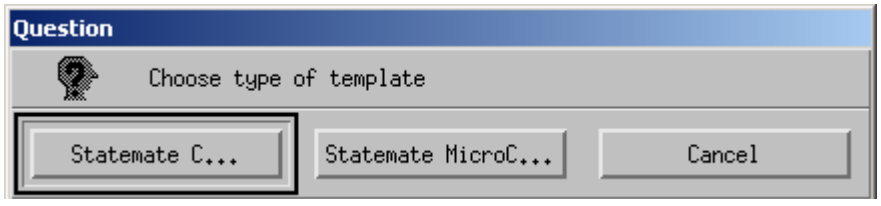
- **K&R C Code**
- **ANSI C Code**

  In the ANSI-C code option, you choose between two options:

  Rational Statemate C

  Rational Statemate Micro C

  Clicking on the Generate Template icon, you get the query:



- **ADA Code**

- **Procedural Statechart -** A specialized derivative of a statechart and does the following:
  - Is executed entirely in one step
  - Must contain a termination connector
  - When called, runs from the default to the termination connector (including any loops) within a single step
- **Flowchart -** A specialized derivative of a flowchart and does the following:
  - Is executed entirely in one step.
  - When called, runs from the start to the end (including any loops) within a single step.
  - May return a value, if used with subroutine of type “Function.”
  - Generic and offpage instances are allowed.
- **Statement Action Language -** A subroutine written using the standard Rational Statemate action statements. Normally, action statements are executed within the context of Rational Statemate semantics (for multiple action statements within a single step, all of the assignments occur at the same time). Within an action language subroutine, all of the assignments occur in the order the statements were written.
- **Truth Table -** Specifies that the action to be taken is to be defined in a truth table. Actions defined in a truth table are considered compound actions and follow the same scoping rules as other textual elements in the model.

  For more information on truth tables and for instructions on how to define a truth table in Rational Statemate, see **Truth Tables**.
- **Lookup Table -** Specifies that the action to be taken is to be defined in a lookup table. Actions defined in a lookup table are considered compound actions and follow the same scoping rules as other textual elements in the model.

  For more information on lookup tables and for instructions on how to define a lookup table in Rational Statemate, see **Truth Tables**.
- **External Tool -** Enables invoking formatting script on ANSI-C code.

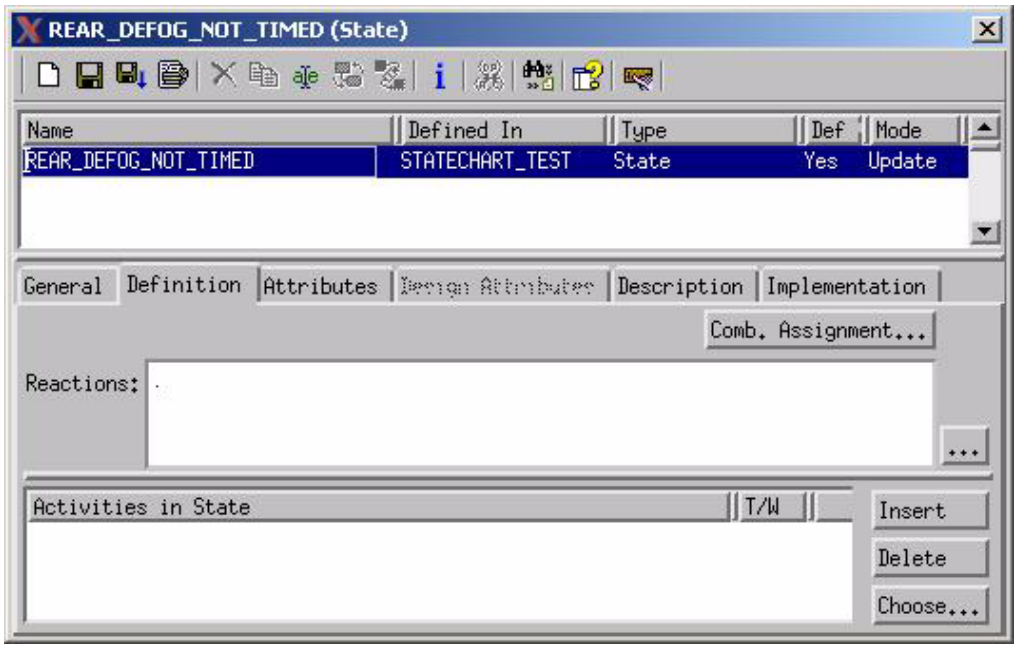
  The preference Generate Template for External Tool Implementation controls the generation of the template for the External Tool subroutine implementation. When the preference is set to “Yes”, the tool will create template before invoking the External Tool editor.

Graphical Elements

The following graphical element types can be created or modified in the **Properties** dialog box:

- State
- Transition
- Instance State of Generic Chart
- Activity
- Instance Activity
- Module
- Use Case

State



The **State** properties dialog box contains the following items:

- **Reactions -** A text box for defining static reactions.

  Static reactions describe the behavior that takes place within a specific state. For example:

  ```
  While in (S1) DO
  [POWER_ON]/tr! (LIGHT_ON);
  COUNTER:=0
  ```

  They also describe actions that occur when there's a transition to enter or exit the associated state. For example:

  ```
  On entering (S1) DO
      /st! (activity_warm_up)
  On exiting (S1) DO
     /sp! (activity_warm_up)
  ```

  Separate multiple reactions in the Properties window with a double semicolon (;;). States that have static reactions are distinguished by a '>' symbol after the chart name (e.g., ALARM>).

- **T/W -** Specifies the start/stop correspondence between a state and an activity: T (Throughout) is equivalent to a static reaction of:

  ```
  entering/);;
  exiting/sp!(A)
  ```

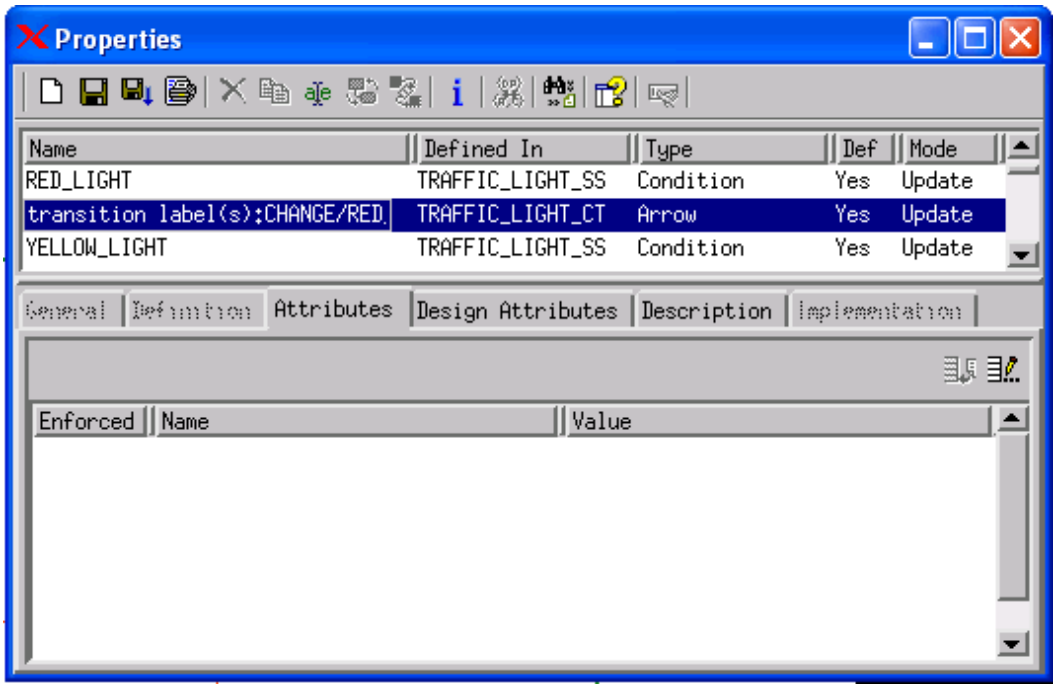
  For a reactive-controlled activity A, exiting the state stops the activity. For a reactive-self activity, there is usually an exit transition from the state triggered by the event stopped (A) This implies that if and when A stops of its own accord, the state will be exited.

  W (Within) indicates that the activity A is activated sometime during the time the system is in the state and is intended to be used as a temporary specification when functional decomposition has not yet reached a point at which Throughout can be used. When the state is exited, A stops (unless, of course, it had stopped earlier for some other reason). However, A does not necessarily start when the state is entered.

- **Insert -** inserts a new line above the selected line.
- **Delete -** deletes the selected line.
- **Choose -** Opens the Activities in State dialog box from which you can select activities to control using the Activities in State list.

Transition

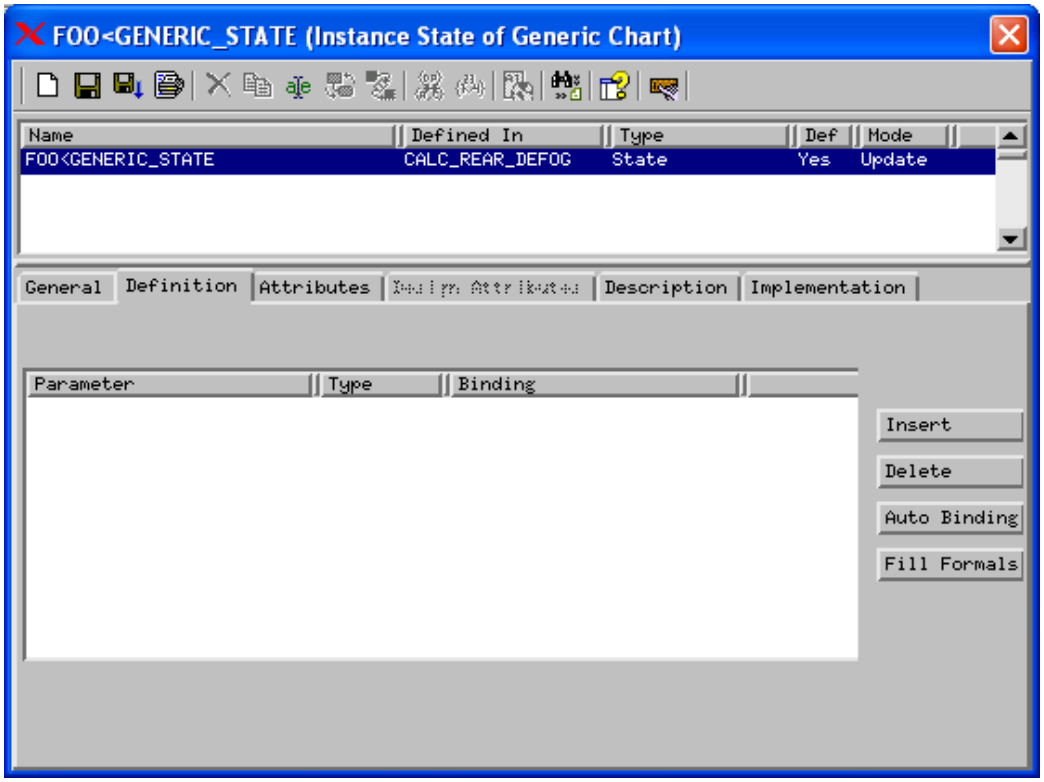
To open the **Transition** properties dialog box, in the Statechart Graphical Editor, select **Tools** > **Extended Properties**.



The **Transition** properties dialog box contains the following items:

- **Attributes** tab
- **Design Attributes** tab
- **Description** tab - A text box for a long description of the transitions.

Instance State of Generic Chart



The **Instance State of Generic Chart** properties dialog box contains the following items:

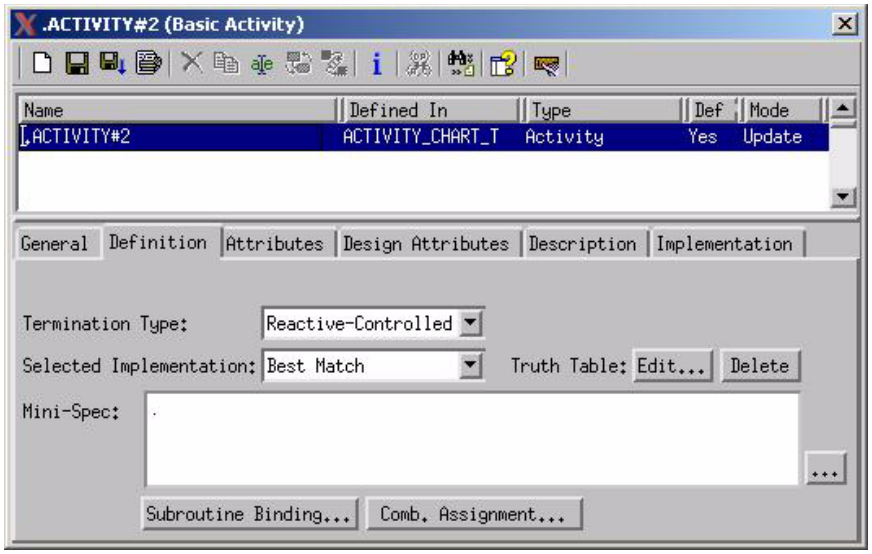
- **Parameter area -** contains the names of actual parameters. An actual parameter is an element or constant, defined in the scope of an instance of a generic chart, that is mapped into a formal parameter of the generic chart during instantiation. The actual binding of an element must match the type of the formal parameter.
- **Insert -** adds a new line above the selected line.
- **Delete -** deletes the selected line.
- **Auto Binding -** determines whether automatic binding is used.
- **Fill Formats -** sets the fill color and type.

Activity

Rational Statemate supports the following activity elements:

- Activity
- Basic activity
- Control activity
- External activity
- Datastore
- Router

Each activity element has its own unique properties, though the operations are essentially the same and designed to be self-explanatory. For the sake of brevity, this manual uses the **Basic Activity** properties dialog box for the example because it is the most inclusive.



The **Activity** properties dialog box contains the following items:

- **Termination Type - A** pull-down menu for selecting a type of termination. A termination expresses the activation rules for an activity. This should be reflected in the mini-spec of the activity. The following terminations are supported:
  - **Reactive-Controlled -** This type of activity is started by the control activity at the next higher level of the activity chart hierarchy. Once started, it will remain active for one or more steps until it is stopped by the same control activity. This type of activity can itself contain a control activity or mini-spec.
  - **Reactive-self -** This type of activity is started by the control activity at the next higher level of the activity chart hierarchy. This type of activity can itself contain a control activity or mini-spec. Once started, it remains active for one or more steps, until it ends by entering a termination connector in its control activity or executing a stop action in its mini-spec.
  - **Procedure-like -** This type of activity is started by the control activity at the next higher level of activity chart hierarchy. Once started, it runs to completion in a single step. This type of activity can itself contain a mini-spec, but can not contain a control activity.
- **Selected Implementation** - A pull-down menu for selecting an implementation. The following implementations are supported:
  - **Mini-Spec -** Specifies that the activity is to be described by a mini-spec.
  - **Subroutine Binding -** Specifies that the activity is to be described by a subroutine.
  - **Truth Table** - Specifies that the activity is to be described by a truth table.

    For more information on truth tables and for instructions on how to define a truth table in Rational Statemate, see **Truth Tables**.
  - **Best Match -** Directs Rational Statemate to select the implementation to use where more than one has been previously specified. The order in which the analysis tool (Simulation or Code Generation) selects an implementation is the order in which the options appear in the menu listing (as shown here). You can override the Best Match by explicitly selecting an implementation option. If the explicitly selected option has not been previously specified, the analysis tool uses the Best Match selection.
  - **None -** No implementation is to be used.
- **Mini-Spec** - If a mini-spec implementation is selected, in this text box, define the fields that contain trigger/action expressions, except for procedure-like activities, which contain action expressions.
- **Is Activity** - A text box to link an activity in one part of a module chart/activity chart hierarchy to another activity in a different activity chart.

- **Implemented by Module** - If linking an activity, in this text box, specify the link from an activity to a lower-level module that implements the functionality of the activity.
- **Subroutine Binding -** Associates a subroutine with the current activity. The subroutine is executed when the activity chart is accessed. The subroutine can be a procedure or a task.
- **Combinational Assignment -** A combinational assignment is the expression used to assign a value to a combinational element with the following syntax:

```
X := Y1 when C1 else
     Y2 when C2 else
     ...
     Ynz
```

  where X is a variable condition or data-item, Y1 to Yn are expressions, and C1 to Cn are condition expressions.
- **Truth Table -** If a truth table is selected as an implementation, the truth table is defined here.
  - **Edit -** a button that brings up the truth table for editing.
  - **Delete -** a button that deletes the truth table.

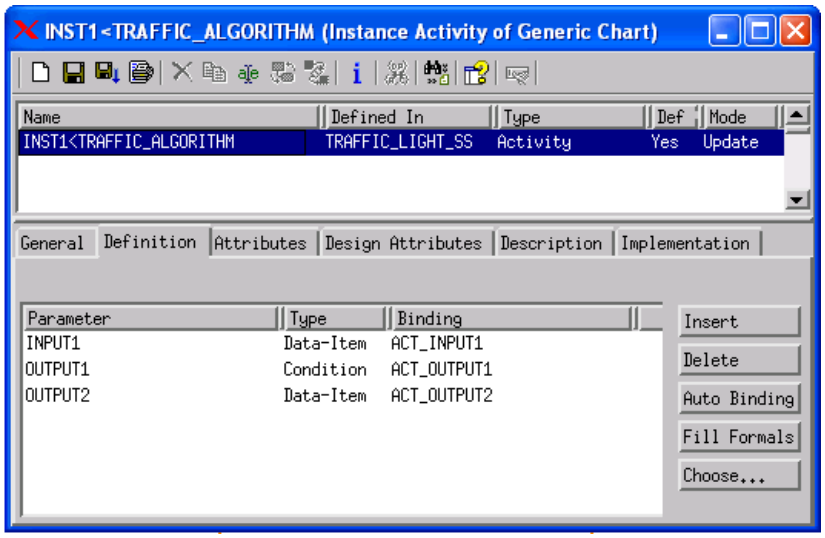
Note: For more information on truth tables and for instructions on how to define a truth table in Rational Statemate, see **Truth Tables**.

Instance Activity

Rational Statemate supports the following instance-activity elements:

- Component Chart
- Offpage Chart

Each instance-activity element has its own unique properties, though the operations are in many cases the same and designed to be self-explanatory.



The **Instance Activity of Generic Chart** properties dialog box contains the following items:

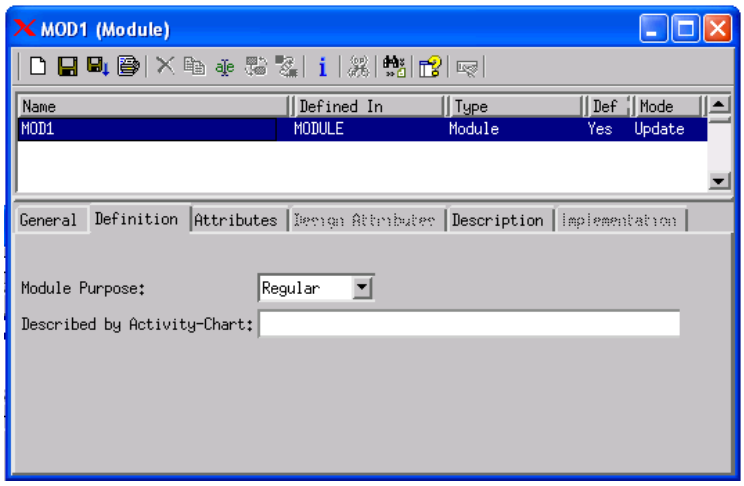
- **Parameter** - See Description Tab for a description.
- **Termination Type** - A pull-down menu. See Activity for a description.

Module

Rational Statemate supports the following module elements:

- Module
- Storage module

For the sake of brevity, this manual uses the **Module** properties dialog box for the example because it is the most inclusive.

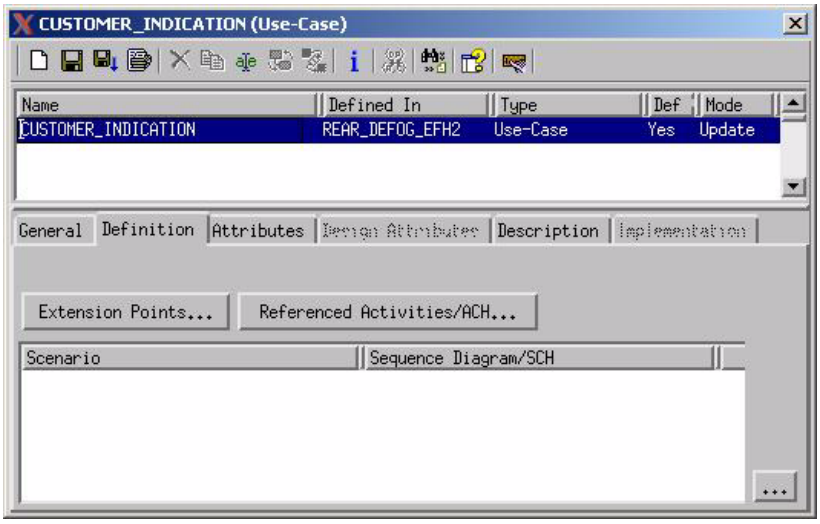


The **Module** properties dialog box contains the following items:

- **Chart** - displays the name of the chart the element is associated with.
- **Name** - displays the name of the element.
- **Synonym** - A text box for adding an alternate name for the element. The name may be used in other expressions to reference the element.
- **Description** - A text box for adding a brief description of the element (up to 80 characters).
- **Module Purpose** - A pull-down menu for selecting from the following:
  - **Regular** (default)
  - **Controller**
  - **Library**
  - **Bus**
- **Described by Activity Chart** - Specifies the activity chart the module is linked to.

Use Case

The use cases in a use case diagram can be described and linked with a Rational Statemate model through the use-case properties.



The **Use Case** properties dialog box contains the following items:

- **Use-Case External Description** - Describes the use case, using a customizable template.
- **Scenario** - List of scenarios associated with the use case. Each scenario can be linked to a sequence diagram and a set of attributes describing it.

The Scenario matrix includes full spreadsheet capabilities, including:

- Scalable cell width and height
- Cell dimensions saved with matrix
- Cell text wrapping

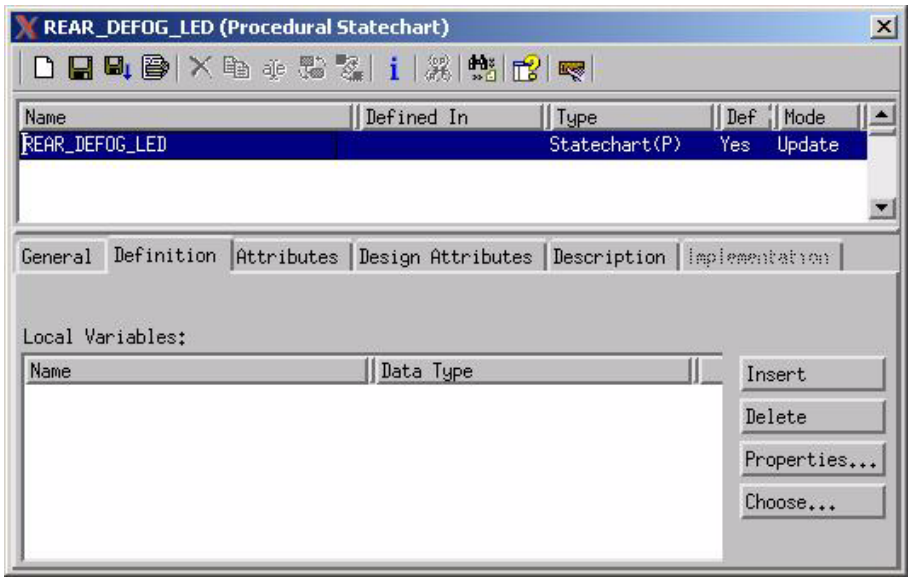
Chart Elements

Rational Statemate supports the following chart elements:

- Statechart
  - Basic
  - Generic
  - Procedural
- Activity chart
  - Basic
  - Generic
- Module chart
  - Basic
  - Generic

For the sake of brevity, this manual describes the most representative and inclusive chart elements.

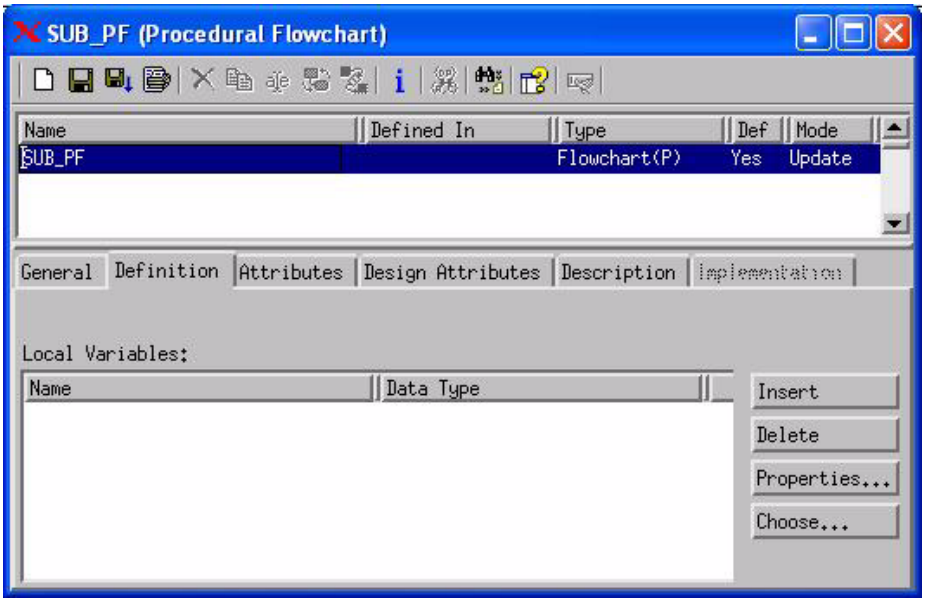
Procedural Statechart



The **Procedural Statechart** properties dialog box contains the following items:

- **Local Variables -** displays the name and data type of local variables for the chart.
- **Insert** - a button that inserts a new line above the selected line.
- **Delete** - a button that deletes the selected line.
- **Properties** - a button that opens the properties dialog box for a selected variable.
- **Choose** - a button that opens a window from which to select and add variables.
- **Info** - a button that displays information on a selected variable.

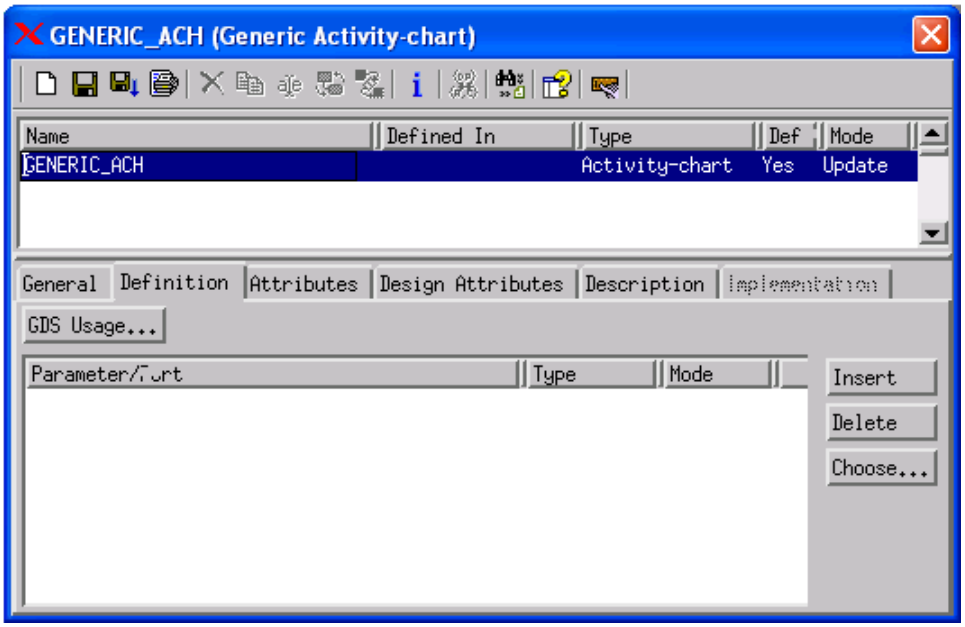
Procedural Flowchart



The **Procedural Flowchart** properties dialog box contains the following items:

- **Local Variables** - displays the name and data type of local variables for the chart.
- **Insert** - a button that inserts a new line above the selected line.
- **Delete** - a button that deletes the selected line.
- **Properties -** a button that opens the properties dialog box for a selected variable.
- **Choose -** a button that opens a window from which to select and add variables.
- **Info -** a button that displays information on a selected variable.

Generic Activity Chart



The **Generic Activity Chart** properties dialog box contains the following items:

- **Name** is a text box that displays the name of the element and consists of alphanumeric characters. This is a read-only field. To change the name, use the **Edit** > **Rename** operation.
- **Version** is the Rational Statemate built-in Configuration Management tool tracks versions using whole numbers (positive integers). When creating a new item and checking it into the databank, Rational Statemate assigns it a version number of “1”. Each time someone checks the item in, Rational Statemate increments the highest existing version number.

  Third-party CM tools may use other systems of version numbering, in which case the version numbers displayed by Rational Statemate conform to the format of the third-party tool.
- **Mode** for update or read-only. Determines whether:
  - Graphic editor editing features and drawing icons are active.
  - Changes made in the Properties window can be saved.
  - Read-only is the only available mode when either of the following apply.
  - You have read-only access to a chart.
  - You have opened a chart on which you do not have a lock.
  - You have already opened a chart in the same session.

- **Modified on/Created by/Created on -** Text boxes that contain the expected information and are read-only.
- **Description -** A text box for adding a brief description of the element (up to 80 characters).

For directions on how to use the **Parameters** section, see **Description Tab**.

Resetting Default Values for Elements

You may reset elements to their default values during runtime using these two operators:

- reset_element(<Element>) resets a single element to its default value
- (reset_all_elements()) resets the entire scope to the elements default value during runtime

Note: These operators are reserved words.

Both reset operators can be used in any place where an assignment expression is allowed, except for combinational assignments.

Both types of execution tools, simulation and the code generators, support these reset operators.

reset_element(TextualElement)

This operator receives only the parameter indicating which textual element to reset. The derived events (wr, ch) are generated.

reset_all_elements()

This operator does not use parameters. When it is called within a subroutine, it executes after exiting the subroutine. The default values are not assigned during the subroutine run. However, no derived events (wr, ch) are generated.

Elements can be excluded from the reset_all_elements operation by adding an attribute to their properties. Elements with an attribute named `STM_RESET_EXCLUDE` and value “Yes” are not reset to their default value when the reset_all_elements operation is executed. The existence of the attribute is considered for fields (i.e., a record field cannot be excluded from reset_all, only the whole record). The `STM_RESET_EXCLUDE` attribute affects the reset_element operation.

Searching for Elements

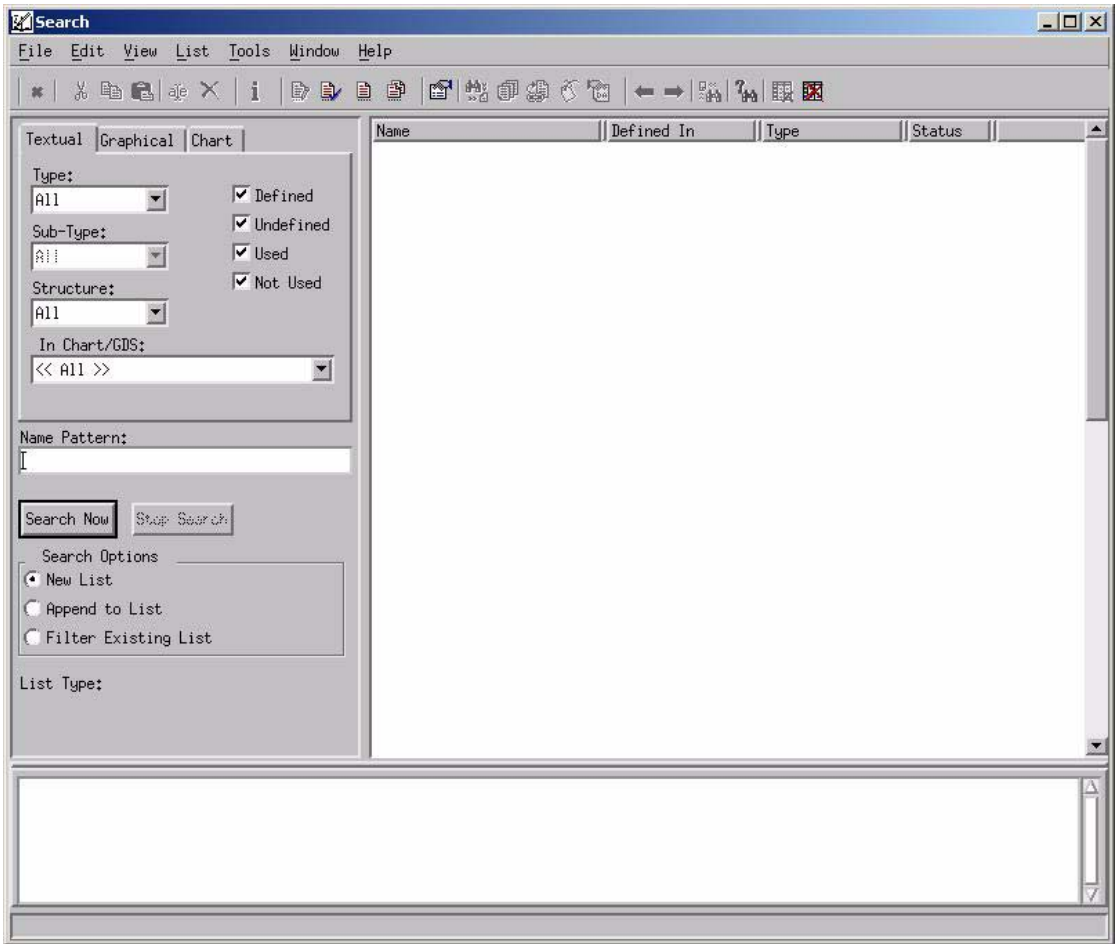
Elements, with specific properties, can be searched for using the **Search** tool. Use the **Search** tool to perform the following tasks:

- **Creating a List of Elements**
- **Saving a List**
- **Accessing a Stored List**
- **Filtering a List of Elements**
- **Appending to a List of Elements**
- **Running an Advanced Query**

Starting the Search Tool

To start the **Search** tool, do one of the following:

- In the Rational Statemate main window, select the **Search** icon.
- In the Rational Statemate main window, select the **Search** tab (shown below).
- In the Rational Statemate main window, select **Tools** > **Search**. The **Search** dialog box displays.


Search
File Edit View List Tools Window Help
Textual Graphical Chart
Type:
All
Sub-Type:
All
Structure:
All
In Chart/GDS:
<< All >>
Defined
Undefined
Used
Not Used
Name Pattern:
Search Now
Stop Search
Search Options
New List
Append to List
Filter Existing List
List Type:
Name
Defined In
Type
Status

Creating a List of Elements

The **Search** tool can create a list of elements.

To create a list of elements:

1. From the **Type** pull-down menu, select a primary element type (textual, graphical, or chart).

   Note: Narrow the scope of the search (filter the list) by using additional options.

2. (Optional) From the **Sub-Type** pull-down menu, narrow the scope of the search by selecting a sub-type.
3. (Optional) From the **Structure** pull-down menu, narrow the scope of the search by selecting a structure.
4. (Optional) From the **In Chart/GDS** pull-down menu, narrow the scope of the search by selecting a specific chart or GDS or by selecting **All** (the default).
5. (Optional) Narrow the scope of the search by enabling or disabling the following buttons (all the buttons are enabled by default):
   - **Defined** - Elements that are defined in a chart of GDS
   - **Undefined** - Elements that are undefined in a chart or GDS
   - **Used** - Elements that are used in a chart or GDS
   - **Not used** - Elements that are not used in a chart or GDS
6. (Optional) Enter a search pattern in the **Name Pattern** text box, using alpha-numeric characters, the underscore, and wildcards.
7. Enable the **New List** button in the Search Options area.
8. Click **Search Now** to create the new list.

The list of elements displays in the right pane. The search results display names that are up to 64 characters in length with descriptions that may be up to 250 characters long.

Note

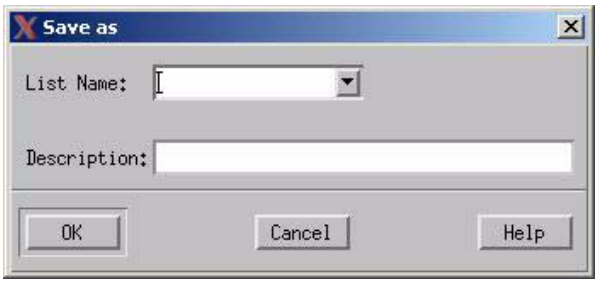
Open the Properties window for an element in the search list by double-clicking on the element.

Saving a List

Once a list is created, it can be saved.

To save a list:

1. On the **Search** tab, select **List** > **Save List As.** The **Save As** dialog box displays.



2. Do the following:

   a. Enter a name for the list in the **List Name** text box, or select a current list to overwrite from the pull-down menu.

      Note: If overwriting an existing list, you are prompted to confirm your decision.

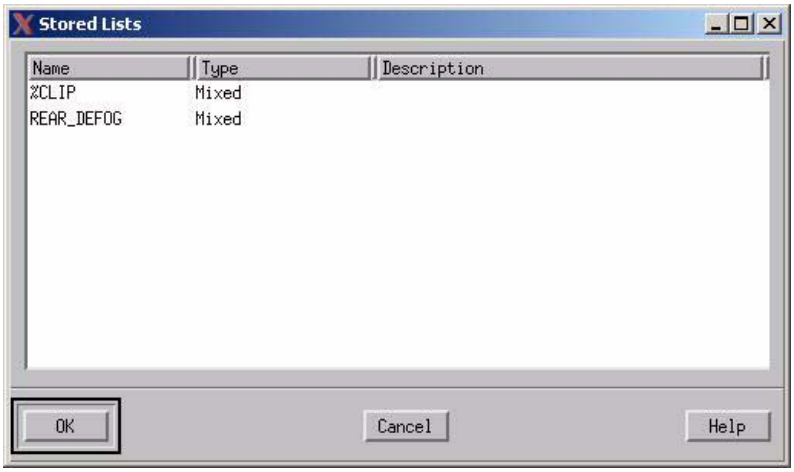
   b. (Optional) Enter a description for this list for future reference in the **Description** textbox.

   c. Click **OK** to confirm your selections and save the list.

Accessing a Stored List

To access a stored list:

1. From the **Search** tab, select **List** > **Open Stored List**. The **Stored Lists** dialog box displays.



2. Select a list to open.
3. Click **OK**. The stored list is displayed in the right pane of the **Search** dialog box.

Filtering a List of Elements

After creating a list, you may need to reduce further the number of elements in the list or group elements by certain categories. This is called filtering.

To filter a list of elements:

1. Open the existing list by following the instructions in **Accessing a Stored List**.
2. Choose filtering criteria by using the options described in **Creating a List of Elements**.
3. Select the **Filter Existing List** button under the **Search Options** area.
4. Click **Search Now** to filter the list.

   The filtered list displays in the right pane of the **Search** dialog box.

Note

You can open the Properties window for an element in the search list by double-clicking on the element.

Appending to a List of Elements

After you create a list, you may need to append additional elements to the list.

To append to a list of elements:

1. Open the existing list by following the instructions in **Accessing a Stored List**.
2. Choose filtering criteria by using the options described in **Creating a List of Elements**.
3. Select the **Append to List** button under the **Search Options** area.
4. Click **Search Now** to append the newly selected elements to the list.

   The expanded list displays in the right pane of the **Search** dialog box.

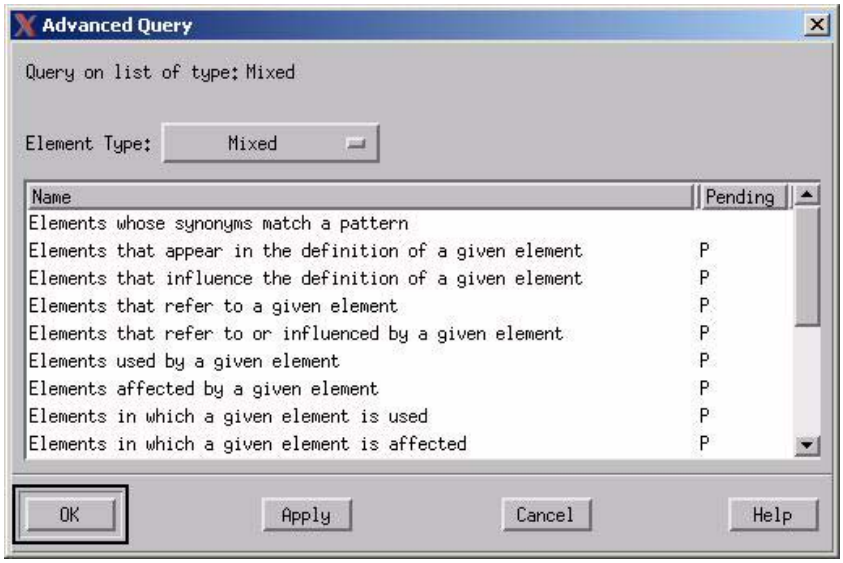
Note

You can open the Properties window for an element in the search list by double-clicking on the element.

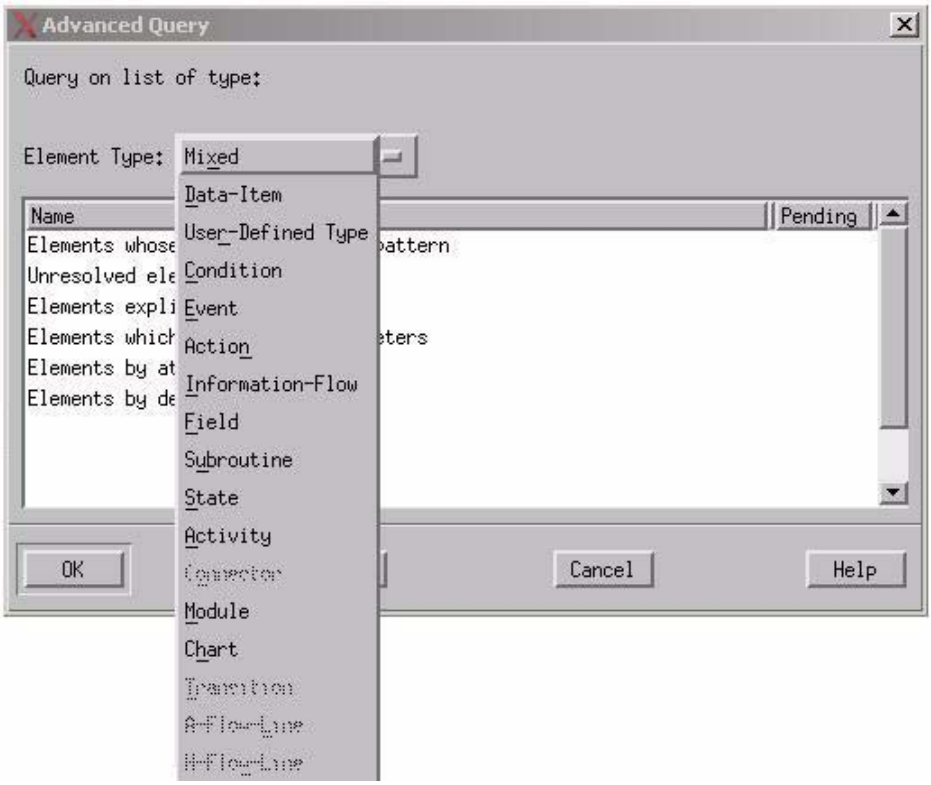
Running an Advanced Query

Sometimes the search criteria in the **Search** dialog box are not granular enough for your needs. When more detailed searches information is needed, used the advanced query. To run an advanced query:

1. With the **Search** dialog box displayed, click the **Advanced Query** icon or select **Tools** > **Advanced Query**. The **Advanced Query** window opens.



2. Select an element type from the **Element Type** pull-down menu. Depending on the element type you select, various query choices appear in the main list of queries.



3. Select a query, then click **Apply** to execute the query.
The query results appear in the right pane of the Search dialog box.

3.1 Some advanced queries, like **Elements whose synonyms match a pattern**, require user input. As a result, when you select these types of advanced queries, a dialog box displays for input.

3.2 To search for Element by attribute:
Enter Attribute name in the Name field;
In the Value field enter:
Explicit value - searches for elements with the specified value for the specified attribute name;
Asterisk ("*") - searches for elements with any value for the specified attribute name;
Empty string ("") - searches for elements with no value for the specified attribute name.

Leaving the Name field empty searches for elements with no attributes at all.

4. When you are finished executing queries, click **OK**.

The following table lists the supported advanced queries by element type.

Element Type	Supported Queries	User Input Required
A-flow Line	Elements whose synonyms match a pattern	Yes
	Elements that are targets of a given a-flow line	Yes
	Elements that are sources of a given a-flow line	Yes
	Elements flowing through a given a-flow line	Yes
	Highest level elements flowing through a given a-flow line	Yes
	Elements labeling a given a-flow line	Yes
Action	Elements whose synonyms match a pattern	Yes
	Elements that appear in the definition of a given action	Yes
	Elements that influence the value of a given action	Yes
	Elements that see a given action	Yes
	Elements that see or are influenced by a given action	Yes
Activity	Elements whose synonyms match a pattern	Yes
	Elements that see a given activity	Yes
	Elements that see or are influenced by a given activity	Yes
	Elements referenced by a given activity	Yes
	Elements referenced by or that influence a given activity	Yes
	Elements used by a given activity	Yes
	Elements affected by a given activity	Yes
	Elements resolved to a given external activity	Yes
Chart/GDS	Elements whose synonyms match a pattern	Yes
	Elements referenced by a given chart	
	Elements referenced by or influence a given chart	Yes
	Elements defined in a given chart	Yes
	Elements unresolved in a given chart	Yes
	Elements defined or unresolved in a given chart	Yes
	Textual elements defined in a given chart	Yes
	Textual elements unresolved in a given chart	Yes
	Textual elements defined or unresolved in a given chart	Yes
	Parameters/ports in a given chart	Yes
	In parameters/ports in a given chart	Yes
	Out parameters/ports in a given chart	Yes
	In/Out parameters/ports in a given chart	Yes
	Analysis Statistics/ports in a given chart	Yes

Element Type	Supported Queries	User Input Required
Condition	Elements whose synonyms match a pattern	Yes
	Elements that appear in the definition of a given condition	Yes
	Elements that influence the definition of a given condition	Yes
	Elements that see a given condition	Yes
	Elements that see or are influenced by a given condition	Yes
	Elements in which a condition is used	Yes
	Elements in which a given condition is affected	[P]
Data-item	Elements whose synonyms match a pattern	Yes
	Elements that appear in the definition of a given data-item	Yes
	Elements that influence the definition of a given data-item	Yes
	Elements that see a given data-item	Yes
	Elements that see or are influenced by a given data-item	Yes
	Elements in which a given data-item is used	Yes
	Elements in which a given data-item is affected	Yes
Data Type	Elements in which a given data-item is used	Yes
	Elements in which a given data-item is affected	Yes
	Elements that see or are influenced by a given data-item	Yes
Data-store	Elements whose synonyms match a pattern	Yes
	Element that see a given data-store	Yes
Event	Elements whose synonyms match a pattern	Yes
	Elements that appear in the definition of a given event	Yes
	Elements that influence the definition of a given event	Yes
	Elements that see a given event	Yes
	Elements that see or influence by a given event	Yes
	Elements in which a given event is used	Yes
	Elements in which a given event is affected	Yes
Field	Elements whose synonyms match a pattern	Yes
	Elements that appear in the definition of a given field	Yes
	Elements that influence the definition of a given field	Yes
	Elements that see a given field	Yes
	Elements that see or are influenced by a given event	Yes
	Elements in which a given event is used	Yes
	Elements in which a given event is affected	Yes
	Elements containing a given field	Yes

Element Type	Supported Queries	User Input Required
Function	Elements whose synonyms match a pattern	Yes
	Elements that see a given function	Yes
	Elements that see or influenced by a given function	Yes
Information-flow	Elements whose synonyms match a pattern	Yes
	Elements that appear in the definition of a given information flow	Yes
	Elements that influence the value of a given information-flow	Yes
	Elements that see a given information-flow	Yes
	Elements that see or are influenced by a given information-flow	Yes
M-flow Line	Elements whose synonyms match a pattern	Yes
	Elements flowing through a given m-flow line	Yes
	Highest level elements flowing through a given m-flow line	Yes
	Elements labeling a given m-flow line	Yes
Mixed	Elements whose synonyms match a pattern	Yes
	Elements that appear in the definition of a given element	Yes
	Elements that influence the value of a given element	Yes
	Elements that see a given element	Yes
	Elements that see or are influenced by a given element	Yes
	Elements used by a given element	Yes
	Elements affected by a given element	Yes
	Elements in which a given element is used	Yes
	Elements in which a given element is affected	Yes
	Unresolved elements	
	Elements explicitly defined	
	Elements that are parameters	
	Elements resolved to an external box	
	Elements by attribute	
Module	Elements whose synonyms match a pattern	Yes
	Elements that see a given module	Yes
	Elements referenced by a given module	Yes
	Elements referenced by or that influence a given module	Yes
	Elements resolve to a given external module	Yes

Element Type	Supported Queries	User Input Required
State	Elements whose synonyms match a pattern	Yes
	Elements that see a given state	Yes
	Elements that see or are influenced by a given state	Yes
	Elements referenced by a given state	Yes
	Elements referenced by or that influence a given state	Yes
	Elements used by a given state	Yes
	Elements affected by a given state	Yes
Subroutine	Elements whose synonyms match a pattern	Yes
	Elements that appear in the definition of a given subroutine	Yes
	Elements that influence the definition of a given subroutine	Yes
	Elements that see a given subroutine	Yes
	Elements that see or are influenced by a given subroutine	Yes
Transition	Elements whose synonyms match a pattern	Yes
	Elements labeling a given transition	Yes
	Elements that are targets of a given transition	Yes
	Elements that are sources of a given transition	Yes
	Elements used by a given transition	Yes
	Elements affected by a given transition	Yes
Undefined	Elements whose synonyms match a pattern	Yes
	Unresolved elements	
	Elements explicitly defined	
	Elements that are parameters.	
User-Defined Type	Elements whose synonyms match a pattern	Yes
	Elements that appear in the definition of a given User-Defined Type	Yes
	Elements that influence the definition of a given User-Defined Type	Yes
	Elements that see a given User-Defined Type	Yes
	Elements see or influence by a given User-Defined Type	Yes

Finding Where Elements are Referenced and Used

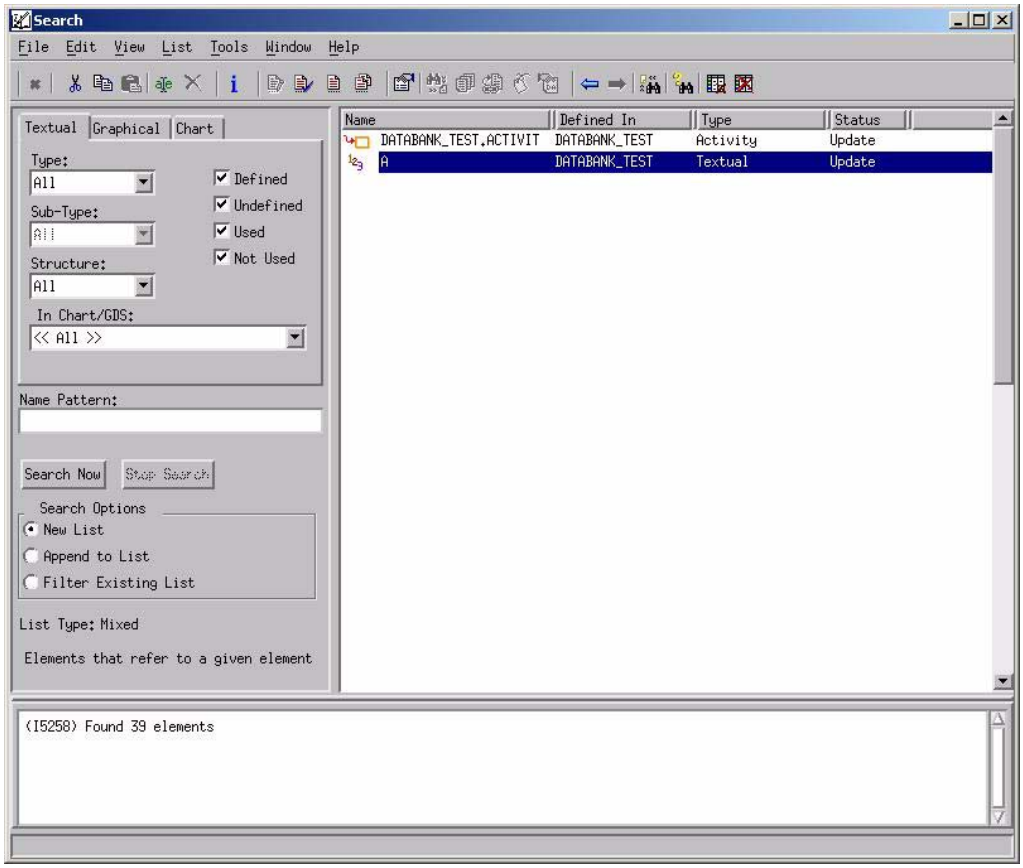
The **Search** tool enables you to track the location in charts where elements are used (called *referencing* an element) and track the various charts in which an element displays (called finding where an element is *used*).

