

Software through Pictures[®] Unified Modeling Language

Release 7.1

Creating UML Models



Software through Pictures Unified Modeling Language Creating UML Models

Release 7.1

Part No. 10-MN502/ST7100-01298/001

December 1998

Aonix reserves the right to make changes in the specifications and other information contained in this publication without prior notice. In case of doubt, the reader should consult Aonix to determine whether any such changes have been made. The software described in this document is furnished under a license and may be used or copied only in accordance with the terms of such license.

Copyright © 1998 by Aonix.™ All rights reserved.

This publication is protected by Federal Copyright Law, with all rights reserved. Unless you are a licensed user, no part of this publication may be reproduced, stored in a retrieval system, translated, transcribed, or transmitted, in any form, by any means, without prior written permission from Aonix. **Licensed users may make copies of this document as needed solely for their internal use—as long as this copyright notice is also reproduced.**

Trademarks

Aonix and the Aonix logo are trademarks of Aonix. ObjectAda is a trademark of Aonix. Software through Pictures is a registered trademark of Aonix. All rights reserved.

HP, HP-UX, and SoftBench are trademarks of Hewlett-Packard Inc. Sun and Solaris are registered trademarks of Sun Microsystems, Inc. SPARC is a registered trademark of SPARC International, Inc. Products bearing SPARC trademarks are based on an architecture developed by Sun Microsystems, Inc. Open Software Foundation, OSF, OSF/Motif, and Motif are trademarks of the Open Software Foundation, Inc. UNIX is a registered trademark in the United States and other countries, exclusively licensed through X/Open Company, Ltd. X Window System is a trademark of X Consortium, Inc. Windows NT is a trademark and Microsoft and Windows are registered trademarks of Microsoft Corporation in the United States and other countries. Adobe, Acrobat, the Acrobat logo, and PostScript are trademarks of Adobe Systems, Inc. Sybase and the Sybase logo are registered trademarks of Sybase, Inc. Adaptive Server, Backup Server, Client-Library, DB-Library, Open Client, PC Net Library, SQL Server, SQL Server Manager, SQL Server Monitor, Sybase Central, SyBooks, System 10, and System 11 are trademarks of Sybase, Inc.



© 1998 Aonix. All rights reserved.

World Headquarters

595 Market Street, 12th Floor
San Francisco, CA 94105
Phone: (800) 97-AONIX
Fax: (415) 543-0145
E-mail: info@aonix.com
<http://www.aonix.com>

Table of Contents

Preface

Intended Audience	xviii
Typographical Conventions	xviii
Contacting Aonix.....	xix
Technical Support	xix
Websites.....	xx
Reader Comments	xx
Related Reading	xx

Chapter 1 **Introduction**

What Are UML Models?.....	1-2
StP/UML Diagram Types.....	1-2
Use Case Diagram	1-3
Class Diagram	1-3
Interaction Diagrams.....	1-3
State Diagrams	1-4
Activity Diagrams	1-4
Implementation Diagrams	1-4
Stereotype Diagrams	1-5
Navigating Between Diagrams.....	1-5
Generating Code	1-6

Capturing External Classes	1-6
Printing.....	1-6
Features of StP/UML	1-7
Using the Editors	1-8
Adding Annotations to a UML Model	1-9
How StP/UML Stores Information	1-9
Diagram, Table, and Annotation Files	1-10
The Repository	1-10
Object Mapping.....	1-11
StP/UML Summary	1-12

Chapter 2 Using the StP Desktop

Starting the StP Desktop	2-2
Starting the StP Desktop	2-2
Understanding the Model Pane.....	2-3
Using StP/UML Desktop Commands.....	2-5
Invoking Commands.....	2-6
File Menu Summary	2-7
Code Menu Summary	2-8

Chapter 3 Creating a Use Case Diagram

What Are Use Cases?.....	3-1
Using the Use Case Editor	3-2
Starting the Use Case Editor	3-3
Navigating to Object References	3-3
Using the Use Case Symbol Toolbar	3-5
Using Display Marks.....	3-5
Creating a Use Case Diagram	3-6
Drawing a Use Case Diagram.....	3-7

Using Extends and Uses Links	3-9
Adding Extensibility Mechanisms	3-12
Annotating a Use Case Symbol	3-15
Decomposing a Use Case	3-18
Validating a Use Case Diagram	3-20