The following sections explain in detail how to perform these operations.

Finding Where Elements are Referenced

To find where elements are referenced:

1. Create a list or open a stored list. For instructions on how to perform these tasks, see **Creating a List of Elements** and **Accessing a Stored List**.
2. In the right pane, highlight the list element or elements you want to track.
3. Either from the right-click menu or in the **Tools** menu, click **Where Referenced**. The **Search** tab then replaces the list with the referenced information.


Search
File Edit View List Tools Window Help
Textual Graphical Chart
Type:
All
Sub-Type:
All
Structure:
All
In Chart/GDS:
<< All >>
Defined
Undefined
Used
Not Used
Name Pattern:
Search Now
Stop Search
Search Options
New List
Append to List
Filter Existing List
List Type: Mixed
Elements that refer to a given element
Name
Defined In
Type
Status
DATABANK_TEST.ACTIVIT
DATABANK_TEST
Activity
Update
A
DATABANK_TEST
Textual
Update
(I5258) Found 39 elements

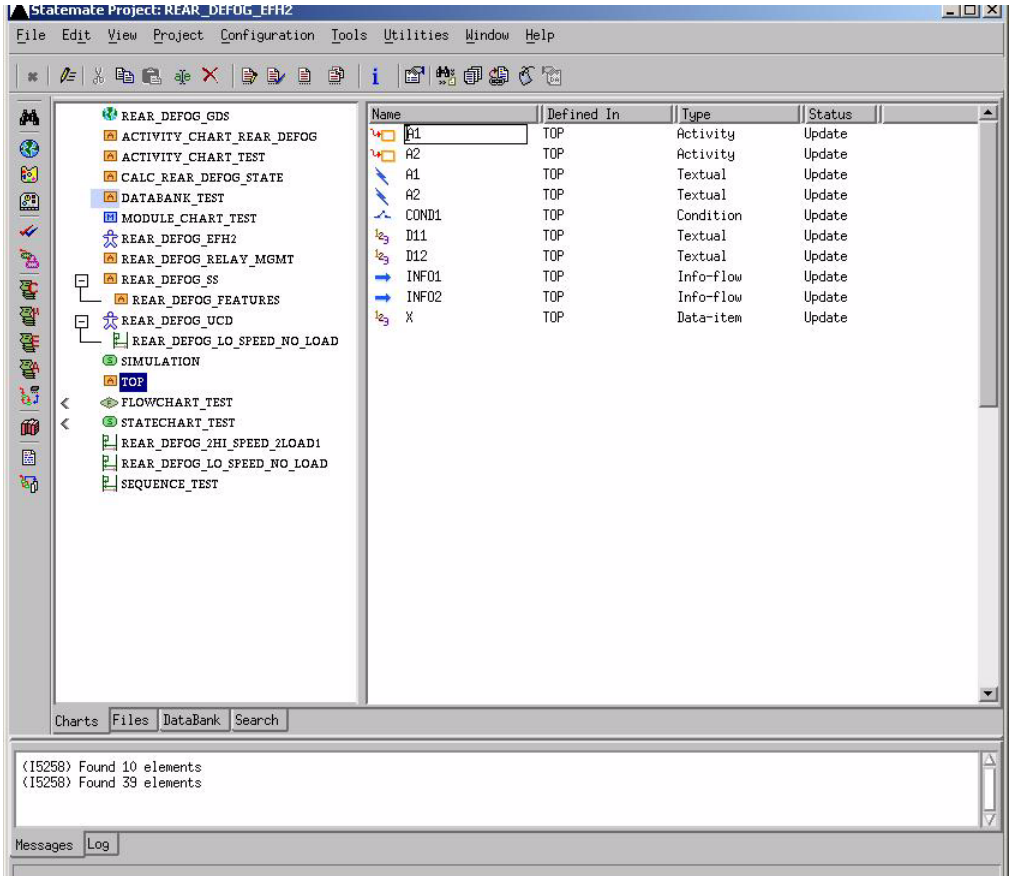
Finding Where Elements are Used

To find where elements are used:

1. Create a list or open a saved list. For instructions on how to perform these tasks, see **Creating a List of Elements** and **Accessing a Stored List**.

2. In the right pane, highlight the list element or elements you want to track.

3. Either from the right-click menu or in the **Tools** menu, select **Where Used**. The **Search** tab is replaced with the **Charts** tab and the charts where the elements are used are highlighted.



Libraries and Components

This section describes the component editor. The topics are as follows:

- Working with Components
- Working with Libraries

A component is an element contained in a library that can be shared across projects, much like a routine in a code library.

A component is defined in a generic activity chart, which is a special activity chart that enables you to define global activities that can be shared within a specific project.

The definition of a component includes three parts:

- **Top-level name and inputs/outputs -** The top-level design contains information on the component's name and inputs/outputs. If the top-level design is changed, all charts that contain instances of that component must be modified by you (as a change here will cause changes to the charts). Use the Check Model tool to detect instances that are not up to date.
- **Behavior -** The behavior of the component is reflected in all the versions of charts that use the component. If the behavior is changed, you do not have to update the graphic editors that contain instances of that component. Analysis tools automatically sense the new behavior, as well.
- **Icon** - The icon is a pixmap that represents the component in the Component Browser. A one-line text is attached to the pixmap to enable you to attach a description to the component. The system then automatically attaches this description of the top-level activity of the generic chart to be the description of the component.

  If the icon is changed, you do not have to update the graphic editors that contain instances of that icon's component. Analysis tools (such as Simulation) do not sense the modification, but the Component Browser is refreshed with the new icon.

Note

Global definition sets (GDSs) can be part of a component. When you create a component, the related GDSs are automatically added to the component's configuration. (A GDS is related if it contains user-defined types or constants that are part of a component definition.).

When you delete a chart from the workarea, GDSs that are no longer relevant (because the last instance of a specific component was deleted) are automatically deleted from the workarea, as well.

Note the following behavior relevant to components:

- The Check Model tool tests the legality of an instantiation of a component by comparing the flows in the activity charts against the component’s specification. Check Model does not “dive” into the components themselves, however.
- The Reports tool reports on the instances themselves, and like all kernel tools, sees only the specifications of components. The Reports tool does not “dive” into the components themselves.
- Dictionary and protocol reports list instances of components with their stubs and their type definition and the short and long descriptions of the components.
- The Simulator tool can simulate components. The Simulator loads the body of components into the database. This is transparent to all other Rational Statemate tools, except the graphic editor in the case that a highlight was explicitly required by you.

  When the Simulation preference **Have Access to Component Elements** is set to **Yes** (default is **No**), access to internal elements in the component is enabled (within Monitors, Do actions, Highlight GEs etc.)
- The Software Code Generators support components. The Code Generators load the body of components into the database. This is transparent to all other Rational Statemate tools.

Working with Components

The following sections discuss components in more detail. For more information on components and libraries, see **Example Components**.

Creating a Component

To create a component:

1. Create a generic chart.

   Generic charts enable reuse of parts of a specification. A generic chart makes it possible to represent common portions of the model as a single chart that can be instantiated in many places, much like a procedure in a conventional programming language.

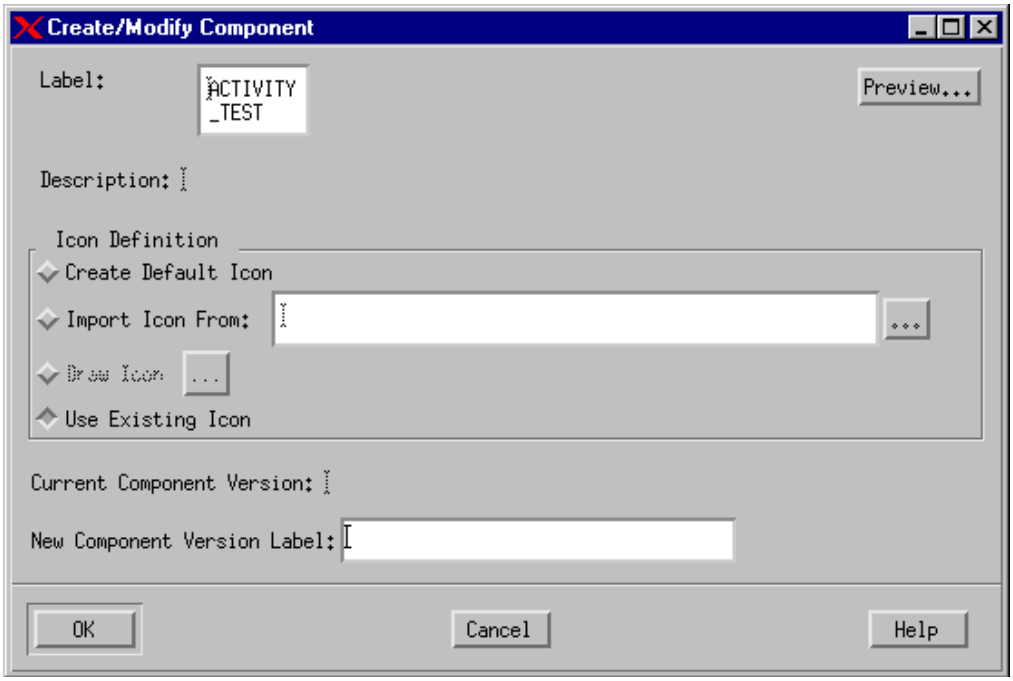
   Generic charts are linked to the rest of the model by parameters; no other elements (besides the definitions in global definition sets) are recognized by both generic charts and other portions of the model.

   For instructions on creating a generic chart, see **Creating a New Chart or Diagram**.

   Note the following:

   - A generic chart can only have one top-level activity.
   - Type definitions of all parameters must be complete. (That is, no unresolved data-items or user-defined types can exist in the chart.)

2. In the Charts, Files, or Databank tab of the Rational Statemate main window, select **Configuration > Create/Modify Component**. The **Create/Modify Component** window opens.



3. From this window:

a. In the **Label** text box, enter a name for the component. The limit is 64 alpha-numeric and underscore characters (_).

b. Multi-Labeling

The Library Component versioning mechanism allows the association of several labels with a single component version. Association of a new label with an existing version is done by editing the space delimited label list in the Version Label column of the **Configuration > Components Version** table.

Note: The same label cannot be associated with different versions. Associating an already associated label with a new version results in the removal of the previous association.

c. The **Description** text box displays the description of the generic chart from the chart's properties and cannot be edited here.

d. In the Icon Definition area create an icon that will appear in the Component Browser to represent the component. You can create the icon in the following ways:

- Select **Create Default Icon** to create a default-size pixmap that contains a rectangle and pins to represent stubs on its edge. The number of pins is similar to the number of stubs up to a limit.
- Select the ellipsis (**...**) beside **Import Icon From** to navigate to an existing icon name.
- Select **Draw Icon,** if the General preference "Icon Editor" has been previously set to an icon editor available for your use. The ellipsis (...) beside the **Draw Icon** button opens this editor on a predefined empty pixmap file.
- **Use Existing Icon** is available when you are modifying an existing component, but do not want to change its icon.

4. Select **Preview** to view the component exactly as it will appear later.
5. Click **OK**.

Inserting a Component

When you insert a component into a chart or diagram, you essentially create a reference to that component. The component cannot be edited when it is inserted. If the component should change you have the choice of updating to the newer version.

Copying a component into a chart is similar to inserting a component, except that there is no linkage between the instance and the component. For more information, see **Copying a Component**.

To insert a component:

1. Click on the Rational Statemate main window.
2. Select the component you want to insert.
3. Open the activity chart where you want to use the component.
4. Right-click and drag the component from the Component Browser to the chart or select **Edit > Insert Component**.

   Note: When a component is inserted into the graphic editor, it is inserted using the current zoom. That is, if the editor is in Zoom x2, the component is inserted in Zoom x2. If the editor is in Full View, the component is drawn in its original size.

5. Connect the I/O pins to flows in the chart.
6. Label the flow line to complete the data binding.

Note

- When inserting a component with global definition sets, the GDSs are loaded in read-only mode to your workarea. This ensures that all global definitions and declarations exist in the new model and that there are no conflicts between the model's definitions and the inserted component's definitions.

  These GDSs are part of the your model, but you cannot check them into the databank. The GDSs are automatically deleted when all references to components that need them are deleted. The imported GDSs are listed in the Chart tab of the Rational Statemate main window, but have a different icon in the tree section.

- When a component is inserted it is read-only. Stubs are also read-only. You cannot modify the stub position, stub name, or stub mode. Only the instance name can be modified. By default, the component name is <COMPONENT_NAME>. You can add an instance name before the <. This is done using the regular Edit Text operations, but when applying the change, the application checks that the <COMPONENT_NAME> name was not changed.

  By default, you cannot change the color or text font of a component. An activity chart Specific Preference **Allow editing of Instance Component** (default **No**) blocks editing operations on component instances. When the preference is set to **Yes**, you can change the size, shape, color and font of the component instances, and the color of stubs. However, you cannot move, delete or change the name of stubs.

  Arrows only enter and exit the head of the stubs.

Copying a Component

Copying a component into a chart is similar to inserting a component, except that there is no linkage between the instance and the component. For more information, see **Inserting a Component**.

Note

When you copy a component to an activity chart, the stubs of the components are removed. All charts of the components are then loaded into the workarea as “new” charts. Charts with names that conflict with existing charts are renamed, and a warning message is issued.

To copy a component:

1. Click on the Rational Statemate main window.
2. Select the component you want to use.
3. Open the activity chart where you want to use the component.
4. In the chart editor, select **Edit > Copy Component**. The component displays within the chart.
5. Position the component by clicking and dragging it to the desired location.

Note

When a component is copied into the graphic editor, it is copied using the current zoom. That is, if the editor is in Zoom x2, the component is copied in Zoom x2. If the editor is in Full View, the component is drawn in its original size.

Previewing a Component

You can control the layout of a component by changing the set of activity chart preferences. As you make changes, you can preview them in real time by selecting **File > Preview Component** from an activity chart editor or from the **Create/ Modify Component** operation in the toolbar of the Rational Statemate main window.

Note

- The component and stubs are displayed with the same graphic attributes (shape, size, color, font, etc.) that will appear when the component is inserted into a chart.
- One activity is displayed using a similar shape as the top-level activity in the generic activity chart.
- The box size is the minimum size required to map all stubs, but not smaller than a predefined (preference) size.
- Notes that reside inside the top-level box are mapped into the component.
- The stub placement is based on the intersection point between flow lines and the top-level activity. The stubs are positioned with equal distance between them. (The distance is a preference.) These (intersecting) arrows are labeled and the label is used for the synthesis. Arrows without labels are ignored.
- Stubs are defined as either information or regular stubs.
- Formal parameters that do not appear on external flows are mapped to stubs (inputs at the left edge, outputs at the right edge of the component).
- If the grid is on (in the generic chart), the synthesis is done according to the above, but also follows the grid rules. (If the distance between stubs contradicts the grid settings, the distance between stubs is enlarged to fit into the next grid point.

Deleting a Component

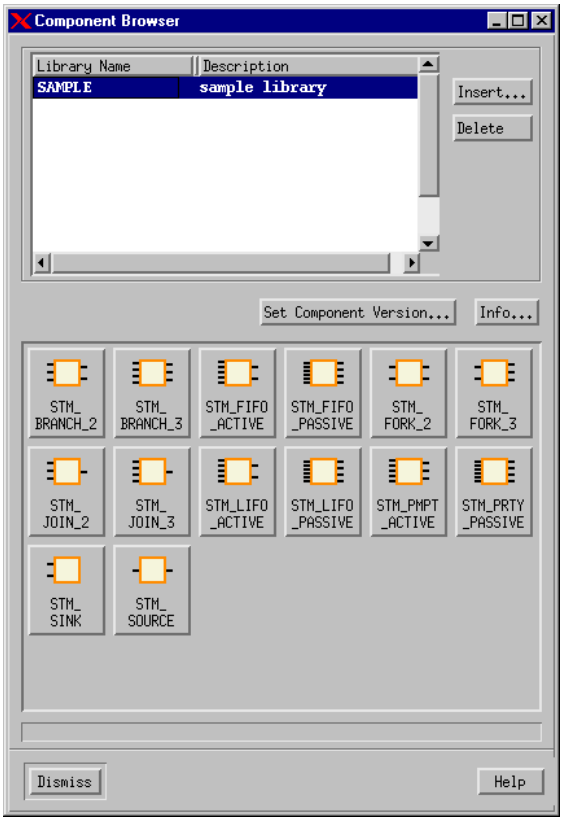
To delete a component:

1. From the Charts, Files, or Databank tab of the Rational Statemate main window, select **Configuration > Remove Component.**
2. Select the component to be deleted.
3. Click **OK** to confirm your choice.

Managing Components

The Component Browser lists any currently available libraries for your project.

To manage components from the Component Browser, in the Charts, Files, or Databank tab of the Rational Statemate main window, select **Configuration > Create/Modify Component** or click . The **Component Browser** window opens.



The Component Browser window lists any currently available libraries for the project. From the Component Browser window you can:

- View the icons representing the components in a library by selecting the library name.
- Read a description of a component and its I/O's by selecting the component's icon and clicking **Info**. This opens the Info window, which contains version information about the component.
- Add additional libraries to your workarea (and not the project as a whole) by clicking **Insert**.
- Delete a library name from your workarea, by clicking **Delete**.

Note

Deleting a library name here does not delete the library itself, or the library from the project (if it had previously been added at the project level).

Working with Libraries

A library is a container for model components. Any Rational Statemate project, new or existing, can be designated as a library.

Together, libraries and components offer a means to speed the process of design specification and to help you create more consistent specifications. Reusing previously created Rational Statemate model elements can save time and allows for more consistent designs.

To define a project as a library, select **Defined As Library** on the Create New Project window or the Modify Project window.

You can add other predefined libraries to a library project during the setup. Also, you can set a general preference to automatically add a predefined list of libraries to any new project (whether or not it is a library).

If you are the project manager, you can establish and modify a list of libraries to automatically become available to any of the project members. (In addition, project members can add libraries to their own workareas. These libraries are only available for the duration of the current Rational Statemate session, however, and only to the specific workarea.)

Adding Libraries to a Project

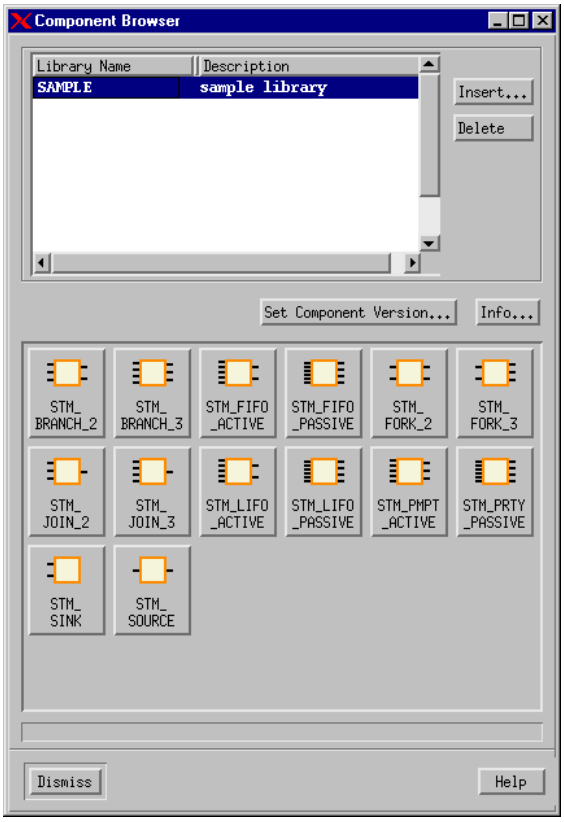
Project managers can make a list of libraries automatically available to any of the project's members. In addition, project members can add libraries to their own workareas.

To add libraries to a project:

1. From any tab on the Rational Statemate main window, select **Project > Project Management.**
2. Click **New** to create a new project or click **Modify** to modify an existing one.
3. Check **Expand** to view the Libraries information.
4. Click **Insert** next to the **Libraries** matrix to add a library. The **Select Libraries** dialog box displays with all the predefined libraries listed in alphabetic order.
5. From the **Select Libraries** dialog box, select the libraries to add to the project.
6. Click **OK**.

Note

Each component in the library is represented by an icon that can be dragged and dropped into an activity chart. The icons also show the number of inputs and outputs for each component. In this **Component Browser** dialog box, the STM_BRANCH_2 icon shows three inputs and two outputs.



For more information on components and libraries, see **Example Components**.

The Router Element

This section describes the Router element. The topics are as follows:

- Router Element for the Activity Charts
- Working with the Router
- Defining Router Properties
- Interface Reporting
- Using Check Model with Router Blocks
- Exporting Router Blocks to Rational DOORS
- Setting Router Preferences

Router Element for the Activity Charts

Activity charts in Rational Statemate offer a drawing element called the *Router*. The router makes creating and maintaining activity charts easier by reducing the need to draw many flow lines between activities.

The router element can be used when you have a large number of flow lines between different activities in your activity chart. It effectively “routes” the signals from a source to a target within the diagram or to another part of the system. Rational Statemate check-model checking is performed on the router blocks to ensure that “what goes in also comes out.” Interface reporting can be performed on activities as well as the router blocks to analyze where signals go within the diagram.

Working with the Router

The router is accessed from an activity chart. For more information on activity charts, see **Activity Charts**. The activity chart editor includes two drawing icons for the router element.

Statemate supports the following two types of routers:

- **Internal router** - The internal router (or simply “router”) represents router blocks used within the scope of the top-most activity in a particular activity chart.

  Internal routers are shown in blue, with a solid line at the top and bottom of the box, and hashed lines for each side.

- **External router** - The external router represents router blocks used outside the scope of the top-most activity in a particular activity chart. Because activity charts are hierarchical, an external router is always resolved to an internal router in a chart higher in the chart hierarchy.

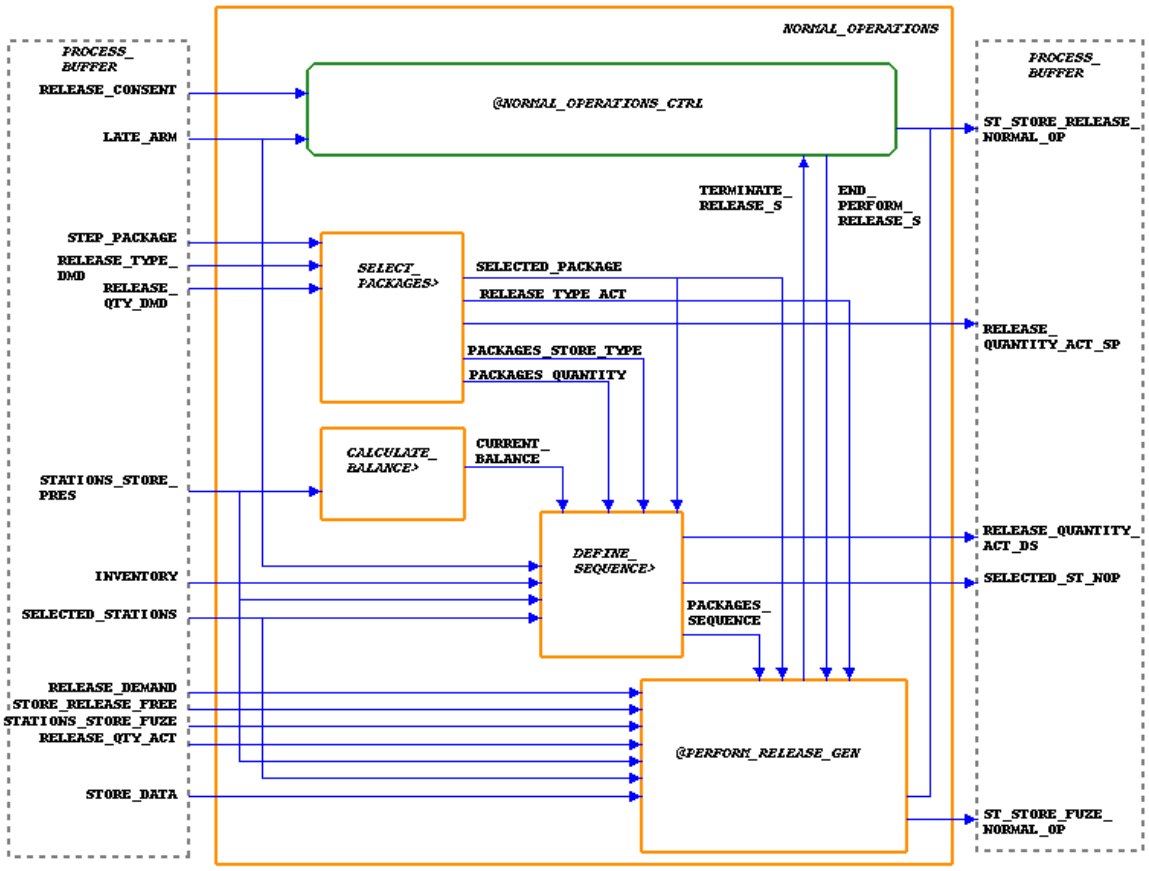
  External routers are shown in blue, with hashed lines at the top, bottom, and each side.

Drawing Router Blocks

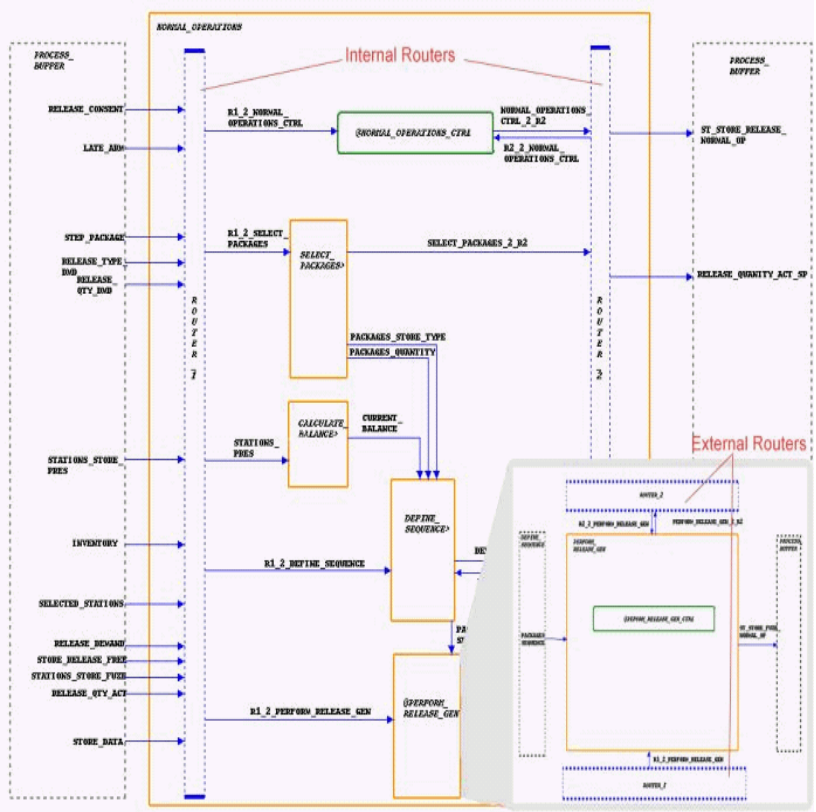
The internal and external router blocks are drawn like the other graphical blocks (activities, data stores, and so on) in the activity chart editor. A default name for the router block is automatically displayed after you draw the block. The name can be changed. For more information, see **Setting Router Preferences**.

Flow Lines can be drawn from a router to an external router, activity, external activity, data store, control activity, or another router.

This is an activity chart that does not use the router. As a result, the chart has an abundance of flow lines going to, from, and even crossing one another to reach their various activity destinations.



This chart uses two router blocks to effectively route the signals going between the external activities and the internal activity blocks.



This chart illustrates the differences between an internal and external router. In the main diagram, *router_1* and *router_2* are both internal. However, in the *perform_release_gen* activity, *router_1* and *router_2* are external routers.

Note

To provide a detailed interface description of those signals, and where they are flowing to and flowing from, use interface reports. For more information, see **Interface Reporting**.

Using Routers to Reduce Flow Lines

Routers can be used to reduce the number of flow lines between different activities in an activity chart. They effectively “route” the signals from a source to a target within the chart or to another part of the model.

Router Rules

The following are the rules for both Internal and External routers:

- Within the chart of the basic/control activity, where a data-element is used/affected, that element must explicitly flow to/from the activity or one of its ancestors.
- In a specific scope, there could be only one Internal Router <XXX>, but there may exist multiple occurrences of External Routers <XXX>, which all resolve to the Internal one.
- An External occurrence of a Router may also exist within the same chart as the Internal Router to which it resolves.
- Since multiple occurrences of a single router may exist in one chart (maximum of one internal, and multiple external occurrences), the flow lines to/from all occurrences in the chart are considered flowing to/from the router in that chart (similar to the current behavior of external activities).
- The source and target activities of each element flowing through an **Internal** Router (or through an external occurrence in the definition chart of the internal router) must be identified within the same chart.This rule also applies to signals that are input to a router.
- All data-flows to/from an External Router (in charts lower in the hierarchy than the router’s definition chart) must have a corresponding data-flow to/from that router in the closest ancestor chart where an occurrence of the router exists (which is not necessarily the parent chart).

  Note: This restriction means that data elements cannot be transferred through a router that is defined higher in the hierarchy than the data element’s definition because then there could not be a “corresponding data flow to/from the router’s occurrence in the closest ancestor chart where such an occurrence exists.” Eventually, the “closest ancestor chart where such an occurrence exists” would be the chart where the internal router is defined.

Compound Flow Lines through Routers

When building compound flow lines in a model with Routers, the tool will identify source, target and flowing elements of data-flows which pass through Routers, and will build the appropriate compound flow lines.

Local flow lines entering an internal router or its external occurrence within the same chart is intersected with the local flow lines exiting the router, and compound local flow lines are created accordingly.

Local Compound Flow Lines through External Routers

Local flow lines entering and exiting an external router will be combined together to a local compound flow line, only when the external router occurrence is within the definition chart of the internal router it resolves to.

Global Compound Flow Lines through Routers

Global compound A-flow lines, which pass through routers, are created only when the router is resolved (i.e., all external occurrences of the router resolve to an existing internal router).

For each element flowing to/from an external router, the tool identifies its real source/target by climbing the hierarchy up to the internal router. On that level, according to the router usage rules, the source/target must be identified. If the identified source/target is an off-page instance box, the tool dives into it to find the actual source/target of the flowing element.

Note

The tool does not create compound flow-lines through routers when both the source and the target of the compound flow-line is a data-store.

When the general preference “Allow Flow-line Loopbacks through Routers” is set to 'No', Statemate does not create compound flow-line through routers when both source and target of the compound flow-line is the same basic activity.

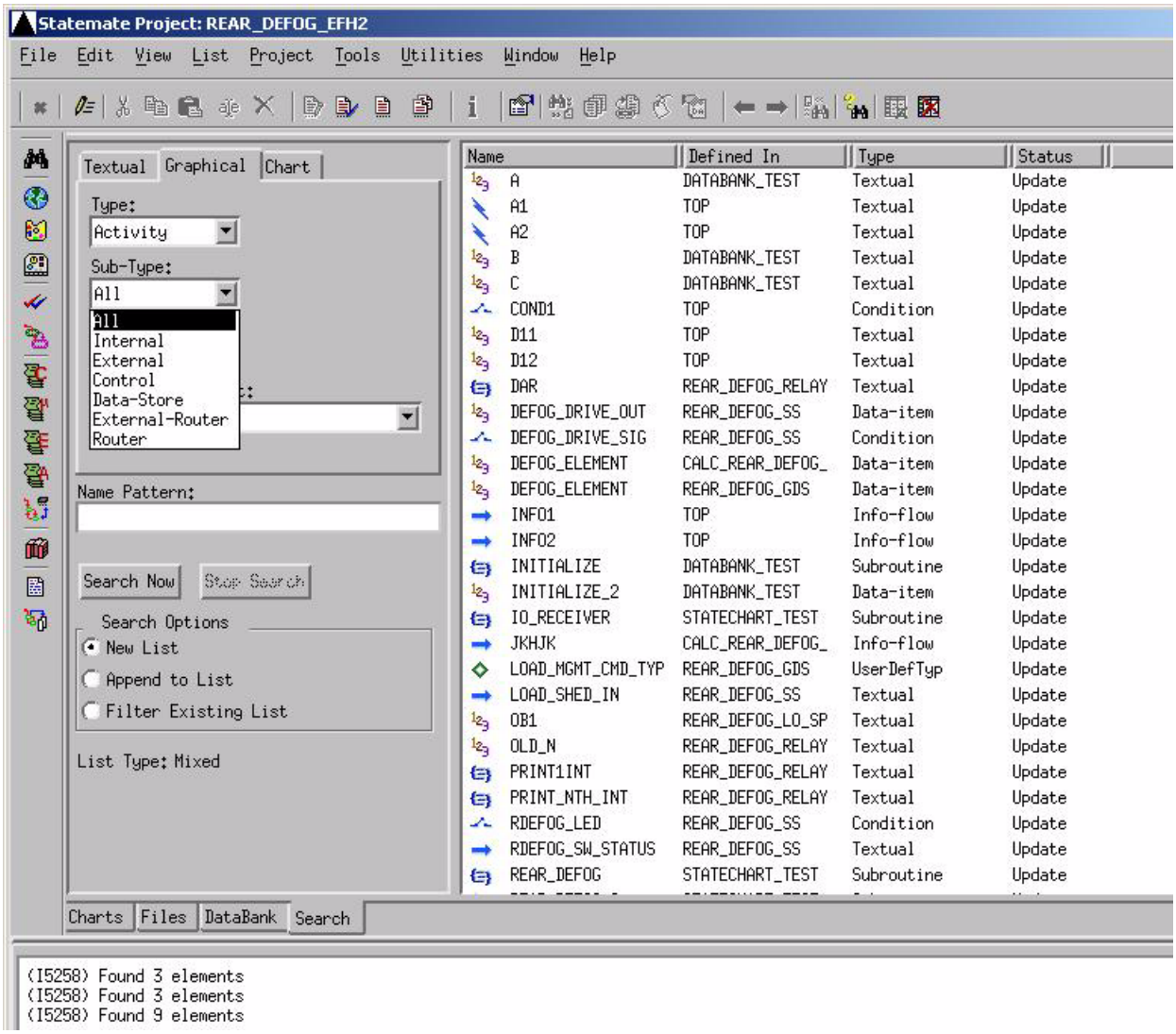
Defining Router Properties

Like other graphical elements in Statemate, internal routers have properties associated with them. In the **Router** properties dialog box:

- Textual description, long description, or attributes are entered.

Note

Router blocks, throughout your Statemate model, can be found using the Search tool. In the **Sub-type** list, click **External-Router** or **Router** (internal). For more information on the Search tool, see **Searching for Elements**.

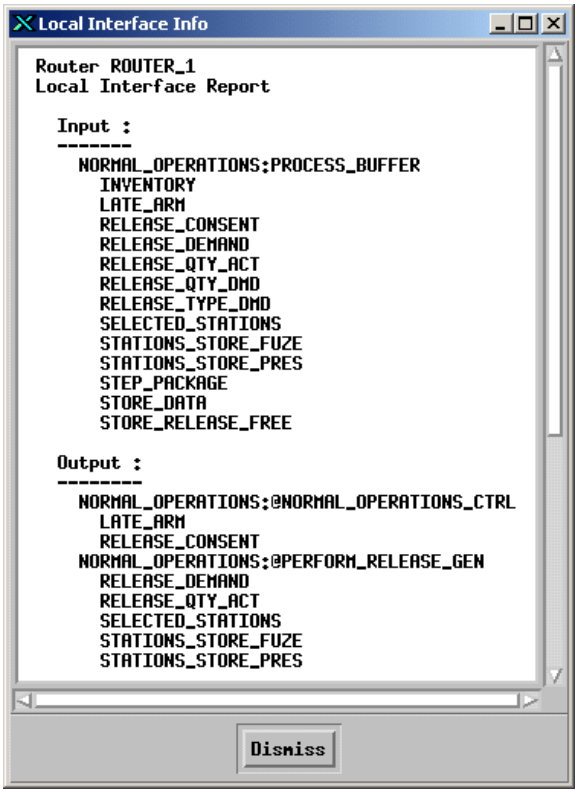

Statemate Project: REAR_DEFOG_EFH2
File Edit View List Project Tools Utilities Window Help
Textual Graphical Chart
Type:
Activity
Sub-Type:
All
All
Internal
External
Control
Data-Store
External-Router
Router
Name Pattern:
Search Now
Stop Search
Search Options
New List
Append to List
Filter Existing List
List Type: Mixed
Name
Defined In
Type
Status
A DATABANK_TEST Textual Update
A1 TOP Textual Update
A2 TOP Textual Update
B DATABANK_TEST Textual Update
C DATABANK_TEST Textual Update
COND1 TOP Condition Update
D11 TOP Textual Update
D12 TOP Textual Update
DAR REAR_DEFOG_RELAY Textual Update
DEFOG_DRIVE_OUT REAR_DEFOG_SS Data-item Update
DEFOG_DRIVE_SIG REAR_DEFOG_SS Condition Update
DEFOG_ELEMENT CALC_REAR_DEFOG_ Data-item Update
DEFOG_ELEMENT REAR_DEFOG_GDS Data-item Update
INFO1 TOP Info-flow Update
INFO2 TOP Info-flow Update
INITIALIZE DATABANK_TEST Subroutine Update
INITIALIZE_2 DATABANK_TEST Data-item Update
IO_RECEIVER STATECHART_TEST Subroutine Update
JKHJK CALC_REAR_DEFOG_ Info-flow Update
LOAD_MGMT_CMD_TYP REAR_DEFOG_GDS UserDefTyp Update
LOAD_SHED_IN REAR_DEFOG_SS Textual Update
OB1 REAR_DEFOG_LO_SP Textual Update
OLD_N REAR_DEFOG_RELAY Textual Update
PRINT1INT REAR_DEFOG_RELAY Textual Update
PRINT_NTH_INT REAR_DEFOG_RELAY Textual Update
RDEFOG_LED REAR_DEFOG_SS Condition Update
RDEFOG_SW_STATUS REAR_DEFOG_SS Textual Update
REAR_DEFOG STATECHART_TEST Subroutine Update
Charts Files DataBank Search
(I5258) Found 3 elements
(I5258) Found 3 elements
(I5258) Found 9 elements

Interface Reporting

You can obtain interface information for routers and activities within activity charts. The interface report shows input and output flow information for each block, and their associated sources and sinks.

Local Interface Report

The local interface report lists the “immediate” activities that the various signals flow to or flow from, as shown in the following figure.



Global Interface Report

The global interface report lists the signals flowing to or from the selected activity and their source/target activities ; these are considered the producers and consumers of the signals.

It also supports the analysis of activity interfaces based on functionality (usage of data). This interface analysis behavior is controlled by an Activity Interface Browser and Reports preference. When this “Functional’ interface is selected, the Global Interface Reports are based on functionality.

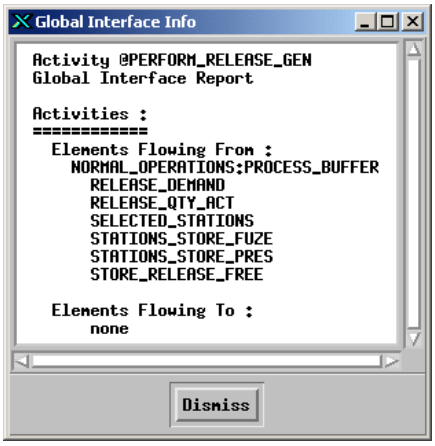
By default the tool displays the basic activity, which is the actual consumer/producer of the input/output signal as source/target. By setting the preference *Show source/target in LCA chart* to Yes, the user can choose to display the ancestor instance of the basic source/target activity in the Least Common Ancestor chart (of the source/target and the selected activity).

Two global interface reports are available.

The first report, called *Global Interface Report (Activities)*, shows the data sorted by the source and sink activities names. An *Activity Interface Browser* supplies a graphical view of the Global Interface Report (Activities) to allow browsing and navigation through the charts hierarchy. To launch this browser, select **Tools > Activity Interface Browser** option. Then, you may select any of these options:

- **Properties**
- **Show in Model** for selected charts/elements
- **Show Interface** for a selected activity

The second report, called *Global Interface Report (Elements)*, shows the data sorted by signal name, as shown in the following figure.

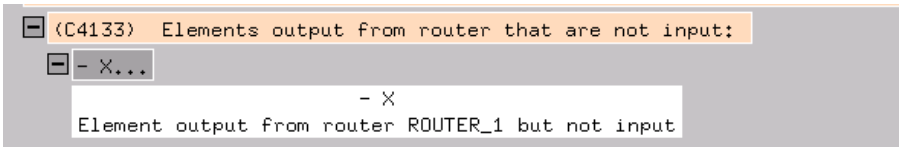


Note

An Activity Interface Browser Reports preference, named *View Interface Info Using External Pager*, controls the dialog box in which the information is shown. When set to No, the tool shows the reports with the internal Info Viewer, as shown in the figures. When set to Yes, the tool shows the reports using the pager defined in the general preferences (**Project > General Preferences**).

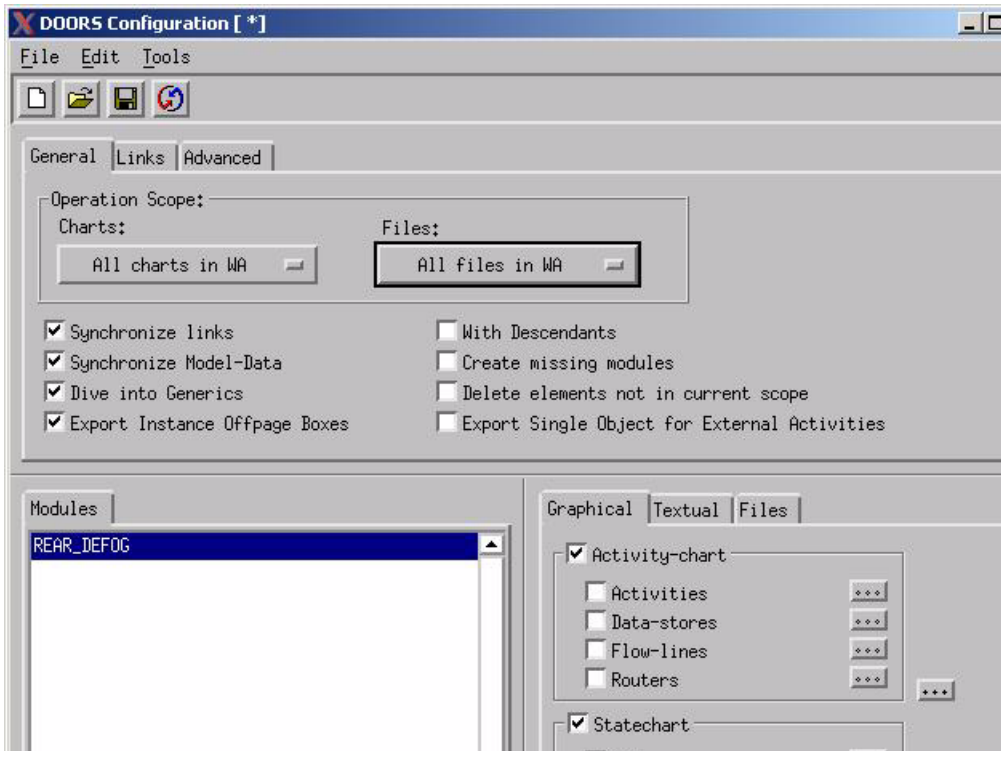
Using Check Model with Router Blocks

Check model performs checks on router blocks to ensure that information flowing into the block also flows out of it. Where external routers have been used, the check model ensures that an original definition of the router exists, based on the chart hierarchy scoping rules. The following figure shows a sample report.



Exporting Router Blocks to Rational DOORS

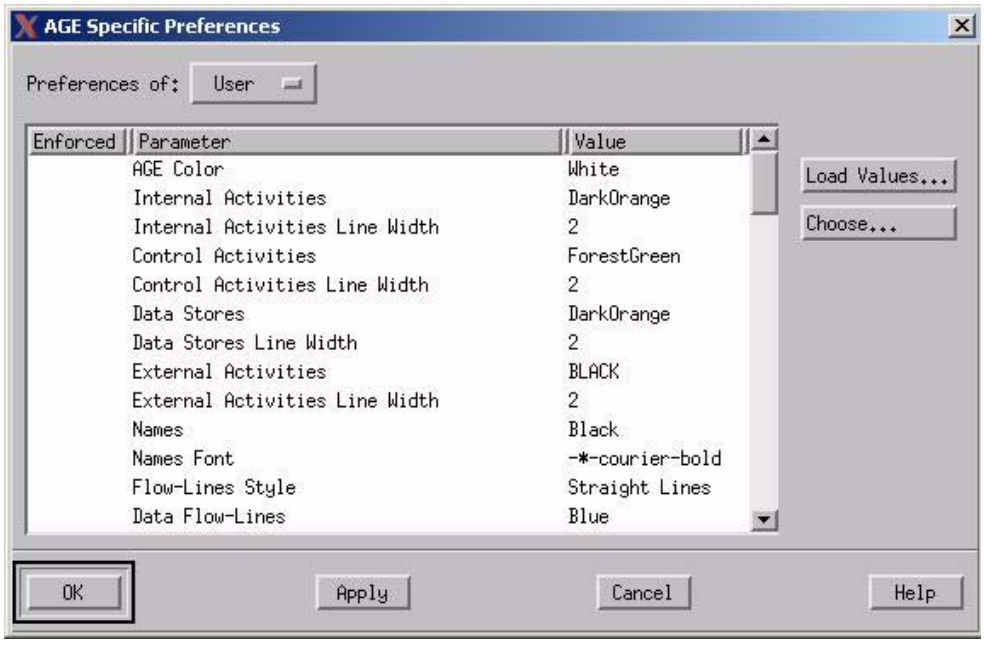
The Rational DOORS interface supports router blocks. To export router blocks to Rational DOORS as elements, select **Routers** in the graphical tab for formal modules, as shown in the following figure.



Setting Router Preferences

There are a number of router preferences available from within the activity chart preference settings. These router preferences follow the same **User** > **Project** > **System** hierarchy rules for settings.

The following figure shows the preference settings for routers.



Internal Router Preferences

The available preferences for internal routers are as follows:

- **Routers** - Specifies the color used for router blocks. This can be any color from the pallet.
- **Routers Horizontal Line Width** - Specifies the horizontal line width for router blocks. This is a value between 0 and 6.
- **Routers Vertical Line Width** - Specifies the vertical line width for router blocks. This is a value between 0 and 6.
- **Routers Horizontal Line Style** - Specifies the horizontal line style. The possible values are Dashed and Solid.
- **Routers Vertical Line Style** - Specifies the vertical line style. The possible values are Dashed and Solid.
- **Routers Fill** - Specifies the color used for a “filled” router block. This can be any color from the pallet.
- **Routers Fill Style** - If this is set to Fill, the router blocks are filled with the color specified in the Routers Fill preference. If this is set to No Fill, the fill color is not used.
- **Auto Router Name Prefix** - Specifies the prefix used when naming router blocks.
- **Routers Name Orientation** - Specifies the orientation used when drawing the router name. The possible values are Horizontal and Vertical.

External Router Preferences

The preferences for an external router are as follows:

- **External Routers** - Specifies the color used for external router blocks. This can be any color from the pallet.
- **External Routers Horizontal Line Width** - Specifies the horizontal line width for external router blocks. This is a value between 0 and 6.
- **External Routers Vertical Line Width** - specifies the vertical line width for external router blocks. This is a value between 0 and 6.
- **External Routers Horizontal Line Style** - specifies the horizontal line style. The possible values are Dashed and Solid.
- **External Routers Vertical Line Style** - specifies the vertical line style. The possible values are Dashed and Solid.
- **External Routers Fill** - specifies the color used for a “filled” external router block. This can be any color from the pallet.
- **External Routers Fill Style** - If this is set to Fill, the external router blocks are filled with the color specified in the External Routers Fill preference. If this is set to No Fill, the fill color is not used.
- **Auto External Router Name Prefix** - specifies the prefix used when naming external router blocks.
- **External Routers Name Orientation** - specifies the orientation used when drawing the name of the external router. The possible values are Horizontal and Vertical.

Global Definition Set Editor

This section describes the Global Definition Set Editor (GDSE). The topics are as follows:

- Creating a New GDS
- Editing an Existing GDS
- GDS Properties

The global definition set editor (GDSE) enables you to create a new global definition set (GDS) or modify the definition of an existing GDS.

A *GDS* is a type of component that contains definitions of user-defined types, as well as constant data-items and conditions. The elements that appear in a GDS are visible in the entire model. Data-types defined in a chart or inherited from a parent chart take precedence over data-types defined in a GDS.

A GDS is similar to a chart in that both are configuration items of the model. That is, both charts and GDSs contain parts of the model and can be saved and loaded separately from other parts. A GDS cannot contain any other graphical or non-graphical information.

There can be several GDSs in one model, but there is no hierarchical relationship between them, or between them and the charts in the model.

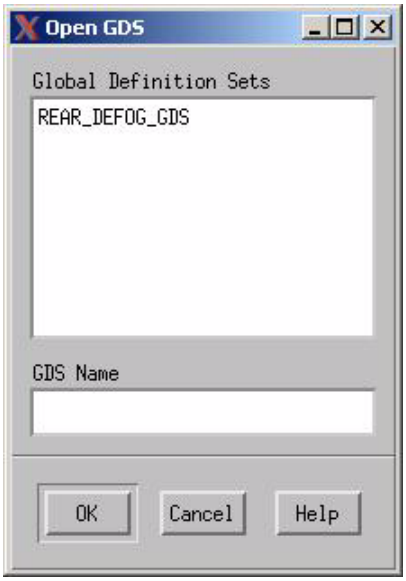
Creating a New GDS

When you create a new GDS, note the following:

- Any user-defined type or constant defined in a GDS is visible throughout the entire workarea.
- You can have multiple GDSs in a workarea.
- A GDS can reference information in another GDS.
- Data types defined in a chart or inherited from a parent chart take precedence over data types defined in a GDS.
- GDS cannot reference information in other GDS in a circular manner. For example, GDS A cannot reference GDS B, if B references information in A.

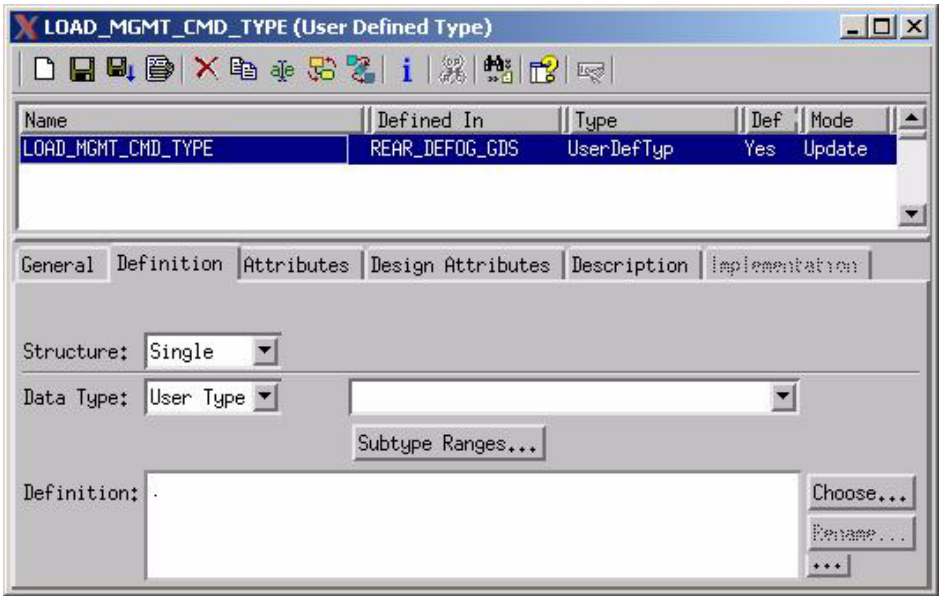
To create a new GDS:

1. In the Statemate main window, click the **GDS Editor** icon . The **Open GDS** dialog box displays.

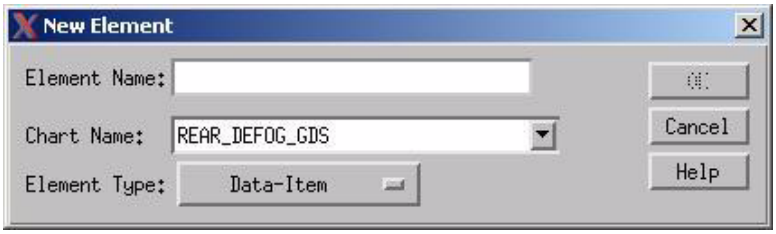


2. Enter a name for the new GDS in the **GDS Name** text box.

3. Click **OK** to confirm your choice. The **GDS Properties** dialog box displays.

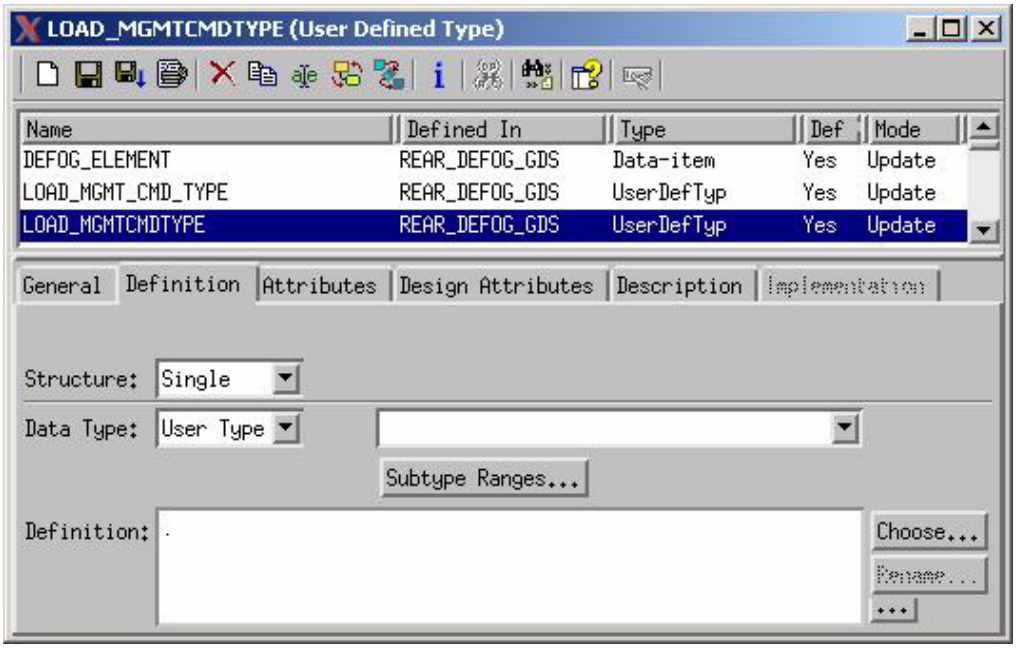


4. In the **GDS Properties** dialog box, click . The **New Element** dialog box displays.



5. In the **Element Name** text box, enter a name for the new element.
6. In the **Chart Name** pull-down menu, select the GDS name entered in Step 2.
7. Select the appropriate type from the **Element Type** pull-down menu.

8. Click **OK**. The **GDS Properties** dialog box refreshes so that you can define the new GDS element.

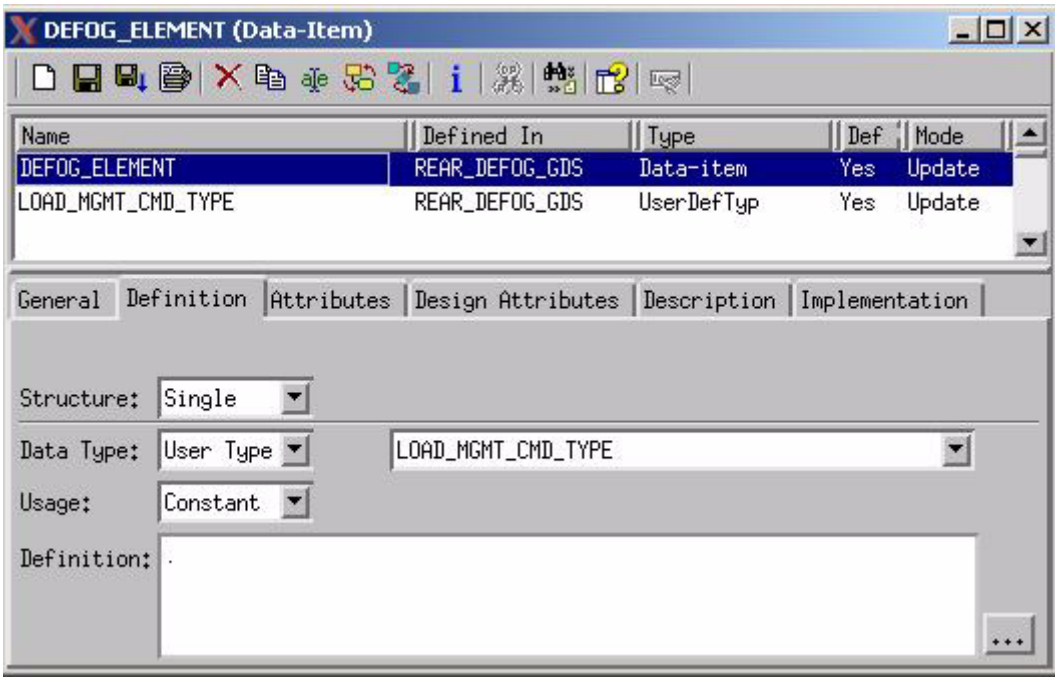


For more information on using the Properties dialog box to define elements, see **Element Properties** and **Libraries and Components**.

Editing an Existing GDS

To edit an existing GDS:

1. In the Statemate main window, click . The **Open GDS** dialog box displays.
2. Click on a **GDS** in the **Global Definition Sets** list.
3. Click **OK** to confirm your choice. The **GDS Properties** dialog box displays.



4. Make the necessary edits. For more information, see **Element Properties** and **Libraries and Components**.

GDS Properties

GDS has the following properties:

- GDS Usage Property
- GDS Visibility Mode Property

GDS Usage Property

The GDS Usage property can be defined for statecharts, activity charts, GDSs, and flowcharts. It cannot be defined for module-charts and sequence diagrams.

This property can be accessed from the chart Properties toolbar .

The GDS Usage form includes two editable lists and one read-only list:

- The **Extend GDS Usage** list includes the GDSs in the workarea that are currently not visible.
- The **Used GDS**(s) list includes the GDSs that are currently visible. It includes a default value, <All-Public-GDS>, which means that all the Public GDSs are visible, but not the Explicit Usage ones.
- The **Inherited Used GDS** list (read-only) is calculated according to the chart hierarchy and the GDS Usage property of the chart's ancestors.

To edit the GDS Usage of a chart (or a GDS), select the chart (or GDS) in the workarea browser and then select **Edit > Properties**.

GDS Visibility Mode Property

The GDS Visibility Mode property is accessed from the GDS Properties.

When set to Public, the information in the GDS is visible to all charts in the workarea that have their GDS Usage property set to <All-Public-GDS> (the default). It is also visible to charts that have the GDS included in the Used GDS(s) list.

When set to Explicit Usage, the information in the GDS is visible only to those charts that have the GDS explicitly included in the Used GDS(s) list.

Reduced GDS

In order to have a workarea that has only necessary data, the user can delete unused GDS elements in the workarea through the Check Model tool, see the *Check Model User's Guide*, or by searching and deleting unused elements. However, the deleted GDS elements may be required later. Statemate also supports a "Filtered Check Out" operation which loads only the necessary data only into the workarea.

A read-only GDS in the workarea is marked as "reduced" when deleting elements from it ("r" for reduced in the files list, replacing the "m" for modified). A reduced GDS can neither be saved to the databank nor locked. To improve workflow, the reduced GDSs can be removed automatically from the workarea using the **Configuration** > **Reduce Workarea** option in the Chart view. For more information, see **Reducing the Workarea**.

Selecting "Filtered Check Out from Databank" while in the "Charts" tab reloads deleted elements from all reduced GDSs, which may solve the resolution of currently unresolved elements in the workarea.

Once marked as reduced, the GDS keeps this status, even after adding elements or modifying existing elements. Only the "Check out from Databank" (either in update or read only mode), "Import" or "Delete from Workarea" operation changes this status.

Note

- This operation may add unused elements to the workarea for elements that their resolution is not conclusive in the filtered check-out calculation.
- In the case of conflicting type definitions, the resolution results of a filtered workarea may be different from the resolution results of a complete workarea. For example, consider the following case:

```
Enum Type OneType {one, two, three}

Enum Type SecondType {one, two, three}

Expression: X := one;
```

  When both `OneType` and `SecondType` definitions exist in the workarea, and `X` is undefined, the right-hand side "`one`" is unresolved. However, if only one of the two type definitions is loaded to the workarea, the tool resolves the right-hand side "`one`" to the loaded type.

MicroC Code Generator

This section describes the MicroC Code Generator in the following topics:

- **Scope Definition** shows the Module structure of the profile.
- **Code Options** describes the available options for code generation.
- **Rational Statemate Block in a Rational Rhapsody Model** explains how a Rational Statemate model can be integrated into a Rational Rhapsody model.
- **Code Optimizations** explains how to optimize generated code.
- **OS Definition Tool** describes customizing the code generator.
- **Supported Targets** lists the operating systems and machines on which the generated code runs.
- **Utilities** describes the code generator utilities.

Scope Definition

The Scope Definition, which is in the Profile Editor main window, shows the Module structure of the profile in a tree format or as a list. Both views show the charts assigned to each module and how they were assigned.

Module Structure

A **Module** is a collection of statecharts and activity charts that comprise a component.

A **Module** signifies a single source file with its functions.

Testbenches

Testbenches are separate Statecharts created outside the specification of the system being developed.

Testbenches trap a specific behavior to test a design's inputs and outputs. It's a "snapshot of a scenario." Testbenches also serve as debuggers and they're visible to all signals in the design without having to draw discrete flows.

Note

Testbenches cannot test generics.

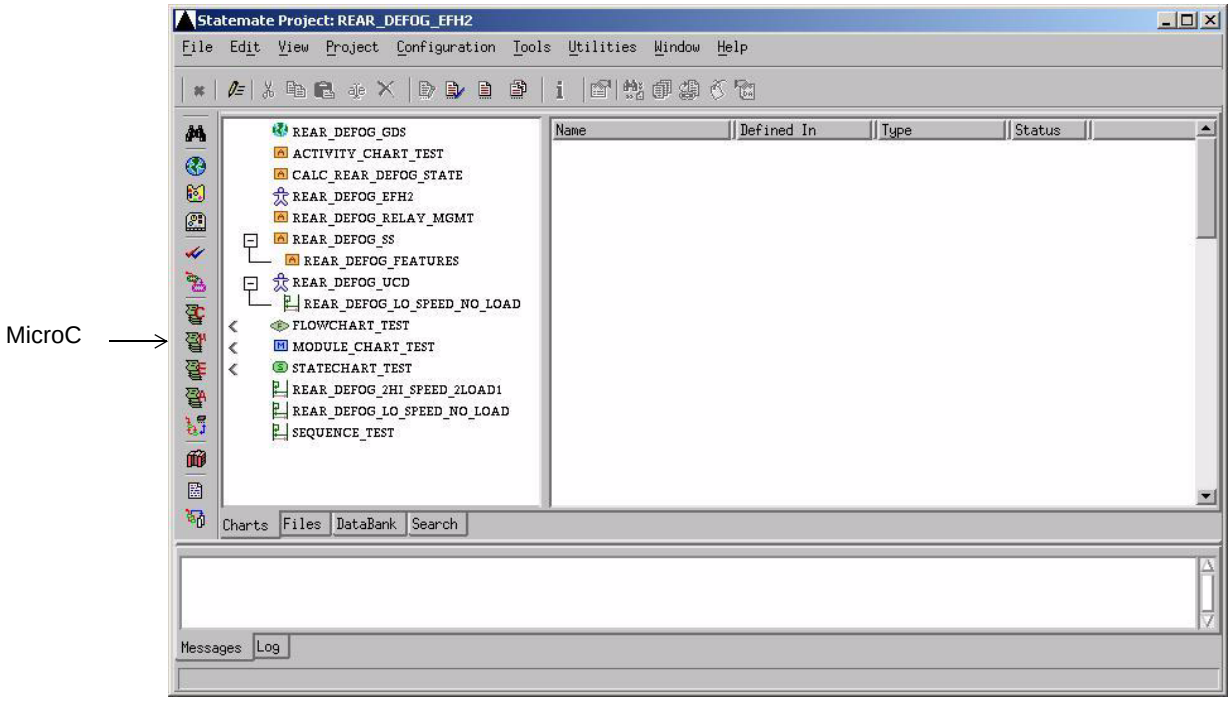
Creating a Sample Profile

This section shows how to create a sample profile and generate code for it.

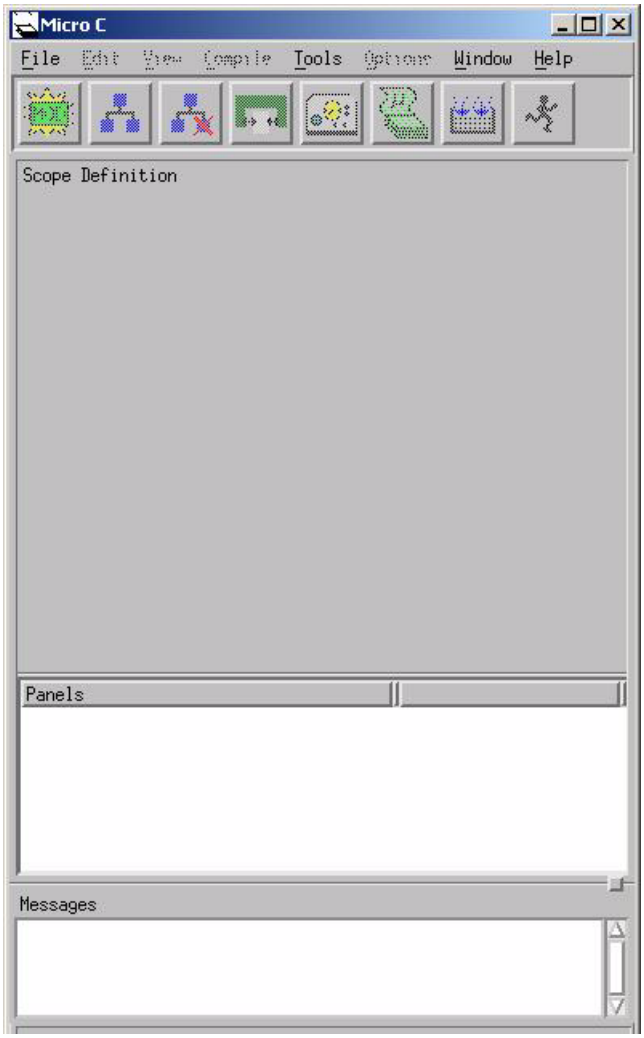
Invoking the Profile Editor

Use the following steps to access the Profile Editor and create a new profile.

1. Select the **MicroC** code generator from the main window.

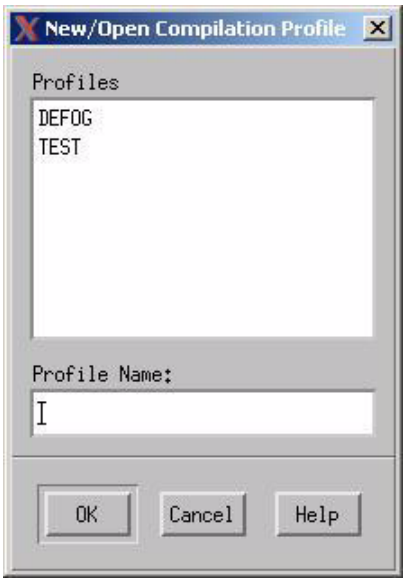


The appropriate Profile Editor displays.



2. Select **File > New Profile**.

   The **New Compilation Profile** dialog displays.



3. Enter the name of the new profile in the **Profile Name** box and click **OK.**

The Profile Editor enables all the menu selections and displays the profile name in the title bar.

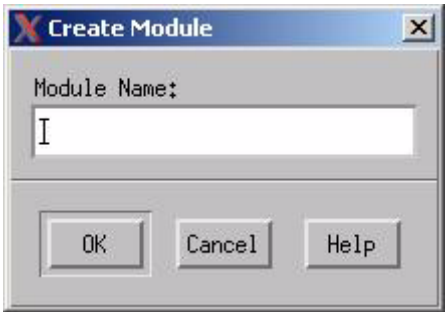
Defining Code Modules

Code module can be structured as desired to meet the needs of the model being simulated. Use the following steps to define the structure of modules that reflects the way you want the code organized. Note that each module may contain one chart, several charts, or a portion of a chart.

1. Click

or select **Edit > Create Module**.

The **Create Module** dialog box opens.



2. Enter the name of the new module and click on **OK**.

Assigning Behavior to the Module

Use the following steps to select charts from the workarea and assign them to the module you want.

1. Click or select **Edit > Add With Descendants**. The Chart Tree opens

2. Select the chart(s) you want to assign to the module by clicking on it. For example, select the Activity Chart REAR_DEFOG.

   Note: To select charts with their descendants in a hierarchy, select only the parent. The Code Generator adds your selection to the profile with its descendants.

3. Select **File > Save** item to save the profile in your workarea.

Splitting Activity Chart Hierarchy

You can create a module from hierarchy sub-trees. For example, referring to the following chart hierarchy:

AA

BB

CC

It is possible to include CC and BB in separate modules A1 and A2 as follows:

Module A1:

Scope AA, CC

Module A2:

Scope BB.

Code Options

This section describes options for the Code-Generator.

All the options below are available from **Statemate MicroC Code Generator > Options > Settings...**

RESET_Data as Function

General Tab > Use Macro for: Dynamic Reset Data

The **General Tab > Use Macro for: Dynamic Reset Data** allows RESET_DATA and RESET_ALL_ELEMENTS to be generated as functions instead of macros. By default, Dynamic Reset Data is checked and macros are generated.

Ignore External Binding

General Tab > Ignore External Binding

The **General Tab > Ignore External Binding** option allows generating code while ignoring external bindings.

It might be used when generating code to run on-host.

Code Generation for Control Activities

General Tab > Use Macros for > Control Activity Implementation

You can generate the code for Control Activities either as a macro or as a function.

To have the code for Control Activities generated as a macro, select the General tab on the Code-Generator Property Sheet. Then select **Use Macros for > Control Activity Implementation**. If not selected, Control Activities are generated as functions.

Enhanced Generated Code-Level Readability and Documentation

General Tab > Use Macro for > Bit Data Items and Conditions

You can generate code for Conditions and Data-items (of type Bits) using the element model-name. Use the **General Tab > Use Macro for > Bit Data Items and Conditions** flag to control the generated mode.

Support Selective GBA

In the compilation profile, distinct flags are defined to control highlighting of Activities, and States/Action Boxes

Byte Orientation Instrumentations

Target Properties Tab > Use Instrumentation

You can control the generation of byte orientation instrumentations (`#ifdef LSBYTE_FIRST` directives) in the code.

Use the Check-Box in: **Target Properties Tab > Use Instrumentation**.

Single-Bit Elements

General Tab > Use Single Bit for

You can control implementation of single-bit elements. You control the code-generation of both Conditions and Events to either use single-bit with bit-mask, or to use a byte. Use the check-boxes in **Use Single Bit for**:

- **Condition** - Turn On/Off generation of Conditions as a bit or as a byte
- **Event** - Turn On/Off generation of Events as a bit or as a byte

User-Code Generation

Application Configuration Tab > Generate User-Code in glob_func.c

Rational Statemate MicroC Code-Generator allows generation of user-code (Functions and Subroutines) code body in the module that the user-code is used in, instead of it being generated into `glob_func.c`.

Start this mode using the check box “Generate User-Code in `glob_func.c`” in the Rational Statemate MicroC Code-Generator Property-Sheet.

When the check-box is checked (the default setting), the tool generates the body code in `glob_func.c`.

When the check box is not selected, the code is generated as follows:

- Regular Subroutine - Body is generated in the Module it belongs to.
- Subroutine defined in a GDS scope - Body is generated in glob_func.c.
- Subroutine defined in a Generic scope:
  - **As function** - Body is generated in the `g_<Generic-Name>.c File.`
  - **Inline** - Body is generated in `glob_func.c.`
- Before and after each section of functions in each module/generic file there are:
  - **Before the section** (and only if there is any Sub to generate), the definition of the API:

    ```
    USER_FUNCTIONS_BODY_DEFINITION_SECTION_HEADER()
    ```

  - **After the section** (and only if there is any Sub to generate), the definition of the API:

    ```
    USER_FUNCTIONS_BODY_DEFINITION_SECTION_FOOTER()
    ```
- Call-Back functions for Elements like Data-Items - Body is generated in `glob_func.c`.

Setting the Time Scale

This section contains the following information:

- **Setting the Time Expression Scale Preference**
- **Working with Multiple Counters**
- **Setting the Time Expression Scale**
- **Defining Counters in a cfg file**

Setting the Time Expression Scale Preference

Time expressions in Rational Statemate model can be scaled, by the MicroC Code Generator. The Micro C Code Generator Preferences preference: **Time Expression Scale**, allows the user to set the time scale for all expressions in the model to one of the following units:

- Counter Ticks (default)
- Seconds
- Milliseconds

The scaled time expressions are used with the following operators:

- `dly()` - Delay
- `tm()` - Timeout
- `sc!()` - Schedule

Working with Multiple Counters

Each time expression calculation is based on a counter, which operates at a fixed tick rate, for example, 1 ms per tick. The units of the expression are translated into ticks according to the Time Expression Scale.

Rational Statemate MicroC lets you work with multiple counters simultaneously, where each counter can run at a different tick rate.

The default counter used by the model is defined in the code generator's profile (**Options > Settings... > OS Tab > Primary Software Counter**),

You can use a different counter with each of the operators using “Time Expression”:

1. For tm(), the 3rd (optional) parameter can specify the counter to be used with this operation – “Primary Software Counter” is used if 3rd parameter is omitted.
2. For sc!(), the counter can be specified using the design-attributes of the related event.
3. For dly(), the “Primary Software Counter” is used, so the user cannot select the counter this operation will use. If such definition is required, use tm() instead. (if the dly() operation is used on a transition exiting from State “S1”, then: dly(10) is similar to tm(en(S1),10))

Setting the Time Expression Scale

Once the user has used a counter in the model, and selected “Time Expression Scale” other than “Counter Ticks”, it is required that the “Time Expression Scale” in the Rational Statemate MicroC Profile be filled in. Each counter has three fields:

- **counter name:** The name of the referenced counter
- **ticks per second**: The number of ticks the counter counts per second
- **ticks per milli-second:** The number of ticks the counter counts per milli-second.

Note that the fields “ticks per second” and “ticks per milli-second” don’t have to be both defined, nor does the ratio between their definitions have to be 1000.

If some necessary data is missing from the “Time Expression Scale,” the Code Generator will issue a warning message.

In the OSI: “MAINLOOP_SC” there are two predefined counters “ms_counter”, and “sec_counter”. Default values are already filled in the “Time Expression Scale Table” for these counters.

If the model does not use the seconds/milliseconds as the “Time Expression Scale”, then these definitions are not required.

When converting to code, the “Time Expression” is automatically scaled to ticks according to the current “Time Expression Scale”:

- **Counter's Ticks**: Nothing is changed - Use the time expression as in model.
- **Seconds:** Multiply the time expression with the following macro:

  “<counter>_TICKS_PER_SEC”
- **Milli - Seconds**: Multiply the time expression with the following macro:

  “<counter>_TICKS_PER_MSEC”

Defining Counters in a cfg file

It is possible to define the counter data in a Target-Configuration (.cfg) file (in the OSI).

The table can be saved with the compilation profile, and it can be read from a configuration file.

The content of the “Time Expression Scale” can be predefined in the “.cfg” files in the OSDT.

The following example shows the section that defines this table in the .cfg file:

```
#Time Expression Scale:
{
  #Counter:"ms_counter"
  #Tick Per Second:"1000"
  #Tick Per Milli Second:"1"
}
{
  #Counter:"sec_counter"
  #Tick Per Second:"1"
  #Tick Per Milli Second:"Error Usage of Counter"
}
```

Generation of Constant Elements with “const” Modifier

When the check-box **Use ‘const’ Keyword to define Constant Values** in the **Target** tab of the Code-Generator Property Sheet is selected, constant elements are generated with a const modifier.

When this option is selected, elements which are defined as Constant are generated with `const` modifier to the `glob_dat.c` and `glob_dat.h` files, instead of being generated as pre-processor macros in the `macro_def.h` file.

Default Data Types

You can define the default data type for the following types:

- Signed Integer
- Unsigned Integer
- Bit-Field
- Floating Point

You define the data types, using the Default Data Types field (from the Code Generator Property Sheet, select the Target Properties tab). Those default data types will be used when declaring data and using bit-wise shift operators.

Generating Code with Extended Documentation

In the setting dialog, selecting **General > Generate extensive code documentation** generates extensively documented code. The code includes extensive comments on each generated function and control expression in the following places:

- Activity function body, Statechart/Flowchart implementation function, Transition to a state, State static reaction, Data declaration, Generated files headers and footers.
- Information from the model, compiled as descriptions:
  - Information of the code-generation profile name, date, version, workarea and project name
  - For Statecharts: Before the implementation function, a textual transition table is added.
  - For Truth-Tables: A textual description of the table is added.
  - For Timeout setting: The expression triggering the code is in comments.

Dynamic Data Initialization

This feature enables the initialization of all the data in the model through calls to the `RESET_DATA` macro in the `TASKINIT` function.The initialization is enabled by the Memory Initializing check box in the Target Properties tab.The `RESET_DATA` macro uses the function `memset`, which should be defined in the environment. In the case where the function `memset` is not defined, you can define the macro `AVOID_MEMSET` and use the function `rimc_mem_set` (defined in the `<profile>.c` file).

OSEK GetResource Usage

When a TASK/ISR has related timeouts, MicroC calls:

1. `GetResource(RES_SCHEDULER)` before the code section that swaps the TASK/ISR event buffer and before the call to `genTmEvent(...)` in on<TIMER>OVERFLOW Tasks (in the `file glob_func.c`).
2. `ReleaseResource(RES_SCHEDULER)` after the code section that swaps the TASK/ISR event buffer and after the call to `genTmEvent(...)` in on<TIMER>OVERFLOW Tasks (in the file `glob_func.c`).

This resource usage can be avoided by clearing the Code Generation Profile option:

Options > OS > Allow GetResource(RES_SCHEDULER) Usage

Note

This applies to OSEK only.

Rational Statemate Block in a Rational Rhapsody Model

The code generator supports integration of a Rational Statemate model (block) into a Rhapsody model with the following prerequisites:

- Rational Rhapsody Developer for C version 7.1.1 or newer
- Rational Statemate 4.2 MR2 or higher on the system and licensed
- License for Rational Statemate MicroC code generator

Required Rational Statemate Model Characteristics

To synchronize a Rational Statemate model with Rational Rhapsody, the Rational Statemate model must have the following characteristics:

- Only one top-level activity
- MicroC profile with only one module

Preparing the Rational Statemate Model

Follow these steps in Rational Statemate to prepare the Rational Statemate model for integration with Rational Rhapsody:

1. Open the Rational Statemate model. If you want to show Rational Statemate animation in Rhapsody, be certain to select the GBA option from the Rational Statemate MicroC profile options.
2. To set the required properties in Rational Statemate before generating code:
   a. Open the Rational Statemate MicroC Code Generator.
   b. Select **Options > Settings > Application Configuration > Application Files**.
   c. In the Application Files dialog box, select both of the **Generate Code in a Single File** and **Generate Code as Statemate Block** items.
   d. Click **OK**.
3. Generate the C code using the Rational Statemate MicroC Code Generator.
4. In the Rational Statemate main interface, select the **Files** tab and follow these steps:
   a. Select a Rational Statemate MicroC code generator profile.

b. Select **Configuration > Create Statemate Block Configuration for Rhapsody > Read mode/Update mode**.

Synchronizing Rational Statemate and Rational Rhapsody

In the Rhapsody system, you must use the StatemateBlock profile as the container for the Statemate model. See the Rational Rhapsody documentation for instructions to create the Rational Rhapsody StatemateBlock and connect the Rational Rhapsody and Rational Statemate models.

The StatemateBlock operates as a black-box for Rational Statemate code within the Rational Rhapsody architecture once it has been connected and synchronized. The StatemateBlock interface of the top-level flowing data within the Rational Statemate model is specified in Rhapsody using flow ports.

The Rational StatemateBlock in Rational Rhapsody automatically synchronizes with the Rational Statemate model and adds or removes flow ports from the Rational StatemateBlock to reflect any changes made in the Rational Statemate top-level flowing data. The synchronization operation uses a Rational Rhapsody Block Configuration containing the following Rational Statemate data:

- Rational Statemate MicroC Profile with a single module
- Rational Statemate charts that are in the scope of the MicroC profile. The top-level chart must have a single top-level (regular) Activity
- Rational Statemate Panels that are in the scope

Troubleshooting Rational Statemate with Rational Rhapsody

When entering information into the Import/Sync Statemate Block dialog box, you may receive one of these error messages. The table below shows the possible error messages and their explanations.

Error Message	Explanation
“Cannot load libraries. Please make sure you are using the correct Statemate installation path.”	Rhapsody was unable to located the stmBlockInterfaceDll.dll in the bin directory of the installation path entered into the dialog box. Correct the **Rational Statemate Installation Path** so that the DLL can be located.
“PM Filepath not found. Please specify a valid PM path.”	The Rational Statemate PM file name entered into the **Rational Statemate PM Location** field must contain “pm.dat” in the name.
“Invalid Statemate Project. Please select a Statemate Project before pushing OK.”	The **Import/Sync** process has checked the Rational Statemate MicroC profile and the Rational Statemate project was not found or the project name did not match the one entered in the Rhapsody dialog box.

Error Message	Explanation
“Invalid Rhapsody Block name. Please select a Statemate Block before pushing OK.”	The **Import/Sync** process has checked the Rational Statemate MicroC profile and cannot locate the Rational StatemateBlock that was entered in the Rhapsody dialog box.
“Invalid Statemate Workarea name. Please select a Statemate Workarea before pushing OK.”	The **Import/Sync** process has checked the Rational Statemate MicroC profile and the Rational Statemate Workarea name was not found that matched the name entered in the Rhapsody dialog box.
“Missing Statemate Block’s charts. Would you like to perform check-out and generate code now?”	The **Import/Sync** process checked for the Rational Statemate code that should have been generated as described in the **Preparing the Rational Statemate Model** section. If it needs to be generated, click **Yes** in this error message box.
“Missing generated code for StatemateBlock. Would you like to generate code now?”	The **Import/Sync** process checked for the Rational Statemate code that should have been generated as described in the **Preparing the Rational Statemate Model** section. If it needs to be generated, click **Yes** in this error message box
“Missing required files. Cannot synchronize with Statemate model.”	If you selected **No** when the system offered to generate the Rational Statemate code (see the two previous error messages), this error message indicates that the Rational Statemate model cannot be synchronized with the Rhapsody until the Rational Statemate code has been generated.

Code Optimizations

The optimization options below can be found in **Statemate MicroC Code Generator > Options > Settings > Optimization Tab.**

Empty Overlapping Tests of State Hierarchy

This feature further optimizes the generated code for Statecharts. The generated code avoids certain tests that are normally performed before a State hierarchy is entere, resulting in a smaller object code size.

Note that although this optimization will reduce the object code size, it might increase the scan time for a transition, because the generated code will be flattened.

Generate All and Generate Only Used

The flag **Generate Model Data** in the compilation profile can be used to control generation of data and functions, where you can choose between generating **All Elements** and generating **Only Used Elements**.

Optimization Algorithms

The following algorithms are used to optimize the code generated by Rational Statemate MicroC:

Inline Setting of the “Need Another Step” Bit

To improve code efficiency, you can specify the criterion No. of Transitions (<= 999, by default). This criterion determines whether the optimization is performed.

Inline Entering and Exiting Reactions

Inlining exiting and entering reaction on transitions is performed according to user defined criteria.

Inlining entering or exiting reactions is based on the following criteria:

- No. Of Statements (<= 5 , by default)
- No. Of Instances (<= 999 , by default)

Example

When having single hierarchy, -T0-> S1 –T1-> S1.S11 –T2-> S1.S11.S111 a transition entering that hierarchy will target directly S111 no having intermediate states S1 and S11.

Behavior definition (semantics): When taking a transition, the transition action is performed, then exiting reaction, from the inner most state that the transition exit to the outer most, then entering reaction is performed, from the outer most state the transition is entering to the inner most.

Without optimization, the following sequence would have been performed as follows:

Consider action Ai on transition Ti, entering reaction Ei for state Si Then:

A0, E1, A1, E11, A2, E111

After “clutching” the sequence will be:

A0, A1, A2, E1, E11, E111

The test models show a 20-30% reduction in ROM size when using those optimization flags.

Reuse of Timeout Variables

To reduce the number of data allocations for the timeout operation, the timeout algorithm merges data allocation for two timeouts that relate to mutually exclusive states.

The optimizer looks for those timeouts and delays that are pending in exclusive states. For those, the same variables might be used.

Example

"Train" states might use a single variable

Clutching Entrance to a State Hierarchy

Rational Statemate MicroC can perform a clutch of steps, intermediate states, and default states when entering state hierarchy. The clutch entrance algorithm steps directly into the lower-most leaf state in the state hierarchy, eliminating intermediate states. All the entering reactions are performed appropriately, according to the state hierarchy.

For more information on code optimization, see the *MicroC Programming Style Guide*.

Additional Optimization Options for Code Generation

- Merge State sequences with no guard on transition.
- Inline default test.

OS Definition Tool

This section contains the following information:

- Design Attributes
- Element Attributes
- Task Execution Mode API and Design Attributes
- Get-Set Functions for Buffered Access Data-Items
- OS Static Configuration
- Defining the Location of the CTD Directory
- APIs
- API Modification Rules
- Upgrading an OSI
- List Support in OSDT
- Generated Data Declaration

Design Attributes

This section contains the following information:

- Design Attribute Notation
- Inheritable Design Attributes
- Special Design Attributes

Design Attribute Notation

The Design-Attributes entry fields can use the **$<**`Attribute-Name`**>** and ?**<...>** (API-like) notation.The $<> and ?<> notation can be used only for Design-Attributes that belong to the same Element. For detailed information on this syntax, see the *MicroC Programming Style Guide*.

Inheritable Design Attributes

An inheritable attribute is one that can be passed from one graphical element (such as an Activity or a State) to another one lower in the hierarchy tree. To designate a Design Attribute as "Inheritable," add its name to the `inheritable_attributes.txt` file, which includes a newline delimited list of "Inheritable Attribute" names. If the file does not exist, create it in the `$STM_ROOT\etc\ctd\<osi-name>` directory).

Special Design Attributes

cg_build_ver

The formal-parameter, `$<cg_build_ver>`, represents the tool version in all APIs.

moduleName, profileName

The following Design Attributes are accessible by every API in the OSDT:

- `moduleName` (accessible by every API that refers to a model object such as Activity, Dataitem etc.).
- `profileName`

CK_predefinedTask

The following predefined Tasks are tagged with the Design-Attribute CK_predefinedTask = yes (for usage in the OSDT)

- TASKINIT
- Timer Overflow Tasks
- GBA Task
- Panel Dispatch Task
- Test Driver Task

Frequency of Activation of Activity

The following Design Attributes are used to define the frequency in which a certain activity is executed per task activation [task runs per Activity executions].

- Use Activation Frequency - yes/no.
- Activation Frequency - The default value is 1 (run every Nth task run).
- Activation Shift - The default value is 0 (a shift of 0, or no shift).

Both the Frequency and the Shift values must be either Decimal, Hexadecimal, or any other literal-named value that starts with a letter.

Element Attributes

Element Attributes can be accessed from the OSDT. The access is enabled by setting the following environment variable: `AMC_AUGMENT_ATTRIBUTES=ON`.

Note

Limitations:

- Only single-line attributes should be used (for all elements)
- Only standard characters should be used (no special controls).

Task Execution Mode API and Design Attributes

The following Design Attributes, defined in the DEFAULT and MAINLOOP_SC OSI's, allow overriding the default Task Run Mode for each Task:

Display Name	Attribute Name
Task Run Mode	CK_taskRunMode
Define Max Steps per Super Step	CK_useMaxStepsPerSuperStep
Max Steps per Super Step	CK_maxStepsPerSuperStep

The following API, available in the `DEFAULT` and `MAINLOOP_SC` OSIs, is located in: **Code Style** > **Variables Naming Style**:

Display Name	Attribute Name
Super Step Steps Counter Name	activityNameid

Get-Set Functions for Buffered Access Data-Items

Enhances the ability to express the value of the Set Value Call and Get Value Call, by allowing the use of the $<Attribute-Name> and ?<...> (API–like) notation.

The $<> and ?<> notation can use only Design-Attributes that belong to the same Element.

For detailed information on this syntax, see the *MicroC Programming Style Guide*.

OS Static Configuration

- Each OS element has 12 API entries in the Static-OS-Configuration Page.
- The selection **Code Generation Profile > Setting > OS Tab** allows a list of input-output file pairs in the Static Configuration files, delimited with semi-colons.

Defining the Location of the CTD Directory

You can define the path that Rational Statemate MicroC uses for the CTD directory, by setting the following environment variable:

```
STM_CTD_DIR = <my path\CTD>
```

APIs

This section describes the various APIs, according to their location.

Code Style

The `Condition Buffer User-Type Name()`: API is located under the “Code Style Page --> Types Naming Style” page. It is available with the ‘default’ OSI and can be loaded to the existing OSI using the “Update From OSI” under the OSDT File menu.

Description

Defines the name of the User-Defined-Type that is used as the type of the buffer generated for Conditions in the Code-Generator's mode: 'Use Separate Buffer for: Conditions' (See the Code-Generator Property-Sheet, General Tab).

The following APIs have a Tab named “Internal Data-Types” in the “API Definitions” page. These APIs are available with the ‘default’ OSI and can be loaded to the existing OSI using the “Update From OSI” under the OSDT File menu.

- `Condition Buffer User-Type Type()`:

  This API defines the type of the User-Define-Type used when generating the Conditions in the 'Buffer per Condition' Mode.

- `Event Buffer Type()`:

  This API defines the type of the Buffer used when generating the Events in the 'Buffer per Event' Mode.

- `Default Signed Integer Type()`:

  This API defines the default integer signed type. It overrides the definition in the Code-Generator property-Sheet.

- `Default Unsigned Integer Type()`:

  This API defines the default integer signed type. It overrides the definition in the Code-Generator property-Sheet.

- `Default Floating Point Type()`:

  This API defines the default Floating Point type. It overrides the definition in the Code-Generator property-Sheet.

- `Bit Field Type()`:

  This API defines the Dada-Type used for Conditions and Bit Data-Items, when generating Conditions/Bits without macros. It overrides the definition in the Code-Generator property-Sheet:

The following APIs, located in the Code-Style tab, can use the full syntax of the OS Customization Tool ($<..> and ?<..>). The parameter `activityNameid` is passed to some of the APIs and refers to the owner Activity. The parameter `udt_or_activity_nameid` is passed to some of the APIs and refers to the owner of the buffer (an Activity or a User-Defined Type). The syntax $<...> and ?<...> can be used in the APIs with the element's Attributes. The Attributes see the element that owns or requests the corresponding API. The APIs are as follows:

- Single Buffer Type Prefix(activityNameid):
- Double Buffer Type Prefix(activityNameid):
- State Variable Prefix(activityNameid):
- Single Buffer Variable Prefix(activityNameid):
- Double Buffer Typedef Prefix(activityNameid):
- Double Buffer Next Variable Prefix(activityNameid):
- Double Buffer Current Variable Prefix(activityNameid):

The `Profile .c File Header(profileName, fileName, genDate, genTime,profileOptions, Project, Workarea, profileVersion)` API is located in the Default OSI: Code Style Page -> File Header/Footer.

This API controls the header of the <profile name>.c file. When checked, the tool looks for the related key word (originated from the OSI user_code.c file) and replaces it with the API definition. The definition of the API is inserted either in the beginning of the file <profile>.c, or in the location marked by the keyword: /* keywordfor <profile>.c header */.

The following APIs (Code-Style page) define the timeout naming method.

- Timeout Event Name(nameid)
- Timeout Time Variable Name(nameid)
- Timeout Mask Name(nameid)
- Timeout Counter Index Name(nameid)

The following APIs (defined in the DEFAULT OSI) are accessible from the Get/Set Functions tab in the API Definitions... page.

- Get function Declare(nameid, returntype, argType, argName)
- Get function Define(nameid, returntype, argType, argName, getElemCode)
- Get function Name(nameid)
- Set function Declare(nameid, returntype, argType, argName)
- Set function Define(nameid, returntype, argType, argName, tstDrvInst,setElemCode)
- Set function Name(nameid)

The following APIs (available in the DEFAULT OSI and located in Code Style -> FileHeader/ Footer) allow customizing of the `<profile>.h` header and footer:

- Generated Profile H File Header(profileName, fileName, genDate, genTime,profileOptions, project, workarea, profileVersion)
- Generated Profile H File Footer(profileName, fileName, genDate, genTime,profileOptions, Project, Workarea, profileVersion)

The following APIs define the generated code naming conventions. They allow control over the actual names and not only the prefix (as the pre-existing APIs did).

- Check for Timeout Function Name(activityNameid)
- Activity Function Name(activityNameid)
- Statechart Function Name(activityNameid)
- Entering Reaction Function Name(activityNameid)
- Exiting Reaction Function Name(activityNameid)
- Generic Chart Function Name(activityNameid)

Customizable Timeouts using OSDT

Allows the user to customize the generated code supporting Timeouts (*tm()/dly()*) using the OS Definition Tool with the following APIs:

- `Timeout Install Define(nameid, tmMaskName, tmMaskVal, tmVariableName, tmMaskVarName, tmVariableType, tmCurrentTickName, tmCounterVarName, tmCounterName)`

  The definition of the Timeout installation, in the file: “macro_def.h”.

- `Timeout Install Call(nameid, time, counterIndex, tmCounterName, tmCurrentTickName)`

  The definition of the call to the install of a Timeout, in file: <module>.c

- `Timeout Test on Expiration Call(nameid, tmCurrentTickName, tmCounterName, tmEventBuffer, tmCounterIndex, genContextVar)`

  The definition of the call to the Timeouts Dispatch function, in the file: <*module*>.*c*.

- `Timeout Test on Expiration Define(nameid, tmCurrentTickType, tmCurrentTickName, tmEventBuffType, tmEventBuffName, tmCounterIndexType, tmCounterIndex, genContextVar, timeoutList)`

  The definition of the Timeouts Dispatch function, in the file: <*module*>.*c*.

- `Timeout Test on Expiration Declare(nameid, tmCurrentTickType, tmCurrentTickName, tmEventBuffType, tmEventBuffName, tmCounterIndexType, tmCounterIndex, genContextVar)`

  The forward declaration of the Timeouts Dispatch function, in the file: *type_def.h*.

- `Timeout verflow Code Per Task(tmMasks, tmDispatchFunc, counterMaxAllowdVal, tmEventBuffName, counterIndex, genContextVar, timeoutList, timeoutVarType, counterValueType)`

  The overflow code related to a specific Task with Timeouts, put in the Overflow-Task, in the file: `glob_func.c`.

These APIs are available for update from the DEFAULT OSI.

Support for Queues

The following APIs are located in the DEFAULT OSI in the OSDT:

- `Queue Descriptor Data Type(nameid, qType)`

  Defines the Data Type declaration of a Queue.

- `Queue Data Type Static Init(nameid, qType)`

  Used to Initialize the Queue statically.

- `Queue Put(nameid, elName)`
- `Queue Urgent Put(nameid, elName)`
- `Queue Get(nameid, elName, statElName)`
- `Queue Peek(nameid, elName, statElName)`
- `Queue Flush(nameid)`
- `Queue Length(nameid)`

Note

The Queue Type can only be simple types as integer and real. Otherwise, a UDT should be used.

Task/ISR APIs

OS APIs

The following API is used for Tasks, in the definition of the macros: `stop_activity, suspend_activity:`

Suspend Task(nameid)

Memory Management: Beginning and Ending Code for Task/ISR

APIs in the OSDT (Memory Management Page) for Task/ISR leading and trailing code:

- `Task/ISR Beginning Code(nameid, profileName):`

  Specifies the code added at the beginning of a Task/ISR body.

- `Task/ISR Beginning Code Entry 2(nameid, profileName):`

  Specifies the code added at the beginning of a Task/ISR body. It is added right after the Task/ISR Beginning Code API definition.

Note

If a Task is using Periodic Activation, the code for this API is generated after the code that handles the periodic activation.

- `Task/ISR Ending Code(nameid, profileName):`

  Specifies the code added at the end of a Task/ISR body.

  - Use this API to put code or function-call in the last line of the generated code in a Task/ISR (before the 'TerminateTask' if defined).
  - This API is defined outside the Super-Step Loop, so it can be used to generate a notification when a given number of steps have been performed.

- The APIs are located in the following OSIs:
  - MAINLOOP_SC
  - MAINLOOP_SC_EXT
  - NATIVE_NT
  - DEFAULT

Memory Management:Task/ISR and Related Activities

- The parameter `nameid` is used in the following APIs on the memory management page, Code - Task/ISR and Related Activities tab:
  - `Task/ISR Opening(nameid)`
  - `Task/ISR Closure(nameid)`
  - `Activity Function Opening(nameid)`
  - `Activity Function Closure(nameid)`
  - `Activity Function Call Opening(nameid)`
  - `Activity Function Call Closure(nameid)`
- The following API is located in the memory management page, Code - Task/ISR and Related Activities tab:

  `Related Function Call(nameid, arglist)`

  This API controls the call style of functions related to the Task/ISR and related Activities, such as cgEnterActions_... and cgExitActions_....

Attribute in Predefined Tasks

All predefined Tasks include the Attribute CK_predefinedTask with the value “yes”. This Attribute can be used in any API definition that relates to a TASK. The predefined tasks are:

- **TASKINIT** - Initialization Task.
- **PANEL_DISPATCH** - Handle Panels, if available.
- **BUFFERED_GBA_TASK** - Handle GBA Instrumentation in buffered mode.
- **TEST_DRIVER_TASK** - Handle Test Driver Instrumentation, if used with a separate task.
- **Timer Overflow Task**s - Handle the overflow of the various timers.
- **Test Bench Task** - Handles the test benches

Statecharts Functions

These are APIs in the Memory Management page that control the beginning and end code of the Statechart function:

- `Statechart Beginning Code(nameid):`

  Description: This line controls the code at the beginning of the Statechart. It will be added as the first executable statement in the Statechart function.

- `Statechart Ending Code(nameid):`

  Description: This line controls the code at the end of the Statechart. It will be added as the last executable statement in the Statechart function. In these APIs, nameid is the name of the Control-Activity whose Function is implemented.

APIs for Function-Declare-Style

'Code - per User Function' Tab

```
User Function Definition Style(nameid, returntype, arglist, shortdescription)
```

Description: This line controls the specific function declaration style.

'Code - Task/ISR and Related Activities' Tab

```
Related Function Declaration Style(nameid, returntype, arglist)
```

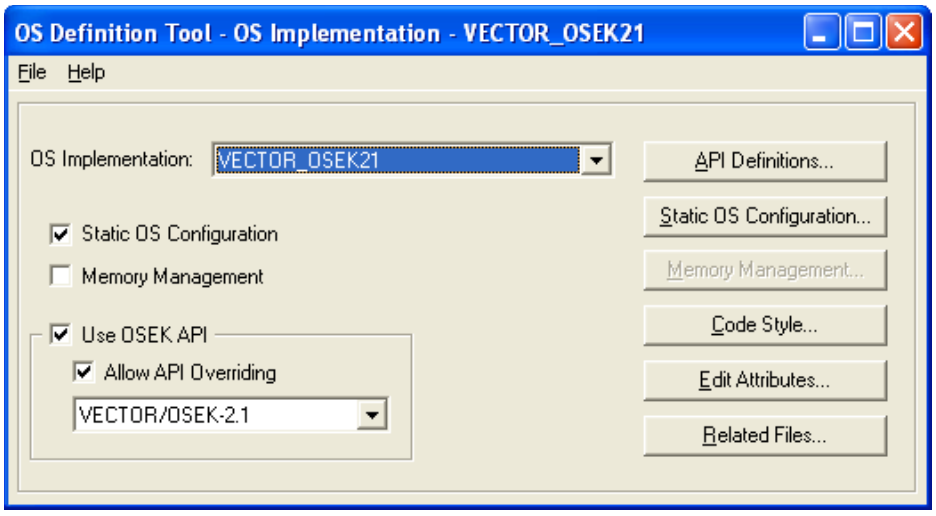
Description: This line controls the declaration style of functions related to theTask/ISR and related Activities, such as cgEnterActions_... and cgExitActions_...

These APIs are available in the following OSIs:

- MAINLOOP_SC
- MAINLOOP_SC_EXT
- DEFAULT:

Customizable OSEK APIs

You can define the OSEK APIs from the OSDT. When the check box **AllowAPI Overriding** in the **Use OSEK API** frame is checked, the API Definitions button is active and enables the API modification.



API Modification Rules

If the API is not checked (selected) in the API Definitions page, the (original) default definition for the API is used. If the API is checked (selected) in the API Definitions page and the Definition is not an empty string, the definition is used. Otherwise, the (original) default definition is used.

Upgrading an OSI

The OSDT supports updating the existing OSI, which might have been created using a previous version of the tool, against a new OSI. The Update OSI operation can be opened by selecting the new menu entry, Update OSI, in the OSDT file menu.

The Update OSI operation performs the following checks:

- Checks APIs per page and per section against the reference OSI to look for APIs that exist in the default but are missing in the current OSI. In this case, you will be prompted to confirm the update.
- Checks for obsolete APIs (those that exist in the current OSI but not in the reference OSI). In this case, you will be prompted to confirm the update (to remove the API).
- A check for new, modified, or obsolete files is performed for the RelatedFiles.

All the changes done by the Update OSI operation are logged in a log file. The log file is located in the CTD directory (`<OSI NAME>DD/MM/YY.txt`). After the operation is completed, the OSI will be marked as “modified.” The changes will take affect only when the OSI is saved. In addition, when you open a workarea, the tool will check whether the latest version of the related files is in use. If not, the tool will prompt you to update the files.:

List Support in OSDT

The OSDT syntax supports list operations on Parameters and Attributes. The operators are available only with the following APIs:

- Timeout Test on Expiration Define
- Timeout Test on Expiration Declare
- Timeout Overflow Code Per Task

The OSDT replaces each element in a list with a template-string and adds it to the definition of the API.

A list is a string operated upon with list operators, having the character '@' used as delimiter between elements in the list.

Syntax

```
@<for> @<list-name> @<begin> $<list-name> @<end>
```

The definition of a list template-string starts with the `@<for>` token and ends with the `@<end>` token. All the text between these two tokens will be replaced with the definition of the API after the template-string is processed.

The string `@<list-name>` between the `@<for>` and the `@<begin>` is the name of the formal argument, being an Attribute or a Parameter, that represent a list of values (a string like: `val_1@val_2@val3`).

The string `$<list-name>` between `@<begin>` and `@<end>` is the string-template which will be used for each one of the list's elements. The `$<list-name>` in this string will be replaced with the current list element.

Example

The Design Attribute CK_workingDay is defined in the Dictionary as

```
MONDAY@TUESDAY@WEDNESDAY@THURSDAY
```

The Design Attribute CK_freeDays is defined in the Dictionary as

```
SUNDAY@FRIDAY@SATURDAY
```

The definition of the API is:

```
============ Beginning of API Definition ================
The Working Days are:
@<for> @<CK_workingDays> @<begin> - $<CK_workingDays> is a working day
@<end>

The Free Days are:
@<for> @<CK_freeDays> @<begin> - $<CK_freeDays> is NOT a working day
@<end>
============ End of API Definition ===================
The resulting string output of the API after processing is:
---------------------------------------
The Working Days are:
- MONDAY is a working day
- TUESDAY is a working day
- WEDNESDAY is a working day
- THURSDAY is a working day

The Free Days are:
- SUNDAY is NOT a working day
- FRIDAY is NOT a working day
- SATURDAY is NOT a working day
---------------------------------------
```

Generated Data Declaration

The tool allows generation of user-defined code between the generated data declaration sections in `glob_dat.h` and `glob_dat.c`.

The user code is defined as a set of APIs in the OSDT, relating to keywords placed in the `type_declare_order.txt` file, located at:

```
ROOT\etc\ctd\<OSI>\type_declare_order.txt.
```

The APIs are available with the ‘default’ or ‘mainloop_sc_ext’ OSI, and can be loaded to existing OSI using the “Update From OSI” under the OSDT File menu.

The APIs are:

API	Key-word
8-bit Declaration Begin Section:	'/* Key word: for 8-bit data declaration begin section */'
8-bit Declaration End Section:	'/* Key word: for 8-bit data declaration end section */'
16-bit Declaration Begin Section:	'/* Key word: for 16-bit data declaration begin section */'
6-bit Declaration end Section:	'/* Key word: for 16-bit data declaration end section */'
32-bit Declaration Begin Section:	'/* Key word: for 32-bit data declaration begin section */'
32-bit Declaration End Section:	'/* Key word: for 32-bit data declaration end section */'
Record Declaration Begin Section:	'/* Key word: for record data declaration begin section */'
Record Declaration End Section:	'/* Key word: for record data declaration end section */'
Other Types Declaration Begin Section:	'/* Key word: for other types declaration begin section */'
Other Types Declaration End Section:	'/* Key word: for other types declaration end section */'
8-bit Extern Declaration Begin Section:	'/* Key word: for 8-bit data extern declaration begin section */'
8-bit Extern Declaration End Section:	'/* Key word: for 8-bit data extern declaration end section */'
16-bit Extern Declaration Begin Section:	'/* Key word: for 16-bit data extern declaration begin section */'
16-bit Extern Declaration End Section:	'/* Key word: for 16-bit data extern declaration end section */'
32-bit Extern Declaration Begin Section:	'/* Key word: for 32-bit data extern declaration begin section */'

API	Key-word
32-bit Extern Declaration End Section:	'/* Key word: for 32-bit data extern declaration end section */'
Record Extern Declaration Begin Section:	'/* Key word: for record data extern declaration begin section */'
Record Extern Declaration End Section:	'/* Key word: for record data extern declaration end section */'
Other Types Extern Declaration Begin Section:	'/* Key word: for other types extern declaration begin section */'
Other Types Extern Declaration End Section:	'/* Key word: for other types extern declaration end section */'

Example

If the file: `'type_declare_order.txt'` looks like:

```
uint8
/* Key word: for 16-bit data declaration begin section */
uint16
uint32
```

The generated `glob_dat.c` looks like:

```
...
uint8 uint8_data;
< Definition of the: ‘16-bit Declaration Begin Section’ API>
uint16 uint16_data;
uint32 uint32_data;
...
```

Limitations

The keywords that are put in the end of the file: 'type_declare_order.txt' are ordered in the following order, instead of the order between themselves:

1. 8-bit
2. 16-bit
3. 32-bit
4. record
5. other

Supported Targets

The following targets are supported:

- Vector/OSEK on Motorola HC12, Fujitsue FM16LX (MB90V595)
- Turbo/OSEK on Motorola HC12
- Fujitsue FM16LX (MB90V595) support in: Vector/OSEK, mainloop_sc and mainloop_sc_ext OSIs

Utilities

This section contains the following information:

- **Remote Panel Server Support**
- **MicroC Design-Level Debugger**

Remote Panel Server Support

You can configure code generated by Rational Statemate MicroC to use a Remote Panel Server to display graphical panels. The executable communicates with the Remote Panel Server via TCP/IP communication protocol.

Upon invocation, the Remote Panel Server communicates the necessary communication parameters to the executable by writing them to the file `rcomm.cfg`.

Note the following:

- The Remote Panel Server must have access to the generated code directory to be able to read the generated panel files.
- In the case where no file system can be shared with the target, you should adapt the `libsvrcom.lib` library code to identify the server.
- When no direct TCP/IP link is available to the target, you should adapt the `libsvrcom.lib` library and `rimc_rpgertl.c` file.

Using the Remote Panel Server

To use the Remote Panel Server, in the Code-Generator profile, select **Settings** > **Application Configuration Tab** and enable the **Use Remote Panel(s)** check box.

If the executable has no access to the Workarea directory:

1. In the Code-Generator profile, select **Settings** > **Application Configuration Tab**.
2. Define the Target Directory. The Target Directory is the current directory of the executable (as “seen” from the viewpoint of the executable).
3. Select **Application Files**.
4. Add the following line to the Additional Sources field:

```
<Rational Statemate Installation directory>
\etc\CTD\default\rimc_rpgertl.c
```

Invoking the Remote Panel

The Remote Panel Server is located in the bin directory and it requires a license.

To set the necessary environment variable definitions, execute the `run_amc.bat` script file (located in the bin directory) at a command prompt as follows:

```
set STM_STAND=ON
set STM_RPANEL_STAND=ON
```

If the executable can access the Workarea directory, enter the following command:

```
switch_rpanel -workarea <workarea>
```

The `rcomm.cfg` file is written by the Remote Panel Server to the Workarea directory and read from there by the executable.

If the executable cannot access the Workarea directory and the Target Directory is defined, enter the following command:

```
switch_rpanel -gen_code_dir <dir>
```

The `rcomm.cfg` file is written by the Remote Panel Server to the `<dir>` directory. The executable reads the `rcomm.cfg` file from the Target Directory.

MicroC Design-Level Debugger

A beta version of the Rational Statemate MicroC Design Level Debugger is available. This debugger can be used with the target-debugger integration package (`AMC_COMMUNICATION_DLL`) to control graphical breakpoints, Break on Transition to State, and Break on Activity on target.

Contact your sales representative for more information.

Rational DOORS RT Interface

This section describes the Rational DOORS RT Interface. The topics are as follows:

- **Configuring the RT Interface**
- **Working with the RT Interface**

Systems engineers typically need to demonstrate that their system design specification and the models they build from that specification map back to the original customer requirements.

If the customer is using Rational DOORS (Requirements and Traceability Management Dynamic Object Oriented Requirements System) to manage the project, Rational Statemate provides a Statemate–to–DOORS interface (RT Interface) that enables the Rational Statemate project manager to export Rational Statemate elements to the Rational DOORS repository.

The RT Interface enables you to:

- Transfer complete Rational Statemate models or subsets of Rational Statemate models into Rational DOORS.
- Keep the information about the Rational Statemate model updated in Rational DOORS.
- Represent a Rational Statemate model in a Rational DOORS formal module so that the Rational Statemate model can be viewed as a special kind of requirements module filled with design elements.
- Support the automatic creation and population of Rational DOORS link modules. These provide the means to link certain Rational Statemate elements within the Rational DOORS module.

The RT Interface is designed to be used in environments where a one-to-one relationship exists between a Rational Statemate project and a Rational DOORS project.

Although it is possible to relate more than one Rational Statemate model to a Rational DOORS project, it is recommend that you do **NOT** do this.

Because the full name of each Rational Statemate chart element (`chart:name`) must be unique within the Rational DOORS/Statemate module, it is recommended that you restrict the mapping to one module, one Rational Statemate project, one Rational DOORS project.

The RT Interface attempts to impose as few restrictions on process management as possible. You determine what elements to export and when they are to be exported, following the guidelines that are presented in this section.

How the RT Interface Works

Setting up the Interface is very simple: the Rational Statemate project manager specifies the name of the corresponding Rational DOORS project; sets up default directories to hold chart plots, log files, and use case description files; and creates a default configuration file.

From the toolbar of the main Rational Statemate window or from the Properties dialog box, you export charts from the Rational Statemate project to the Rational DOORS project. The first time you export a set of charts, Rational Statemate automatically creates a formal module in the Rational DOORS database with a user-defined name, into which *shadow copies* of the exported charts and chart elements are placed.

Shadow copies contain the element name, type, and description fields.

Note

In the case of a Rational Statemate transition, which has no explicit name, the shadow copy is given a pseudonym.

For more information, see **Working with the RT Interface**.

Exporting Data

Before you export Rational Statemate data to Rational DOORS, determine what modules are suitable and necessary to export and then configure them for export (see **Preparing Rational Statemate Elements for Export to Rational DOORS**).

Once you have configured a project for export, to export the data to Rational DOORS:

Note

The initial export is performed once for each module.

1. Check-out (with locks) the Rational Statemate charts to be exported.
2. Define a Rational DOORS module and select the appropriate charts and elements (using the main Rational Statemate window and the RT Interface).
3. Export the charts using the RT Interface.
4. Save the configuration of the chart elements and the module as a Rational DOORS configuration file.

5. Check-in the Rational Statemate charts and unlock them if necessary.

For more detailed instructions, see **Exporting Rational Statemate Data to Rational DOORS**.

Re-Exporting and Synchronizing Data

During Rational Statemate design work, model elements frequently change, and in certain instances may be deleted altogether. As a result, it is necessary to re-export and synchronize the data in the Rational Statemate project with the data stored in the Rational DOORS repository.

To re-export and synchronize data:

1. Check-out (with locks) the Rational Statemate charts to be exported.
2. Open the appropriate saved Rational DOORS configuration file.
3. Set up the RT Interface to synchronize the data.
4. Synchronize the data.

   Modified elements overwrite existing elements (with user-prompting for confirmation) and deleted elements are deleted – both from the Rational DOORS repository and from the corresponding shadow copy.
5. Check in the Rational Statemate charts and unlock them if necessary.

Note

You must re-export Rational Statemate data to Rational DOORS manually. No automatic mechanism (like a makefile) triggers the re-exporting and synchronization of data when new Rational Statemate elements are created or when existing ones are changed or deleted.

For more detailed instructions, see **Exporting Rational Statemate Data to Rational DOORS**.

Methodology Guidelines

In a typical Rational Statemate project, there are multiple users and multiple workareas. All of the workareas in the project can be used with the same Rational DOORS database. However, because some workareas may contain outdated versions of the model, it may be helpful to:

- Divide the Rational Statemate model into groups of charts, and assign responsibility for editing and updating the charts to a variety of people working on the project.
- Designate a single workarea as the Rational DOORS update workarea, and do all the updates from this location.

The amount of data that can be exported to Rational DOORS can be overwhelming. You should therefore determine beforehand what modules and how much data you will be exporting.

It is recommended that when you configure a module for export, you limit the exported data to a bare minimum to provide concise linking to the requirements.

For example, in a simple scenario, users typically define three separate element modules and a links module. The modules are used to contain use cases, sequence diagrams, and activity charts. The links module contains linksets, which define the explicit relationships (links) between, for instance, use cases and sequence diagrams, or sequence diagrams and activity charts.

Configuring the RT Interface

This section explains how to configure the RT Interface on your operating system, and consists of the following:

- **Preliminary Requirements**
- **Configuring the RT Interface on Windows**

Note

During the Rational Statemate installation you are prompted to configure the RT Interface. If you have already configured the RT Interface during a previous Rational Statemate installation, you can ignore this section.

Preliminary Requirements

Before configuring the RT Interface, you must have all of the following software licensed on your system:

- Rational Statemate
- Rational DOORS
- The RT Interface

Configuring the RT Interface on Windows

To configure the RT Interface on Windows NT, you must define parameters in the following batch files:

- `run_stmm.bat` - starts Rational Statemate.
- `doors.bat` - starts Rational DOORS in batch mode during export operations in Rational Statemate.
- `run_doors.bat` - starts Rational DOORS from the icon on the Rational Statemate main window.

Note

Depending on how Rational Statemate and Rational DOORS are on your system or network, you may require privileges to configure the RT Interface.

To edit these batch files:

1. Locate the Rational Statemate `bin` directory by doing the following:

   a. Select **Start** > **Programs** > **IBM Rational** > **Rational Statemate <version #>** > **Statemate <version #>**.

   b. Right-click on **Statemate** <version #>.

   In the Rational Statemate Properties window, the **Target** field contains the full path of the file `run_stmm.bat` in the Rational Statemate `bin` directory.

2. Change your working directory to the directory you specified.

Edit run_stmm.bat

1. Copy `run_stmm.bat` to `run_stmm.ORIG.bat`.
2. Using the text editor of your choice, open `run_stmm.bat` and search for the word "DXLPORTNO." This brings you to the Rational DOORS section of the file.
3. To enable the RT Interface, remove the `@rem` prefix from the following SET commands:

```
@rem SET DXLPORTNO=5093
@rem SET DXLIPHOST=%COMPUTERNAME%
@rem SET STM_RT_TOOL=DOORS
@rem SET RT_MAIN_TOOL=%STM_ROOT%\BIN\RUN_DOORS
@rem SET DOORSHOME=<doors home>
@rem SET DOORSS_BATCH_FILE=doorss.bat
@rem SET STM_DOORS_USE_SYS_LOGIN=ON
@rem SET PATH=%PATH%;%DOORSHOME%\bin
@rem SET DOORSDATA=C:\gbmodels\DOORS 51\data
@rem SET RT_NO_OLE=ON
```

   Note: If you do not want to insert the chart as an OLE object in Rational DOORS, do not remove the `@rem` in front of `RT_NO_OLE`.
   Remove the `@rem` from the `STM_DOORS_USE_SYS_LOGIN` line if DOORS is configured to "Use system usernames"

4. After uncommenting the SET commands, specify values for the following parameters:

   - **DXLPORTNO**

     Set to `DXLPORTNO=5093`.

     Typically, 5093 is the default TCP-IP port for Rational DOORS. However in some instances, such as when the database is located on server, the 5093 number may already be assigned. If necessary, check with the network administrator to ensure that 5093 is not assigned.

   - **DXLIPHOST**

     Set this parameter to the hostname of the system on which Rational DOORS is running. If Rational DOORS is running locally on your system, you can set this parameter to `DXLIPHOST=localhost`.

     If Rational DOORS is not available locally, contact your network administrator for the name of the remote system running Rational DOORS.

- **DOORSHOME**

  Set this parameter to the absolute path to the top-level directory where Rational DOORS is available. If, for example, DOORS is on your `C` drive in the `Program Files` directory, define this parameter as follows:

  ```
  SET DOORSHOME=C:/Program Files/Doors
  ```

5. Save and close the file.

Edit doorss.bat

1. Copy `doorss.bat` to `doorss.ORIG.bat`.
2. Using the text editor of your choice, open `doorss.bat` for writing and set the following parameters:
   - When DOORS is configured to "Use system usernames", remove the "-user %2" from the doorss.bat command line.
3. When you are satisfied with your changes, save and close the file.

Edit GetDoorsVer.bat file

The GetDoorsVer.bat is located in the bin directory of the Statemate installation path.
When DOORS is configured to “Use system usernames”, remove the “-user %1 -password %2” from the command line in this file.

Edit run_doors.bat

1. Copy `rundoors.bat` to `rundoors.ORIG.bat`.
2. Using the text editor of your choice, open `rundoors.bat` for writing. You will see lines similar to the following:

```
@rem This file should contain two lines:
@rem First line sets the environment variable LM_LICENSE_FILE
@rem Second line starts the Doors tool, e.g D:\DOORS\BIN\DOORS.
@rem If DOORS is installed in the program files directory,
@rem the path must be in quotes, "D:\PROGRAM FILES\BIN\DOORS".
@rem Modify these two lines to fit your environment
@echo off
SET LM_LICENSE_FILE=<full-path of your doors license file>
<full path command to start doors>
```

3. Make the following changes in the file so that the operating system can find the Rational DOORS license and then start Rational DOORS:

- `SET LM_LICENSE_FILE=<full-path of your DOORS license file>`

  Replace the string `<full-path of your DOORS license file>` with the absolute path of the Rational DOORS license file.

- <full path command to start doors>

  Replace this string with the absolute path to the Rational DOORS executable and enclose the line in double quotes(“ ”). If, for example, the Rational DOORS executable is on the C drive, enter the following line:

  ```
  “C:\Program Files\Doors\bin\DOORS.exe”
  ```

Working with the RT Interface

This section explains how to perform the following tasks in Rational Statemate using the RT Interface:

- Associating a Rational Statemate Project with a Rational DOORS Project
- Setting Preferences
- Exporting Rational Statemate Data to Rational DOORS
- Configuring Filtering by Attribute
- Configuring Linksets for Export
- Creating Multiple Linksets in a Single Link Module
- Re-Exporting Rational Statemate Data to Rational DOORS

Associating a Rational Statemate Project with a Rational DOORS Project

This section explains how to associate a Rational Statemate project with a Rational DOORS project.

To associate a Rational Statemate project with a Rational DOORS project:

1. Ensure that you have a corresponding Rational DOORS project.

   The Rational DOORS project must exist in order to export Rational Statemate elements. For information on how to create a DOORS project, consult your DOORS documentation.

   Additionally, the Rational DOORS Project name must match the RT Project name of the Rational Statemate project. It is case sensitive and cannot contain white space characters, because white space characters are not supported by Rational Statemate.

2. Start Rational Statemate.

3. From the Rational Statemate main window, complete **one** of the following:

   - If you are creating a new project, select **File > New Project**.
   - If the project already exists, select **Project > Project Management > Modify**.

4. Fill in the **RT project:** field with the name of the corresponding DOORS project.

5. Click **OK** to confirm your selections.

Setting Preferences

This section describes how to set defaults within the RT Interface, so that data is saved to common areas.

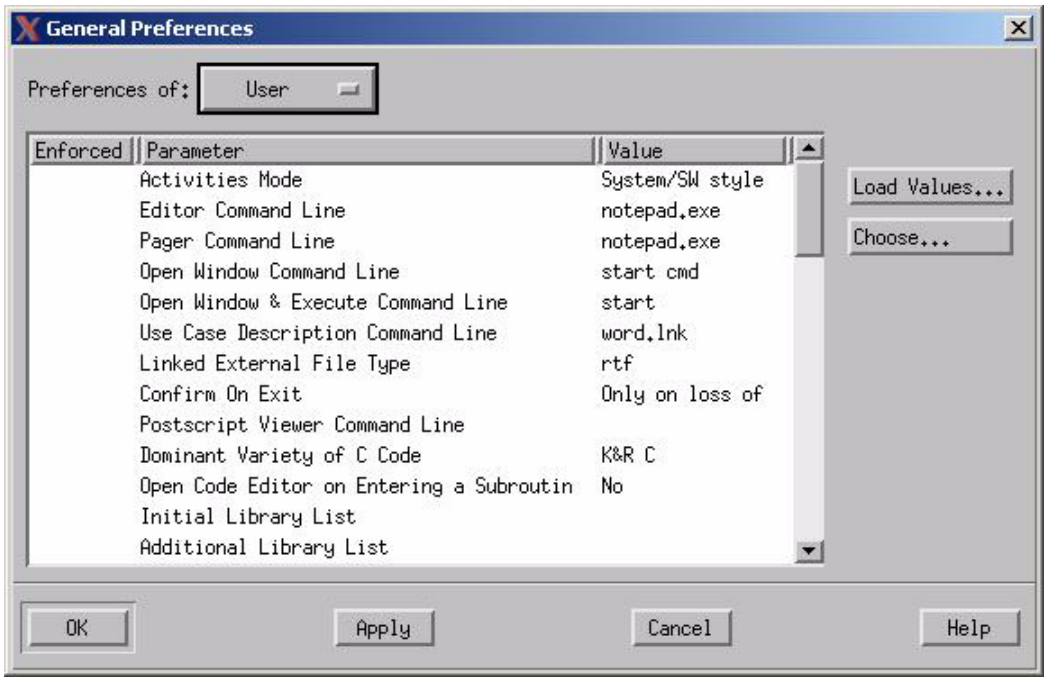
The following sections explain how to perform the following tasks:

- **Setting Preferences for Chart Plots**
- **Setting Preferences for External Use-Case Files**
- **Setting Preferences for Log Files**
- **Setting Up a Default Configuration File**

Setting Preferences for Chart Plots

To set up preferences for chart plots:

1. From the Rational Statemate main window, select **Project** > **General Preferences** > **Preferences of** > **User**. The General Preferences window opens.



2. Highlight the **Default Postscript Device,** click on the text in the **Value** field, and then select **WORD** from the pull-down menu that opens.
3. Select **Apply**.
4. Select **OK** to confirm your choice.
5. Create a directory to hold the chart plots, such as `C:\Doors\DoorsPlots`.

   Note: Do not use any white spaces in the directory name; otherwise, DOORS is not able to resolve the link.

6. Set up an RT Interface to point to the chart plots directory, by doing the following:

   a. From the Rational Statemate main window, click or select **Tools** > **RT Interface** > **DOORS I/F**. The Doors Configuration dialog box opens.

      Note: When you select **DOORS I/F**, a warning splash screen displays, which can be ignored. Click **OK** to delete the splash screen.

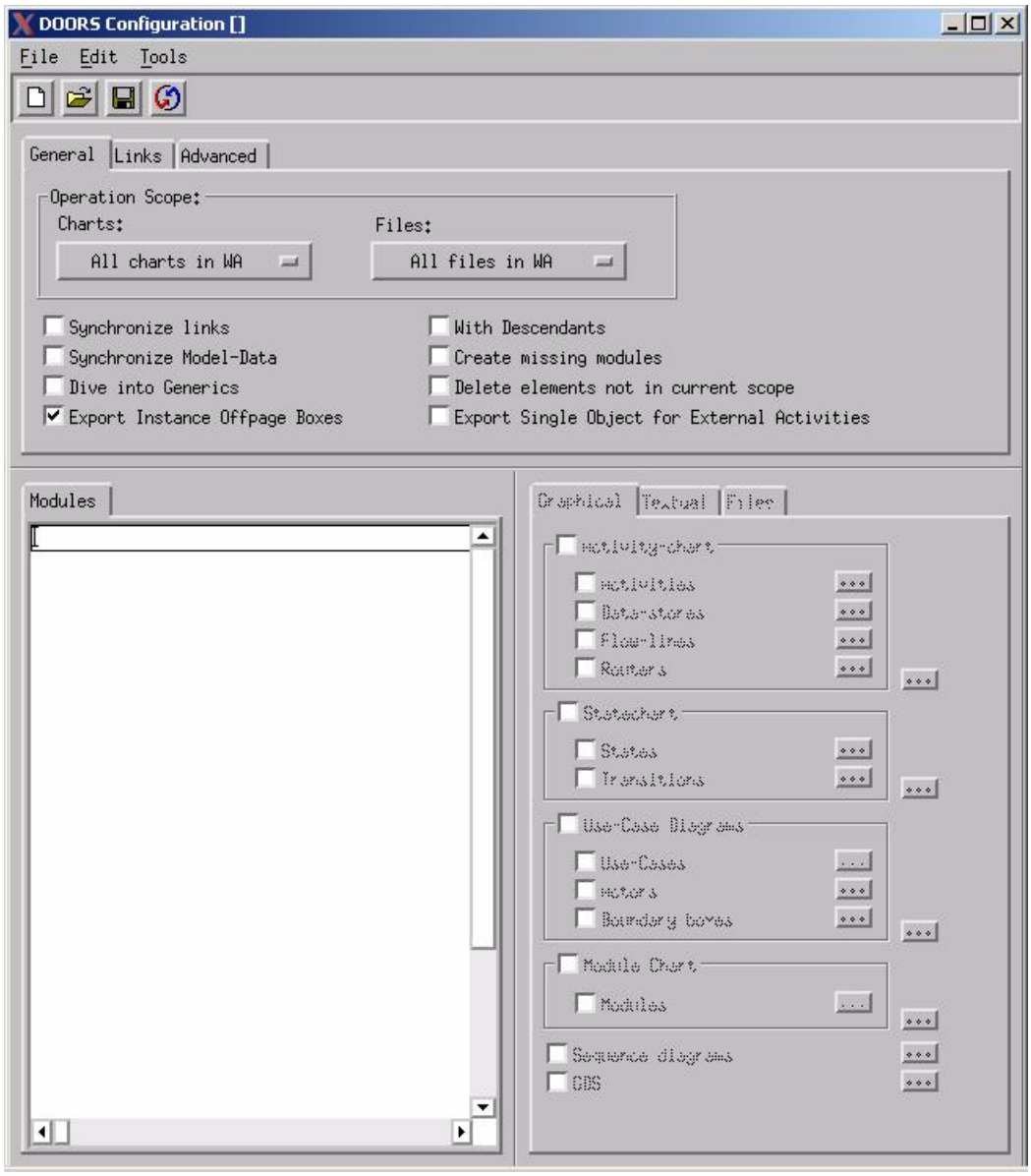
   b. Select the **Advanced** tab.

   c. Click **Export Chart Plot file path.**

   d. Select the tab to the right of the blank path window, which opens an Explorer window.

   e. Navigate to the chart plots directory and click **OK**.

The path to the folder is entered into the blank window.


DOORS Configuration []
File Edit Tools
General Links Advanced
Operation Scope:
Charts:
All charts in WA
Files:
All files in WA
Synchronize links
Synchronize Model-Data
Dive into Generics
Export Instance Offpage Boxes
With Descendants
Create missing modules
Delete elements not in current scope
Export Single Object for External Activities
Modules
Graphical Textual Files
Activity-chart
Activities
Data-stores
Flow-lines
Routers
Statechart
States
Transitions
Use-Case Diagrams
Use-Cases
Actors
Boundary boxes
Module Chart
Modules
Sequence diagrams
ODS

Setting Preferences for External Use-Case Files

You set preferences for external use-case files if you want to link use-case external description files in a DOORS module. This enables you to select a link in DOORS and open the external description file in Microsoft Word.

To set preferences for external use-case files:

1. Create a directory to hold the use-case files, such as `C:\Doors\UseCaseFiles`.

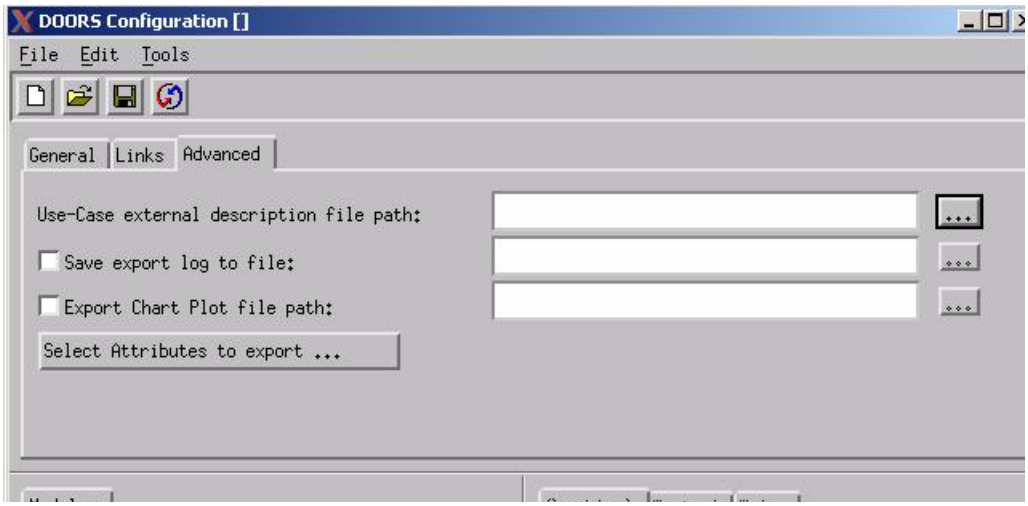
   Note: Do not use any white spaces in the directory name; otherwise, DOORS will not be able to resolve the link.

2. Set up the RT Interface to point to the use-case files directory by doing the following:

   a. From the Rational Statemate main window, click or select **Tools > RT Interface > DOORS I/F**.

   Note: When you select **DOORS I/F**, a warning splash screen displays, which can be ignored. Click **OK** to delete the splash screen.

   b. Select the **Advanced** tab.



   c. Click to the right of the blank path window for the **Use-Case external description file path, which opens an Explorer** window.

   d. Navigate to the case files directory and click **OK**.

The path to the folder is entered into the blank window.

Setting Preferences for Log Files

Setting preferences for log files enables you to specify a name and directory for a log file that contains a list all the elements that were exported to the Rational DOORS module, along with the shadow ID and its status.

To set preferences for a log file:

1. Create a directory to hold the log files, such as `C:\Doors\DoorsLogFiles`.

   Note: Do not use any white spaces in the directory name; otherwise, Rational DOORS will not be able to resolve the link.

2. Set up an RT Interface to point to the log files directory by doing the following:

   a. From the Rational Statemate main window, click or select **Tools** > **RT Interface > DOORS I/F**.

      Note: When you select **DOORS I/F**, a warning splash screen displays, which can be ignored. Click **OK** to delete the splash screen.

   b. Select the **Advanced** tab.

   c. Click **Save export log to file**.

   d. Select the tab to the right of the blank path window for the **Use-Case external description file path**, which opens an Explorer window.

   e. Navigate to the log files directory and click **OK**.

   f. Click **Save**.

The path and name of the file are entered into the blank window.

Setting Up a Default Configuration File

It is recommended that once you have configured the necessary export operations, you capture these settings in a default configuration file. You can use the default configuration file as the basis for later, customized export configurations. Typically a configuration file is saved once an export profile has been created.

To set up a default configuration file:

1. Create a directory to hold the configuration file, for example: `C:\Doors\ConfigFiles`.

   Note: Do not use any white spaces in the directory name; otherwise, Rational DOORS will not be able to resolve the link.

2. Save the current configuration settings that Rational Statemate has saved in its own named file as a new file with a name you determine, by doing the following:

   a. From the Rational Statemate main window, click or select **Tools > RT Interface** > **DOORS I/F**.

      Note: When you select **DOORS I/F**, a warning splash screen displays, which can be ignored. Click **OK** to delete the splash screen.

   b. Select **File** > **Save As**. A browsing window opens.

   c. Browse to the configuration file directory you created in Step 1 and enter a name for the default configuration file, ensuring that it has a

To choose which elements to export, set the **Operation Scope** on the Charts or Files tab to **one** of the following:

- All charts/files in the WorkArea (WA)
- WAB Selection Only

  This setting requires the user to highlight elements in the Charts or Files tab of the Rational Statemate main window to limit the files to be exported.

To define how elements are to be exported, select one of the following buttons:

- **Synchronize Links** - When linkset modules are defined, creates or updates the links between Rational Statemate elements. For more information, see **Configuring Linksets for Export**.
- **Synchronize Model-Data -** forces the Rational DOORS module to update.

  Note: You must set this button when exporting elements to the Rational DOORS module.

- **Dive into Generics -** exports instances of any generic chart, if selected; otherwise the generic chart is seen in the module as a separate element.
- **With Descendants -** exports only particular branches of a tree in the hierarchy of the charts on the Charts tab of the Rational Statemate main window. It is used in conjunction with the WAB Selection Only button.
- **Create Missing Modules -** creates new modules, if the target modules do not yet exist. It is generally used only on the first instance of an export to a module.
- **Delete elements not in current scope -** clears the Rational DOORS module of fully defined textual elements that are outside the current export scope.

  Note: Do not use this button unless you are sure what data elements will be removed from the DOORS module.

- **Export Instance Offpage Boxes** - The DOORS interface option “Export Instance Offpage Boxes” controls the export of offpage instance boxes. By default, that option is set and all offpage instance boxes are exported.

  When that option is not set the following behavior occurs:

  - Offpage instance boxes are not exported to Rational DOORS.
  - Generic instance boxes are exported to Rational DOORS.
  - Statecharts that see a Control Activity with siblings have in their “objectInfo” field the value: “@Controlling Statechart” (instead of the instance).
  - Statecharts that see a Control Activity without siblings have in their “objectInfo” field the value: “@Statechart” (instead of the instance).
  - Flowcharts that see a Control Activity with siblings have in their “objectInfo” field the value: “@Controlling Flowchart.”
  - Flowcharts that see a Control Activity without siblings have in their “objectInfo” field the value: “@Flowchart.”

Configuring Attributes for Export

Attributes are an intrinsic part of systems engineering. They can be used to define many aspects of an element’s non-functional requirements, such as security classification, standards, quality of service requirements.

The RT Interface can be configured to export the element attributes.

All elements exported to a Rational DOORS module have the attributes types exported with them, whether or not an attribute is set in the original Rational Statemate model. If an attribute is not set, the value field will be empty.

The exported attribute values can be inspected in the DOORS module by selecting an element, right-clicking on it to open the Properties window, and finally, selecting the Attributes tab.

To configure Rational Statemate to export the element attributes:

1. Create a set of attributes for elements in the Rational Statemate model. For more information, see Creating a List of Elements.

2. From the Statemate main window, click or select **Tools** > **RT Interface** > **DOORS I/F**.

   Note: When you select **DOORS I/F**, a warning splash screen displays, which can be ignored. Click **OK** to delete the splash screen.

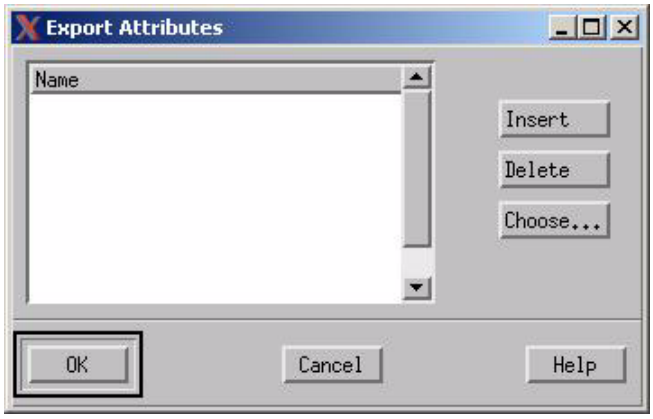
3. Select the **Advanced** tab.

4. Click **Select Attributes to export**.

5. Click **Choose** from the **Export Attributes** dialog box.

6. Select the attributes to export from the **Export Attributes** dialog box by pressing and holding **CTRL** and clicking the attributes.

7. Click **OK** to confirm and close the window. The **Export Attributes** dialog box displays, displaying the list of attributes.



   Note: The list can be modified using the **Delete** and **Insert** buttons or by manually typing in attribute types.

8. Click **OK** to close the window.

Note

To avoid mistakes, do not type in the attribute name because the RT Interface expects the correct case for the attribute type to export the values.

Configuring Filtering by Attribute

Filtering by attribute is extremely useful, if you need to export limited sets of data to a specific module.

For example, if you need to export a module of elements with an attribute type **security classification** and a value of **unclassified**, you can set up a filter in the RT Interface to export any element with that particular attribute value to a specific Rational DOORS module.

To configure Statemate to filter by attributes:

1. From the Statemate main window, click or select **Tools > RT Interface** > **DOORS I/F**.

   Note: When you select **DOORS I/F**, a warning splash screen displays, which can be ignored. Click **OK** to delete the splash screen.

2. Set up the target module. For information on how to perform this task, see **Preparing Rational Statemate Elements for Export to Rational DOORS**.

3. In the Operation Scope area, select **Charts** or **Files**. Then, select **All in WA** or **WAB Selection Only**. If you choose WAB Selection Only, select the elements you want to include in the operation.

4. Select the chart types to export (attribute filters are only active when an element has been selected).

   Note:

   - There are a series of buttons in the Chart Type window, to the right of the element types, with three dots in them. These are the attribute filters.
   - The button furthest to the right relates to attributes associated with a chart.
   - The buttons inside the Chart Type boxes relate to attributes associated with the graphical elements on a chart.

5. Click the **Attribute Filter** button associated with the element you wish to export. The Attribute Filter window opens.

6. Enter the attribute type into the box on the left side of the window (use the exact case of the attribute).

7. Enter the attribute value associated with the attribute type you wish to filter on the right side of the window (use the exact case of the attribute value).

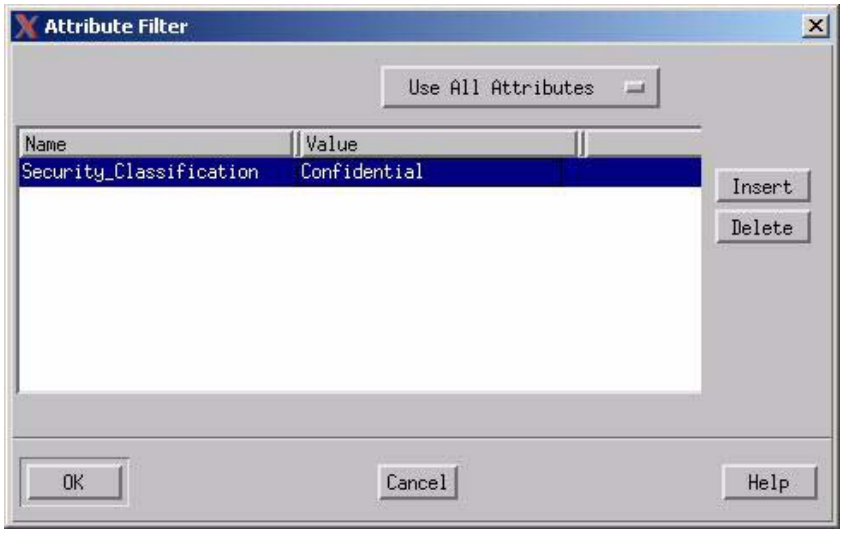
Note: If more than one attribute is to be used as the basis for a filter, the **Use All Attributes** or the **Use Any Attributes** button in the Attribute Filter window is extremely useful.

These buttons do the following:

- **Use All Attributes -** An element to be exported must have all the attributes mentioned in the filter set to the values defined (an AND operation).
- **Use Any Attributes -** An element to be exported can have any of the attributes mentioned in the filter set to the values defined (an OR operation).

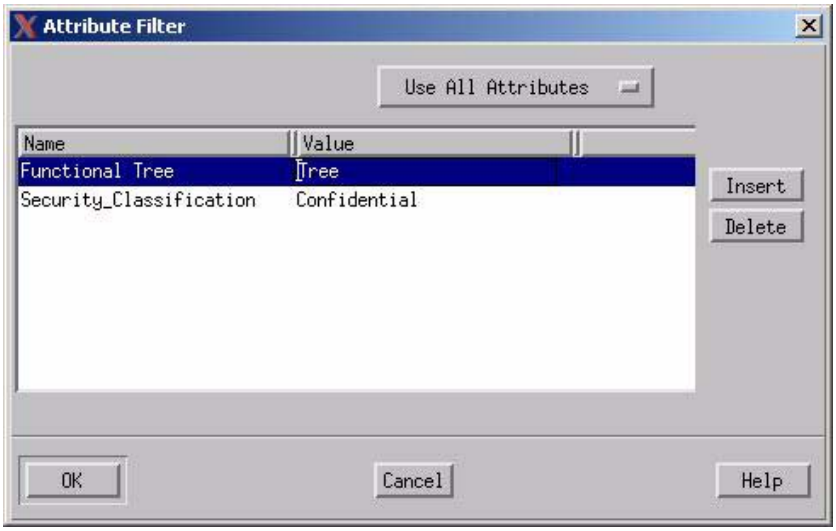
Note: When filtering charts by attribute, it is also worth exporting the relevant attributes.

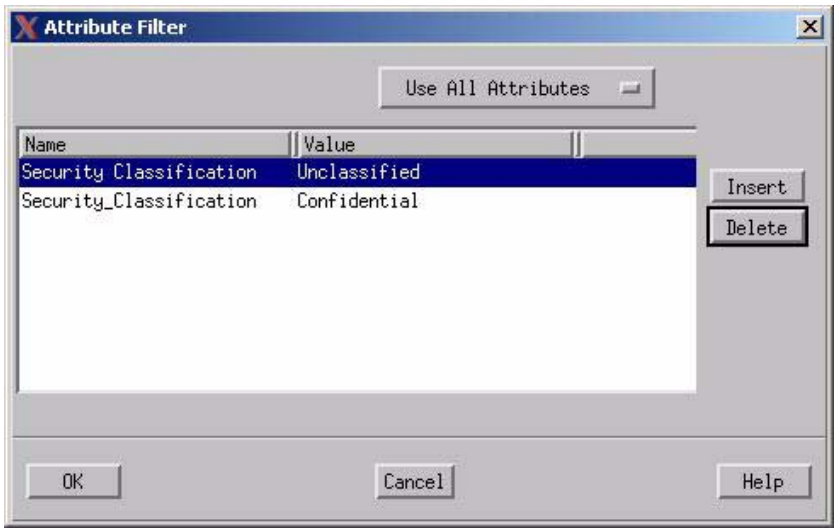
The following figure shows how to use a general filter to export only an activity chart with a **Security Classification** of **Confidential** to the DOORS module `Activity_charts`.



The following figure shows how to use the AND/OR filter to export only activity charts with a **Security Classification** of **Confidential** *and* a **Functional Area** of **Three** to the DOORS module `Activity_charts`.

Note: Because the graphical element `Activities` has been set to export, another filter can be applied to it. In this instance, all the graphical activity elements with either a **Security Classification** of **Confidential** *or* **Top Secret** are exported.





8. Click **OK** to save the settings.

Configuring Linksets for Export

Linksets are used to define the traceability between different elements in DOORS. Typically they define the explicit links between requirements (contained in a requirements module) and model elements (contained in a separate module). The linksets themselves are contained within a links module. More than one linkset can be contained in a single links module.

The RT Interface can be configured to export these links to DOORS. Links are supported between the following:

- Use cases and sequence diagrams.
- Use cases and the statecharts that can be used to define the use-case behavior.
- Sequence diagrams and the activity charts that appear in them as lifelines.
- Definitions of data elements and the sequence diagrams where they are used as messages.
- Use cases and their scenarios.

Note: When exporting links, you must also export the relevant diagrams to their respective modules.

To configure a linkset for export:

Note

For the sake of brevity, this book explains the steps for configuring a linkset between a use case and a sequence diagram. The process is almost identical for other linksets, apart from the selection of the radio buttons controlling which linkset to export and the selection of the relevant modules to create each linkset type.

1. From the Statemate main window, click or select **Tools > RT Interface** > **DOORS I/F**.

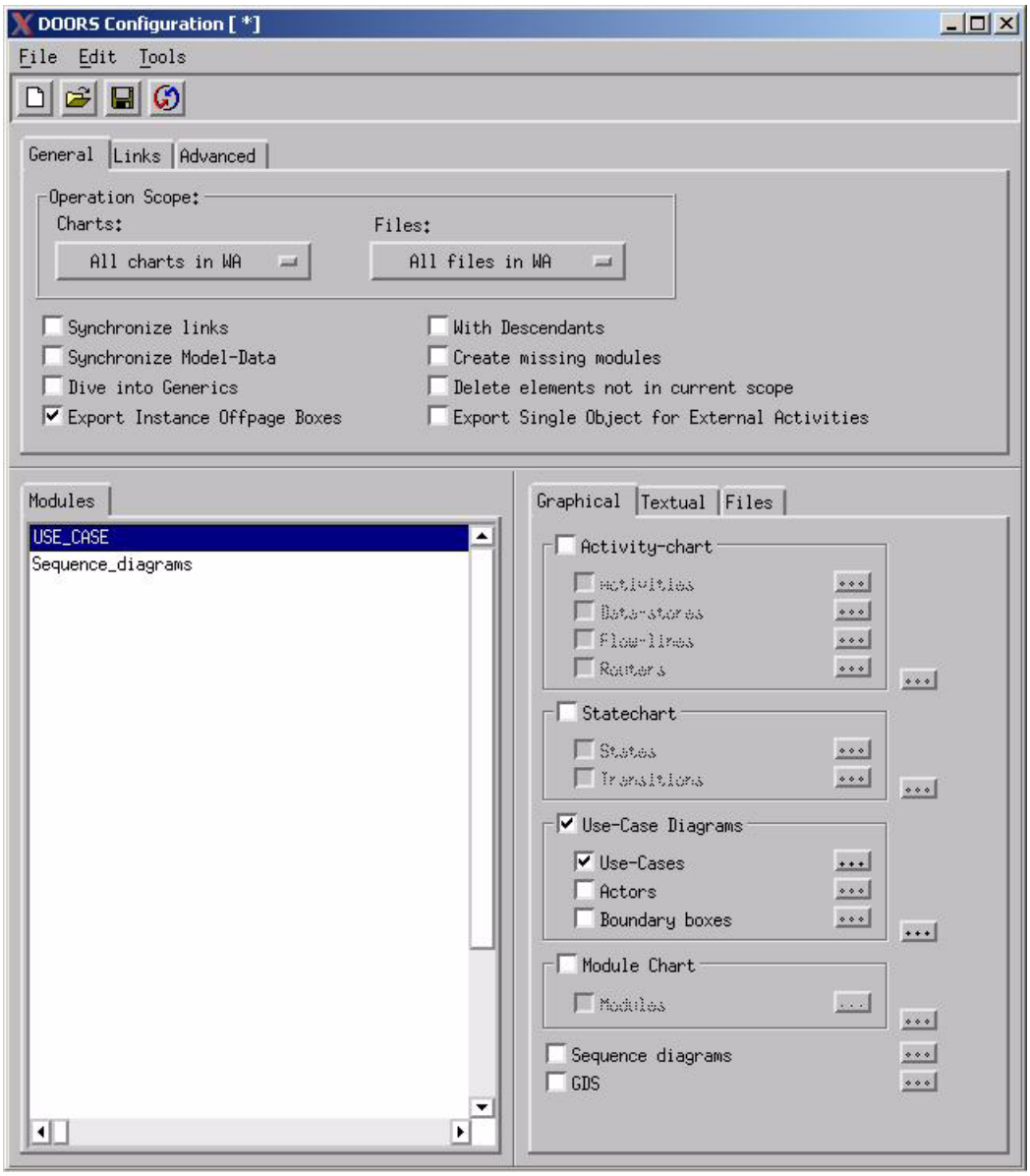
   Note: When you select **DOORS I/F**, a warning splash screen displays, which can be ignored. Click **OK** to delete the splash screen.

2. Create a module named `Use_Cases` in the Modules window.
3. Select **Use-Case Diagrams** from the Graphics tab.
4. Select **Use Cases** element from the Graphics tab.
5. Create a module named `Sequence_diagrams` in the Modules window.
6. Select **Sequence Diagram** from the Graphics tab.
7. Set the **Operation scope** to **All charts in WA**.
8. Set **Synchronize model data** to be ON.

9. Set **Synchronize links** to be ON (creates the linksets automatically).

10. Set **Create missing modules** to be ON (creates the links module as well as the data modules).

    The Rational DOORS configuration window should look similar to the following figure:



11. Set up the linkset module, define the type of links to be created, and source the target modules by following these steps:

a. Select the **Links** tab.

b. Right-click in the Link Modules window.

c. Add a new module named `Link_Module`.

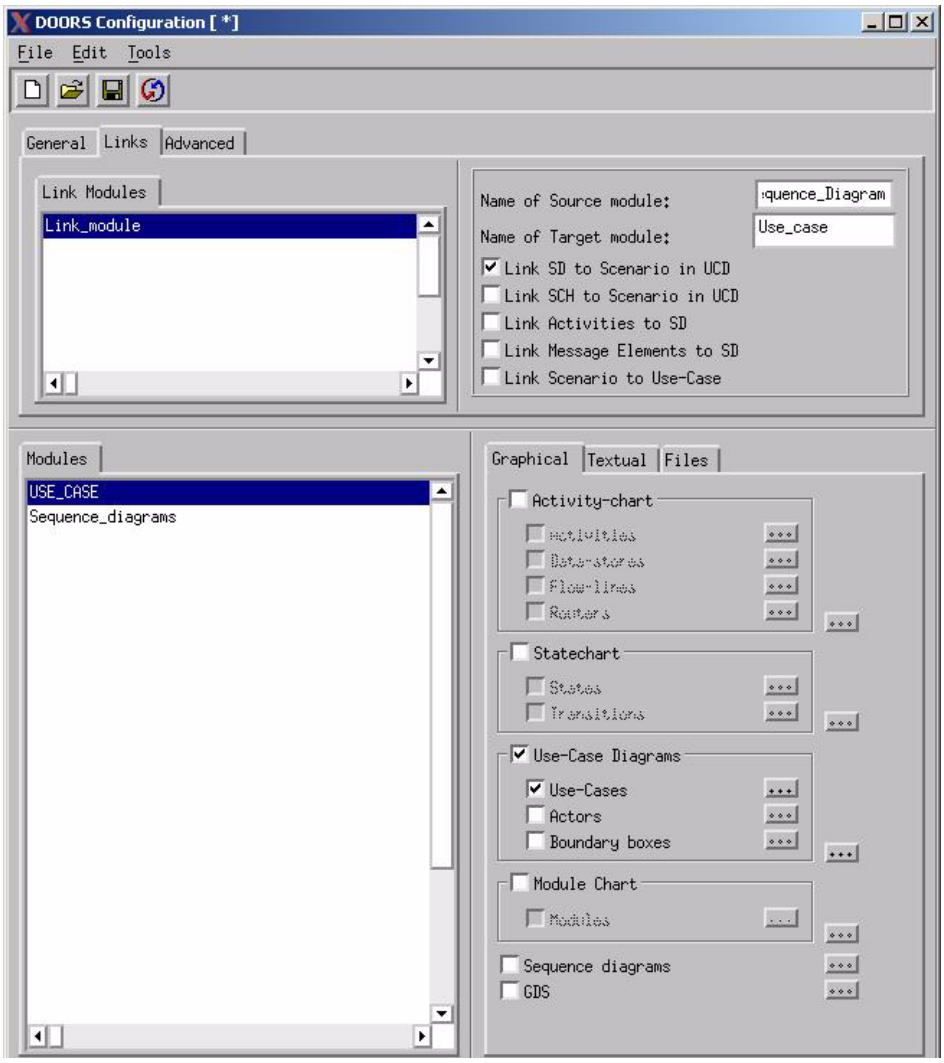
d. Select the radio button for **Link SD to Scenario in UCD**.

e. Enter the name of the **Source module** (for example, `Sequence_Diagrams`).

f. Enter the name of the **Target module** (for example, `Use_Cases`).

Note: Ensure that the case of the module names is correct.

The DOORS configuration window should look similar to the following figure:



12. Once the configuration is complete, you can export the linksets. For information on exporting to DOORS, see **Exporting Rational Statemate Data to Rational DOORS**.

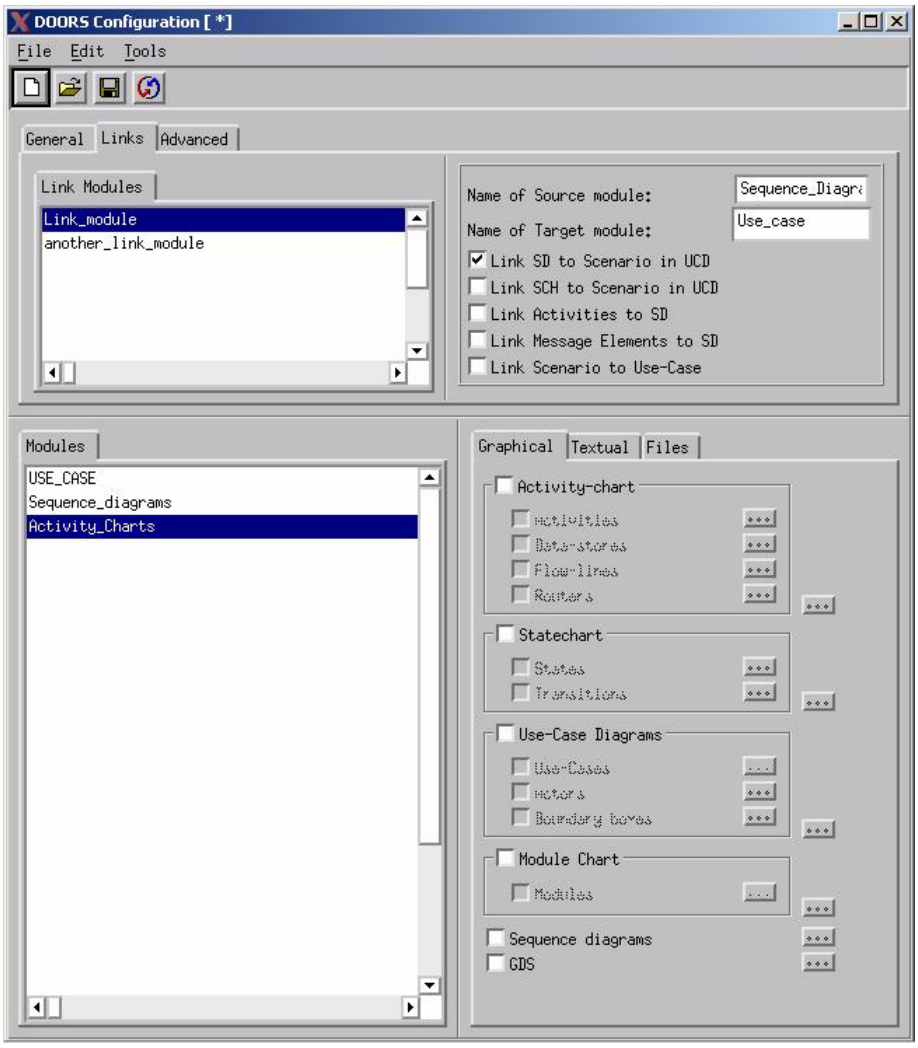
Note: You can check the relevant DOORS module by going into the sequence-diagram module, then selecting a sequence diagram.

A red right-hand arrow displays. Right-click on the arrow, and the links to the use case diagram are shown as an unloaded object.

Select an **unloaded object** and the relevant use case diagram module is opened at the correct use case. The reverse link is seen as a yellow left-hand arrow.

Creating Multiple Linksets in a Single Link Module

It is possible – and relatively easy – to create multiple linksets in one link module. You use the same module name in the link modules window, but you select a different linkset with the relevant source and target modules, as illustrated in the following figure:



Exporting

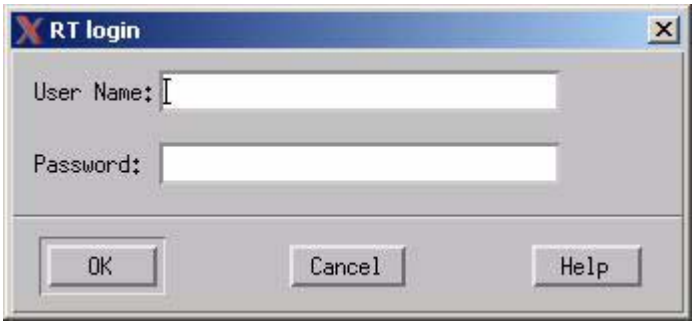
To export Statemate elements to Rational DOORS:

1. Ensure that you have completed all of the necessary preliminary work, as outlined in **Setting Preferences** and **Preparing Rational Statemate Elements for Export to Rational DOORS**.

2. If you are exporting elements to an existing Rational DOORS module, ensure that the DOORS module is closed.

3. Select **Tools > Synchronize data with DOORS** or click on the box to right of the **Save** icon in the Rational DOORS Configuration window.

   The RT Login window opens, as shown in the following figure.



4. To start the export, enter your DOORS **User name** and **Password,** then click **OK.**

Re-Exporting Rational Statemate Data to Rational DOORS

In most cases, you need to re-export data at regular intervals as the data changes and model baselines are set and reset. There are two ways to re-export data to Rational DOORS:

- Use a saved configuration file.
- Reset the profile of elements and modules to be exported manually.

Using a Saved Configuration File

To use a saved configuration file:

1. Access a saved configuration file (see **Accessing a Saved Configuration File**.)
2. Synchronize the data (see **Synchronizing the Data**.)

Accessing a Saved Configuration File

To access a saved configuration file:

1. From the Statemate main window, click or select **Tools** > **RT Interface**> **DOORS I/F**.

   Note: When you select **DOORS I/F**, a warning splash screen displays. Click **OK** to delete the splash screen.

2. Select **File** > **Save As**. A browsing window displays.
3. Browse to the configuration file directory created in **Setting Preferences for External Use-Case Files**.
4. Select a configuration file and click **Open**. The buttons in the **RT Interface** are set up.

   Note: Once a configuration is retrieved, select the chart scope to be exported from the workarea browser.

   Configuration files do not save the scope of elements to be exported because the files are available to all users who have access to the folder. The files can be applied to any project.

Rational DOORS Interface Support for Transitions

The Rational DOORS interface supports the inclusion of the following information for Transitions:

- Attributes
- Design-Attributes
- Long Description

The “Object CreationStamp” attribute helps identify the Statemate element corresponding to a Rational DOORS object. This attribute has the following characteristics:

- The “Object CreationStamp” attribute is part of all elements in the modules, but it is updated only when the object is modified and thus updated.
- Any existing Rational DOORS objects (created before Statemate 4.3) in the modules have an empty value for the “Object CreationStamp” attribute.
- The attribute is available only for elements that were created in Statemate 4.2 and later. Elements that were created with Statemate before version 4.2, have “0” as their attribute values.
- This attribute has the value “0” for scenarios.

Synchronizing the Data

To synchronize the data:

1. To begin synchronizing, do **one** of the following:
   - Select **Tools** > **Synchronize data with DOORS**.
   - Click **Save** in the **DOORS Configuration** dialog box.

   The **RT Login** dialog box displays.
2. Enter your Rational DOORS **User name** and **Password** and click **OK.** The **RT Interface Manager** dialog box displays and data starts to be resysnchronized.

   If an exported data element has changed, the RT Interface overwrites the original version of the data in the Rational DOORS module. If an element has been deleted from the scope of the data to be exported, a **Question** dialog box displays, giving you the option of deleting the DOORS shadow element.

The following options are available:

- **Yes** - deletes the Rational DOORS shadow element.
- **Yes to All** - deletes all the Rational DOORS shadow elements**.**

Note: Select this option to prevent this question from being asked for every shadow element.

- **No** or **No to All** - keeps all versions of the data in the DOORS shadow.
- **Rename** – renames the chart shadow and its direct descendants in DOORS. Use this option when the chart was renamed in Statemate after it was exported.
- **Abort Export –** aborts the current export of data, stopping the RT Interface manager.

  In order to keep the Statemate data synchronized against its Rational DOORS Object, the Rational DOORS Object includes the attributes “Object CreationStamp.” The value of this attribute is a number representing the time in which the corresponding element was created in the Statemate model.

Truth Tables

This section provides information on using truth tables in Statemate. The topics are as follows:

- **Executing Truth Tables**
- **Defining a Truth Table**

A *truth table* is a tabular representation of inputs, resulting outputs, and actions. Each table can contain one or more input or output columns and one action column. In Statemate, you can use truth tables in the body of actions and activities and as an additional language to describe procedures.

The following sections describe the format and contents of truth tables. The topics are as follows:

- **Special Characters**
- **Input Columns**
- **Output Columns**
- **Action Column**

Format and Content of Truth Tables

This section contains the following information:

- Special Characters
- Input Columns
- Output Columns
- Action Column

Special Characters

The following table lists the characters that have special meanings in truth tables.

Character	Meaning
*	Don't care
+	Event generated (input or output)
-	Event not generated (input)

Input Columns

The input columns of a truth table are similar to the following:

CO_1	CO_2	DI_1	REC_1	ARR_1
True	False	1	REC_2	{1,2,3}
False	False	2	*	*
True	False	3	*	*
False	True	5	*	*

Each column in the input section of the table is associated with an input. An input can be either a Statemate element or expression. Subroutine parameters and globals can be used as inputs when the truth table is a subroutine implementation body.

Compound elements can be used as inputs. For example, `CO_2` can be defined as `D1>5` and `in (STATE_1)`.

Entries in the input section can be:

- Literals
- Statemate elements
- Expressions
- Empty
- Don’t care (*)

Each input section of a row represents a Boolean expression. The Boolean expresses an AND of the values for each of the inputs that does not have a “Don’t Care” value.

Note

Input cells that are left blank are considered as “Don’t Care” items by the simulation and code generation tools.

For example:

```
Row 1

CO_1 and not CO_2 and DI_1==1 and REC_1==REC_2 and
ARR_1=={1,2,3}

Row 2

not CO_1 and not CO_2 and DATA_1==2
```

Valid Input Elements

Inputs to truth tables can be conditions or data items. Data items include:

- Integers
- Reals
- Bits
- Bit-arrays
- Strings
- Records
- Record fields
- Enumerated types
- Arrays of the previously listed types
- Elements of arrays
- Subroutine calls
- User-defined types built of the previously listed types

Note

There is no literal syntax for the following types: records, unions, and arrays of complex types. The only legal comparison in the input section for these elements is another element of the same type.

Input Column Header Operators

A value in a truth-table input column header can be prefixed with one of the following operators: <, >, <=, >=, !=, \=, == . A column cell is evaluated by comparing its value with the column header value using the operator. The default operator is ==.

For example, a value of <X in an input column header causes a column cell with value Y to be evaluated as TRUE only when Y<X.

Invalid Input Elements

The following elements *cannot* be used as inputs:

- Unions
- Records that contain unions
- Arrays of unions
- Fields of unions
- Slices of arrays or bit-arrays
- Queues
- States
- Activities

Output Columns

The output columns of a truth table are similar to the following:

CO_3	DATA_2
True	100
False	-1
True	1
False	2

Each output column must be a Statemate element. Local elements, subroutine parameters, and subroutine global elements can be outputs when the truth table is a subroutine implementation body.

Entries in the cells of the output section can be:

- Literals
- Statemate elements
- Statemate expressions
- Empty

Empty entries in the output section indicate outputs that are not changed when the related row is executed. Unchanged items are not “written.”

Output Elements

Primitive conditions and data items can be used as outputs for truth tables.The following elements *cannot* be used as outputs:

- Compounds
- Slices of arrays
- Slices of bit-arrays
- Queues
- Activities
- States
- Actions

Note

The same element can appear in the table as both an input and an output.

Action Column

In the action column, you can include any action expression that is legal in the context of the truth table.

The action column is similar to the following:

Action
AN1;AN2
AN3
X:=X+Y

Executing Truth Tables

The following sections describe how truth tables are executed. The topics are as follows:

- **Default Row**
- **Row Execution**
- **Truth Table Contents for Activities and Actions**
- **Truth Table Contents for Subroutines**
- **Micro-step Execution of Procedure Truth Tables**
- **Execution of Action Truth Tables**
- **Factorization of Cells**

Default Row

Optionally, you can add a default row to the truth table. This row contains no input values and is executed only if none of the previous rows in the table have been executed.

Row Execution

Statemate evaluates a truth table as follows:

- When a truth table is executed, Statemate evaluates it row-by-row, starting at the top of the table and proceeding downward to the end.
- The first row whose input expression evaluates to TRUE is “fired.”
- Once the row is fired, all the outputs listed in the output section of that row are generated and the action section is executed.
- If any output columns are blank, the related outputs are not changed. Unchanged items are not “written.”
- The order of execution is from left to right, first outputs and then actions. This is relevant only for truth tables that implement procedures.
- If the table contains a default row, and if during the evaluation of the table no other row has fired, the default row is fired.
- If the table does not contain a default row and no row fires during the evaluation of the table, a warning message is displayed during simulation and no output elements are changed.

Truth Table Contents for Activities and Actions

Truth tables associated with actions or activities can include any legal Statemate action expressions, including, for example:

- References to named actions
- Assignments
- Generation of events
- Operations on activities (within Statecharts)

Note

The semantics of the action section in this context is the “regular” Statemate semantics. A race condition occurs when the same element is assigned both from the output section and the action section.

Truth Table Contents for Subroutines

Truth tables defined as subroutines can include any Statemate action expression that is legal in a subroutine body. They cannot contain references to named actions or other actions, such as scheduled actions or actions on activities and events. They can contain references to local elements, subroutine parameters and globals.

The semantics of the action section in this context is the subroutine action language semantics, that is, all assignments are done immediately. Because an element can be assigned more than once in the output and action sections, order of execution is from left to right to avoid race conditions.

Micro-step Execution of Procedure Truth Tables

Assignments are made within truth tables following the micro-step and immediate update semantics of all functions and procedures.

This means that as soon as an assignment is made, it is available to be used. This does not affect the evaluation of the rows, because only one row fires each time the table is executed. It does, however, affect assigned values, if an output refers to another output that has already been assigned.

In the following example, both **DATA_2** and **DATA_3** receive the value “5” when the row fires, regardless of the previous value of **DATA_2**.

Inputs			**Outputs**	
CO_1	**CO_2**	**DATA_1**	**DATA_2**	**DATA_3**
*	*	*	5	DATA_2

Execution of Action Truth Tables

The following applies to truth tables that are either

- Bound to activities
- Defined as action bodies

Assignments are made within truth tables following the Statemate step semantics. New values are sensed only at the next step. Writing twice to the same element flags a write/write race error. This means that after an assignment is made, it is not available for immediate use.

In the following example, **DATA_2** receives the value 5 and **DATA_3** receives the previous value of **DATA_2**.

Inputs			**Outputs**	
CO_1	**CO_2**	**DATA_1**	**DATA_2**	**DATA_3**
*	*	*	5	DATA_2

Factorization of Cells

You can group vertically adjacent cells together and have the same value applied to the entire group. This grouping, called factorization, can be applied to all three columns in the truth table (inputs, outputs, and actions).

Factorizing Inputs

While factorization of inputs is a labor saving device, it also affects the logic of the table and how it is implemented in code. The table is evaluated from top to bottom, and from left to right. The generated code, as well as the simulator, matches this behavior.

Here is an example of factorization:

Inputs					**Outputs**	
CO_1	CO_2	DL_1	REC_1	ARR_1	CON_3	DATA_2
True	True	1	REC_2	{1,2,3}	True	100
		2	*	*	True	-1
	False	3	*	*	True	1
		5	*	*	True	2
False	*	*	*	*	False	0

This is the resulting logic, shown as pseudo-code:

```
if CO_1 then
if CO_2 then
if DI_1=1 and REC_1=REC_2 and ARR_1 = {1,2,3} then
tr!(CON_3); DATA_2:=100;
else
if DI_1=2 then
tr!(CON_3); DATA_2:=-1;
else
if DI_1=3 then
tr!(CON_3); DATA_2:=1;
else
if DI_1=5 then
tr!(CON_3); DATA_2:=2;
else
fs!(CON_3); DATA_2:=0;
```

Note that factorization of inputs is allowed from left to right only. Reading from left to right, each subsequent factorization of inputs must be a subset of all those to the left, as this example illustrates.

The next two examples show incorrect implementations of factorization to further illustrate the points explained above.

Incorrect factorization - Example 1:

Inputs		Outputs	
CO_1	**CO_2**	**CON_3**	**DATA_2**
True	True	True	100
True		True	-1

Note that in this example, input column 2 is a subset of input column 1, and this is not allowed.

Incorrect factorization - Example 2:

Inputs		Outputs	
CO_1	CO_2	CON_3	DATA_2
True	1	True	100
	2	False	-1
False		True	1
	3	False	2

Note that in this example, column 2 was not built so that each factorization is a subset of all those to the left of it.

Factorizing Outputs and Actions

You can also factorize output and action rows to repeat the same pattern. In contrast to factorization of input rows, this does not affect the code generated from the truth table, and is a labor-saving device only. Here is an example:

Inputs	Outputs			
DATA_1	CON_3	DATA_2	DATA_3	DATA_4
True	True	0	1	DATA_5
				DATA_6
False				DATA_7
				DATA_9
				DATA_10
				DATA_11

Defining a Truth Table

In Statemate, you can define truth tables in the Properties window for the following elements:

- **Action**

  Any work done as a result of:

  - Making a transition in a statechart.
  - Executing a static reaction within a state.
  - Executing a mini-spec within an activity.

- **A single action can consist of the following**:
  - Making an assignment
  - Generating an event
  - Invoking a defined (named) action
  - Several special types of expressions (starting/stopping/ suspending activities, clearing history, and so on.)

- **Activity**

  The primary graphical object in activity charts that represent a function in the functional view of the system. An activity represents something that transforms inputs into outputs.

  - There are three types of activities:
    - Internal activities (solid rectangle)
    - External activities (dashed rectangle)
    - Control activities (rounded rectangle)
  - Activities can be allocated to modules (structure) and can contain statecharts. You can specify the behavior of an activity by connecting it to a subroutine.
    - Procedure-like activities can be connected to procedures within any of the languages supported.
    - Internal primitive activities (reactive-controlled and reactive-self) can be connected to tasks (no mini-specs or decomposition is allowed).
    - External activities can only be connected to tasks.

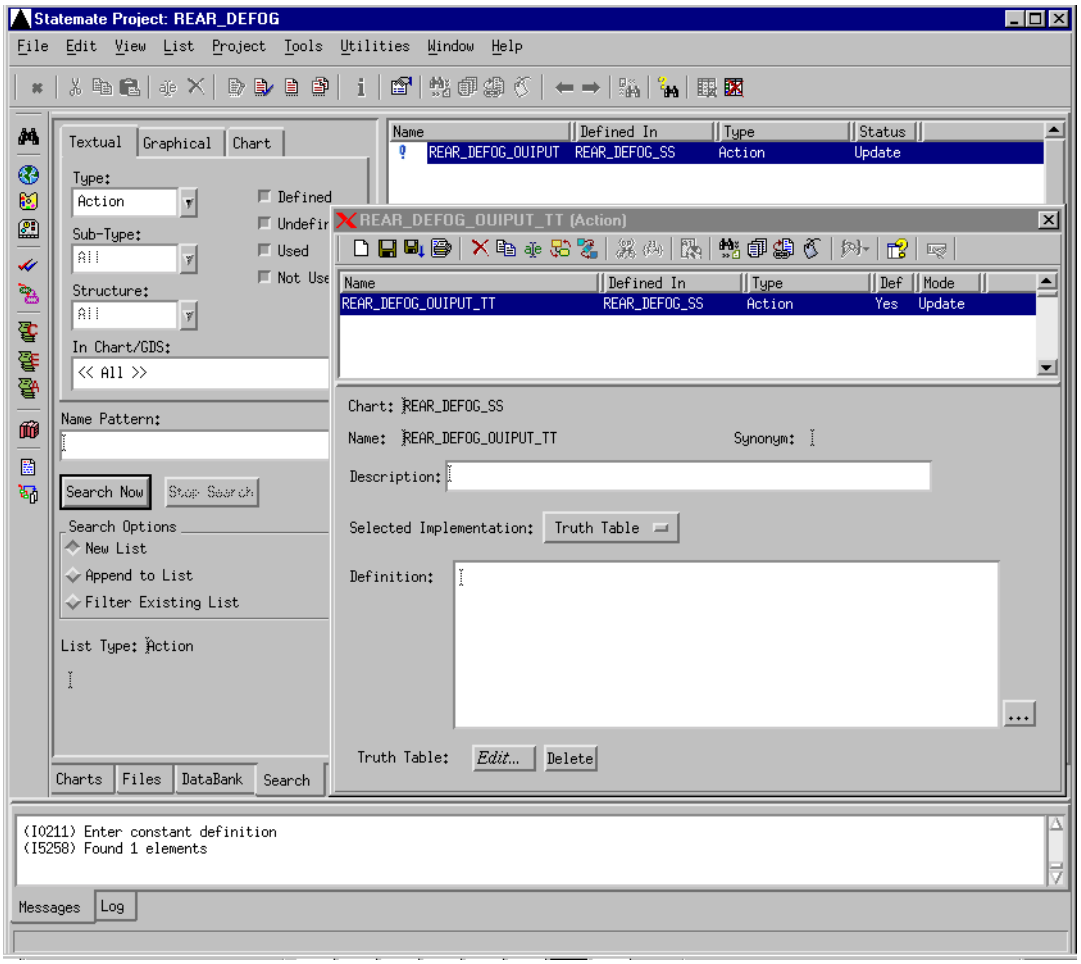
- **Procedure**

  A subroutine that has no return value but can have multiple parameters. Each parameter can be INPUT, OUTPUT, or INPUT/OUTPUT.

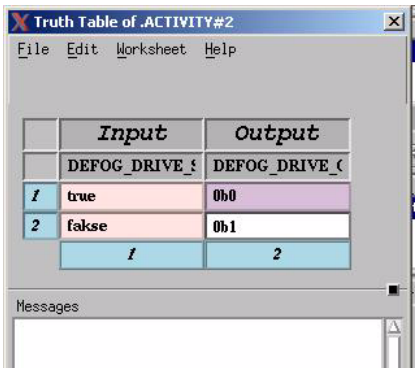
To define a truth table:

1. From the **Search** window or a chart, access the Property window for the element for which you want to define a truth table.

   The following figure shows the Property window for an action element, accessed from the Search tab.



2. Select **Truth Table** from the **Selected Implementation** pull-down menu.
3. Click **Edit** to start the truth-table editor, as shown in the following figure.



4. Using the **Worksheet** option in the toolbar, you can perform the following tasks:
   - **Insert Row** - adds one row above the selected row.
   - **Insert Column** - adds one column to the left of the selected column.
   - **Redefine Table** - displays the Redefine Table dialog box. Allows you to change the number of inputs or outputs or the number of rows, or adds or removes the action section.
   - **Remove Selected** - removes the selected rows or columns.
   - **Add Default Row** - adds a default row. (Because a truth table can only include one default row, this option is not available if the table already has a default row.) Note that the Input columns in the default row are read-only, but you can edit both the output and action columns.
   - **Join Cells** - joins two adjacent cells within the same column.
   - **Split Cells** - splits a previously united cell back into two separate cells.
   - **Local Variables** - This option is available only for truth tables attached to subroutines. When you select this option, a dialog box opens. In this dialog box, you can define the name and data type for the local variables.
5. Select **File > Save**, then **File > Exit** to confirm your selections and exit the editor.

Lookup Tables

This section provides information on using a lookup table as an implementation of a subroutine in Statemate. The topics are as follows:

- Defining a Lookup Table
- Example of a Lookup Table

Lookup tables support non-linear “Y=F(X)” functions that are so common in the world of micros. Typically, these non-linear functions are used to represent characteristic curves of valves in a table structure. Such a table may consist of a list of pairs of digitizing points, Xi, and its corresponding value, Fi. The data might be imported from any ASCII data file. A choice is given whether to perform (linear) interpolation between points, or to use a histogram like mode. In addition, saturation values might be defined, for the upper and lower range bounds, as well as a search order to support performance sensitive scenarios.

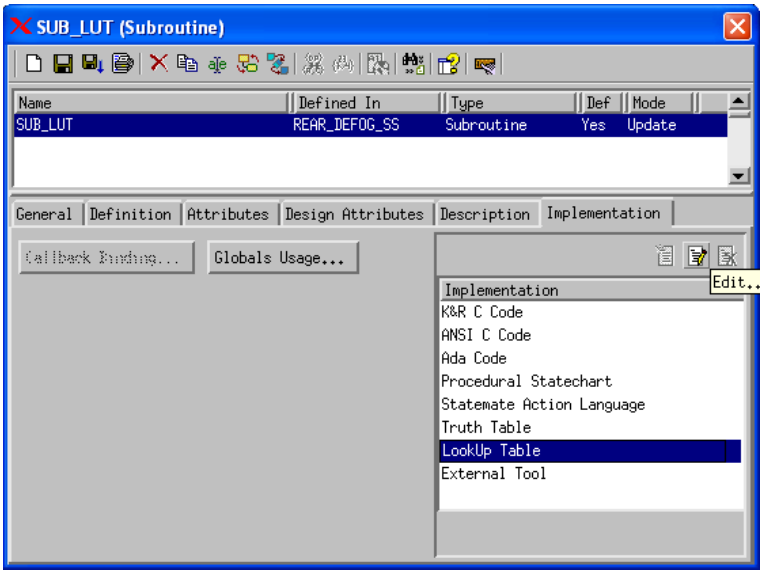
Defining a Lookup Table

To define a lookup table:

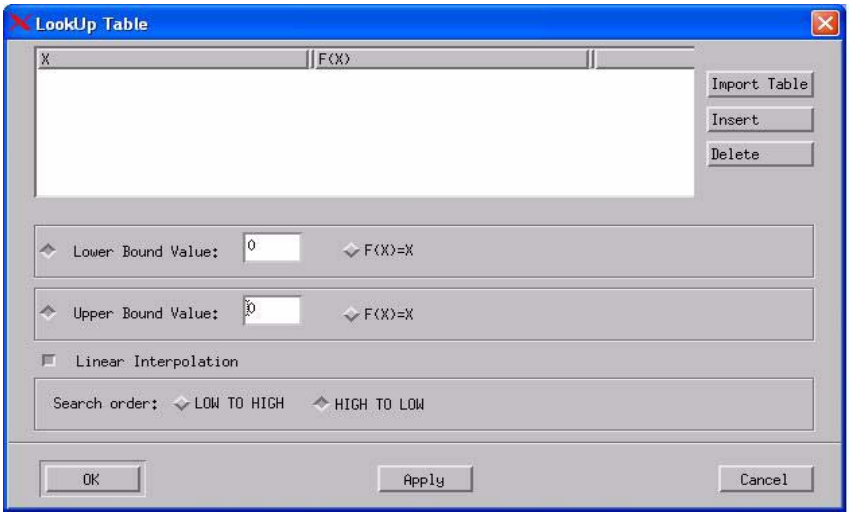
1. From the Search window or a chart, access the Property window for the element for which you want to define a lookup table.

   The following figure shows the Property window for an action element, accessed from the Search tab.

2. Select **Lookup Table** from the **Implementation** tab.



3. Click **Edit** to start the lookup table editor, as shown in the following figure.



4. Fill in the Lookup Table values
5. Fill in the Lookup Table parameters according to the following table:

Option	Description
Lower Bound Value	The lowest possible interpolated value.
Upper Bound Value	The highest possible interpolated value.
Linear Interpolation	Defines if interpolation is linear or not
Search order	Direction that the lookup table is evaluated.

Note

The independent variable X can be either a number or the value of an expression. When X is the value of an expression, its values must be non-decreasing from the beginning to the end of the table.

Example of a Lookup Table

For example, consider the following definition of such a function with return value defined to be “Real” and input defined to be “Integer”: In “Interpolation”, High to Low mode, Lower Bound=0, Upper Bound =4.

X	F(X)
1	1
10	2
100	3
1000	4

Example Components

This section provides reference information on a set of example components. The topics are as follows:

- Overview
- Example Component Library

Overview

Libraries and components offer a means to speed up the design process and to help you create more consistent specifications:

- A library is a container for model components.
- A component is a model element that contains behavior and uses input and output parameters to communicate information.

You can add a component to a model by selecting it from a library and placing it into an activity chart. For more information on working with libraries and components, see Libraries and Components.

The example component library contains a set of components you can use as building blocks for basic system architecture performance modeling and analysis. You can download the component library from the product Web site. The download file includes instructions for setting up the component library on your system.

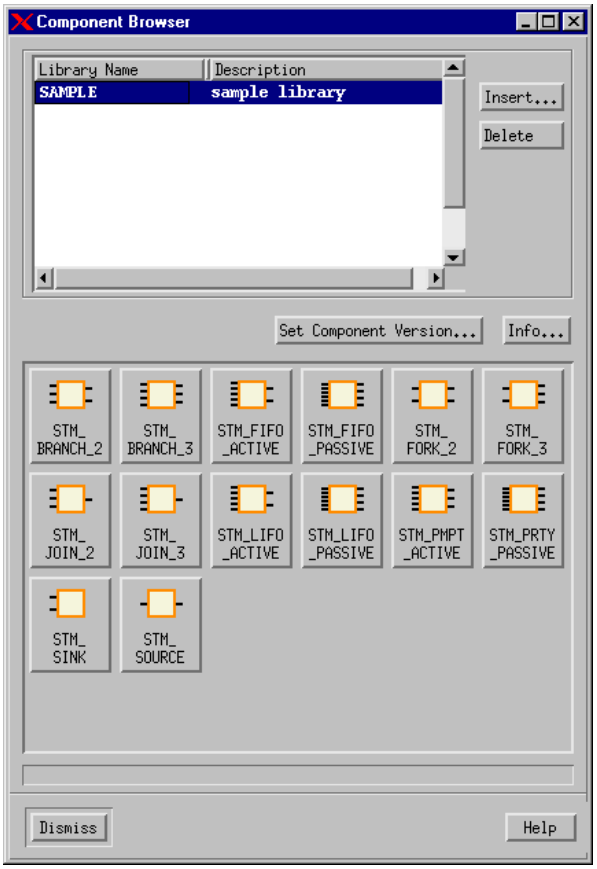
The component library is provided as a learning aid, and is not a supported product. However, it can be freely used, modified, and distributed as needed.

Note

The example component library include subroutines with C code implementations, so a C compiler must be operational on your computer in order for these components to be used in a simulation.

Example Component Library

Each component in a library is represented by an icon that you can drag and drop into an activity chart. The icons also show the number of inputs and outputs for each component. For example, in the following figure, the `STM_BRANCH_2` icon shows three inputs and two outputs.



For more information on working with libraries and components, see **Libraries and Components**.

The following section contains reference information for these components:

- **STM_BRANCH_2**
- **STM_BRANCH_3**
- **STM_FORK_2**
- **STM_FORK_3**
- **STM_JOIN_2**
- **STM_JOIN_3**
- **STM_FIFO_ACTIVE**
- **STM_FIFO_PASSIVE**
- **STM_LIFO_ACTIVE**
- **STM_LIFO_PASSIVE**
- **STM_PMPT_ACTIVE**
- **STM_PRTY_PASSIVE**
- **STM_SINK**
- **STM_SOURCE**

STM_BRANCH_2

Branches transactions to one of two outputs.

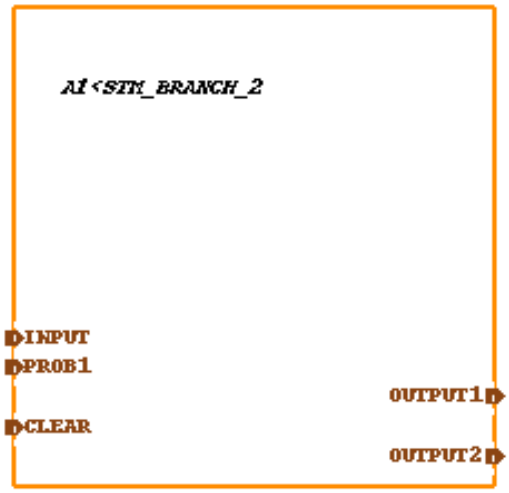
Description

Generates `STMM_TRANSACTION_TYPE` transactions at the OUTPUT1 or OUTPUT2 port. When an INPUT arrives, the component generates a uniform real number between 0.0 and 100.0.

If the number is:

- Less than or equal to `PROB1`, `OUTPUT1` is set to `INPUT`.
- Greater than `PROB1`, `OUTPUT2` is set to `INPUT`.

When you insert the `STM_BRANCH_2` component into an activity chart, the component is represented by the following graphic image.



I/O Stubs

Name	Mode	Format
INPUT	input	STMM_TRANSACTION_TYPE
PROB1	input	Real
CLEAR	input	Event
OUTPUT1	output	STMM_TRANSACTION_TYPE
OUTPUT2	output	STMM_TRANSACTION_TYPE

Analysis Statistics

Formal Parameter	Type
ANALYZE	Condition

If you set **ANALYZE**, the following statistics are gathered:

Variable	Data Type	Description
Local variables for analysis		
OUTPUT1_INTERVAL	Real array	OUTPUT1 arrival interval
OUTPUT1_INTERVAL_MIN	Real array	
OUTPUT1_INTERVAL_MAX	Real array	
OUTPUT1_INTERVAL_MEAN	Real array	
OUTPUT2_INTERVAL	Real array	OUTPUT2 arrival interval
OUTPUT2_INTERVAL_MIN	Real array	
OUTPUT2_INTERVAL_MAX	Real array	
OUTPUT2_INTERVAL_MEAN	Real array	
TRANSACTIONS_OUTPUT1	Integer array	Number of OUTPUT1 transactions

Variable	Data Type	Description
TRANSACTIONS_OUTPUT2	Integer array	Number of OUTPUT2 transactions
Totals for analysis		
T_OUTPUT1_INTERVAL	Real	
T_OUTPUT1_INTERVAL_MIN	Real	
T_OUTPUT1_INTERVAL_MAX	Real	
T_OUTPUT1_INTERVAL_MEAN	Real	
T_OUTPUT2_INTERVAL	Real	
T_OUTPUT2_INTERVAL_MIN	Real	
T_OUTPUT2_INTERVAL_MAX	Real	
T_OUTPUT2_INTERVAL_MEAN	Real	
T_TRANSACTIONS_OUTPUT1	Integer	
T_TRANSACTIONS_OUTPUT2	Integer	

To access these values in the panels or in the simulation monitor, use the following naming convention: `<Component_Name>^value_name`

For example, you access the `T_TRANSACTIONS_OUTPUT1` for component `BRANCH` as `BRANCH^T_TRANSACTIONS_OUTPUT1`.

`CLEAR` clears the statistics.

Termination Type

Reactive Controlled

STM_BRANCH_3

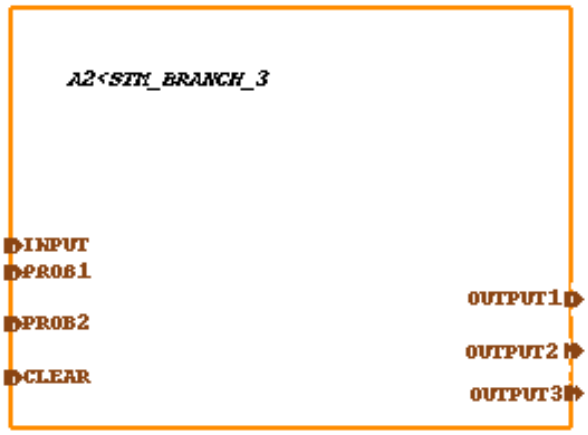
Branches transactions to one of three outputs.

Description

Generates `STMM_TRANSACTION_TYPE` transactions at the `OUTPUT1`, `OUTPUT2`, or `OUTPUT3` port. When an `INPUT` arrives, the component generates a uniform real number between 0.0 and 100.0. If the number is:

- Less than or equal to `PROB1`, `OUTPUT1` is set to `INPUT`.
- Greater than PROB1, but less than or equal to `(PROB1+PROB2)`, `OUTPUT2` is set to `INPUT`.
- Greater than `(PROB1+PROB2)`, `OUTPUT3` is set to `INPUT`.

When you insert the `STM_BRANCH_3` component into an activity chart, the component is represented by the following graphic image.



I/O Stubs

Name	Mode	Format
INPUT	input	STMM_TRANSACTION_TYPE
PROB1	input	Real
PROB2	input	Real
CLEAR	input	Event
OUTPUT1	output	STMM_TRANSACTION_TYPE
OUTPUT2	output	STMM_TRANSACTION_TYPE
OUTPUT3	output	STMM_TRANSACTION_TYPE

Analysis Statistics

Formal Parameter	Type
ANALYZE	Condition

If you set **ANALYZE**, the following statistics are gathered:

Variable	Data Type	Description
Local variables for analysis		
OUTPUT1_INTERVAL	Real array	OUTPUT1 arrival interval
OUTPUT1_INTERVAL_MIN	Real array	
OUTPUT1_INTERVAL_MAX	Real array	
OUTPUT1_INTERVAL_MEAN	Real array	
OUTPUT2_INTERVAL	Real array	OUTPUT2 arrival interval
OUTPUT2_INTERVAL_MIN	Real array	
OUTPUT2_INTERVAL_MAX	Real array	

Variable	Data Type	Description
OUTPUT2_INTERVAL_MEAN	Real array	
OUTPUT3_INTERVAL	Real array	OUTPUT3 arrival interval
OUTPUT3_INTERVAL_MIN	Real array	
OUTPUT3_INTERVAL_MAX	Real array	
OUTPUT3_INTERVAL_MEAN	Real array	
TRANSACTIONS_OUTPUT1	Integer array	Number of output1 transactions
TRANSACTIONS_OUTPUT2	Integer array	Number of output2 transactions
TRANSACTIONS_OUTPUT3	Integer array	Number of output3 transactions
Totals for analysis		
T_OUTPUT1_INTERVAL	Real	
T_OUTPUT1_INTERVAL_MIN	Real	
T_OUTPUT1_INTERVAL_MAX	Real	
T_OUTPUT1_INTERVAL_MEAN	Real	
T_OUTPUT2_INTERVAL	Real	
T_OUTPUT2_INTERVAL_MIN	Real	
T_OUTPUT2_INTERVAL_MAX	Real	
T_OUTPUT2_INTERVAL_MEAN	Real	
T_OUTPUT3_INTERVAL	Real	
T_OUTPUT3_INTERVAL_MIN	Real	
T_OUTPUT3_INTERVAL_MAX	Real	
T_OUTPUT3_INTERVAL_MEAN	Real	
T_TRANSACTIONS_OUTPUT1	Integer	
T_TRANSACTIONS_OUTPUT2	Integer	
T_TRANSACTIONS_OUTPUT3	Integer	

To access these values in the panels or in the simulation monitor, use the following naming convention: `<Component_Name>^value_name`

For example, you access the `T_TRANSACTIONS_OUTPUT1` for component `BRANCH` as `BRANCH^T_TRANSACTIONS_OUTPUT1.`

`CLEAR` clears the statistics.

Termination Type

Reactive Controlled

STM_FORK_2

Forks transactions to two outputs.

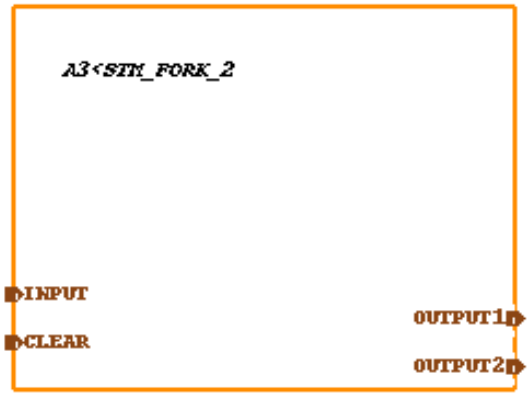
Description

Generates `STMM_TRANSACTION_TYPE` transactions at the `OUTPUT1` and `OUTPUT2` ports. When an INPUT arrives, the component sets `OUTPUT1` and `OUTPUT2` to `INPUT`.

Note

To synchronize the forked transactions, use the `STM_JOIN_2` component.

When you insert the `STM_FORK_2` component into an activity chart, the component is represented by the following graphic image.



I/O Stubs

Name	Mode	Format
INPUT	input	STMM_TRANSACTION_TYPE
CLEAR	input	Event
OUTPUT1	output	STMM_TRANSACTION_TYPE
OUTPUT2	output	STMM_TRANSACTION_TYPE

Analysis Statistics

Formal Parameter	Type
ANALYZE	Condition

If you set **ANALYZE**, the following statistics are gathered:

Variable	Data Type	Description
Local variables for analysis		
OUTPUT_INTERVAL	Real array	OUTPUT arrival interval
OUTPUT_INTERVAL_MIN	Real array	
OUTPUT_INTERVAL_MAX	Real array	
OUTPUT_INTERVAL_MEAN	Real array	
T_OUTPUT_INTERVAL_MEAN	Real	
T_TRANSACTIONS_OUTPUT	Integer	

To access these values in the panels or in the simulation monitor, use the following naming convention: <Component_Name>^value_name

For example, you access the `T_TRANSACTIONS_OUTPUT` for component `FORK` as `FORK^T_TRANSACTIONS_OUTPUT`.

`CLEAR` clears the statistics.

Termination Type

Reactive Controlled

STM_FORK_3

Forks transactions to three outputs.

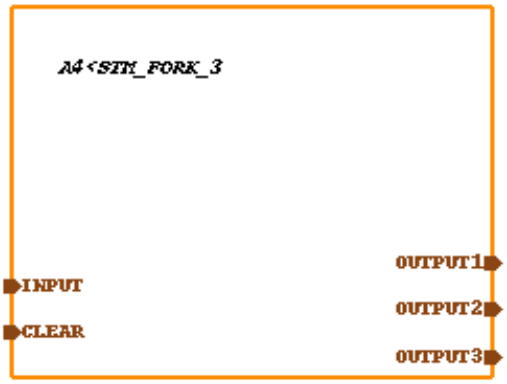
Description

Generates `STMM_TRANSACTION_TYPE` transactions at the OUTPUT1, OUTPUT2, and OUTPUT3 ports. When an INPUT arrives, the component sets OUTPUT1, OUTPUT2, and OUTPUT3 to INPUT.

Note

To synchronize the forked transactions, use the `STM_JOIN_3` component.

When you insert the `STM_FORK_3` component into an activity chart, the component is represented by the following graphic image.



I/O Stubs

Name	Mode	Format
INPUT	input	STMM_TRANSACTION_TYPE
CLEAR	input	Event
OUTPUT1	output	STMM_TRANSACTION_TYPE
OUTPUT2	output	STMM_TRANSACTION_TYPE
OUTPUT3	output	STMM_TRANSACTION_TYPE

Analysis Statistics

Formal Parameter	Type
ANALYZE	Condition

If you set **ANALYZE**, the following statistics are gathered:

Variable	Data Type	Description
Local variables for analysis		
OUTPUT_INTERVAL	Real array	OUTPUT arrival interval
OUTPUT_INTERVAL_MIN	Real array	
OUTPUT_INTERVAL_MAX	Real array	
OUTPUT_INTERVAL_MEAN	Real array	
TRANSACTIONS_OUTPUT	Integer array	Number of OUTPUT transactions
Totals for analysis		
T_OUTPUT_INTERVAL	Real	
T_OUTPUT_INTERVAL_MIN	Real	
T_OUTPUT_INTERVAL_MAX	Real	
T_OUTPUT_INTERVAL_MEAN	Real	
T_TRANSACTIONS_OUTPUT	Integer	

To access these values in the panels or in the simulation monitor, use the following naming convention: `<Component_Name>^value_name`

For example, you access the `T_TRANSACTIONS_OUTPUT` for component `FORK` as `FORK^T_TRANSACTIONS_OUTPUT`.

CLEAR clears the statistics.

Termination Type

Reactive Controlled

STM_JOIN_2

Joins (synchronizes) two transactions.

Description

Generates `STMM_TRANSACTION_TYPE` transactions at the `INPUT1` and `INPUT2` ports.

Note

Use this component in conjunction with the `STM_FORK_2` component to synchronize the forked transactions.

When the `INPUT1` and `INPUT2` ports receive a transaction, the component generates the `OUTPUT` transaction. (The `OUTPUT` is set to the most recent input, which should be the same as the other input.).

When you insert the `STM_JOIN_2` component into an activity chart, the component is represented by the following graphic image.

I/O Stubs

Name	Mode	Format
INPUT1	input	STMM_TRANSACTION_TYPE
INPUT2	input	STMM_TRANSACTION_TYPE
CLEAR	input	Event
OUTPUT	output	STMM_TRANSACTION_TYPE

Analysis Statistics

Formal Parameter	Type
ANALYZE	Condition

If you set **ANALYZE**, the following statistics are gathered:

Variable	Data Type	Description
Local variables for analysis		
INPUT1_INTERVAL	Real array	INPUT1 arrival interval
INPUT1_INTERVAL_MIN	Real array	
INPUT1_INTERVAL_MAX	Real array	
INPUT1_INTERVAL_MEAN	Real array	
INPUT2_INTERVAL	Real array	INPUT2 arrival interval
INPUT2_INTERVAL_MIN	Real array	
INPUT2_INTERVAL_MAX	Real array	
INPUT2_INTERVAL_MEAN	Real array	
RESPONSE_TIME	Real array	Time between fork and join
RESPONSE_TIME_MIN	Real array	
RESPONSE_TIME_MAX	Real array	
RESPONSE_TIME_MEAN	Real array	
TRANSACTIONS_OUTPUT	Integer array	Number of OUTPUT transactions

Variable	Data Type	Description
Totals for analysis		
T_INPUT1_INTERVAL	Real	
T_INPUT1_INTERVAL_MIN	Real	
T_INPUT1_INTERVAL_MAX	Real	
T_INPUT1_INTERVAL_MEAN	Real	
T_INPUT2_INTERVAL	Real	
T_INPUT2_INTERVAL_MIN	Real	
T_INPUT2_INTERVAL_MAX	Real	
T_INPUT2_INTERVAL_MEAN	Real	
T_RESPONSE_TIME	Real	
T_RESPONSE_TIME_MIN	Real	
T_RESPONSE_TIME_MAX	Real	
T_RESPONSE_TIME_MEAN	Real	
T_TRANSACTIONS_OUTPUT	Integer	

To access these values in the panels or in the simulation monitor, use the following naming convention: `<Component_Name>^value_name`

For example, you access the `T_TRANSACTIONS_OUTPUT` for component `FORK` as `FORK^T_TRANSACTIONS_OUTPUT`.

CLEAR clears the statistics.

Termination Type

Reactive Controlled

STM_JOIN_3

Joins (synchronizes) three transactions.

Description

Generates `STMM_TRANSACTION_TYPE` transactions at the `INPUT1`, `INPUT2`, and `INPUT3` ports.

Note

Use this component in conjunction with the `STM_FORK_3` component to synchronize the forked transactions.

When the `INPUT1`, `INPUT2`, and `INPUT3` ports receive a transaction, the component generates the OUTPUT transaction. (The OUTPUT is set to the most recent input, which should be the same as the other inputs.).

When you insert the `STM_JOIN_3` component into an activity chart, the component is represented by the following graphic image.

I/O Stubs

Name	Mode	Format
INPUT1	input	STMM_TRANSACTION_TYPE
INPUT2	input	STMM_TRANSACTION_TYPE
INPUT3	input	STMM_TRANSACTION_TYPE
CLEAR	input	Event
OUTPUT	output	STMM_TRANSACTION_TYPE

Analysis Statistics

Formal Parameter	Type
ANALYZE	Condition

If you set **ANALYZE**, the following statistics are gathered:

Variable	Data Type	Description
Local variables for analysis		
INPUT1_INTERVAL	Real array	INPUT1 arrival interval
INPUT1_INTERVAL_MIN	Real array	
INPUT1_INTERVAL_MAX	Real array	
INPUT1_INTERVAL_MEAN	Real array	
INPUT2_INTERVAL	Real array	INPUT2 arrival interval
INPUT2_INTERVAL_MIN	Real array	
INPUT2_INTERVAL_MAX	Real array	
INPUT2_INTERVAL_MEAN	Real array	
INPUT3_INTERVAL	Real array	
INPUT3_INTERVAL_MIN	Real array	INPUT3 arrival interval

Variable	Data Type	Description
INPUT3_INTERVAL_MAX	Real array	
INPUT3_INTERVAL_MEAN	Real array	
RESPONSE_TIME	Real array	Time between fork and join
RESPONSE_TIME_MIN	Real array	
RESPONSE_TIME_MAX	Real array	
RESPONSE_TIME_MEAN	Real array	
TRANSACTIONS_OUTPUT	Integer array	Number of output transactions
Totals for analysis		
T_INPUT1_INTERVAL	Real	
T_INPUT1_INTERVAL_MIN	Real	
T_INPUT1_INTERVAL_MAX	Real	
T_INPUT1_INTERVAL_MEAN	Real	
T_INPUT2_INTERVAL	Real	
T_INPUT2_INTERVAL_MIN	Real	
T_INPUT2_INTERVAL_MAX	Real	
T_INPUT2_INTERVAL_MEAN	Real	
T_INPUT3_INTERVAL	Real	
T_INPUT3_INTERVAL_MIN	Real	
T_INPUT3_INTERVAL_MAX	Real	
T_INPUT3_INTERVAL_MEAN	Real	
T_RESPONSE_TIME	Real	
T_RESPONSE_TIME_MIN	Real	
T_RESPONSE_TIME_MAX	Real	
T_RESPONSE_TIME_MEAN	Real	
T_TRANSACTIONS_OUTPUT	Integer	

To access these values in the panels or in the simulation monitor, use the following naming convention: <Component_Name>^value_name

For example, you access the `T_TRANSACTIONS_OUTPUT` for component `JOIN` as `JOIN^T_TRANSACTIONS_OUTPUT`.

`CLEAR` clears the statistics.

Termination Type

Reactive Controlled

STM_FIFO_ACTIVE

A first-in-first-out active service resource component.

Description

Accepts `STMM_TRANSACTION_TYPE` transactions on the `INPUT` port, and processes them in a first-in-first-out scheme.

- If the service is idle, `INPUT` is immediately serviced.
- If the service is busy, `INPUT` is queued and serviced when it is the oldest transaction in the queue.
- **DIST** specifies the distribution and parameters used to calculate the service time for each transaction class.
- If **INT** occurs while a transaction is in service, the servicing stops, and the transaction is sent to `I_OUTPUT`.
- If the transaction completes servicing without interruption, then it is sent to the `S_OUTPUT`.
- **CLEAR** clears (resets) contents of the resource.

When you insert the `STM_FIFO_ACTIVE` component into an activity chart, the component is represented by the following graphic image.

I/O Stubs

Name	Mode	Format
INPUT1	input	STMM_TRANSACTION_TYPE
CLEAR	input	Event
UPDATE	input	Event
DIST	input	STMM_TRANS_CLASS_ARRAY_DIST
INT	input	Event
S_OUTPUT	output	STMM_TRANSACTION_TYPE
I_OUTPUT	output	STMM_TRANSACTION_TYPE

Analysis Statistics

Formal Parameter	Type
ANALYZE	Condition

If you set **ANALYZE**, the following statistics are gathered

Variable	Data Type	Description
Local variables for analysis		
QUEUE_LENGTH	Integer	Length of the queue
QUEUE_LENGTH_MIN	Integer	
QUEUE_LENGTH_MAX	Integer	
QUEUE_LENGTH_MEAN	Real	
RESPONSE_TIME	Real array	Time between INPUT and OUTPUT
RESPONSE_TIME_MIN	Real array	
RESPONSE_TIME_MAX	Real array	
RESPONSE_TIME_MEAN	Real array	

Variable	Data Type	Description
SERVICE_UTILIZATION	Real-Percent	Resource utilization
SERVICE_UTILIZATION_MIN	Real-Percent	
SERVICE_UTILIZATION_MAX	Real-Percent	
SERVICE_UTILIZATION_MEAN	Real-Percent	
TRANSACTIONS_INPUT	Integer array	Number of INPUT transactions
TRANSACTIONS_INTERRUPTED	Integer array	Number of interrupted transactions
TRANSACTIONS_SERVICED	Integer array	Number of serviced transactions
TRANSACTION_INTERVAL	Real array	INPUT arrival interval
TRANSACTION_INTERVAL_MIN	Real array	
TRANSACTION_INTERVAL_MAX	Real array	
TRANSACTION_INTERVAL_MEAN	Real array	

Variable	Data Type	Description
Totals for analysis		
T_RESPONSE_TIME	Real	
T_RESPONSE_TIME_MIN	Real	
T_RESPONSE_TIME_MAX	Real	
T_RESPONSE_TIME_MEAN	Real	
T_TRANSACTIONS_INPUT	Integer	
T_TRANSACTIONS_INTERRUPTED	Integer	
T_TRANSACTIONS_SERVICED	Integer	
T_TRANSACTION_INTERVAL	Real	
T_TRANSACTION_INTERVAL_MIN	Real	
T_TRANSACTION_INTERVAL_MAX	Real	
T_TRANSACTION_INTERVAL_MEAN	Real	

To access these values in the panels or in the simulation monitor, use the following naming convention: `<Component_Name>^value_name`

For example, you access the `QUEUE_LENGTH` for component `ACT_RES1` as `ACT_RES1^QUEUE_LENGTH`.

CLEAR clears the statistics.

`UPDATE` makes sure the statistics values are up-to-date. In some cases, the statistics are only updated when events are triggered within the resource, so `UPDATE` ensures the statistics are up-to-date regardless of the triggering events.

Termination Type

Reactive Controlled

STM_FIFO_PASSIVE

A first-in-first-out passive resource component.

Description

Accepts `STMM_TRANSACTION_TYPE` transactions on the INPUT port, and processes them in a first-in-first-out scheme.

- **RELEASE** - The component also accepts `STMM_TRANSACTION_TYPE` transactions on the `RELEASE_` port and processes them immediately.
- **USAGE** - This port specifies the amount of resource used by a transaction based on the class.
  - If there is enough of the resource available (**AVAIL**) to process an input transaction, the resource is allocated, and **ALLOCD** is set to the input transaction.
  - If there are insufficient resources, the input transaction is placed in the FIFO queue until sufficient resources are available.
  - When a transaction arrives on the `RELEASE_` port, the resources associated with the transaction are released (based on the value of USAGE), and **FREED** is set to the release transaction.

When you insert the `STM_FIFO_PASSIVE` component into an activity chart, the component is represented by the following graphic image.

I/O Stubs

Name	Mode	Format
INPUT	input	STMM_TRANSACTION_TYPE
RELEASE_	input	STMM_TRANSACTION_TYPE
USAGE	input	STMM_TRANSACTION_TYPE
CLEAR	input	Event
UPDATE	input	Event
SIZE	input/output	Real
ALLOCID	output	STMM_TRANSACTION_TYPE
FREED	output	STMM_TRANSACTION_TYPE
AVAIL	output	Real
SIZE	input/output	Real

Analysis Statistics

Formal Parameter	Type
ANALYZE	Condition
INITIAL_RESOURCE_SIZE	(Data-Item) Real

If you set **ANALYZE**, the following statistics are gathered:

Variable	Data Type	Description
Local variables for analysis		
AVAIL	Real	Available resource units
INPUT_INTERVAL	Real array	INPUT arrival interval
INPUT_INTERVAL_MIN	Real array	

Variable	Data Type	Description
INPUT_INTERVAL_MAX	Real array	
INPUT_INTERVAL_MEAN	Real array	
QUEUE_LENGTH	Integer	Length of the queue
QUEUE_LENGTH_MIN	Integer	
QUEUE_LENGTH_MAX	Integer	
QUEUE_LENGTH_MEAN	Real	
RELEASE_INTERVAL	Real array	Release arrival interval
RELEASE_INTERVAL_MIN	Real array	
RELEASE_INTERVAL_MAX	Real array	
RELEASE_INTERVAL_MEAN	Real array	
RESOURCE_IN_USE	Real array	Resource units in use
RESOURCE_IN_USE_MIN	Real array	
RESOURCE_IN_USE_MAX	Real array	
RESOURCE_IN_USE_MEAN	Real array	
RESPONSE_TIME	Real array	Time in queue
RESPONSE_TIME_MIN	Real array	
RESPONSE_TIME_MAX	Real array	
RESPONSE_TIME_MEAN	Real array	
SIZE	Real	Total resource units
TRANSACTIONS_ALLOCD	Integer array	Number of allocated inputs
TRANSACTIONS_FREED	Integer array	Number of freed releases
USAGE_TIME	Real array	Time between input and freed
USAGE_TIME_MIN	Real array	
USAGE_TIME_MAX	Real array	
USAGE_TIME_MEAN	Real array	
Totals for analysis		
T_INPUT_INTERVAL	Real	
T_INPUT_INTERVAL_MIN	Real	
T_INPUT_INTERVAL_MAX	Real	
T_INPUT_INTERVAL_MEAN	Real	

Variable	Data Type	Description
T_RELEASE_INTERVAL	Real	
T_RELEASE_INTERVAL_MIN	Real	
T_RELEASE_INTERVAL_MAX	Real	
T_RELEASE_INTERVAL_MEAN	Real	
T_RESOURCE_IN_USE	Real	
T_RESOURCE_IN_USE_MIN	Real	
T_RESOURCE_IN_USE_MAX	Real	
T_RESOURCE_IN_USE_MEAN	Real	
T_RESPONSE_TIME	Real	
T_RESPONSE_TIME_MIN	Real	
T_RESPONSE_TIME_MAX	Real	
T_RESPONSE_TIME_MEAN	Real	
T_TRANSACTIONS_ALLOCD	Integer	
T_TRANSACTIONS_FREED	Integer	
T_USAGE_TIME	Real	
T_USAGE_TIME_MIN	Real	
T_USAGE_TIME_MAX	Real	
T_USAGE_TIME_MEAN	Real	

To access these values in the panels or in the simulation monitor, use the following naming convention: `<Component_Name>^value_name`

For example, you access the `QUEUE_LENGTH` for component `PASS_RES1` as `PASS_RES1^QUEUE_LENGTH`.

`CLEAR` clears the statistics.

`UPDATE` makes sure the statistics values are up-to-date. In some cases, the statistics are only updated when events are triggered within the resource, so `UPDATE` ensures the statistics are up-to-date regardless of the triggering events.

Termination Type

Reactive Controlled

STM_LIFO_ACTIVE

A last-in-first-out active resource component.

Description

Accepts `STMM_TRANSACTION_TYPE` transactions on the INPUT port, and processes them in a last-in-first-out scheme.

- If the service is idle, INPUT is immediately serviced.
- If the service is busy, INPUT is queued, and serviced when it is the youngest transaction in the queue.
- **DIST** specifies the distribution and parameters used to calculate the service time for each transaction class.
- If **INT** occurs while a transaction is in service, the servicing stops, and the transaction is sent to `I_OUTPUT`.
- If the transaction completes servicing without interruption, it is sent to the `S_OUTPUT`.

When you insert the `STM_LIFO_ACTIVE` component into an activity chart, the component is represented by the following graphic image.

CLEAR clears (resets) the contents of the resource.

I/O Stubs

Name	Mode	Format
INPUT1	input	STMM_TRANSACTION_TYPE
CLEAR	input	Event
UPDATE	input	Event
DIST	input	STMM_TRANS_CLASS_ARRAY_DIST
INT	input	Event
S_OUTPUT	output	STMM_TRANSACTION_TYPE
I_OUTPUT	output	STMM_TRANSACTION_TYPE

Analysis Statistics

Formal Parameter	Type
ANALYZE	Condition

If you set **ANALYZE**, the following statistics are gathered:

Variable	Data Type	Description
Local variables for analysis		
QUEUE_LENGTH	Integer	Length of the queue
QUEUE_LENGTH_MIN	Integer	
QUEUE_LENGTH_MAX	Integer	
QUEUE_LENGTH_MEAN	Real	
RESPONSE_TIME	Real array	Time between input and output
RESPONSE_TIME_MIN	Real array	
RESPONSE_TIME_MAX	Real array	
RESPONSE_TIME_MEAN	Real array	

Variable	Data Type	Description
SERVICE_UTILIZATION	Real-Percent	Resource utilization
SERVICE_UTILIZATION_MIN	Real-Percent	
SERVICE_UTILIZATION_MAX	Real-Percent	
SERVICE_UTILIZATION_MEAN	Real-Percent	
TRANSACTIONS_INPUT	Integer array	Number of input transactions
TRANSACTIONS_INTERRUPTED	Integer array	Number of interrupted transactions
TRANSACTIONS_SERVICED	Integer array	Number of serviced transactions
TRANSACTION_INTERVAL	Real array	INPUT arrival interval
TRANSACTION_INTERVAL_MIN	Real array	
TRANSACTION_INTERVAL_MAX	Real array	
TRANSACTION_INTERVAL_MEAN	Real array	
Totals for analysis		
T_RESPONSE_TIME	Real	
T_RESPONSE_TIME_MIN	Real	
T_RESPONSE_TIME_MAX	Real	
T_RESPONSE_TIME_MEAN	Real	
T_TRANSACTIONS_INPUT	Integer	
T_TRANSACTIONS_INTERRUPTED	Integer	
T_TRANSACTIONS_SERVICED	Integer	
T_TRANSACTION_INTERVAL	Real	
T_TRANSACTION_INTERVAL_MIN	Real	
T_TRANSACTION_INTERVAL_MAX	Real	
T_TRANSACTION_INTERVAL_MEAN	Real	

To access these values in the panels or in the simulation monitor, use the following naming convention: `<Component_Name>^value_name`

For example, you access the `QUEUE_LENGTH` for component `ACT_RES1` as `ACT_RES1^QUEUE_LENGTH`.

`CLEAR` clears the statistics.

`UPDATE` makes sure the statistics values are up-to-date. In some cases, the statistics are only updated when events are triggered within the resource, so `UPDATE` ensures the statistics are up-to-date regardless of the triggering events.

Termination Type

Reactive Controlled

STM_LIFO_PASSIVE

A last-in-last-out passive resource component.

Description

Accepts `STMM_TRANSACTION_TYPE` transactions on the INPUT port, and processes them in a last-in-first-out scheme.

- **RELEASE_** – The component also accepts `STMM_TRANSACTION_TYPE` transactions on the `RELEASE_` port and processes them immediately.
- **USAGE** – This port specifies the amount of resource used by a transaction based on the transaction class.
  - If there is enough of the resource available (AVAIL) to process an input transaction, the resource is allocated, and ALLOCD is set to the input transaction.
  - If there are insufficient resources, the input transaction is placed in the LIFO queue until sufficient resources are available.
  - When a transaction arrives on the `RELEASE_` port, the resources associated with the transaction are released (based on the value of USAGE), and FREED is set to the release transaction.

When you insert the `STM_LIFO_PASSIVE` component into an activity chart, the component is represented by the following graphic image.

I/O Stubs

Name	Mode	Format
INPUT	input	STMM_TRANSACTION_TYPE
RELEASE_	input	STMM_TRANSACTION_TYPE
USAGE	input	STMM_TRANSACTION_TYPE
CLEAR	input	Event
UPDATE	input	Event
SIZE	input/output	Real
ALLOCID	output	STMM_TRANSACTION_TYPE
FREED	output	STMM_TRANSACTION_TYPE
AVAIL	output	Real
SIZE	input/output	Real

Analysis Statistics

Formal Parameter	Type
ANALYZE	Condition
INITIAL_RESOURCE_SIZE	(Data-Item) Real

If you set **ANALYZE**, the following statistics are gathered:

Variable	Data Type	Description
Local variables for analysis		
AVAIL	Real	Available resource units
INPUT_INTERVAL	Real array	INPUT arrival interval
INPUT_INTERVAL_MIN	Real array	

Variable	Data Type	Description
INPUT_INTERVAL_MAX	Real array	
INPUT_INTERVAL_MEAN	Real array	
QUEUE_LENGTH	Integer	Length of the queue
QUEUE_LENGTH_MIN	Integer	
QUEUE_LENGTH_MAX	Integer	
QUEUE_LENGTH_MEAN	Real	
RELEASE_INTERVAL	Real array	Release arrival interval
RELEASE_INTERVAL_MIN	Real array	
RELEASE_INTERVAL_MAX	Real array	
RELEASE_INTERVAL_MEAN	Real array	
RESOURCE_IN_USE	Real array	Resource units in use
RESOURCE_IN_USE_MIN	Real array	
RESOURCE_IN_USE_MAX	Real array	
RESOURCE_IN_USE_MEAN	Real array	
RESPONSE_TIME	Real array	Time in queue
RESPONSE_TIME_MIN	Real array	
RESPONSE_TIME_MAX	Real array	
RESPONSE_TIME_MEAN	Real array	
SIZE	Real	Total resource units
TRANSACTIONS_ALLOCD	Integer array	Number of allocated inputs
TRANSACTIONS_FREED	Integer array	Number of freed releases
USAGE_TIME	Real array	Time between input and freed
USAGE_TIME_MIN	Real array	
USAGE_TIME_MAX	Real array	
USAGE_TIME_MEAN	Real array	
Totals for analysis		
T_INPUT_INTERVAL	Real	
T_INPUT_INTERVAL_MIN	Real	
T_INPUT_INTERVAL_MAX	Real	
T_INPUT_INTERVAL_MEAN	Real	

Variable	Data Type	Description
T_RELEASE_INTERVAL	Real	
T_RELEASE_INTERVAL_MIN	Real	
T_RELEASE_INTERVAL_MAX	Real	
T_RELEASE_INTERVAL_MEAN	Real	
T_RESOURCE_IN_USE	Real	
T_RESOURCE_IN_USE_MIN	Real	
T_RESOURCE_IN_USE_MAX	Real	
T_RESOURCE_IN_USE_MEAN	Real	
T_RESPONSE_TIME	Real	
T_RESPONSE_TIME_MIN	Real	
T_RESPONSE_TIME_MAX	Real	
T_RESPONSE_TIME_MEAN	Real	
T_TRANSACTIONS_ALLOCD	Integer	
T_TRANSACTIONS_FREED	Integer	
T_USAGE_TIME	Real	
T_USAGE_TIME_MIN	Real	
T_USAGE_TIME_MAX	Real	
T_USAGE_TIME_MEAN	Real	

To access these values in the panels or in the simulation monitor, use the following naming convention: <Component_Name>^value_name

For example, you access the `QUEUE_LENGTH` for component `PASS_RES1` as `PASS_RES1^QUEUE_LENGTH`.

`CLEAR` clears the statistics.

`UPDATE` makes sure the statistics values are up-to-date. In some cases, the statistics are only updated when events are triggered within the resource, so `UPDATE` ensures the statistics are up-to-date regardless of the triggering events.

Termination Type

Reactive Controlled

STM_PMPT_ACTIVE

A preemptive priority-based active resource component.

Description

Accepts `STMM_TRANSACTION_TYPE` transactions on the INPUT port, and processes them in a preemptive priority-based scheme.

The transaction's priority is specified by `INPUT_PRIORITY_LEVEL`.

- If the service is idle, INPUT is immediately serviced.
- If the service is busy, INPUT is queued, and serviced when it is the highest priority transaction in the queue.
- **DIST** specifies the distribution and parameters used to calculate the service time for each transaction class.
- If **INT** occurs while a transaction is in service, the servicing stops, and the transaction is sent to `I_OUTPUT`.
- If a higher priority transaction arrives while a transaction is in service, the servicing stops, and the transaction is either:
  - Put back into the queue if its `EXIT_ON_PREEMPT` is FALSE, or
  - Sent to the `P_OUTPUT` if its `EXIT_ON_PREEMPT` is TRUE.
- If the transaction completes servicing without interruption or preemption, it is sent to the `S_OUTPUT`.

When you insert the `STM_PMPT_ACTIVE` component into an activity chart, the component is represented by the following graphic image.

I/O Stubs

Name	Mode	Format
INPUT1	input	STMM_TRANSACTION_TYPE
CLEAR	input	Event
UPDATE	input	Event
DIST	input	STMM_TRANS_CLASS_ARRAY_DIST
INT	input	Event
S_OUTPUT	output	STMM_TRANSACTION_TYPE
I_OUTPUT	output	STMM_TRANSACTION_TYPE
P_OUTPUT	output	STMM_TRANSACTION_TYPE

Analysis Statistics

Formal Parameter	Type
ANALYZE	Condition

If you set **ANALYZE**, the following statistics are gathered:

Variable	Data Type	Description
Local variables for analysis		
QUEUE_LENGTH	Integer	Length of the queue
QUEUE_LENGTH_MIN	Integer	
QUEUE_LENGTH_MAX	Integer	
QUEUE_LENGTH_MEAN	Real	
SERVICE_UTILIZATION	Real-Percent	Resource utilization
SERVICE_UTILIZATION_MIN	Real-Percent	
SERVICE_UTILIZATION_MAX	Real-Percent	
SERVICE_UTILIZATION_MEAN	Real-Percent	
RESPONSE_TIME	Real array	Time between input and output
RESPONSE_TIME_MIN	Real array	
RESPONSE_TIME_MAX	Real array	
RESPONSE_TIME_MEAN	Real array	
TRANSACTIONS_INPUT	Integer array	Number of input transactions
TRANSACTIONS_INTERRUPTED	Integer array	Number of interrupted transactions
TRANSACTIONS_PREEMPTED	Integer array	Number of preempted transactions
TRANSACTIONS_SERVICED	Integer array	Number of serviced transactions
TRANSACTION_INTERVAL	Real array	INPUT arrival interval
TRANSACTION_INTERVAL_MIN	Real array	
TRANSACTION_INTERVAL_MAX	Real array	
TRANSACTION_INTERVAL_MEAN	Real array	
Totals for analysis		
T_RESPONSE_TIME	Real	
T_RESPONSE_TIME_MIN	Real	
T_RESPONSE_TIME_MAX	Real	
T_RESPONSE_TIME_MEAN	Real	
T_TRANSACTIONS_INPUT	Integer	

Variable	Data Type	Description
T_TRANSACTIONS_INTERRUPTED	Integer	
T_TRANSACTIONS_PREEMPTED	Integer	
T_TRANSACTIONS_SERVICED	Integer	
T_TRANSACTION_INTERVAL	Real	
T_TRANSACTION_INTERVAL_MIN	Real	
T_TRANSACTION_INTERVAL_MAX	Real	
T_TRANSACTION_INTERVAL_MEAN	Real	

To access these values in the panels or in the simulation monitor, use the following naming convention: `<Component_Name>^value_name`

For example, you access the `QUEUE_LENGTH` for component `ACT_RES1` as `ACT_RES1^QUEUE_LENGTH`.

`CLEAR` clears the statistics.

UPDATE makes sure the statistics values are up-to-date. In some cases, the statistics are only updated when events are triggered within the resource, so UPDATE ensures the statistics are up-to-date regardless of the triggering events.

Termination Type

Reactive Controlled

STM_PRTY_PASSIVE

A priority-based passive resource component.

Description

Accepts `STMM_TRANSACTION_TYPE` transactions on the INPUT port, and processes them in a priority-based scheme.

- **RELEASE_** – The component also accepts `STMM_TRANSACTION_TYPE` transactions on the `RELEASE_` port and processes them immediately.
- **USAGE** – This port specifies the amount of resource used by a transaction based on the class.
  - If there is enough of the resource available (**AVAIL**) to process an input transaction, the resource is allocated, and **ALLOCD** is set to the input transaction.
  - If there are insufficient resources, the input transaction is placed in the queue until sufficient resources are available.
  - When a transaction arrives on the `RELEASE_` port, the resources associated with the transaction are released (based on the value of USAGE), and **FREED** is set to the release transaction.
- **SIZE** – `INITIAL_RESOURCE_SIZE` is the initial size of the resource. A change in the size of the resource can be requested by modifying SIZE. An increase in size is always allowed, but a decrease is only allowed when the new size is greater than or equal to the amount of resource in use.

When you insert the `STM_PRTY_PASSIVE` component into an activity chart, the component is represented by the following graphic image.

I/O Stubs

Name	Mode	Format
INPUT	input	STMM_TRANSACTION_TYPE
RELEASE	input	STMM_TRANSACTION_TYPE
USAGE	input	STMM_TRANSACTION_TYPE
CLEAR	input	Event
UPDATE	input	Event
SIZE	input/output	Real
ALLOCID	output	STMM_TRANSACTION_TYPE
FREED	output	STMM_TRANSACTION_TYPE
AVAIL	output	Real
SIZE	input/output	Real

Analysis Statistics

Formal Parameter	Type
ANALYZE	Condition
INITIAL_RESOURCE_SIZE	(Data-Item) Real

If you set **ANALYZE**, the following statistics are gathered:

Variable	Data Type	Description
Local variables for analysis		
QUEUE_LENGTH	Integer	Length of the queue
QUEUE_LENGTH_MIN	Integer	
QUEUE_LENGTH_MAX	Integer	
QUEUE_LENGTH_MEAN	Real	
AVAIL	Real	Available resource units

Variable	Data Type	Description
SIZE	Real	Total resource units
INPUT_INTERVAL	Real array	INPUT arrival interval
INPUT_INTERVAL_MIN	Real array	
INPUT_INTERVAL_MAX	Real array	
INPUT_INTERVAL_MEAN	Real array	
RELEASE_INTERVAL	Real array	Release arrival interval
RELEASE_INTERVAL_MIN	Real array	
RELEASE_INTERVAL_MAX	Real array	
RELEASE_INTERVAL_MEAN	Real array	
RESOURCE_IN_USE	Real array	Resource units in use
RESOURCE_IN_USE_MIN	Real array	
RESOURCE_IN_USE_MAX	Real array	
RESOURCE_IN_USE_MEAN	Real array	
RESPONSE_TIME	Real array	Time in queue
RESPONSE_TIME_MIN	Real array	
RESPONSE_TIME_MAX	Real array	
Totals for analysis		
T_INPUT_INTERVAL	Real	
T_INPUT_INTERVAL_MIN	Real	
T_INPUT_INTERVAL_MAX	Real	
T_INPUT_INTERVAL_MEAN	Real	
T_RELEASE_INTERVAL	Real	
T_RELEASE_INTERVAL_MIN	Real	
T_RELEASE_INTERVAL_MAX	Real	
T_RELEASE_INTERVAL_MEAN	Real	
T_RESOURCE_IN_USE	Real	
T_RESOURCE_IN_USE_MIN	Real	
T_RESOURCE_IN_USE_MAX	Real	
T_RESOURCE_IN_USE_MEAN	Real	
T_RESPONSE_TIME	Real	
T_RESPONSE_TIME_MIN	Real	
T_RESPONSE_TIME_MAX	Real	

Variable	Data Type	Description
T_RESPONSE_TIME_MEAN	Real	
T_TRANSACTIONS_ALLOCD	Integer	
T_TRANSACTIONS_FREED	Integer	
T_USAGE_TIME	Real	
T_USAGE_TIME_MIN	Real	
T_USAGE_TIME_MAX	Real	
T_USAGE_TIME_MEAN	Real	

To access these values in the panels or in the simulation monitor, use the following naming convention: `<Component_Name>^value_name`

For example, you access the `QUEUE_LENGTH` for component `PASS_RES1` as `PASS_RES1^QUEUE_LENGTH`.

`CLEAR` clears the statistics.

`UPDATE` makes sure the statistics values are up-to-date. In some cases, the statistics are only updated when events are triggered within the resource, so `UPDATE` ensures the statistics are up-to-date regardless of the triggering events.

Termination Type

Reactive Controlled

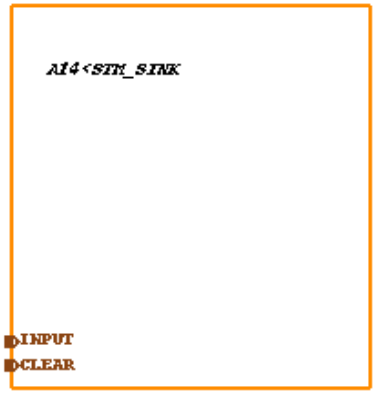
STM_SINK

Receives transactions.

Description

Receives `STMM_TRANSACTION_TYPE` transactions at the INPUT port.

When you insert the `STM_SINK` component into an activity chart, the component is represented by the following graphic image.



I/O Stubs

Name	Mode	Format
`INPUT`	input	`STMM_TRANSACTION_TYPE`
`CLEAR`	input	Event

Analysis Statistics

Formal Parameter	Type
ANALYZE	Condition

If you set **ANALYZE**, the following statistics are gathered:

Variable	Data Type	Description
Local variables for analysis		
INPUT_INTERVAL	Real array	INPUT arrival interval
INPUT_INTERVAL_MIN	Real array	
INPUT_INTERVAL_MAX	Real array	
INPUT_INTERVAL_MEAN	Real array	
RESPONSE_TIME	Real array	Time between source and sink
RESPONSE_TIME_MIN	Real array	
RESPONSE_TIME_MAX	Real array	
RESPONSE_TIME_MEAN	Real array	
TRANSACTIONS_INPUT	Integer array	Number of input transactions
Totals for analysis		
T_INPUT_INTERVAL	Real	
T_INPUT_INTERVAL_MIN	Real	
T_INPUT_INTERVAL_MAX	Real	
T_INPUT_INTERVAL_MEAN	Real	
T_RESPONSE_TIME	Real	
T_RESPONSE_TIME_MIN	Real	
T_RESPONSE_TIME_MAX	Real	
T_RESPONSE_TIME_MEAN	Real	
T_TRANSACTIONS_INPUT	Integer	

To access these values in the panels or in the simulation monitor, use the following naming convention: `<Component_Name>^value_name`

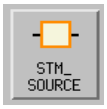
For example, you access the `T_TRANSACTIONS_INPUT` for component `SOURCE1` as `SOURCE1^T_TRANSACTIONS_INPUT`.

CLEAR clears the statistics.

Termination Type

Reactive Controlled

STM_SOURCE

Generates transactions. 

Description

Generates `STMM_TRANSACTION_TYPE` transactions at the OUTPUT port.

When you insert the `STM_SOURCE` component into an activity chart, the component is represented by the following graphic image.

I/O Stubs

Name	Mode	Format
`OUTPUT`	output	`STMM_TRANSACTION_TYPE`
`CLEAR`	input	Event

Analysis Statistics

Formal Parameter	Type
`ANALYZE`	Condition

If you set **ANALYZE**, the following statistics are gathered:

Variable	Data Type	Description
Local variables for analysis		
OUTPUT_INTERVAL	Real array	OUTPUT arrival interval
OUTPUT_INTERVAL_MIN	Real array	
OUTPUT_INTERVAL_MAX	Real array	
OUTPUT_INTERVAL_MEAN	Real array	
TRANSACTIONS_OUTPUT	Integer array	Number of output transactions
Totals for analysis		
T_OUTPUT_INTERVAL	Real	
T_OUTPUT_INTERVAL_MIN	Real	
T_OUTPUT_INTERVAL_MAX	Real	
T_OUTPUT_INTERVAL_MEAN	Real	
T_TRANSACTIONS_OUTPUT	Integer	

To access these values in the panels or in the simulation monitor, use the following naming convention: `<Component_Name>^value_name`

For example, you access the `T_TRANSACTIONS_OUTPUT` for component `SOURCE1` as `SOURCE1^T_TRANSACTIONS_OUTPUT`.

CLEAR clears the statistics.

Termination Type

Reactive Controlled

AUTOSAR Generator

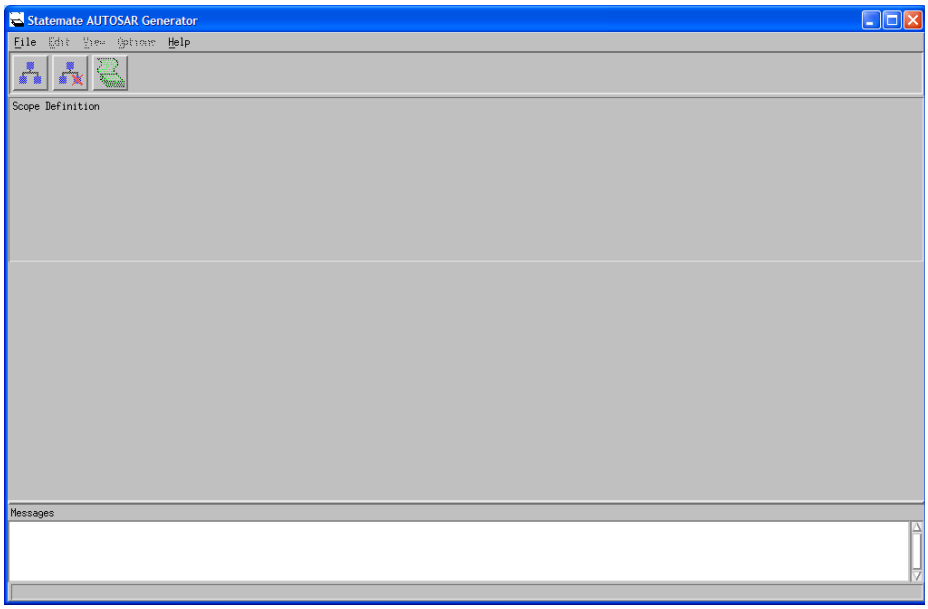
Rational Statemate AUTOSAR (AUTomotive Open System ARchitecture) Generator (SAG) tool enables defining an Atomic Software Component implementation using a Rational Statemate model and generate code accordingly, as well as a Software Component Description XML file.

With SAG, you can view the runnable entities that were defined in the model, design the ports and interfaces of the software component, control the data exchange method, and set various definitions regarding AUTOSAR

Note

You must have an AUTOSAR license to access the application.

From the Rational Statemate application, select **Tools > Statemate AUTOSAR Generator** click **AUTOSAR** on the toolbar. The AUTOSAR application opens (see the following figure):



Overview of the AUTOSAR Interface

This section provides an overview of the AUTOSAR interface, including menu and toolbar options.

AUTOSAR Menus

The following table describes the AUTOSAR menus:

Menu	Options	Description
File	New Profile	Opens The New Profile dialog box.
	Open Profile	Opens an existing profile.
	Close	Closes the open profile.
	Save	Saves changes to the profile.
	Save as	Saves as a new file.
	Print Profile Report	Prints the profile report.
	Profile Management	Opens SAG Management dialog box.
	Exit	Closes AUTOSAR.
Edit	Rename Software Component	Renames a component.
	Rename Internal Behavior	Renames an IB.
	Add to Scope	Adds component to scope.
	Remove from Scope	Removes a component from scope.
	Select	
View	Messages	Allows you to view or hide the messages displayed at the bottom of the Rational Statemate AUTOSAR Generator dialog box.
	Tool Bar	Allows you to view or hide the Rational Statemate AUTOSAR Generator tool bar.
Compile	Generate Code	Gnerates AUTOSAR Software Component Code and XML.
	File Management	Opens a menu for Showing, Deleting, Copying, Exporting, and Printing the generated files.
	Edit	Opens the generated code folder.
Options	Set Time Configuration	Opens the Timer Configuration dialog box.
	Settings	Opens the Properties for Code Generation dialog box.

Menu	Options	Description
Help	List of Books Index Contents on Help on Window	Opens the help system.

AUTOSAR Toolbar Options

The following table lists the AUTOSAR toolbar options:

Toolbar Icon	Description
	Add Chart - Click to add an Activity Chart to the scope.
	Remove Selected Object - Click to remove selected object from tree view.
	Generate Code - Click to generate code.

Scope Definition Area

The Scope Definition area supports one AUTOSAR Atomic Software Component Type (SWC). There is always one SWC when opening the profile.

Under a SWC, as part of it, there is one instance of Internal-Behavior (IB). There is always one IB instance when opening the profile.

Note

The name of the SWC and the IB is case-sensitive and may be renamed accessing the right-click pop-up menu. By default, the name of the SWC is atomicSoftwareComponent_1 and the name if the IB is internalBehavior_1.

The following design-attributes for an Internal-Behavior are accessed using the right-click popup menu):

- **Multiple Instantiation** - Defines whether the Software Component described by this Internal Behavior is Multiple Instantiated or not. This determines whether the RTE APIs should use the instance parameter or not.
- **Group I/O Elements to a Single Port** - When set to 'yes', a single port is created for all Input elements of the same R.E. Another port is created for all Output elements of the same R.E. that the “On Interface” is set to “yes”. When set to 'no', each I/O element (i.e., each Input line or Output line) will define a separate port and relating interface.
- **Runnables Entities run in Exclusive Area** - When set to 'yes', all Runnable Entities in the scope will be defined as “Run in Exclusive Area”. In this case, an additional Design Attribute, “Exclusive Area Name”, is opened for setting the Exclusive Area's Name. The default name is `Exclusive_Area`.
- **Use '&' For Input Array Parameter** - Defines an IN parameter of type array shall be passed using '&' or not.
- **Use '&' For Output Array Parameter** - Defines an OUT parameter of type array shall be passed using '&' or not.

You should add to the scope the Charts containing Activities tagged as Runnable Entities.Those Activities that are Runnable Entities define the implementation of the Internal-Behavior instance of the regarded SWC. After adding the Charts and Activities (that are defined as Runnable Entities) to the scope, double-click on any of the Runnable Entities in the Scope Definition area to display the input and output flowing elements in the lower half of the screen.

The lower half of the screen displays the Inputs (Left hand side) and the Outputs (Right hand side) handled by each of the Runnable Entities. This is referred to as the “I/O Definition Area”. The I/O Definition Area contains two tabs: "Main" and "Client/Server"

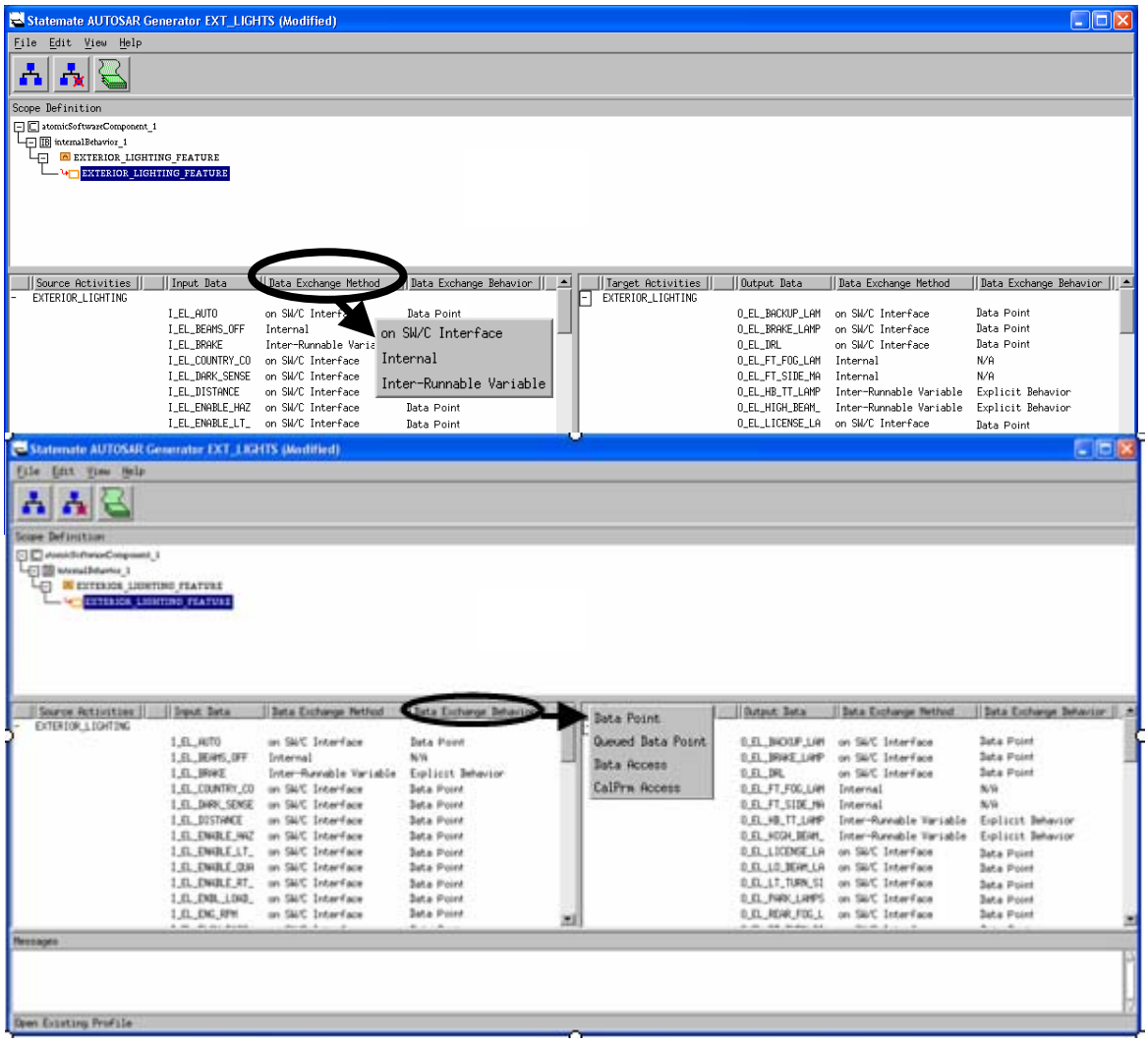
The "Main" tab

There are four columns in the the "Main" tab:

- **Source/Target Activities (Read-Only)** - displays the selected Activity, representing a Runnable entity.
- **Input/Output Data (Read-Only)** - The input/output Flow Line to/from the selected Activity. From SAG point of view, the Flow Line represents all the Data Elements (Data Items, Conditions and Events) that it contains, and all the definitions that are set for a the Flow line will effect its Data Elements also.

  SAG uses Flow-Lines for defining a Require-Ports (for input) or Provide-Ports (for output), with matching Sender-Receiver Interfaces. More specifically, if an Info-Flow flows on the Flow-Line, the Info-Flow is presented, and a proper interface is defined on a port carrying the same name. When a Data-Item, Condition or Event flows on the Flow-Line, it defines the interface of the relating port.
  The user may use an external Interface to type a Port, by setting an interface reference to the design-attribute External Interface Reference, available for Info-Flows (for S-R Interface) and Subroutines (C-S Interface). In this case, the Interface will not be defined in the XML, and the port will refer to the provided reference.
  The Design-Attribute "Its Port" of Data-Item, Condition or Event may be used to determine the Port's name. If the value of the Design-Attribute "Its Port" is empty or "none", the Port name will be determined by the Flow-Line label.
- **Data Exchange Method** - This field can get one of the following values:
  - **On SW/C Interface (default)** - Notifies that the data is part of the Interface of the Software Component.
  - **Internal** - Notifies that the data is internal to the model.
  - **Inter-Runnable Variable** - Notifies that the data is used as part of Inter-Runnable Variables.
  - **By Design Attribute** - Notifies that the Data Exchange Method is specifically set to each Data-Item, Condition or Event of this port, via the Design-Attribute "Data Exchange Method".

See the following figure:



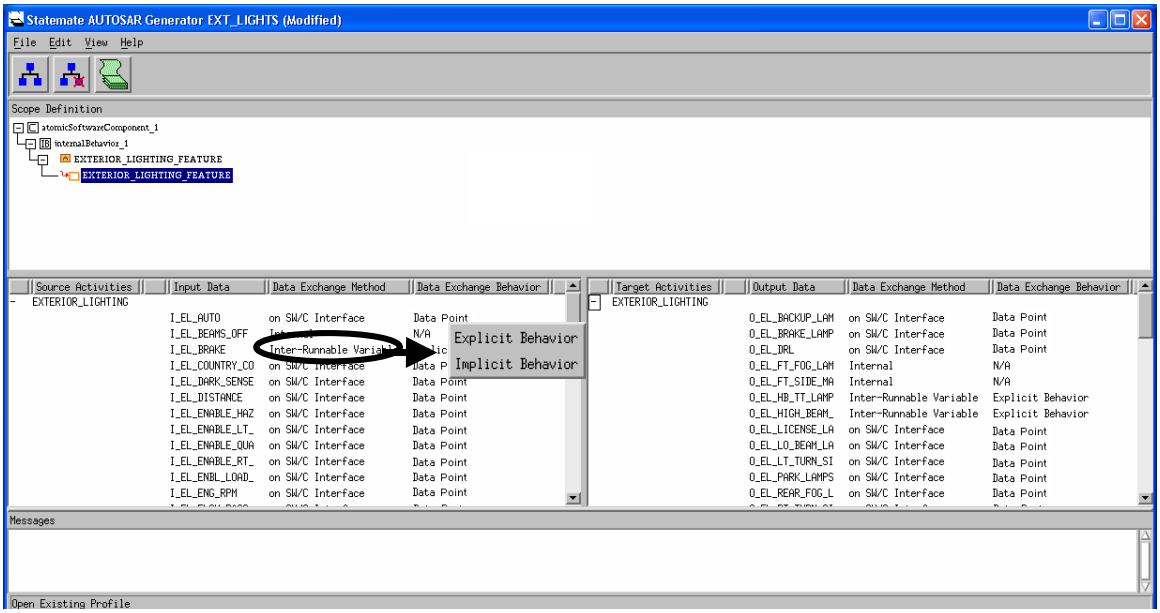
- **Data Exchange Behavior** - This field can get one of the following values:
  (When Data Exchange Method is set to On SW/C Interface - see the following figure):
  - **Data Point (Default)** - Notifies that the data is accessed by a Data Receive Point (for input) or a Data Send Point (for output), and is on a Sender-Receiver Interface.
  - **Queued Data Point** - Notifies that the data is queued and accessed by a Data Receive Point or a Data Send Point, and is on a Sender-Receiver Interface.
  - **Data Access** - Notifies that the data is accessed by a Data Read Access (for input) or a Data Write Access (for output), and is on a Sender-Receiver Interface.
  - **CalPrm Access** - Notifies that the data is accessed by a CalPrm Access , and is on a CalPrm-Interface. For Input (left-side) elements only.

- **By Design Attribute** - Notifies that the Data Exchange Behavior is specifically set to each Data-Item, Condition or Event of this port, using the Design-Attribute "Data Exchange Behavior"

When Data Exchange Method is set to Internal: N/A.

When Data Exchange Method is set to Inter-Runnable Variable (refer to the following figure):

- **Explicit Behavior (default)** - Notifies that the Inter-Runnable Variable has explicit behavior.
- **Implicit Behavior** - Notifies that the Inter-Runnable Variable has implicit behavior.
- **By Design Attribute** - Notifies that the Data Exchange Behavior is specifically set to each Data-Item, Condition or Event of this port, using the Design-Attribute "Data Exchange Behavior"



The following table lists the RTE APIs that are used in the different cases:

Data Exchange Method	On SW/C Interface				
Data Exchange Behavior	Data Point	Queued Data Point	Data Access	CalPrm Access	Mode Declaration Group
Input	Rte_Read	Rte_Send	Rte_IRead	Rte_Calprm	Rte_Mode
Output	Rte_Write	Rte_Receive	Rte_IWrite	-	Rte_Switch
Data Exchange Method	Inter-Runnable Variable				
Data Exchange Behavior	Explicit Behavior			Implicit Behavior	
Input	Rte_IrvRead			Rte_ IrvIRead	
Output	Rte_IrvWrite			Rte_ IrvIWrite	

The "Client/Server" tab

The "Client/Server" tab consists of two parts, the upper part is for adding and defining Required Ports (input, or Client) and the lower part is for Provided-Ports (output, or Server).

Both the Provided Part and the Required Part contain of the following columns:

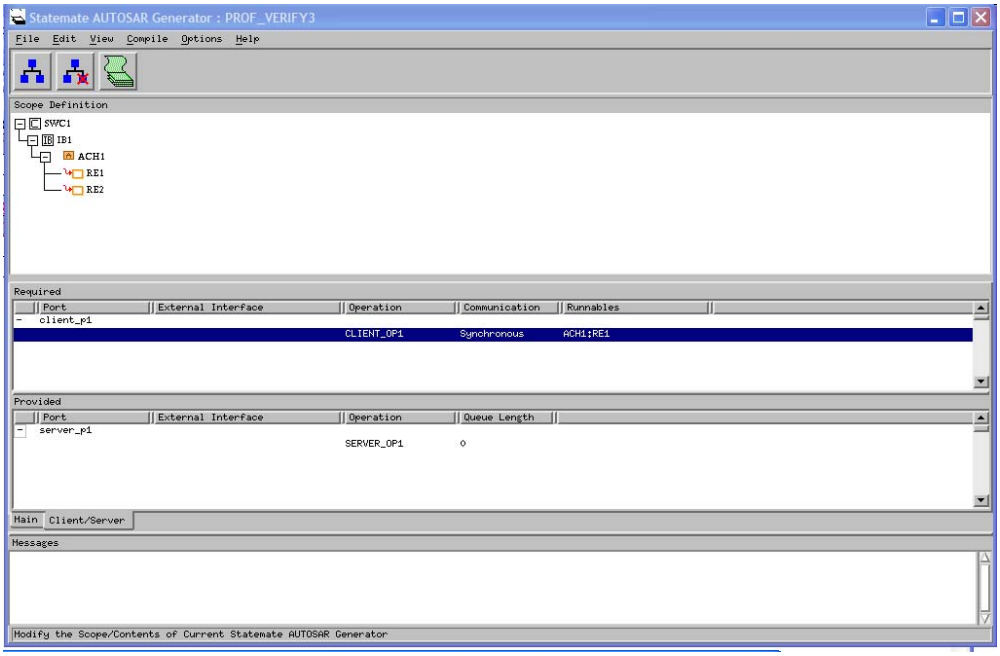
- **Port** - Allows the user to create a new Provided/Required Port, to be typed by a Client-Server Interface.
- **External Interrface** - Allows the user to select an externally define Client-Server Interface to type the current Port. If the user selects "none", a new Interface will be defined in the generated ARXML.
- **Operation** - Allows the user to select one or more operations from the Statemate model, to be added to the Client-Server Interface.

The Required part contains of the following additional columns:

- **Communication** - Specifies the communication model - Synchronous or Asynchronous (Currently only Synchronous mode is supported by SAG).
- **Runnables** - Specifies the list of Runnable Entities that access a certain operation.

The Provided part contains of the following additional column:

- **Queue Length** - Specifies the list of size of the queue to be used for a certain operation.


Statemate AUTOSAR Generator : PROF_VERIFY3
File Edit View Compile Options Help
Scope Definition
SWC1
IB1
ACH1
RE1
RE2
Required
Port
External Interface
Operation
Communication
Runnables
client_p1
CLIENT_OP1
Synchronous
ACH1:RE1
Provided
Port
External Interface
Operation
Queue Length
server_p1
SERVER_OP1
0
Main
Client/Server
Messages
Modify the Scope/Contents of Current Statemate AUTOSAR Generator

Generating Code and XML Description

The tool generates code using MicroC code generator.

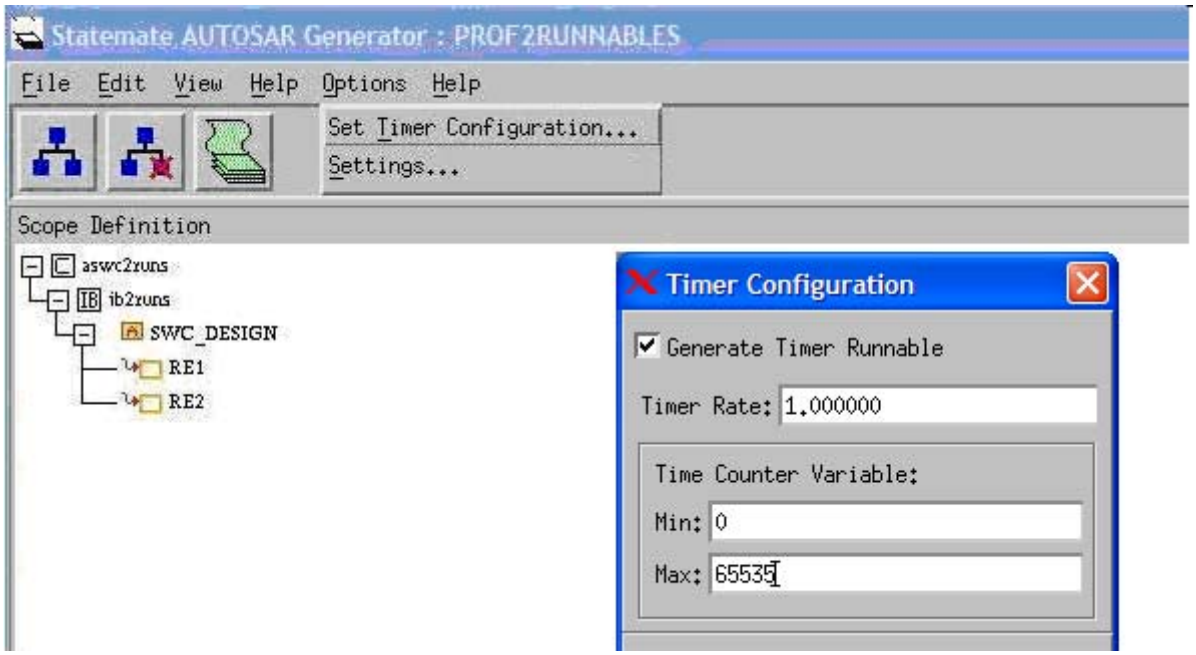
- A source file and a header file that are the implementation of the Software Component. The files are generated in: <WA>/prt/<profile-name>.
- Options on the generated code may be controlled via **Options > Setting.**

The tool generates an AUTOSAR XML file. This is done using the Static OS configuration APIs of the OSDT. Some naming conventions are used. For example, Provide/Require Interfaces - Are yield from the ports' names with the postfix “i”.

Timeouts

Timeouts are a Rational Statemate feature of SAG. In order to handle timeouts, SAG generates an additional Runnable Entity that functions as a time counter. This Runnable Entity, called TIME_COUNTER, will be activated periodically by a dedicated Timing Event. TIME_COUNTER increments an Inter-Runnable-Variable, called CounterTime, on every run.

You can control the creation of the Timer Runnable Entity and set the rate period of its Timing Event, using the GUI menu **Options >Set Timer Configuration**. See the following figure:



The timer Runnable Entity is generated under the following conditions:

- The check box “Generate Timer Runnable” is checked.
- The model uses timeouts.

The following example is the Runnable Entity's entry function implementation in usercode.c:

```
void
TIME_COUNTER(void)
{
        uint16 curTime;

        Rte_Enter_Timer_Exclusive_Area();
        curTime = Rte_IrvRead_TIME_COUNTER_CounterTime();
        Rte_IrvWrite_TIME_COUNTER_CounterTime(curTime++);
        if(curTime == 0){
                onCounterTime_OVERFLOW();
        }
        Rte_Exit_Timer_Exclusive_Area();
}
```

SAG allows the user to define the Defining Minimum and Maximum values for the CounterTime variable, using the dialog options->Set Timer Configuration.

In-Out Elements

SAG automatically identifies Data Elements that are used both as inputs, i.e. on incoming info-flow representing an AUTOSAR Required-Port, and as outputs, i.e. on outgoing info-flow representing an AUTOSAR Provided-Port.

Input access of such Data Elements will be implemented regularly, meaning via a Sender-Receiver Interface using the appropriate RTE API's. Output Access of such Data Elements will be implemented using an operation named Write<DataName>, on a Client-Server Interface typing a dedicated Required-Port.

All such operations will be gathered into a single Port and a single Interface. The Port's name may be customized using the OSDT API: Code Style->Type Naming Style->Implicit (Generated) R-Port Name.

SAG Implementation of AUTOSAR Features

The following sections describe the AUTOSAR Features:

- **Exclusive Areas**
- **Timing Events**
- **Data Types**
- **Services**
- **Data Send Points, Data Receive Points, Data Read Access, Data Write Access**
- **Inter Runnable Variables**
- **Mode Declaration Groups**

Exclusive Areas

SAG uses up to three Exclusive Areas:

- **Model_Exclusive_Area** - for general usage in the generated code of the model.
- **Timer_Exclusive_Area** - created when the Timer Runnable Entity is generated and used only by it (see **Timeouts**).
- **Exclusive_Area** (rename-able) - created when Runnable Entities run in Exclusive Area is set to 'yes' for the Internal-Behavior.

Timing Events

SAG uses two types of Timing Events:

- **TIME_COUNTER_TMEV** - created when the Timer Runnable Entity is generated and used only by it (see **Timeouts**).
- **Specific Timing Event** - for each Runnable Entity that has its Design-Attribute “Schedule Periodically” set to `yes`. You may define the period of the Runnable Entity, using the Design-Attribute `Period`. The name of the Timing Event is “<Runnable Entity Name>_TMEV”.

The following is an example of XML description of Timing Event:

```
<TIMING-EVENT>
  <SHORT-NAME>RE2_TMEV</SHORT-NAME>
  <START-ON-EVENT-REF DEST="RUNNABLE-ENTITY">/../../RE2</START-ON-EVENT-REF>
  <PERIOD>546</PERIOD>
</TIMING-EVENT>
```

Data Types

SAG supports only basic Data Types for SW/C Interface items and Inter-Runnable Variables. The supported Data Types are:

- INTEGER-TYPE
- REAL-TYPE
- CHAR-TYPE
- STRING-TYPE
- BOOLEAN-TYPE
- OPAQUE-TYPE
- ARRAY-TYPE
- RECORD-TYPE

The types will be declared in the SWC description file (XML) according to the following rules:

- By Default, each Data-Element Definition in the XML defines one Data-Type declaration, named `<Data Element Name>_type`.
- In order to have more than one Data-Element using the same Data-Type, those Data-Elements must be typed using a Rational Statemate User-Defined-Type (UDT).
- In order to use an externally defined type, AUTOSAR basic type or other, you can provide the full path to the external type, using the Design-Attribute `AR Type Reference`, available for Data-Items, Conditions, Events, and Subroutine-Parameters.
- A single Boolean type will be defined in the XML for all Conditions and another single Boolean type will be defined for all Events.
- SAG supports automatic linkage to external type for to Data-Elements that answer the following requirements:
    - o Have no specific minimum or maximum values.
    - o Have the Design Attribute "C Data Type" set to a value.
    - o In order to map the value of C Data Type to an AUTOSAR type, it must be written within the body of one of the API's at the OS Definition Tool->API Definition->AUTOSAR Types API's.

Example 1:Default behavior:

Data Element Prototype Declaration:

```
<DATA-ELEMENT-PROTOTYPE UUID="DCE:59700696-b7a3-468d-9f76-79e3948493dc">
    <SHORT-NAME>Data1</SHORT-NAME>
    <TYPE-TREF DEST="INTEGER -TYPE">/prof2res/ Data1_type</TYPE-TREF>
```

```
    <IS-QUEUED>false</IS-QUEUED>
</DATA-ELEMENT-PROTOTYPE>
```

Data Type Declaration:

```
<INTEGER-TYPE UUID="DCE:98823982-dbbe-4e7d-8f32-830a9fc39719">
    <SHORT-NAME>Data1_type </SHORT-NAME>
    <LOWER-LIMIT INTERVAL-TYPE="CLOSED">0</LOWER-LIMIT>
    <UPPER-LIMIT INTERVAL-TYPE="CLOSED">255</UPPER-LIMIT>
</INTEGER-TYPE>
```

Example 2- Using a UDT:

Data Elements Prototype Declaration:

```
<DATA-ELEMENT-PROTOTYPE UUID="DCE:59700696-b7a3-468d-9f76-79e3948493dc">
    <SHORT-NAME>Data1</SHORT-NAME>
    <TYPE-TREF DEST="INTEGER -TYPE">/prof2res/ MY_TYPE1</TYPE-TREF>
    <IS-QUEUED>false</IS-QUEUED>
</DATA-ELEMENT-PROTOTYPE>

DATA-ELEMENT-PROTOTYPE UUID="DCE:59700696-b7a3-468d-9f76-79e3948493dc">
    <SHORT-NAME>Data1</SHORT-NAME>
    <TYPE-TREF DEST="INTEGER -TYPE">/prof2res/ MY_TYPE1</TYPE-TREF>
    <IS-QUEUED>false</IS-QUEUED>
</DATA-ELEMENT-PROTOTYPE>
```

Data Type Declaration:

```
<INTEGER-TYPE UUID="DCE:98823982-dbbe-4e7d-8f32-830a9fc39719">
    <SHORT-NAME>/ MY_TYPE1</SHORT-NAME>
    <LOWER-LIMIT INTERVAL-TYPE="CLOSED">0</LOWER-LIMIT>
    <UPPER-LIMIT INTERVAL-TYPE="CLOSED">255</UPPER-LIMIT>
</INTEGER-TYPE>
```

Example 3 - Using an externally defined type:

Data Element Prototype declaration:

```
<DATA-ELEMENT-PROTOTYPE UUID="DCE:43981504-1a29-42bb-8ddc-998a3c4c08d2">
    <SHORT-NAME>DI_IN1</SHORT-NAME>
    <TYPE-TREF DEST="INTEGER-TYPE">/Autosar/UInt8</TYPE-TREF>
    <IS-QUEUED>false</IS-QUEUED>
</DATA-ELEMENT-PROTOTYPE>
```

Note

The Data Type is not declared in the xml.

Services

SAG supports accessing AUTOSAR Services using Client-Server Interfaces. In order to access Services, you may define a Rational Statemate Subroutine with the following characteristics:

- External implementation.
- Design Attribute "Is Service" = "yes".
- Design Attribute "Its Port", free text, defining the name of the R-Port.

As a result, in the SWC Description (XML) file SAG generates A Require-Port, named after the “Its Port” attribute, referring to a Client-Server Interface. And a Client-Server Interface, named “i<Port Name>”, and tagged as Service. This Interface contains Operations based on the definition of Rational Statemate Subroutines, which belong to that Port/Interface, including Arguments definitions.

Example:

```
<CLIENT-SERVER-INTERFACE UUID="DCE:bec17ae8-ca1a-4048-a11a-3d8b10252a3b">
 <SHORT-NAME>iservice1Port</SHORT-NAME>
 <IS-SERVICE>true</IS-SERVICE>
 <OPERATIONS>
   <OPERATION-PROTOTYPE UUID="DCE:5f150618-6f28-4b7a-97cd-b530c456e465">
    <SHORT-NAME>SERVICE1</SHORT-NAME>
    <ARGUMENTS>
     <ARGUMENT-PROTOTYPE UUID="DCE:2af69675-924f-45fb-b45a-e00b1626e6d7">
      <SHORT-NAME>IN_PAR1</SHORT-NAME>
      <TYPE-TREF DEST="INTEGER-TYPE">/Autosar/UInt16</TYPE-TREF>
```

```
      <DIRECTION>IN</DIRECTION>
     </ARGUMENT-PROTOTYPE>
     <ARGUMENT-PROTOTYPE UUID="DCE:d36f528f-a935-4b82-92c0-6a24b0e7a35d">
     <SHORT-NAME>OUT_PAR2</SHORT-NAME>
     <TYPE-TREF DEST="REAL-TYPE">/prof2res/MY_REAL_TYPE</TYPE-TREF>
     <DIRECTION>OUT</DIRECTION>
    </ARGUMENT-PROTOTYPE>
    <ARGUMENT-PROTOTYPE UUID="DCE:8b89a05d-8559-4f2f-8b61-fd24079863c9">
     <SHORT-NAME>INOUT_PAR3</SHORT-NAME>
      <TYPE-TREF DEST="OPAQUE-TYPE">/prof2res/INOUT_PAR3_type</TYPE-TREF>
     <DIRECTION>INOUT</DIRECTION>
    </ARGUMENT-PROTOTYPE>
   </ARGUMENTS>
  </OPERATION-PROTOTYPE>
 </OPERATIONS>
<POSSIBLE-ERRORS/>
</CLIENT-SERVER-INTERFACE>
```

Data Send Points, Data Receive Points, Data Read Access, Data Write Access

As mention in the GUI and common-workflows section, you may select Data to be used on the Software component Interface as queued or non-queued Data Send Point (Explicit Write) or Data Receive Point (Explicit Read), or as Data Read Access (Implicit Read) or Data Write Access (Explicit Write), or as CalPrm Access.

This results in corresponding definitions in the XML and generated code. See the following examples.

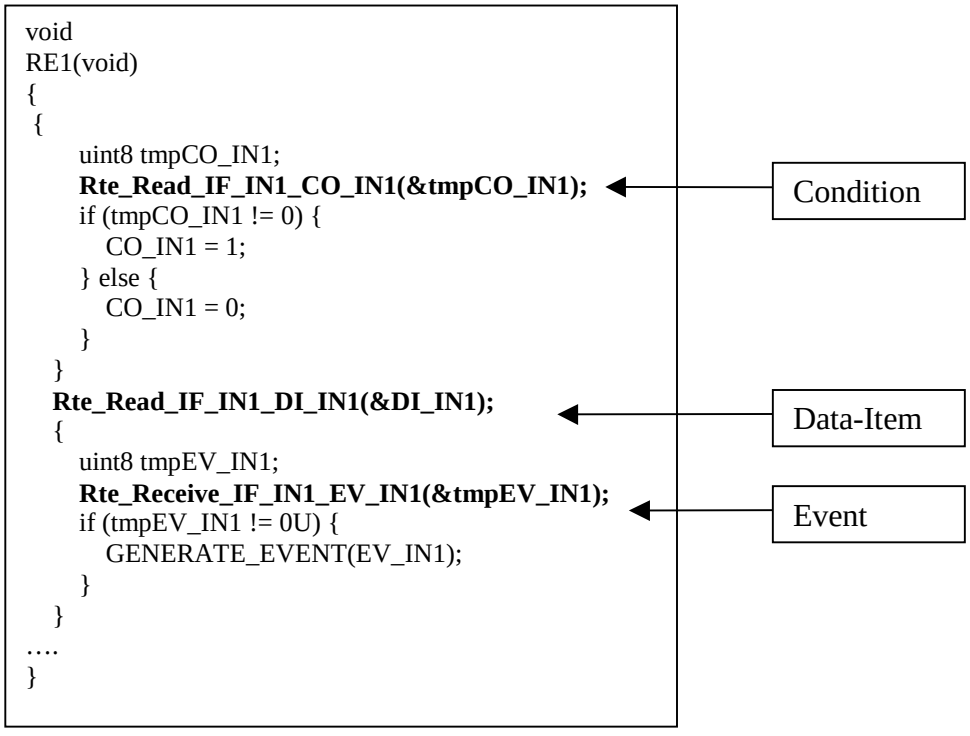
Each of these five Data Exchange Behavior options may be implemented as one of two Statemate MicroC Access Modes: "Buffered" or "Direct Using Get/Set API's". This is configured with a corresponding set of Design-Attributes on the Internal-Behavior (Default behavior in braces):

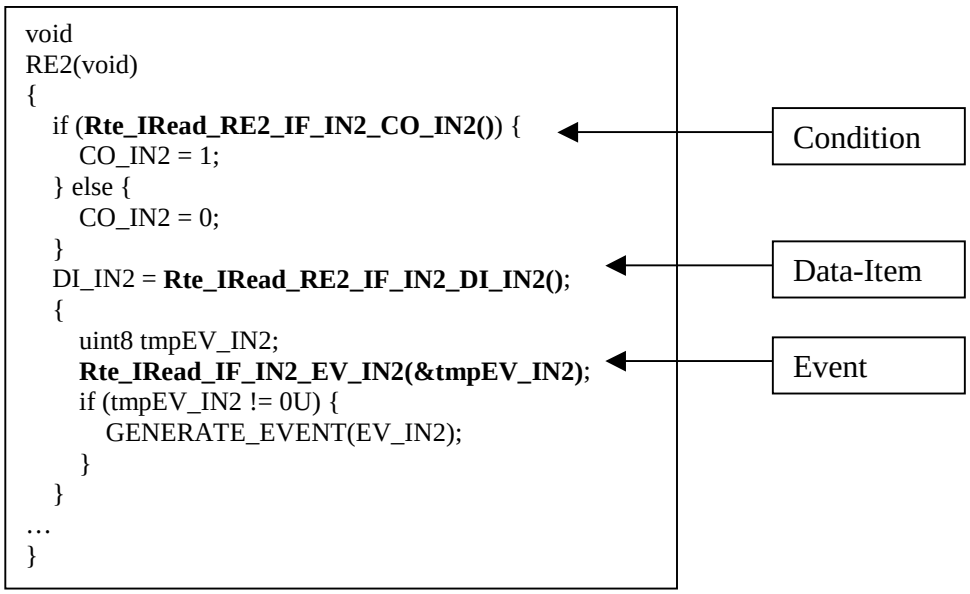
- CalPrm Access Mode (Buffered)
- Explicit Read Access Mode(Buffered)
- Implicit Read Access Mode (Direct Using Get/Set API's)
- Explicit Write Access Mode(Direct Using Get/Set API's)
- Implicit Write Access Mode (Direct Using Get/Set API's)

Each of these Design-Attributes may be set with the following 3 values:

- Buffered Access
- Direct Using Get/Set API's Access
- By Design Attribute - When "By Design Attribute" is selected, SAG will refer the Design-Attribute "Double Buffered" in the following way:
    - o When "Double Buffered" is set to "yes", Buffered access will be used.
    - o When "Double Buffered" is set to "no", Direct Using Get/Set API's access will be used.

In case of using Explicit Read/Write, you may want to assign the return value of the RTE API assign to a variable. This can be achieved by using a Design-Attribute "RTE Status Variable Name" available for Data-Item, Condition and Event. A non-empty value of this Design-Attribute specifies the name of a variable that will be assigned with the return value of the Explicit Read/ Write RTE API's. Its default value is empty string. It is the user's responsibility to define such variable (Data-Item) in the model , with the correct settings, e.g.: Exact Type = Std_ReturnType.




void
RE2(void)
{
if (Rte_IRead_RE2_IF_IN2_CO_IN2()) {
CO_IN2 = 1;
} else {
CO_IN2 = 0;
}
DI_IN2 = Rte_IRead_RE2_IF_IN2_DI_IN2();
{
uint8 tmpEV_IN2;
Rte_IRead_IF_IN2_EV_IN2(&tmpEV_IN2);
if (tmpEV_IN2 != 0U) {
GENERATE_EVENT(EV_IN2);
}
}
…
}
Condition
Data-Item
Event

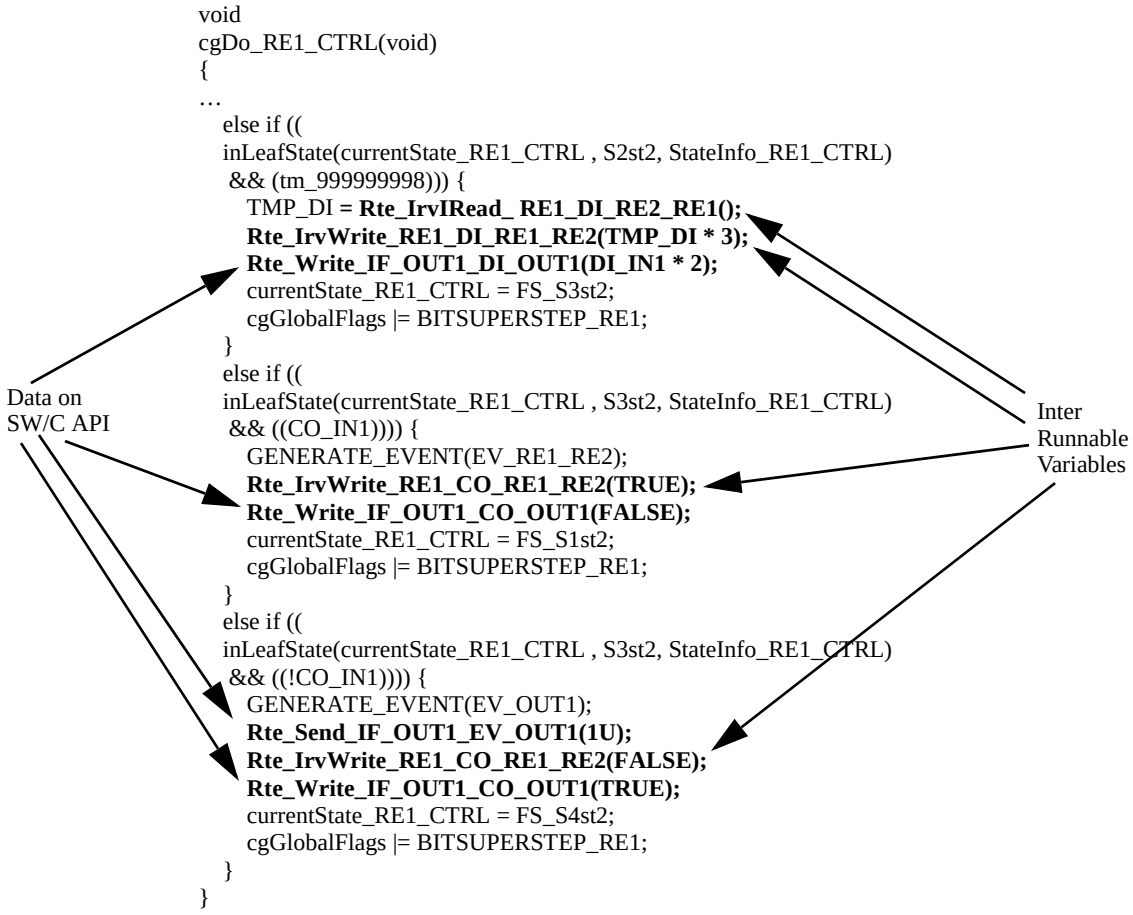
Inter Runnable Variables

You may select Data to be used as Inter Runnable Variables. This results in corresponding definitions in the XML and generated code. Below is an example displays a Data Exchange in Statechart code:

Each of Explicit or Implicit IRV behavior options may be implemented as one of two Statemate MicroC Access Modes: "Buffered" or "Direct Using Get/Set API's". This is configured with a corresponding set of Design-Attributes on the Internal-Behavior (Default behavior in braces):

- "Explicit IRV Access Mode (Direct Using Get/Set API's)
- "Implicit IRV Access Mode (Direct Using Get/Set API's)

For possible values of these Design-Attributes, see previous sections.



Mode Declaration Groups

- Mode Declaration Group will be implemented by SAG using an enumerated User Defined Type with Design-Attribute "Mode Declaration Group" set to "yes", and having a data item of this type flowing on a certain info flow.
- "An input data will be accessed via the RTE API Rte_Mode_...() and output via the API Rte_Switch_...()
- "The XML will include definitions of ModeDeclarationGroup and ModeDeclaration. In addition, it will define ModeSwitchPoint and ModeSwitchComSpec for output data.

The AUTOSAR RTE OS ImplementationsI

The OSI's "autosar_rte_210" and "autosar_rte_310" are used to creating Rational Statemate AUTOSAR models. These OSI's include definitions of the various APIs and XML definitions (using the "API Definitions" and the "Static OS Configuration" capabilities). In addition, the OSI's defines the set of design-attributes for the relevant Rational Statemate types (Activities, Data-Items, Conditions, etc.) for complying with AUTOSAR entities.

Important Notes

- An Atomic Software Component contains one Internal Behavior.
- Client Server Interfaces are not supported.
- Concurrent invoking of Runnable Entities is not supported.
- Allow-NAN for Real Data Types is not supported.

Creating an AUTOSAR Project

To create an AUTOSAR project, complete the following:

1. Select **File > New**.
2. In the New Project dialog:
   a. Specify the project's name.
   b. Specify the databank location for the project.
   c. Select an **AUTOSAR_RTE_OSI** from the **OS Implementation** list.
   d. Click **OK**.

Technical Support

All IBM® Rational® Statemate® customers receive support from IBM Rational Software Support and resources.

Contacting IBM Rational Software Support

If the self-help resources have not provided a resolution to your problem, you can contact IBM Rational Software Support for assistance in resolving product issues.

Prerequisites

To submit your problem to IBM Rational Software Support, you must have an active Passport Advantage® software maintenance agreement. Passport Advantage is the IBM comprehensive software licensing and software maintenance (product upgrades and technical support) offering. You can enroll online in Passport Advantage from http://www.ibm.com/software/lotus/passportadvantage/howtoenroll.html.

- To learn more about Passport Advantage, visit the Passport Advantage FAQs at http://www.ibm.com/software/lotus/passportadvantage/brochures_faqs_quickguides.html.
- For further assistance, contact your IBM representative.

To submit your problem online (from the IBM Web site) to IBM Rational Software Support, you must additionally:

- Be a registered user on the IBM Rational Software Support Web site. For details about registering, go to http://www.ibm.com/software/support/.
- Be listed as an authorized caller in the service request tool.

Contacting Support

To contact IBM Rational Software Support:

1. Locate your **ICN** (IBM customer number). It is required for support requests.
2. Determine the business impact of your problem. When you report a problem to IBM, you are asked to supply a severity level. Therefore, you need to understand and assess the business impact of the problem that you are reporting.

   Use the following table to determine the severity level.

Severity	Descriptions
1	The problem has a *critical* business impact: You are unable to use the program, resulting in a critical impact on operations. This condition requires an immediate solution.
2	This problem has a *significant* business impact: The program is usable, but it is severely limited.
3	The problem has *some* business impact: The program is usable, but less significant features (not critical to operations) are unavailable.
4	The problem has *minimal* business impact: The problem causes little impact on operations or a reasonable circumvention to the problem was implemented.

3. Describe your problem and gather background information, When describing a problem to IBM, be as specific as possible. Include all relevant background information so that IBM Rational Software Support specialists can help you solve the problem efficiently. To save time, know the answers to these questions:

   - What software versions were you running when the problem occurred?

     To determine the exact product name and version, use the option applicable to you:

     – Start the IBM Installation Manager and choose **File** > **View Installed Packages**. Expand a package group and select a package to see the package name and version number.

     – Start your product, and choose **Help** > **About** to see the offering name and version number.
   - What is your operating system and version number (including any service packs or patches)?
   - Do you have logs, traces, and messages that are related to the problem symptoms?
   - Can you recreate the problem? If so, what steps do you perform to recreate the problem?

- Did you make any changes to the system? For example, did you make changes to the hardware, operating system, networking software, or other system components?
- Are you currently using a workaround for the problem? If so, be prepared to describe the workaround when you report the problem.

4. Submit your problem to IBM Rational Software Support. You can submit your problem to IBM Rational Software Support in the following ways:

   - **From the Support Web site**: Go to the IBM Rational Software Support Web site at https://www.ibm.com/software/rational/support/ and in the Rational support task navigator, click **Open Service Request.** Select the electronic problem reporting tool, and open a Problem Management Record (PMR), describing the problem accurately in your own words.
   - **Request assistance through e-mail**: send the e-mail to the support address for your region:

     `sw_support_emea@nl.ibm.com`

     `sw_support@us.ibm.com`

     `sw_support_ap@au1.ibm.com`
   - For more information about opening a service request, go to http://www.ibm.com/software/support/help.html
   - You can also open an online service request using the IBM Support Assistant. For more information, go to http://www.ibm.com/software/support/isa/faq.html.
   - **By phone**: For the phone number to call in your country or region, go to the IBM directory of worldwide contacts at http://www.ibm.com/planetwide/ and click the name of your country or geographic region.
   - **Through your IBM Representative**: If you cannot access IBM Rational Software Support online or by phone, contact your IBM Representative. If necessary, your IBM Representative can open a service request for you. You can find complete contact information for each country at http://www.ibm.com/planetwide/.

If the problem you submit is for a software defect or for missing or inaccurate documentation, IBM Rational Software Support creates an Authorized Program Analysis Report (APAR). The APAR describes the problem in detail. Whenever possible, IBM Rational Software Support provides a workaround that you can implement until the APAR is resolved and a fix is delivered. IBM publishes resolved APARs on the IBM Rational Software Support Web site daily, so that other users who experience the same problem can benefit from the same resolution.

Reporting Rational Statemate Problems from the Software

When Rational Statemate is running, you may want to use the problem reporting facility from the Statemate Help menu.

To send an automated problem report:

1. In Rational Statemate, choose **Help > Generate Support Request** to open the Generate Support Request dialog box.
2. Review the **Statemate Information** and **System Information** areas to verify their accuracy.
3. From the **Impact** drop-down list box, select the severity of the problem.
4. In the **Summary** box, summarize the problem.
5. In the **Problem** box, type a detailed description of the problem.
6. If possible, take a snapshot of the problem and attach it to the problem report. Click the **Statemate Window Snapshot** button or **Screen Snapshot** button, whichever is applicable, and select the snapshot file from wherever you have it on your machine.
7. If possible, add the model, active component, files, and/or a video capture by using the buttons in the **Attachment Information** area.
8. Include an item description for each item in the **Attachment Information** area, if needed.
9. Click **Preview and Send** to submit the report.

The problem report is recorded in the Rational Statemate case tracking system and put into a queue to be assigned to a support representative. This representative works with you to be certain that your problem is solved.

Note

If your Rational Statemate system crashes, it displays a message asking if you want to send a problem report to technical support about this crash. If you select to send the report, the system displays the same online form that is available from **Help > Generate Support Request**. However, this form contains information about the crash condition in addition the information that is usually filled in describing your system. Add any more information that you can to help the support staff identify the problem and then click **Preview and Send** to submit the report.

Glossary of Rational Statemate Terminology

a-flow line

Refers to any **flow line** found in an **activity chart** (either **control-flow** or **data-flow**). This term is used by the **properties**, **Documentor**, and **Dataport**.

access level

A mask that protects **configuration item**s within the **databank** according to your **preferences**.

The possible values are as follows:

- **None** - The item cannot be checked out.
- **Read-only** - The item can be checked out without a **lock** and used or modified, but cannot be checked in.
- **Update** - The item can be checked out with a lock (which guarantees that other users cannot change it while you are working on it) and checked in.

You can assign an access level to the following groups:

- **Owner** - The **project member** who first created the configuration item in the databank.
- **Group** - Project members who belong to the same group as the owner (according to the definitions set by the operating system).
- **Others** - All project members.

The databank directory structure is created with “world read/write” access, but specific items within the structure can be protected according to your preferences. All versions of a configuration item belong to the same owner and have the same access level.

See also **read-only editing mode**.

action

Any task done as a result of one of the following:

- Taking a **transition** in a **statechart**
- Executing a **static reaction** within a **state**
- Executing a **mini-spec** within an **activity**, **truth table**, or **subroutine**

A single action can consist of making an assignment, generating an **event**, invoking a defined (named) action, or several special types of expressions (starting, stopping, or suspending activities; clearing history; and so on).

action expression

One or more **action**s executed on the same **transition**, **static reaction**, or **mini-spec**. The actions can be listed sequentially, in which case they are separated by semicolons, or nested, as the required logic dictates. All actions on the same transition/static reaction/mini-spec occur simultaneously.

In addition, special **context variable**s are available to enable the definition of actions that are executed sequentially instead of simultaneously. These context variables are prefaced by a dollar sign ($).

action language

A **subroutine** written using the standard Statemate **action** statements. Normally, action statements are executed within the context of Statemate semantics (for multiple action statements within a single **step**, all the assignments occur at the same time). Within an action language subroutine, all the assignments occur in the order the statements are written.

activity

The primary graphical object in an **activity chart** that represent a **function** in the functional view of the system. An activity represents something that transforms input into output.

There are three types of activities:

- Internal activities (shown as a solid rectangle)
- External activities (shown as a dashed rectangle)
- Control activities (shown as a rounded rectangle)

Activities can be allocated to **module**s (structure) and can contain **statechart**s. You can specify the behavior of an activity by connecting it to a **subroutine**, as follows:

- Procedure-like activities can be connected to **procedure**s within any of the supported languages.
- Internal primitive activities (**reactive-controlled** and **reactive-self**) can be connected to **task**s (no **mini-spec**s or decomposition is allowed).
- External activities can be connected only to tasks.

activity termination type

Applies only when **Activity Style** is set to **Software style**. This means that activities in your **model** can be started and stopped. The alternative is **Hardware style**, in which activities are always active. This affects **simulation**, **code generation**, and **Check Model**.

There are three possible activity termination types:

- **Procedure-like** - When started, the **activity** will run to completion in a single **step**. This type can contain only **mini-spec**s.
- **Reactive-controlled** - The activity will be stopped by the **control activity** that started it. This type can contain either mini-specs or control activities.
- **Reactive-self** - The activity self-terminates at an appropriate time. This type can contain either mini-specs or control activities.

All three types of activities are started by their respective control activity.

activity chart

Describes the functional view of the system using activities as the primary building block. A system description can contain one or more activity charts. Activity charts, which may be connected to **module-chart**s, describe the functionality of individual modules. Activity charts can be hierarchically connected to other activity charts.

In addition, activity charts can be connected to **statechart**s. Statecharts either define the behavior of individual activities or control groups of activities like a **control activity**.

See also **activity chart graphic editor (AGE)**.

activity chart graphic editor (AGE)

The graphical editing tool used to create and edit **activity chart**s. Statemate **graphic editor**s (GEs) are more than simple drawing packages; they are language-sensitive graphic editors.

actual parameter

The **element** or **constant**, defined in the scope of an **instance** of a **generic chart**, that is mapped into a **formal parameter** of the generic chart during **instantiation**. The type of the actual parameter must match the type of the formal parameter.

aggregate element

An **array**, **queue**, **union**, or **record**.

alias

A named reference to a slice of a **bit-array** or to a slice of a bit-array that is a member of an **array** of bit-arrays. In addition, aliases can be slices of other aliases.

Aliases can be sensed or written. Writing to an alias changes only the bits of the bit-array that the alias refers to. A **bit** or a bit-array can be an alias.

Events and **condition**s cannot be specified as an alias. Integers and **real**s can be defined as an alias only if the definition specifies a single bit. (In this case, an automatic conversion is applied.) Arrays, **queue**s, **record**s, and **union**s cannot be aliases.

Examples:

```
(bit-array) ADDR => WORD(4..15)
(bit-array) DATA1 => PACKET(1)(16..31)
(bit) PARITY => ADDR(5)
```

And-state

Represent parallel behavior. And-states are sometimes called *concurrent* or *orthogonal* **state**s. When a **statechart** enters one And-state, it simultaneously enters all other And-states at that level of the state hierarchy. And-states can be subdivided into smaller states (decomposition).

array

A one-dimensional (vector) grouping of **data-item**s, **event**s, or **condition**s under a single name whose individual **element**s are addressed through a reference index.

array element

An individual **data-item**, **event**, or **condition** in an **array**. The **element** is referenced by the syntax `A(i)`, where `A` is the array and `i` is the index to the array.

array slice

A reference to a consecutive group of individual **array element**s. The slice is referenced by the syntax `A(i..j)`, where `A` is the array and `i` and `j` are indices into the array.

asynchronous

In Statemate **model**s, **transition**s are made on a **step**, but the step does not consume any time. This is typically used with software or system-level design.

attribute

User-defined information stored with an **element**'s **properties** entry. Each attribute has a name and can have a value. For example, an attribute can represent a security classification level or the name of a requirements document.

You can define attributes using the Attributes dialog box in the property sheet. The dialog box displays an editable list of attributes and contains a button that loads attribute definition files.

Note the following:

- You can list the attributes of all the elements in your **workarea** using the reports tool. The Property and Attribute reports include information about attributes.
- Support for attributes is available in the **Dataport** and the **Documentor**.
- You can use the **Check Model** tool to check for inconsistencies between the attributes defined in an attributes definition file and those stored in the properties. The checks report enforced attributes that are missing, attributes without a value, and values that are inconsistent with type definitions.

attribute definition file

A text file containing predefined **attribute**s that you can apply to an **element** using the **Load Attributes** button in the Attributes dialog box in the property sheet. Before using the **Load Attributes** button, you must first specify the directory containing attribute definition files as a preference in Statemate.

basic activity

An **activity** with no **descendant**s; the most primitive activity.

binding

See **interactor binding**.

bit

Holds a single, binary value. Its **literal**s are `0b1` and `0b0`.

bit-array

An **array** of **bit**s. The length and indices are specified when the **element** is defined. The defaults values are: length 32 bits long, right index (most significant bit) 31, and left index (least significant bit) 0.

Bit-arrays are treated as unsigned numbers by Statemate. Both implicit and explicit conversion to other Statemate types is supported.

callback

A mechanism that associates a **subroutine** with an **element**.

chart

A general term for one of the graphical editors within Statemate.

Check Model

Tests all or selected **chart**s in the **model** for violations in syntax and semantics. It also checks the relation between charts.

- The tool tests for two types of errors:
- **Correctness** - detects inconsistencies in the model, such as an **Or-state** without a default entrance.
- **Completeness** - detects redundancy and incompleteness in the model, such as unresolved **element**s.

CM

See **configuration management**.

code generation

Statemate can create code from a **model** via a code profile editor. The resulting code can be compiled and executed on the development host.

combinational assignment

The expression used to assign a value to a **combinational element**. The syntax is as follows:

```
X = Y1 when C1 else
    Y2 when C2 else
    ...
    Yn
```

In the syntax, `X` is a variable **condition** or **data-item**, `Y1` to `Yn` are expressions, and `C1` to `Cn` are **condition expression**s.

combinational element

Represents **asynchronous** behavior, that is, an **element** whose value is "continuously" assigned (rather than evaluated once each **step**).

combinational logic

Enables you to model **asynchronous** logic in your designs. The flexible syntax for **combinational element**s enables you to specify asynchronous **function**s, from simple combinational logic gates to multiplexors, transparent latches, and so on.

component

An **element** contained in a **library**. A component root is a generic **activity**-chart. The definition of a component includes the following parts:

– **Top-level name and input/output**

The top-level design contains information on the component's name and input/output. If you change the top-level design, all **chart**s that contain **instance**s of that component must be modified by you. Use the **Check Model** tool to detect instances that are not up-to-date.

– **Behavior**

The behavior of the component is reflected in all the versions of charts that use the component. If the behavior is changed, the **graphic editor**s that contain instances of that component do *not* have to be modified by you. Analysis tools automatically sense the new behavior as well.

Note: Note that this is true for inserted (but not copied) components.

– **Icon**

The icon is a pixel map used to represent the component in the component browser. A one-line text **label** is attached to the pix map to enable you to attach a description to the component. The system then automatically attaches this description of the top-level activity of the **generic chart** as the description of the component.

If the icon is changed, the graphic editors that contain instances of that icon's component do not have to be modified by you. Analysis tools do not sense the modification, but the component browser is refreshed with the new icon.

Note: Global definition sets can be part of a component. When you create a component, the related GDSs are automatically added to the component's configuration. (A GDS is related if it contains user-defined types or **constant**s that are part of a component definition.)

composition connector

A **connector** that can be used only with a **record** or **union** **data-item**. The composition connector directs the **component**s of a record to two different target activities. For example, a **data-store** called RANGE has a record type of data-item with two components called LOW_LIMIT and HIGH_LIMIT. If the **flow** is going out from the data-store, the composition connector splits the record into the two components.

The composition connector can also go in the other direction, where multiple **flow line**s labeled with the record's components enter the connector and the single **flow line** denoting the record flow emanates from it.

compound

An **element** defined as an expression in terms of other elements. The type of the expression must match the type of the element.

Compounds can only be sensed; they cannot be written or changed directly. To change a compound, you must change the value of the elements used to define the compound.

Events, **condition**s, and simple **data-item**s (**integer**, **real**, **string**, **bit**, or **bit-array**) can be specified to be compounds. Array, **queue**s, **record**s, and **union**s cannot be compounds.

The following examples are valid:

```
(real) VOLUME => (4.0/3.0) * PI * R**2.0
(condition) BOOL1 => CON1 or CON2 and not in(STATE1)
```

The following example is invalid:

```
(condition) BOOL2 => 2 * PI * R
```

compound flow

A mechanism consisting of multiple **flow**s and **connector**s that transfers information between activities. This reduces the number of arrows and makes the **chart**s easier to read and understand.

condition

A persistent signal whose value is either TRUE or FALSE that indicates something occurring over a span of time. For example, the light is on. All conditions are enclosed in square brackets (for example, `[C]`).

The **action** `tr! (C)` has the effect of setting the truth value of condition `C` to true, and the corresponding action `fs! (C)` sets it to false.

Compare this with **event**.

condition connector

Used in **statechart**s, this emphasizes a choice based on a **condition** (for example, `x>1, x<1, x=1`). The **trigger**s must be mutually exclusive.

condition expression

A **compound** expression that can contain **variable**s and **context variable**s.

configuration file

A version “snapshot” of the **workarea**. Typically, configuration files are saved to the **databank** where they can be retrieved and executed at a later time to reload the snapshot of the design to the current workarea.

See also **configuration management** and **configuration item**.

configuration item

Elements stored in the **databank** as ASCII files. Multiple versions of the same configuration item can exist in the databank. All versions of a configuration item belong to the same owner and have the same access permissions.

See also **configuration file** and **configuration management**.

configuration management

Also called CM, change management, revision management, and source control. Each Statemate **project** has a common repository area called the **databank**. The databank is both centralized and permanent. That is, information in the databank belongs to the entire project and represents the current working design.

As you rework your **chart**s, each modification is made to your **workarea**. When you want to permanently store a new version of your changes to allow others to share them, you save the modified charts to the databank. And, through a locking mechanism, you are ensured that your work will not conflict with the interests of the other **project member**s.

While working on a project, you can perform configuration management operations such as check-ins and checkouts using the workarea browser and the **databank browser**. Statemate provides a built-in CM tool similar to those found in products designed specifically for the task of configuration management. In addition, Statemate allows you to transparently substitute a different CM tool when you create a project.

See also **configuration file**, **configuration item**, and **configuration management tool**.

configuration management tool

The tool used to perform **configuration management**. When you create a new **project**, you can choose Statemate's built-in CM tool or one of several widely-used, third-party CM tools. If you prefer a different tool, templates are available so you can create your own simple, script-based interface to your tool of choice.

See also **configuration file**, **configuration item**, and **configuration management**.

connector

A circular or oval graphical object used in **chart**s to join and divide arrows or to allow an arrow to exist on multiple pages. Its purpose is to reduce the number of arrows in, and to clarify, a specification.

constant

A value that does not change. A constant must be defined in terms of a **literal** of the corresponding type or expression of literals. It cannot be defined in terms of an expression of other **element**s even if the other elements are constants.

Constants can only be sensed. They cannot be changed or written. Conditions and simple **data-item**s (**integer**, **real**, **string**, **bit**, and **bit-array**) can be specified as constants. Arrays, **queue**s, **record**s, **union**s, and **event**s cannot be constants.

constant literal

An alphanumeric expression assigned to a constant **element**. For example:

```
X= 'Hello'
Y = 5
```

context variable

A **variable** that is evaluated and updated sequentially within an **action expression**. A context variable is preceded by a dollar sign ($).

control activity

A reference to a **statechart** or a **flowchart** that controls the activities within the same **activity chart**. A control **activity** cannot have any subactivities and is specified by an off-page statechart. An at-sign symbol (@) precedes the title of control activities.

control-flow

Carry information or signals used in making control decisions (for example, commands or synchronization messages).Two types of **flow line**s are allowed in **activity chart**s: data flow lines (drawn as solid arrows) and control flow lines (drawn as dashed arrows).

data-flow

Carry information used in computations and data-processing operations. Two types of **flow line**s are allowed in **activity chart**s: data flow lines (drawn as solid arrows) and control flow lines, drawn as dashed arrows.

data-item

A unit of information that can be one of the following:

- Numeric - Integer or **real**
- String
- Bit or **bit-array**
- User-defined type (UDT)
- Record
- Union

data-store

Stores information on activities for later use. Data-stores can be used to total large volumes of data, continuously accumulating over time. Data-stores are always basic; they cannot contain other data-stores or activities.

data type

Statemate supports the following data types:

- **bit**
- **bit-array**
- **enumerated type**
- **field**
- **integer**
- **real**
- **record**
- **string**
- **union**
- **user-defined type (UDT)**

databank

The directory structure in which **project** data is stored. The databank contains several subdirectories that hold different types of Statemate objects, such as **chart**s, **panel**s, and **configuration file**s. Objects in the databank are stored as ASCII files.

A project databank can be placed in any directory to which you have read and write access. Many sites designate a special location for project data; check with your system administrator.

See also **configuration management**.

databank browser

Enables you to navigate through your project's **databank**. The window displays a complete or partial list of the **configuration item**s in the databank, along with detailed information about the selected item (if any). The information includes a list of all existing versions of the item from which you can choose specific versions on which to apply configuration menu operations.

Dataport

A Statemate utility that enables you to extract information from the Statemate database. You can use this information to create plots, generate reports, and analyze data.

deadlock condition

A set of **condition**s that remain true forever.

deep history connector

Indicates an entrance to the most recently visited **state** or configuration at the lowest level in the hierarchy. This type of **history connector** has an asterisk after it (H*).

default connector

Specifies the starting point at each hierarchical level within a **statechart** (assuming that an explicit **transition** was not taken).

defined element

Each **element** in a **model** belongs to a specific **chart** and is defined in that chart via the **properties**. Elements defined in one chart can be used in another chart. A textual element is clearly “visible” in the chart where it is defined; it is also visible in all the **descendant** charts in the chart hierarchy.

See also **unresolved element**.

descendant

A **chart** that has a parent. The terms *descendants* and *ancestors* denote subactivities and parent activities, respectively, on any level of nesting. Activities that have no descendants are called *basic*.

diagram connector

Connects a target and a source that are far from each other. Using this type of **connector** eliminates the need for long arrows.

Document Generation Language (DGL)

In the **Documentor**, you use DGL to write a program that designs how your documents will appear and what information they will contain.

Documentor

A Statemate tool used to design and produce documentation of the system you are designing. The documents can include textual and graphical information from a variety of sources, including your **project** database and information outside your **workarea**.

Documentor include file

A statement used to include files from outside the **project** database within your documents. Such include files consist of textual information, diagrams, plots, and so on.

Documentor template

A design program that you write using **Document Generation Language (DGL)**. The template contains instructions as to what information to include in the report and how to format it.

DOORS

Provides access to the DOORS requirements tracing tool, if available.

element

A **model** is made up of elements, which can be:

- **Textual** - Actions, **condition**s, **data-item**s, **event**s, **field**s, information flows, **subroutine**s, and user-defined types
- **Graphical** - Activities, **module**s, and **state**s
- **Chart** - Activity charts, **control activity**s, **global definition set (GDS)**, **module-chart**s, and **statechart**s

The **properties** contains information about each element in the model.

enumerated type

A type of **constant**. You can define a **user-defined type (UDT)** to be an enumerated type, then define a set of constant values for this type. For example, you can define an enumerated type `COLOR`, then define `COLOR` as `{RED, YELLOW, BLUE, GREEN, ORANGE}`. You can use these enumerated values in expressions. For example:

```
/X:=RED;
```

environmental activity

A special type of external **activity** defined through the DDE to be external to all levels of the **chart** hierarchy. This type of **external activity** is external to the entire system being developed.

event

Instantaneous signals used for synchronization purposes. They indicate that something has happened.

Events occur in a precise instant in time, and if not immediately sensed they are lost. Events “live” for the duration of one **step** only, and that step is in the one following in which they occur.

Compare with **condition**.

external activity

Any **activity** outside the scope of the top-most activity in a particular **activity chart**. Because activity charts are hierarchical, an external activity is usually resolved to a box in a **chart** higher in the chart hierarchy.

However, an external activity might be resolved to a box that is an **internal activity** at a higher level. In this case, it remains an external activity when referenced in the lower chart.

field

In addition to basic types, a **data-item** can be a composition of named **component**s, referred to as *fields*, each of which can be a data-item of any type or a **condition**.

Statemate supports two kinds of compositions: **record**s and **union**s. The entire construct is referenced by its name (for example, on a **flow line**), whereas a particular field is referenced using dot notation, as follows:

```
<record/union reference>.<field reference>
```

flow

Transfers information between activities and **module**s. They can be single pieces of information called **data-flow***s*, or groups of information called i*nformation flows*.

See also **a-flow line**, **compound flow**, **information-flow**, and **m-flow**.

flowchart

A flowchart represents a process graphically. It includes the entire process from start to finish, showing inputs, pathways and circuits, and action or decision points.

flow line

A labeled arrow that visually represents a **flow**. The **label** on a flow line denotes either a single information **element** that flows along the line or a group of such elements (an **information-flow**).

for loop

An iterative **action** that makes it possible to access the individual **array** **component**s in successive order. The syntax is as follows:

```
for $I in N1 to N2 loop
   A
end loop
```

fork connector

A type of **connector** that represents a single information **element** flowing from one source to several targets. A **joint connector** is the opposite type of connector.

formal parameter

An **element** defined (via the **properties**) in the generic **chart** with a type and mode. The formal parameter is a placeholder for the actual element being mapped to it. It is used to connect a **generic instance** to its scope or to define the special characteristics of the particular **instance**.

See also **actual parameter**.

function

A **subroutine** that returns a value and can have multiple **parameter**s. All function parameters are inputs.

See also **procedure**.

GBA

See **graphical back animation (GBA)**.

GDS

See **global definition set (GDS)**.

GDS editor

Enables you to create a new global definition set (GDS) or access an existing one.

generic chart

Enable you to reuse parts of a specification. Using a generic **chart**, you can represent common portions of the **model** as a single chart that can be instantiated in many places (similar to a **procedure** in a conventional programming language).

Generic charts are linked to the rest of the model via **parameter**s; no other **element**s (besides the definitions in global definition sets) are recognized by both generic charts and other portions of the model.

generic instance

An **instance** of a **generic chart** used for a specific portion of a **model**. The "<" notation indicates that a box is an instance of a generic chart. For example, a box named `I<G` denotes that the box `I` is an instance of the generic chart `G`.

Instances must be basic and truly generic. They cannot contain **descendant**s, behavioral information, **static reaction**s, or **combinational assignment**s.

global definition set (GDS)

A type of **component** that contains definitions of user-defined types as well as **constant** **data-item**s and **condition**s. The **element**s that appear in a GDS are visible in the entire **model**. Data types defined in a **chart** or inherited from a **parent chart** take precedence over **data type**s defined in a GDS.

A GDS is similar to a chart in that both are **configuration item**s of the model. That is, both charts and GDSs contain parts of the model and can be saved and loaded separately from other parts. A GDS cannot contain any other graphical or non-graphical information.

There can be several GDSs in one model, but there is no hierarchical relationship between them, nor between them and the charts in the model.

See also **GDS editor**.

graphic editor

The graphic editors enable you to create and modify **statechart**s, **procedural statechart**s, **module-chart**s, and **activity chart**s.

graphical back animation (GBA)

The Simulator highlights the **chart**s as they are executed. However, once you generate code, you lose that graphical feedback. GBA provides graphical highlighting similar to Simulation, but from generated code.

hardware activation style

A behavior pattern where the activities' **component**s are always active. In these cases, the **activity** does not need a **control activity**, and all of the subactivities start and stop when the parent does.

history connector

A **connector** (H) that indicates an entrance to the last **state** visited residing at the same hierarchical level as the connector.

See also **deep history connector**.

hook

A software code generator mechanism that enables you to "hook" user-actions or **procedure**s to any change in the specification during execution. This is useful to tie

your external environment to the behavior represented by the generated code. Unlike **stub**s, which simply serve as placeholders, hooks generate **callback** **function**s and actually communicate with external code.

Infer device

For **combinational element**s that model latching behavior, the **Infer Transparent Latches For** option can be used to generate code that will cause the synthesis tool to infer a transparent latch instead of combinational feedback.

information-flow

Represents a container for other **element**s (**condition**s, events, **data-item**s, and other information flows). They reduce the number of **flow line**s, which makes the **chart** more readable and easier to understand.

instance

A repetition of a generic **chart**. It can be repeated multiple times in a **model**, but it must be a primitive box that does not contain **static reaction**s or **mini-spec**s.

instantiation

The act of creating an **instance** of a generic **chart**. Each instance has its own actual-to-**formal parameter** binding.

integer

Any natural number, 0, or the negative of any natural number. The maximum value of integers allowed is dependent on the architecture of the machine on which Statemate is being executed. On a 32-bit machine, it is `(2**31)-1`.

interactor

Graphical representation of an input/output device in the **panel graphic editor** that has a predefined behavior. The following types of interactors are available: push buttons, menu buttons, radio buttons, sliders, displays, lamps, meters, and knobs.

interactor binding

Establishes a connection between a **panel** graphical object and **element**s in the simulated **model**. The three possible modes of binding used for **interactor**s are IN, OUT, and IN/ OUT. For animation of user-defined graphical objects, only the OUT binding is allowed.

interface report

A report that graphically presents the input and output of a **module** or **state**. You can generate this type of report using the reports tool or **Documentor**.

internal activity

Any **activity** within the scope of the top-most activity in a particular **activity chart**.

See also **external activity**.

joint connector

A type of **connector** that represents a single information **element** flowing from several sources to one target. A **fork connector** is the opposite type of connector.

junction connector

A type of **connector** that reduces the number of lengthy **flow line**s by connecting different **element**s together. These elements then form a single flow line that emanates from or enters a common box or connector.

label

The name on a **flow line** that denotes either a single information **element** that flows along the line or a group of such elements.

library

A container for **model** **component**s. Any Statemate **project**, new or existing, can be designated as a library.

Together, libraries and components offer a means to speed the process of design specification and to help you create more consistent specifications. Reusing previously created Statemate model **element**s is more efficient and makes for more consistent designs.

list report

A report that presents either:

- All **element**s of a particular type
- A table of all names and synonyms for elements in the input list

You can generate this type of report using either the reports tool or the **Documentor**.

literal

One of the following:

- **Character literal** - A quoted sequence of characters.
- **Numeric literal** - A sequence of digits, an optional decimal point, and an optional negation sign.

For example:

```
real - 3.1234
integer - 5
string - 'abcdefg'
```

Also called a *literal constant*.

local data

Subroutines that are implemented in the Statemate **action language**, or a **procedural statechart** that uses local data **element**s. These elements are like local **variable**s in any programming language, and can have name and type definitions.

lock

Guarantees that other users cannot change the item in the **databank** while you are working on it. After your changes are complete, you can check it into the databank and either release or continue to hold the lock. No **configuration item** can be locked by two or more users simultaneously.

When you create a new configuration item (a **chart** or file that you have created but not yet stored in the databank), it is implicitly locked by you. You can, therefore, always save such a configuration item into the databank.

See also **configuration management**.

m-flow

Refers to a **flow line** in a **module-chart**. This term used by the **properties**, **Documentor**, and the **Dataport**.

mini-spec

A definition of an **activity**'s behavior entered into the **properties**. The mini-spec is activated when the associated activity is active and stops when the associated activity stops.

You define mini-specs in the property sheet. The syntax is similar to **static reaction**s (a list of reactions of the form `trigger/action`, separated by a double semicolon (;;).

States that have mini-specs are distinguished by a ">" symbol after the **chart** name (for example, `ALARM>`).

model

Represents the system under design.

The heart of the specification stage is the construction of the system **model**. Statemate modeling is especially effective for **reactive system**s, whose behavior can be very complex, causing the specification problem to be notoriously elusive and error-prone. Most real-time systems, for example, are reactive in nature.

A system model constitutes a tangible representation of the system's conceptual and physical properties, and serves as a vehicle for the specifier and designer to capture their thoughts. In some ways, it is like the set of plans drawn by an architect to describe a house. It is used mainly for communication, but should also facilitate inspection and analysis. The modeling process involves conceiving of the **element**s relevant to the system and the relationships between them, and representing them using specific, well-defined languages. When the model reflects some pre-existing descriptions (for example, requirements written in natural language), it is useful to keep track of how the **component**s of the developing model are derived from the earlier descriptions.

module

The primary graphical object used in **module-chart**s. Modules are used to represent the structure of the system. There are two types of modules: internal (shown as a solid rectangle) and external (shown as a dashed rectangle).

The functionality of a module is shown by describing it by an **activity chart**.

- The primary structural building block of **Verilog** code. This term also refers to structural **component**s when defining Verilog Code Generation Profiles in Statemate.
- The term used to see structural components when defining C or Ada **code generation** profiles in Statemate.

module-chart

Describes the structural view of the system using modules as the primary building block. A system description can contain one or more module-charts. They are located at the top of the **chart** hierarchy in a system **model**.

Module-charts can be connected to **activity chart**s, which describe the functionality of individual modules.

module-chart graphic editor (MGE)

The graphical editing tool used to create and edit **module-chart**s. Statemate graphical editors (GEs) are more than simple drawing packages; they are language-sensitive graphical editors.

monitor

A **simulation** debugging tool that displays a table of the textual and graphical **element status** during a simulation. The monitor can be used as an output device to display element status and an input device that accepts input stimuli during simulation. The various characteristics of a monitor window can be saved in a simulation profile, which enables you to reuse the monitor in other simulation sessions.

multi-value logic

An HDL option that provides the capability to do extensive behavioral **simulation**. Using graphical **testbench**es, along with the multi-value logic constructs, enables you to test design functionality in a true hardware environment.

Multi-value logic makes it easier to model multiple logic drivers, bus contention, resolution problems, and many other real-world situations.

N2-chart report

A report that presents the flow of information between activities and modules. You can generate this type or report using the reports tool or the **Documentor**.

non-determinism

Occurs when there are multiple, legal exits from a single **state**. If more than one state is true, the system does not know which **transition** to take.

off-page chart

A decluttering mechanism that decomposes a **chart** into several pages. The contents of the box **element** (**activity**, **state**, or module) can be drawn in a separate chart. The box element is called an *instance box* and the associated chart is called an *off-page chart*.

A box refers to its off-page chart by using the @ sign in its name. For example, a box named `P@C` denotes that the box `P` is decomposed into chart `C`. If you want to use the same name for the box and the off-page chart, omit the first name. Thus, `P@C` becomes `@C`.

off-page connector

Connects arrows that appear on separate pages. You can use an alphanumeric string to **label** these **connector**s. A useful convention is to label the connector with the name of the source or target of the arrow in the instance **chart**.

one hot state vector encoding

Allows a single storage **element** to represent each **state**. This option significantly reduces the amount of decode logic in the resulting circuit, which improves circuit performance.

optimize state translation

Eliminates unnecessary levels by assigning the minimum number of storage **element**s while maintaining the hierarchy. However, this option will not eliminate all levels in all cases. For example, it cannot eliminate a level that contains a **static reaction**.

Or-state

Enables you to represent sequential behavior. The Or-state is similar to the **state**s used in traditional state diagrams or finite state machines.

The **statechart** can be in one, and only one, Or-state at any one time (at a particular level of the state hierarchy). Or-states can be subdivided into smaller states (decomposition).

overload state vector

Makes the generated HDL code perform faster than when you do not overload, but it takes up more space. Overload when your design requires more speed; do not overload when your design requires less area.

panel

Provides a visual interface to the simulated **model** or generated code for debugging and prototyping purposes. It is built of predefined **interactor**s and user-defined shapes. Dynamic behavior of these graphical objects is defined through their binding to **element**s of the model.

Typically, panels represent the user interface to a system. However, it is also quite common for the panels to show a logical representation of a system. For example, a panel could show the routing of packets through a communications network, or the failure states of valves and pumps in an aircraft fuel system.

panel graphic editor

Enables you to create a realistic mock-up that represents the system you are designing. The **panel** is associated with a **model** and is animated during model execution.

parameter

An **element** that links a **generic chart** to the rest of the **model**. No other elements (besides the definitions in global definition sets) are recognized by both generic charts and other portions of the model.

parameter binding

The way that an **instance** of a **generic chart** is connected to the scope of the instance during **instantiation**. It involves the binding of **actual parameter**s to **formal parameter**s of the generic chart.

See also **actual parameter** and **formal parameter**.

parent chart

A **chart** that has **descendant**s or subactivities. Parent charts are sometimes called *ancestors*.

playback

During a **simulation**, you can record the commands and save them in the form of a **simulation control language (SCL)** program, which can be run like a normal SCP.

See also **simulation control playback (SCP)**.

predefined type

Five primitive types of **data-item**s are predefined in the language of Statemate: **integer**, **real**, **bit**, **bit-array**, and **string**.

preferences

Customizations applied to the Statemate work environment. You can set preferences individually or by loading preestablished settings.

Project managers and system managers can set preferences that are enforced for users at your site. When you set your own preferences in a specific area, you can obtain information on which preferences have been enforced at your site.

procedural statechart

A specialized derivative of a **statechart**. Procedural statecharts:

- Are executed entirely in one **step**.
- Must contain a termination **connector**.
- When called, run from the default to the **termination connector** (including any loops) within a single step.

procedure

A **subroutine** that has no return value but can have multiple **parameter**s. Each parameter can be INPUT, OUTPUT, or INPUT/OUTPUT.

See also **function** and **task**.

procedure-like termination type

A type of **activity** that is started by the contro at the next higher level in the **activity chart** hierarchy. Once started, it runs to completion in a single **step**. This type of activity can contain a **mini-spec**, but cannot contain a control-activity.

See also **activity termination type** and **procedure**.

project

The main unit of work organization in Statemate. A project consists of data and users who can access that data. In general, **project member**s have access to all data in a project, but any objects can be protected from write or read access.

A project includes the following:

- **project name**
- **project manager**
- **databank**
- **configuration management tool**
- **requirements traceability**
- **project member**s
- **library** (optional)

A project is intended to be accessed by multiple users; a **workarea** is intended to be accessed by a single user.

project manager

A Statemate user who controls access to a **project** (can add or remove members from the project).

Each project can only have one project manager associated with it. By default, the project manager is also a **project member**. The project manager does not need to be an **SMAN**.

project member

A Statemate user who has access to a particular **project**. A user can belong to many projects simultaneously.

Having access to a project means that the project member can view the list of objects stored in the **databank**. Read or write access to individual objects within the databank is granted on a per-object basis, according to user **preferences**.

Only the **project manager** can add or remove project members.

project name

The name of the **project**. Project names must be unique at your site. Project names must begin with a letter, and can consist of letters, numbers, and underscores. Lowercase letters are automatically converted to uppercase. Names in Statemate, including project names, are not case-sensitive and cannot contain spaces.

Leaving the CAPS LOCK key off is the preferred setting.

properties

Store textual information for the Statemate **element**, such as a description, **attribute**s, and relationships with other elements. The element information can be formal (possessing some semantics that is relevant to the **model** and its behavior) or informal.

Some kinds of textual information are relevant to all types of elements, such as a one-line short description and an unlimited, textual, long description. These narrative additions, especially the long description, can be used to provide information about the element in an informal language, for the record.

In addition, the general mechanism of an attribute pair, name, and value can be used to associate special characteristics with the element. The properties can also be used to associate a synonym with the element, usually a shorter name that is easier to incorporate into a detailed **chart**.

property report

A type of report that extracts basic **element** information from the **properties**. You can produce a property report for all element types. The reports tool and the **Documentor** can generate this type of report.

property sheet

Provides a mechanism to search the **properties** to create lists of **element**s so you can examine and modify listed elements using the property sheet.

protection level

See **access level**.

query

The **properties** includes a set of pre-programmed queries that represent specific relationships between specification **element**s or elements with a specific **attribute**. The output of the query is a pending list, which can be passed, as input, to subsequent queries.

You can perform three types of queries:

- Search the entire **workarea**. These queries search for element names and synonyms.
- Use the pending list as input. Once used as input, the subsequent query produces another pending list that replaces the original.
- Operate on either the workarea or the pending list.

queue

An ordered, unlimited collection of **data-item**s, all of the same **data type**. This data type can be any predefined or **user-defined type (UDT)**. A queue is essentially a single structure that holds many **element**s. You can add elements to either the front or the back, but can retrieve them only from the front of the queue.

The following **action**s affect queues:

- `fl!` - Clears the queue.
- `get!` - gets information about an element. `get` actions are performed when they are encountered.
- `peek!` - copies the queue's head element without removing it.
- `put!` - adds the value of the expression to the end of the queue. `put` actions are accumulated and performed at the end of the **step**. This scheme reduces the chances of **racing**.
- `q_length` - Returns the length of the queue.
- `uput!` - adds the value of the expression to the front of the queue.

racing

Occurs when a **condition** or **data-item** is:

- Modified and used at the same point in time.
- Modified more than once at the same point in time.

reactive system

A typical reactive system exhibits the following distinctive characteristics:

- It continuously interacts with its environment, using inputs and outputs that are either continuous in time or discrete. The inputs and outputs are often **asynchronous**, meaning that they can arrive or change values unpredictably at any point in time.
- It must be able to respond to interrupts (high-priority **event**s), even when it is busy doing something else.
- Its operation and reaction to inputs often reflect stringent time requirements.
- It has many possible scenarios of operation, depending on the current mode of operation and the current values of its data, as well as its past behavior.
- It is often based on interacting processes that operate in parallel.

Examples of reactive systems include on-line interactive systems, such as automatic teller machines (ATMs) and flight reservation systems; computer embedded systems, such as avionics, automotive and telecommunication systems; and control systems, such as chemical and manufacturing systems.

reactive-controlled

A type of **activity** started by the control at the next higher level in the **activity chart** hierarchy. Once started, it remains active for one or more **step**s until it is stopped by the same **control activity**. This type of activity can contain a control activity or **mini-spec**.

See also **activity termination type**.

reactive-self

A type of **activity** started by the control at the next higher level in the **activity chart** hierarchy. Once started, it remains active for one or more **step**s until it terminates itself by entering a **termination connector** in its **control activity** or executing a stop **action** in its **mini-spec**. This type of activity can contain a control activity or mini-spec.

See also **activity termination type**.

read-only access

An **access level** that allows a **configuration item** to be checked out without a **lock**. The checked out item is automatically placed in read-only editing mode.

See also **access level**.

read-only editing mode

A mode in which all **graphic editor** viewing and tool launch features are active, but editing features and drawing icons are disabled. This is the only available mode when you are using **read-only access** and a **chart** is opened more than once in the same session.

In the property sheet, you can make changes, but the changes cannot be saved (unless you **lock** the item).

See also **configuration management**.

real

A floating-point number. In Statemate, you can see values using either the form `nnn.mmm` or `n.mmm E+ee`. The values allowed are dependent on the architecture of the machine on which Statemate is being executed. This is usually in the range `-1.0 E+38` to `1.0E +38`.

real-time operating system (RTOS)

The low-level operating system that controls basic system **function**s such as memory management, interrupt management, and disk access. In an RTOS, the assumption is that all tasks will be executed in such a way that they are (essentially) happening in "real time," with no artificial delay between the initiation and execution of an **action**. For example, a real-time control system will appear to process inputs in virtually the same moment that they are made, so any output action required as a consequence of the input will happen immediately thereafter with no detectable delay.

In actuality, delays are inescapable in the sequential nature of digital control systems and software execution. However, the delays are kept to less than a "critical time" period that is short enough to be imperceptible to the system user. Guaranteeing that this "virtual real-time execution" occurs in a consistent and predictable manner is the job of the RTOS. Simulation **model** code must be compiled specifically to function in the hardware and software environment provided by the RTOS.

record

A **data type** that consists of several **field**s of possibly different predefined or user-defined types. When a **data-item** is declared to be a record, it is defined to contain all of its fields. A record is analogous to a `structure` in C and a `record` in Ada. To access a field in a record, use both the record name and field name separated by a period. For example:

```
RECORD_NAME.FIELD_NAME
```

See also **union**.

register transfer level (RTL)

A code style that can be both simulated and synthesized.

regular chart

A non-generic **chart**.

See also **generic chart**.

requirements traceability

See **DOORS** and **RTM**.

RTM

Provides access to the RTM requirements tracing tool, if available.

SCL

See **simulation control language (SCL)**.

SCP

See **simulation control playback (SCP)**.

simulation

Enables you to execute a graphical **model**. You can verify the behavior of your design by examining the animation of the graphical **element**s in your design. You can also modify and examine the values of the textual elements in your design.

simulation control language (SCL)

An optional file that can be recorded (like a **trace file**) when you are simulating a **statechart**. It is a record of the keystrokes you pressed during a **simulation** and is sometimes called a **playback**.

simulation control playback (SCP)

A program of **simulation control language (SCL)** commands, created by recording a **simulation**. You can then use the SCP to control a simulation in batch mode. An SCP animates the **statechart**s and **activity chart**s in the simulation scope in the same manner as interactive simulation. You can use SCPs to facilitate the entry of large amounts of data and to automate scenario-based executions.

single

A **data-item** that is neither an **array** nor a **queue**.

SMAN

A Statemate manager, who can:

- Modify or delete any **project**.
- Define or undefine other SMANs.

A user does not need to have special system capabilities to be a SMAN (that is, they do not need to be “root”). SMAN activities do not modify system files.

See also **project manager**.

state

The primary graphical object used in **statechart**s. States represent behavior of the system or part of the system. States in a statechart differ from states shown in more traditional state diagrams or finite state machines (FSM) in two ways:

- They can be divided into substates hierarchically.
- They represent parallel state behavior.

There are two types of states:

- **And-state**s - Shown by dividing an **Or-state** into substates with a dashed line. And-states show concurrent, or parallel, behavior.
- **Or-states** - Similar to states in traditional FSM. The statechart can be in only one Or-state at the same level of hierarchy at one time. Or-states are represented by a rounded rectangle.

See also **And-state** and **Or-state**.

statechart

Describes the system's behavior over time, including:

- The dynamics of activities
- Their control and timing behavior
- The **state**s and modes of the system
- The **condition**s and **event**s that cause modes to change and other occurrences to take place

In addition, it provides answers to questions about causality, concurrency, and synchronization.

Statecharts constitute an extensive generalization of state-transition diagrams. They allow for multi-level states, decomposed in an and/or fashion, and therefore support economical specification of concurrency and encapsulation. They incorporate a broadcast communication mechanism, timeout and delay operators for specifying synchronization and timing information, and a means for specifying **transition**s that depend on the history of the system's behavior.

Each **element** in the statechart has an entry in the **properties**, which can contain additional information. For example, an event entity can be used to define a **compound** event by an expression involving other events and conditions.

statechart graphic editor

The graphical editing tool used to create and edit **statechart**s. Statemate graphical editors (GEs) are more than simple drawing packages; they are language-sensitive graphical editors.

static reaction

Describe the behavior that takes place within a specific **state**. For example:

```
While in (S1) DO
   [POWER_ON]/tr! (LIGHT_ON);
   COUNTER:=0
```

Static reactions also describe **action**s that occur when there's a **transition** to enter or exit the associated state. For example:

```
On entering (S1) DO
   /st! (activity_warm_up)
On exiting (S1) DO
   /sp! (activity_warm_up)
```

You define static reactions in the **Reaction** **field** in the property sheet (DDE). Separate multiple reactions in the DDE with a double semicolon (;;). States that have static reactions include a ">" symbol after the **chart** name (for example, ALARM>).

status

During **simulation**, the status consists of the following information:

- The status of activities in the scope (active, hanging, or inactive)
- The set of **state**s the system is in (the configuration)
- The values of all **condition**s and **data-item**s in the scope
- The **event**s generated in the previous simulation **step**
- The time delays until each scheduled **action** and **timeout event** occur
- The history of the states

Context **variable**s (whose names begin with a dollar sign), are not part of the status of the system. They do not retain their value from one step to another.

When restoring a status, the simulation tool checks the consistency between the current simulation scope and the one in which the status was saved. When the two scopes are coincident, all saved values are restored.

The values of **compound** **element**s are not saved. In addition, because you might want to use different global or local clocks during the restoration, the Show Future command might show different times than when the status was saved.

To effectively use the restore status facility when the stored status is a subset of the restored status or vice versa, the following points apply:

- Changes in the hierarchies of activities' and/or states cause the saved status to become unrestoreable. This includes cases when a state or **activity** is added, removed, or when it changes its location in the hierarchy.
- When a textual element is deleted, its saved value is ignored at the time of restoration. When a new textual element is added, its current value remains unchanged after the restoration.

status file

Records the **status** of the **simulation** in a non-text file for future reference. This is useful to:

- Backtrack to a certain point in the simulation (for example, nondeterministic solutions).
- Continue your work later.
- Use the current status in another simulation scope.

step

A change in the system **status** in response to external stimuli or internal changes. A step can be triggered by an **action** (internal or external) or by a timeout **event** occurring as a result of incrementing time.

A **simulation** step is a two-stage process:

- The stimulus to the system occurs via actions or **timeout event**s.
- The system reacts by processing **transition**s, **static reaction**s, and **mini-spec**s.

When the simulation execution begins, and before the first step is performed, the default initial status of the system is as follows:

- When using software style activities, the activities in the top-level hierarchy in the scope are active.
- When using hardware style activities, all activities in the scope are active.
- The system is not in any of its **state**s.
- All primitive **condition**s are false. All primitive numeric **data-item**s are zero and **string** data-items are blank.
- No events are generated.
- No timeout events or actions are scheduled.
- States have no history.

After the first simulation step is taken, the system status is as follows:

- The state configuration includes the default states of the **statechart**s connected to any active control **activity**, or defined to be a **testbench**.
- All other **element**s of the system status are modified in accordance with actions performed on **default connector**s or by static reactions on entrances into these states.

storage module

Stores information on **module**s for later use. It is analogous to a **data-store** in an **activity** except it is used in a module. Data-stores can be used to total large volumes of data, continuously accumulating over time. Data-stores are always basic; they cannot contain other data-stores or modules.

string

Holds any number of characters up to its defined length. The default length for a string is 80 characters.

structure

The structure of a **data-item** can be one of the following:

- **single**
- **array**
- **queue**

See also **data type** and **usage**.

stub

An empty entity where you can place handwritten or vendor-supplied code.

subroutine

You can define **function**, **procedure**, and **task** subroutines using:

- K&R C
- ANSI C
- Ada
- Statemate Action Language
- Procedural **statechart**s (for **procedure**s only)

You can use subroutines in the following ways:

- Within a **model** as part of **trigger**s and **action**s
- Connected to activities to describe their implementation
- Connected to Statemate **element**s as **callback**s

In addition, any C code that has been used to describe subroutines within a model can automatically be included within the generated code.

Subroutines have textual information like any other Statemate element, including long descriptions, **attribute**s, short descriptions, and so on.

subroutine parameter

Subroutines can have **parameter**s, which are analogous to **formal parameter**s in any other programming language. Parameters have a type, name, and mode.

superstep

Sometimes, as a reaction to external changes, the system is able to perform more than one **step** without additional external stimuli. Each step in such a series of steps, except for the initial one, is triggered by changes the system itself produced in the previous step. This chain of steps continues until the system reaches a **status** from which it cannot advance without further external input or without advancing the clock. Such a status is called a *stable status*. The progression from one stable status to another is called a *superstep*.

switch connector

Emphasizes a choice based on **event**s (for example, `input_1`, `input_2`, and `input_3`). The **trigger**s must be mutually exclusive.

synchronous

Typically used in conjunction with hardware circuit design. In Statemate **model**s, **transition**s are made on a clock. Every transition consumes one clock period and every **step** consumes one clock cycle.

task

A special form of **procedure** connected to activities for C and Ada only. Task **parameter**s can be inputs, outputs, or I/Os.

See also **subroutine**.

termination connector

A **connector** that denotes the termination of a **statechart**. This connector can appear anywhere in the statechart, and is considered a final **state** with no exits. If the statechart is the definition **chart** for a control, the **activity** associated with the **control activity** will be stopped.

test file

The **simulation** tool enables you to record input and output **element**s into files. The input file can be used as a test vector for the simulation. In other words, you can record an input file and then run the simulation again while reading inputs from that file.

The output file can be used as a benchmark for the simulation. In other words, you can create (record and edit) an output file that represents some expected or desired result. That file can then be compared to output files generated during subsequent simulations, including those that read from a (test vector) input file.

Only textual elements can be recorded.

testbench

A separate **statechart** created outside the specification of the system being developed. Testbenches can see any **element** in the **model** because the scoping rules do not apply to them. This enables a testbench to trap a specific behavior to test a design's inputs and outputs. It can be thought of as a "snapshot of a scenario."

Testbenches serve as debuggers and are visible to all signals in the design (discrete **flow**s are not necessary). However, testbenches cannot test generics.

timeout event

Triggers a **transition** based on the passage of time since a specified **event** occurred. The syntax is as follows:

```
tm (E,T)
```

In this syntax, `E` is an **event** and `T` is an **integer** expression. This expression defines a new event, which will occur `T` time units after the latest occurrence of the event `E`.

to-control connector

A **connector** that connects only to the **control activity**. Using this type of connector eliminates the need for long arrows.

trace file

An optional file that can be recorded when you are simulating a **statechart**. It captures the raw data of a **simulation**, which you can present in the following ways:

- Display on the screen.
- Create reports.
- Generate a waveform.
- Copy the data to another file.

In batch mode, simulation uses the commands `set trace` and `cancel trace` to toggle the tracing facility. The trace file is closed when one of the following commands is entered: **Exit**, **Restart Simulation**, or **Rebuild Simulation**.

transition

An **event** that makes the **model** leave one **state** and enter another. Label each transition with the **trigger** that causes it to be taken and, optionally, with an **action**. Separate the **trigger** from the action with a slash, as follows:

```
trigger/action
```

trigger

Causes the movement from one **state** to another.

tristate

One signal that can be driven by many sources, one source at a time. The syntax for a tristated signal is as follows:

```
output:=signal1
   when enable1
      else signal2
   when enable2
      else 0bZ
```

truth table

A tabular representation of inputs, resulting outputs, and **action**s. A truth table can also represent the behavior of an **activity** and the definition of a named action. It is similar to a **mini-spec**.

union

A **data type** that consists of several **field**s of possibly different **predefined type**s or user-defined types. When a **data-item** is declared to be a union, it is not defined to contain all of its fields. Rather, it is defined to contain one of its possible fields at any point in time. A union is analogous to a `union` in C and a `variant record` in Ada.

You access a field in a union using both the union name and field name separated by a period. For example:

```
UNION_NAME.FIELD_NAME
```

See also **record**.

unresolved element

Often, it is useful to be able to see **element**s that have not yet been defined in the **properties**. Such a situation might occur in intermediate stages of the specification process. A simple example is the use of an external **event** as a **trigger** in a **statechart** before the **activity chart** that defines that event is constructed.

When you define an unresolved element, the properties make a preliminary suggestion, based on the appropriate choices. For textual elements, the type defaults to textual.

Graphical elements that have unresolved references to them do not have property entries and cannot be defined.

usage

A Statemate information **element** can have one of four usages: **variable**, **constant**, **alias**, and **compound**. Certain usages are restricted based on how the element is referred to in the **model** and on the type of the element.

user-defined type (UDT)

A data-type that consists of several **field**s of possibly different types. A user-defined type is analogous to the `typedef` statement in C or the `type is` statement in Ada.

User-defined types are often required to be visible throughout the entire **model**, so they are usually defined in a global definition set.

variable

Holds a value. A variable is not defined in terms of any other **element** or expression. Variables can be both written and sensed. A variable is the default usage for any Statemate information element. Any Statemate textual element except for **information-flow**s can be a variable: **event**s, **condition**s, **data-item**s (**integer**, **real**, **string**, **bit**, **bit-array**, **record**, **union**, **array**, and **queue**).

Verilog

A tool that provides access to the hardware **code generation** tool. This tool automatically generates source code that is optimized for target Electronic Design Automation (EDA) tools downstream. The generated code reflects the same behavior as the original **model** and can be used as input to a Verilog simulator.

version number

The Rational Statemate built-in **configuration management** (CM) tool tracks versions using whole numbers (positive **integer**s). When you create a new item and check it into the **databank**, Statemate assigns it a version number of 1. Each time someone checks the item in, Statemate increments the highest existing version number.

Third-party CM tools might use other systems of version numbering, in which case the version numbers displayed by Statemate conform to the format of the third-party tool.

See also **configuration management**.

watchdog

See **testbench**.

Waveform

A **simulation** tool that enables you to communicate with a simulation and display changes as they occur.

while loop

An iterative **action** that iterates until some **condition** becomes false. The `break` action can be used to “jump out” of the loop without completing the iteration.

workarea

A private directory structure associated with a user and a **project** that enables each **project member** to work independently. You can design and redesign without making irrevocable changes to the current working or released design. As you rework your **chart**s, modifications are made to your workarea.

To permanently store your changes and allow others to share them, save the modified charts to the **databank**. Through a locking mechanism, you are ensured that your work will not conflict with that of the other project members.

A user can have multiple workareas associated with any project, but any one workarea can be associated with only one project.

See also **workarea browser**.

workarea browser

Displays the **chart**s and other files in the **workarea** and provides an easy way to open and edit them. In addition, it provides support for **project** **configuration management**.

Index

A

Access
  level 461
  read-only 490
  saved configuration file 364
Action 197, 462
  column 372
  expression 462
  language 226, 462
  properties 219
  relationships 200
Action in Box, creating 169
Activities
  global interface report 280
Activity 202, 463
  basic 466
  control 472
  environmental 475
  external 476
  internal 480
  naming 138
  properties 233
  termination type 463
Activity charts 136, 204, 206, 464
  accessing 136
  editor 464
  filtered check out 93
  generating from 164
  generic 242
  hierarchy 300
  icons 138
  in workarea 40
  modules 295
  properties 242
Activity preferences 84
Activity-chart
  graphic editor preferences 54
Activity-chart graphic editor preferences
  age specific preferences 54
  component preferences 57
Actors 152, 202
  creating 154
Actual parameter 464
Ada language
  code generation 17
  record 198
  selected implementation 225
  subroutines 99, 197
  task 225
  user-defined type 196
  variant record 198
Adding
  change description 15
Advanced Carriage Return is New Line 122
Advanced Enabled Reshaping 122
Advanced Fill Boxes 122
Advanced Fill Colors 122
Advanced query 250
AGE 464
Age specific preferences 55
Aggregate element 464
Alias 464
Align to Grid 118, 182
And sub-type 203
And-state 465
  naming 148
Animation
  built-in properties 173
  graphical back 478
  interactors and drawing elements 174
  sequence diagrams 167
Arc
  drawing 187
  drawing filled 187
Arrange
  panel editor 181, 182
Arrays 199, 200, 465
  bit 466
  element 465
  slice 465
Arrows
  creating 169
  creating start point 169
  labeling 169
Assignments 197, 226
  combinational 141, 235, 243, 467
Association, creating 154
Asynchronous 465
Attributes 16, 466
  definition file 466
  design 16, 151, 314
  edit 213

Object CreationStamp 365
tab properties 213
Auto panel specific preferences 74, 77
AUTOSAR 435, 436
creating a project 456
data 451
data types 448
exclusive areas 447
features 447
generating code 445
IB exclusive area 447
implementation 447
implementing STATEMATE 445, 446
inter runnable variables 454, 455
model exclusive area 447
scope definition area 437
services 450
timer exclusive area 447
toolbars 437
autosar_rte_210 OSI 456
AUTOSAT
notes 456
timing events 447

B

Basic
activity 466
structure of graphical elements 203
types 201
Best match 226
Bind 184
Binding 301
Bindings 466
check 184
group 191
individual 191
interactor 479
interactors 188
parameter 485
report 184
Bit 198, 466
Bit preferences 78
Bit_Array preferences 79
Bit-array 198, 466
Boundary box 202, 204
creating 154
Box
boundary 202
creating 169
drawing 187
drawing filled 187
filling 187
Browsers 2
activity interface 280
component 17
databank 473
workarea 503

C

C language
prototype code generator 17
selected implementation 225
task 225
Callback 467
Change permissions 19
Changes
automatic tracking 98
description 15
tracking 98
types to track 99
view all 15
Characters, special 368
Charts 467
activity 204, 464
check out 97
creating 123
element types 204
elements 204, 205, 240
exporting 96
GDS 205
generic 477
hierarchy in quick-edit mode 207
info 120
list of elements 239
load faults 96
module 483
module chart 205
off-page 484
parent 486
properties 120
read-only check out 97
regular 492
searching 210
sequence diagram 205
setting preferences for plots 343
tab 4
Check Model 17, 120, 467
router checks 282
Check model
preferences 68
Check out
charts 97
filtered 93
with descendents 93
Checking
bindings 184
Circle
drawing 187
drawing filled 187
Close 110
Closing a project 36
CM 471

Code generation 467
  Ada 17
  MicroC 17, 295
  options 301
  prototype for C 17
  Rational Statemate block in Rational Rhapsody 308
  support for reset operator 243
  with extended documentation 306
Column
  action 372
  input 368
  output 371
Combinational
  assignments 141, 235, 243, 467
  element 467
  logic 468
Components 259, 468
  browser 17
  copying 112, 265
  creating 261
  deleting 267
  insert 112
  inserting 263
  managing 267
  previewing 266
Composition connector 140, 469
Compound 469
Compound flow 469
Condition 196, 470
  and event properties 222
  connector 470
  creating connector 149
  deadlock 473
  expression 470
  relationships 200
Condition preferences 82
Conditional assignment, creating 150
Conditions
  default value 201
Configuration
  create new 94
  file 470
  item 470
  management 89, 471
  management tool 471
  purging items 92
  Rational DOORS 348
Configuration management 90
  checking in and out 97
  information 16
Connectors 471
  composition 469
  condition 470
  deep history 474
  default 474
  diagram 170, 474
  end 169
  fork 477
  history 478
  joint 480
  junction 480
  off-page 484
  switch 498
  termination 499
  to-control 500
Constant 471
Constant literal 472
Context variable 472
Control
  activity 138, 472
  creating flow 139
  labeling flow line 148
  reactive 490
  sub-type 203
Control-flow 472
Copy 15
Copying 209
  components 112, 265
  mirror X axis 183
  mirror Y axis 183
  workarea 40
Creating
  action in box 169
  actors 154
  And-line 148
  arrows 169
  arrows, start point 169
  association 154
  AUTOSAR project 456
  boundary box 154
  box 169
  chart 123
  combinational assignment 141
  composition connector 140
  composition connector in module charts 144
  condition connector 149
  conditional assignment 150
  control activity 138
  decision 169
  deep history connector 149
  default transition 148
  diagram connectors 140, 144, 149, 170
  diagrams 123
  display interactor 186
  end connectors 169
  event 148
  extended relationship 154
  external activity 139
  external life-line 162
  external module 143
  external router block 140
  free text 141, 145, 150, 155, 163, 170
  generalization relationship 154
  history connector 149

horizontal choice 186
include relationship 154
instance box 169
internal activity 138
internal module 143
internal router block 140
junction connector 140, 144, 149
knob interactor 186
lamp interactor 186
Life-Line 162
message label 162
message note 162
meter 186
OR state 148
Order Insignificant Line 162
Partition Line 162
project 30
push button interactor 186
Referenced Sequence Diagram Line 162
slider interactor 186
sub-chart 135
switch 169
switch connector 149
termination connector 149
timing constraint 162
timing constraint note 162
to-control connector 140
use case diagrams 154
vertical choice 186
workarea 37
Creating elements 207
Cut 15
Cut and paste 209

D

Data
flow 139, 144, 472
initialization of dynamic 307
item 196, 200, 220, 221, 472
local 481
store 141, 145, 203, 472
type 473
Data items 200
default value 201
Data read access 451
Data receive points 451
Data send points 451
Data types
AUTOSAR 448
Data write access 451
Databank 90, 473
automatic refresh preference 93
browser 473
check out items from 93
information 16
locking items 95
tracking changes 98
databank
search feature 7
Databank browser preferences 63
Database
error diagnostics 100, 102
Dataport 473
Deadlock condition 473
Decision
creating 169
writing an expression 169
Deep history connector 149, 474
Default
connector 474
transition 148
truth table row 372
Defined element 474
Definition
file attribute 466
GDS 478
Delete 15
Deleting
components 267
output devices 46
project 36
unused elements from workarea 40
workarea 41
Descendant 474
Description tab properties 218
Descriptions
length in search results 246
long 16, 231
Design attributes 16, 314
Design Attributes tab properties 217
DGL 474
Diagrams
activity charts 164, 206
connector 474
creating 123
creating connector 140
creating connectors 144, 149, 170
flowcharts 205, 206
procedural statecharts 240
sequence 159, 164, 205, 206
statecharts 206
use case 152, 154, 206
use case properties 238
Directories
temporary workarea 42
Displaying, project settings 34
Dive Group 184
Document Generation Language 474
Documentation
code generation with 306
Documentor 17, 474
include file 474
template 475

DOORS 475
  exporting router elements 282
Drawing
  arc 187
  box 187
  filled arc 187
  filled line segments 187
  filled polygon 187
  graphics editor 127
  line 187
  line segments 187
  oval 187
  polygon 187
  router blocks 273
Drawing areas 2

E

Editing 15
  properties 208
  quick mode 207
  track changes while 15
Editing mode
  read-only 491
Editors
  activity chart 464
  GDS 17, 287, 477
  graphic 17, 478
  module-chart 483
  panel 17, 485
  profile 296
  setting preferences 89
  statechart 494
Elements 91, 196, 475
  aggregate 464
  appending to lists 249
  array 465
  chart 204
  combinational 467
  creating 207
  creating lists of 246
  data 275
  defined 474
  filtering lists of 249
  finding referenced 256
  finding where used 258
  graphical 202
  mass edit 209
  modifying 207
  output 371
  quick-edit mode 207
  Rational DOORS exporting 363
  read only in workarea 40
  re-exporting Rational DOORS 364
  reset to default 243
  router 271
  saving lists of 247
  searching 244
  textual 196, 201
  unresolved 207, 501
Embedded Rapid Prototyper 17
Embedded rapid prototyper preferences 72, 85
Enable Scale Text 122
Entering
  existing label interactor 187
  free text 187
Enum preferences 80
Enumerated type 475
Enum-type 171, 198
Environmental activity 475
Errors
  chart load faults 96
  database 100, 102
  report 100
Establishing
  control flow 139
  data flow 139, 144
Event preferences 81
Events 197, 475
  creating 148
  relationships 200
  timeout 500
Execution
  of truth tables 373
  sub-type 203
Exporting
  charts 96
  databank files 96
  Rational DOORS elements 363
  router blocks to DOORS 282
Expression
  action 462
  condition 470
Extended relationship
  creating 154
External
  activity 139, 476
  life-line 162
  module 143
  router 272
  router preferences 285
  sub-type 203
External router sub-type 203

F

Factorization
  actions 377
  cells 375
  inputs 375
  outputs 377
Field 197, 476
  relationships 200
Fields properties 223, 224

Files
configuration 470
exporting 96
in databank 90
include 474
link to external 16
purging 92
saved configuration 364
status 496
tab 5
test 499
trace 500
Filled line
drawing 187
Filtered check out
GDS 293
Floating-point 491
Flow 476
compound 469
control 472
data 472
in 481
information 479
Flow chart 168
Flow lines 461, 476
naming 139, 144
reducing number of 275
Flow ports
for StatemateBlock 309
Rational Statemate 309
Flowchart
accessing 168
icons 168
Flowchart graphic editor preferences 57, 59
Flowcharts 205, 206, 226, 476
procedural 170, 241
subroutine implementation 170
Flow-lines
compound 276
global compound 276
local compound 276
for loop 476
Fork connector 477
formal parameter 477
Free text
creating 141, 145, 150, 155, 163, 170
Function 477

G

GBA 478
GDS 205, 478
component part 260
create new 288
editing 291
editor 17, 287, 477
filtered check out 93, 293
reduced 40, 293
usage property 292
visibility mode property 292
General preferences 49
General tab properties 212
Generalization relationship 154
Generating code
AUTOSAR 445
Generation
code 467
document language 474
Generic chart 477
Generic instance 477
Global definition set 478
Global Definition Set (GDS) 93, 205, 260, 287
Global interface 120
Global interface report 280
browser 280
Graphic editor 478
Graphic editor preferences 62
Activity-chart 54
graphic editor preferences
Statechart 52
Graphical
editors 17
Graphical back animation (GBA) 478
Graphical elements
"and" sub-types 203
Activity 233
activity 202, 204
actor 202, 204
basic structures 203
boundary box 202, 204
control sub-types 203
data-store 203
execution sub-types 203
external router sub-types 203
external sub-types 203
Instance Activity 236
Instance Activity of Generic Charts 236
Instance State of Generic Chart 232
internal sub-types 203
Module 237
module 204
modules 202
non-basic structures 203
OR sub-types 203
router sub-types 203
State 230
state 202, 204
storage sub-types 203
transition 231
use case 202, 238
use case diagrams 204
Graphics editors 105, 110
charts 123
drawing charts 127

edit 112
layout 116
options 121
starting 123
tools 118
view 113, 179
Gravity Setting 121, 185
Groups
dive 184
surface 184
top 184

H

Hardware activation style 478
Help
generate support request 460
History
deep 474
History connector 478
creating 149
Hook 478
Horizontal Spacing 118

I

I 15
IBM
Passport Advantage 457
Icons
activity chart 138
Edit Attributes 213
flowchart 168
Label Existing Interactor 187
Module chart 142
Panel Builder 192
Select mode 141, 145, 150, 155, 163, 170, 187
Implementation tab 227
Include file
Documentor 474
Include relationship
creating 154
Infer device 479
In-flow 481
Info-flow 197
relationships 200
Information-flow 479
Initialization
dynamic data 307
Input column 368
Input device 479
Input element
for truth tables 369, 370
Instance 479
generic 477
Instance Activity properties 236
Instance Box
creating 169
naming 169
Instantiation 479
Integer 479
Integers 197
Inter runnable variables 454, 455
Interactor 479
binding 479
Interfaces
global 120
global report 280
local info 120
local report 279
report 480
reporting 279
Internal activity 480
creating 138
Internal module
creating 143
Internal router 272
preferences 284
Internal sub-type 203
Invoking RT Interface 133
Item
configuration 470
data 472

J

Joint connector 480
Junction connector 480
creating 140, 144, 149

L

Label 480
existing arrows 169
Language
action 462
document generation 474
simulation control 492
Libraries 31, 259
adding to project 269
working with 268
Library 480
License report 25
Life-Line
creating 162
Limitations
change tracking 99
quick-edit mode 207
textual elements' default value 201
Line
drawing 187
Line segments
drawing 187
Link to External File 16

Linksets
  exporting 358
List report 480
Literal 480
  constant 472
Local data 481
Local interface report 279
Local interfaces 120
Lock 481
Locking
  databank items 95
Log tab 11
Logic
  combinational 468
  multi-value 483
Long description 16
Lookup tables 226
Loop
  for 476
  while 503

M

Main window
  icons 17
Make Group 184
Management
  configuration 90
Menus
  pop-up 12
Message
  label 162
  note 162
Messages
  error 100, 102
  tab 11
Meter
  creating 186
MGE 483
MicroC code generator 295
  preferences 71
Mini-spec 481
Mirror
  X Axis 183
  Y Axis 183
Model 482
Models
  check 17
Modify chart usage 19
Modifying
  chart elements 239
  output devices 45
  project 32
Modifying elements 207
Module 295, 483
  naming 143
  storage 497
Module chart 142
  accessing 142
  icons 142
Module charts 205
  composition connector 144
Module properties 237
Module-chart 483
  graphic editor 483
Modules 202
  assigning behavior 299
  defining code 298
Monitor 483
Moving
  workarea 39
Multi-value logic 483

N

N2-chart report 484
Name
  project 488
Names
  length in search results 246
Naming
  activity 138
  AND state 148
  control flow 148
  flow line 139, 144
  instance box 169
  module 143
  OR state 148
New 110
Non-determinism 484

O

Off-page chart 484
Off-page connector 484
One hot state vector encoding 484
Open 110
  parent 110
  sub-chart 110
Open References 16
Opening
  project 35
  workarea 39
Operations
  not support in quick-edit mode 207
Optimize state translation 484
OR state
  creating 148
  naming 148
OR sub-type 203
Order Insignificant Line
  creating 162
Or-state 485
OSEK 307

Output column 371
Output device 479
Output devices
  deleting 46
  modifying 45
  setting up 43
Oval
  drawing 187
Overload state vector 485

P

Panel 485
  graphics editor 485
Panel Builder
  working with 192
Panel builder preferences 74
Panel editor
  accessing 175
  binding interactors 188
  edit 178
  file option 177
  group 184
  icons 186
  layout 181
  options 185
  tools 184
  transform 183
  working with 175
Panel graphic editor preferences 61
Parameter 485
  actual 464
  binding 485
  formal 477
Parameters
  generic 201
  preferences 87
  subroutine 201, 498
Parent chart 486
Partition Line
  creating 162
Passport Advantage 457
Paste 15
Permission 461
Playback
  simulation control 492
Playback SCL 486
Polygon
  drawing 187
Polygons
  drawing filled 187
Predefined type 486
Preference
  router 283
Preference Management 122
Preferences 46, 486
  activity 84
  Activity-chart graphic editor 54
  age specific 55
  auto panel specific 74, 77
  automatic refresh databank 93
  Beautify Indent Size 128
  bit_array 79
  bits 78
  check model 68
  component 57
  condition 82
  databank browser 63
  embedded rapid prototyper 72, 85
  Enum 80
  events 81
  external router 285
  flowchart graphic editor 57, 59
  for tracking changes 99
  general 49
  graphic editor 62
  loading predefined 86
  panel builder 74
  panel graphic editor 61
  properties 66
  Rational Statemate MicroC code generator 71
  Rational Statemate prototype C Code Generator 70
  real 77
  RT interface preferences 73
  sequence diagram graphic editor 59
  setting parameter 87
  simulation 64
  state 83
  Statechart graphic editor 52
  string preferences 77
  user-case diagram graphic editor 60
Preferences management 52
Preserve Selection 122
Print 110
Procedural statechart 486
Procedure 486
Procedure-like termination type 487
Profiles
  editor 295, 296
  name a new 298
  new 297
  sample 296
Project 487
  closing 36
  creating 30
  creating AUTOSAR 456
  deleting 36
  display settings 34
  manager 487
  member 487
  modifying 32
  name 488
  opening 35
Projects 29

members 31
multiple users 30
Rational DOORS 31
select configuration management 31
select shared area 31
Properties 16, 113, 184, 488
action 219
Activity 233
condition and event 222
Data-Item 220
Description tab 218
Design Attributes tab 217
display dialog box 208
editor 222
element types 195
Fields 223, 224
GDS usage 292
GDS visibility mode 292
General tab 212
generic activity chart 242
Instance Activity 236
Instance Activity of Generic Chart 236
Instance State of Generic Chart 232
Module 237
preferences 66
procedural statecharts 240
router 277
router elements 277
State 230
subroutine 225
subroutines 120, 209
transition 231
use case diagrams 238
user-defined type 221
Property
report 488
Property sheet 210, 488
activity graphical elements 233
chart elements 240, 242
graphical elements 229
information flow 224
instance activity 236
instance state of generic chart 232
modifying graphical elements 229
modifying textual elements 218
module graphical elements 237
subroutines 225
textual elements 219, 221
textual elements fields 223
Protection level 461
Prototype C Code Generator 17
Purge 92

Q

Query 489
Queue 199, 489
Queues 201

R

Racing 489
Ratioanl DOORS RT 17
Rational DOORS 333
associating projects to 342
attributes filtering 355
attributes to configure for export 353
configuration file 348
creating multiple linksets 362
exporting data 334, 350
exporting elements 363
exporting linksets 358
log files 347
methodology 336
Object CreationStamp attribute 365
operating system configuration 337
preferences for chart plots 343
preparing elements for export 350
project 31
re-exporting data 335
re-exporting elements 364
requirements 337
setting preferences 343, 347
shadow element 365
support for transitions 365
synchronizing data 335
Rational Rhapsody 308
synchronizing with Rational Statemate 309
Rational Statemate
Action language 226
block in Rational Rhapsody 308
development environment 1
flow ports 309
interface 1
main window 2
preferences 46
Prototype C Code Generator preferences 70
synchronizing with Rational Rhapsody 309
troubleshooting 309
Rational Statemate AUTOSAR generator 435
Rational Statemate MicroC code generator
preferences 71
Reaction
static 495
Reactive system 490
Reactive-controlled 490
Reactive-self 490
Read-only access 490
Read-only editing mode 491
Real 198, 491
Real preferences 77
Real-time operating system 491
Record 198, 209, 491
Record fields

default value 201
Records 200
Reduce workarea 40
Re-exporting
Rational DOORS elements 364
Referenced Sequence Diagram Line
creating 162
References
open 16
Reframe 118, 182
Refresh
databank 93
Register transfer level 492
Regular chart 492
Rename 15
Replicate 117, 181
Report
global interface 280
interface 480
list 480
local interface 279
N2-chart 484
on interfaces 279
properties 488
Reports 17
binding 184
database errors 100
Requirements traceability 492
Reserved words
reset operators 243
Reset
element default 243
exclude 243
pivot point 183
resizing elements 130
Revision management 90
Rotate 183
Router 271
drawing blocks 273
elements 271
external type 272
interface reporting 279
internal 272, 284
preferences 283
preferences for external 285
properties 277
working with 272, 301, 311
Router blocks
creating external 140
creating internal 140
exporting to DOORS 282
Router sub-type 203
Routers
compound flow-lines 276
external 275
internal 275
reduce flow lines 275
rules 275
RT Interface
invoking 133
RT interface preferences 73
RTL 492
RTOS 491

S

SAG 435
Save 110
Scale 183
Scenarios 365
SCL 492
playback 486
Scope definition area
AUTOSAR 437
SCP 492
Screen snapshot 460
Search 17
accessing stored list 248
advanced query 250
appending lists to 249
filtering lists of elements 249
finding used elements 258
lists of elements 246
locating referenced elements 256
results 246
saving lists of elements 247
starting 244
tab 10
Searching
charts 210
Select mode 141, 145, 150, 155, 163, 170, 187
Self
reactive 490
Sequence diagram 110
Sequence diagram graphic editor preferences 59
Sequence diagrams 159, 164, 205, 206
accessing 161
and properties 166, 167
animation 167
drawing 163
icons 162
integrating with model 164
Services
AUTOSAR 450
Session Status dialog box 25
Setting
pivot point 183
preferences 46
Rational DOORS preferences 343
router preferences 283
Setting preferences
editors 89
utilities 89
Setting up

output devices 43
Settings
project 34
Shadow copy 334
Simulation 17, 120, 492
control language 492
control playback 492
highlights option 122
support for reset operator 243
Simulation preferences 64
Single 492
Single structures 199
Slice array 465
SMAN 493
Snapshot 460
Source code control 471
Source control 90
Special characters 368
Starting
graphics editor 123
State 493
State preferences 83
State properties 230
Statechart graphic editor preferences 52
Statecharts 146, 148, 204, 206, 494
accessing 147
associating with activity 151
design attributes 151
graphics editor 494
modules 295
procedural 240, 486
test benches 151
States 202
AND 148
OR 148
Static reaction 495
Status 495
file 496
Step 496
superstep 498
STM 492
STM_BRANCH_3 391
STM_FIFO_ACTIVE 406
STM_FIFO_PASSIVE 410
STM_FORK_2 395
STM_FORK_3 397
STM_JOIN_2 399
STM_JOIN_3 402
STM_LIFO_ACTIVE 414
STM_LIFO_PASSIVE 418
STM_PMPT_ACTIVE 422
STM_PRTY_PASSIVE 426
STM_SINK 430
STM_SOURCE 432
STM_TMP_DIR variable 42
Storage module 497
Storage sub-type 203
Store
data 472
Stored list
accessing 248
Stretching elements 130
String 198, 497
String preferences 77
Structure 497
Stub 497
Sub-chart 110
creating 135
opening 110, 123, 134
Subroutines 99, 197, 498
flowcharts implemented as 170
parameter 498
parameters 201
properties 120, 209, 225
relationships 200
truth tables 374
Superstep 498
Surface Group 184
Switch
connector 149, 498
creating 169
expression 169
Synchronization
Rational Statemate and Rational Rhapsody 309
Synchronizing data 365
Synchronous 499

T

Tables
lookup 226
truth 226, 235
Tabs 2, 4
Charts 4
Files 5
Log 11
Messages 11
Search 10
Task 499
Tasks 225
Technical support
new customers 457
Template
Documentor 475
Termination connector 499
creating 149
Termination type
activity 463
procedure-like 487
Test benches
types 151
Test file 499
Testbenches 296, 499
Text

adjust indention of 128
creating note 141
entering 187
note 155
reformat with Beautify 128
transition note 150
use case description 152
Textual elements
action 197, 219
array structures 199
bit 198
bit-array 198
condition 196
condition and event 222
Data-Item 220
data-item 196
default values 201
enum-type 198
event 197
field 197
Fields 224
fields 223
info-flow 197
integer 197
queue structures 199
real 198
record 198
single structures 199
string 198
subroutines 197, 225
union 198
user defined type 221
user-defined 196
user-defined type 198, 221
Timeout event 500
Timeouts 445, 446
Timing constraint
line 162
note 162
Timing events
AUTOSAR 447
To-control connector 500
creating 140
Toolbar
AUTOSAR 437
Toolbars 2
Tools
external 226
Top Group 184
Trace file 500
Track Changes 120
Tracking
automatic change 98
change types for 99
changes 98
changes preferences 99
changes while editing 15
limitations 99
Transform
copy mirror X axis 183
copy mirror Y axis 183
mirror X axis 183
mirror Y axis 183
reset pivot point 183
rotate 183
scale 183
set pivot point 183
Transition priority 146
Transitions 500
default 148
properties 231
Rational DOORS support for 365
Trigger 500
Tristate 501
Troubleshooting
Rational Statemate with Rational Rhapsody 309
Truth tables 226, 235, 501
action column 372
actions 373
activities 373
default row 372
defining 378
execution 373
factorization of actions 377
factorization of cells 375
factorization of inputs 375
factorization of outputs 377
input columns 368
input elements 369, 370
output columns 371
output elements 371
special characters 368
subroutines 374
Types
data 473
enumerated 475
predefined 486
procedure-like termination 487
user-defined 200
user-defined basic 201
user-defined enum 201

U

UDT 502
Un-Group 184
Union 198, 209, 501
Unions 200, 201
Unresolved element 501
Usage 502
Use case diagrams 152, 202, 205, 206
"and" properties 155
accessing 152
attributes templates 156

creating 154
graphical elements 204
icons 154
linking to scenarios 157
properties 238
Use-case diagram graphic editor preferences 60
User-defined types 196, 200, 502
basic 201
enum 201
properties 221
Rational Statemate element 198
relationships 200
Utilities
database diagnostics 100, 102
setting preferences 46, 89
setup output devices 43

V

Variables 502
context 472
STM_TMP_DIR 42
Vector
one hot state encoding 484
overload state 485
Verilog 502
Version
changes 15
Version number 502
Vertical Spacing 118
Video capture 460
Viewing
all changes 15
last version-changes 15

W

Watchdog 499
Waveform 503
while loop 503
Workareas 37, 503
browser 503
copying 40
creating 37
deleting 41
moving 39
opening 39
reducing 40
shared 41
sharing & locking 42
temporary directory 42
unused elements 40
Writing
decision expression 169
switch expression 169