Chapter 4 Creating a Class Diagram

What Are Class Diagrams?	4-2
Using the Class Editor	4-2
Starting the Class Editor	4-4
Navigating to Object References	4-4
Using the Class Editor Symbol Toolbar	4-7
Using the UML Menu	4-7
Using Display Marks	4-9
Classes	4-12
Class Symbols	4-13
Class Names	4-13
Inserting and Labeling Classes	4-13
Inserting and Labeling Attributes	4-14
Inserting and Labeling Operations	4-16
Representing Parameterized and Instantiated Classes	4-18
Representing Interfaces	4-20
Adding Details about Classes	4-22
Using the Class Properties Dialog Box	4-22
Summary of Class Properties	4-24
Specifying Class Visibility	4-25
Setting Class Multiplicity	4-25
Designating Abstract Classes	4-26
Designating External Classes	4-27
Adding Parameters to a Parameterized Class	4-27

Designating a Stereotype	4-28
Specifying Constraints on Classes.....	4-28
Adding Tagged Values.....	4-29
Representing Objects (Class Instances).....	4-29
Representing and Labeling an Object Instance	4-30
Creating an Object Diagram.....	4-31
Creating Relationships	4-31
Changing Relationship Types	4-32
Creating Associations.....	4-33
Inserting and Labeling Associations.....	4-33
Inserting Reflexive Associations.....	4-34
Creating N-ary Associations	4-34
Creating Association Classes	4-36
Creating Or Associations.....	4-37
Designating Roles	4-38
Adding Details about Associations.....	4-39
Using the Association Properties Dialog Box.....	4-39
Setting Association Multiplicity	4-42
Adding Qualifiers.....	4-43
Setting Aggregation.....	4-44
Adding Directionality to Roles	4-46
Adding Ordering to Multiplicity.....	4-47
Adding Stereotypes	4-47
Adding Constraints	4-48
Adding Tagged Values.....	4-48
Adding Details to N-ary Associations.....	4-49
Using Specialized Relationships.....	4-49
Using Generalization Relationships.....	4-50
Using Binds Relationships.....	4-51
Using Dependency Relationships	4-52

Using Implements Relationships	4-52
Adding Details to Specialized Relationships	4-53
Creating Class Packages	4-55
Labeling Class Packages.....	4-55
Identifying Class Packages.....	4-55
Creating a Class Package.....	4-56
Package Scope Display Mark.....	4-57
Public Classes in a Class Package	4-57
Designating View Points.....	4-57
Navigating to ViewPoints	4-59
Changing View Points	4-60
Validating a Class Diagram.....	4-60
Checking a Diagram.....	4-60
Checking the Repository	4-61
Updating a Class Diagram	4-61
Updating a Class Table	4-63

Chapter 5 Defining Classes with the Class Table Editor

What Are Class Tables?.....	5-1
Using the Class Table Editor	5-2
Starting the Class Table Editor.....	5-3
Navigating to Object References	5-3
Using the UML Menu	5-5
Parts of the Class Table Editor	5-6
Hiding and Showing Table Sections	5-8
Choosing Valid Values for Cells	5-12
Adding Annotations for Member Functions.....	5-12
Defining and Modifying Classes	5-12
Defining a Class With a Class Table.....	5-12
Constructing a Class From the Repository	5-14

Displaying Inherited Operations.....	5-16
Validating a Class Table	5-18
Checking a Table	5-18
Checking the StP Repository.....	5-18
Updating a Class Table.....	5-19
Updating a Class Diagram	5-19
Deleting a Class From the Repository.....	5-19

Chapter 6 Creating a Sequence Diagram

What Are Sequence Diagrams?.....	6-1
Using the Sequence Editor.....	6-2
Starting the Sequence Editor.....	6-3
Navigating to Object References	6-3
Using the Sequence Editor Symbol Toolbar.....	6-5
Using the UML Menu	6-6
Using Display Marks.....	6-8
Drawing Sequence Diagrams.....	6-9
Representing Passive and Active Objects	6-10
Representing an Actor.....	6-11
Understanding Message Links	6-12
Types of Message Synchronization	6-13
Drawing and Labeling Links between Objects	6-14
Inserting Additional Messages	6-16
Resequencing Messages.....	6-16
Reordering Messages	6-17
Adjusting Symbols in a Sequence Diagram.....	6-17
Adding Details to Messages.....	6-19
Using the Properties Dialog Box	6-19
Summary of Message Properties.....	6-21
Setting Creation.....	6-22

Setting Destruction	6-22
Setting Transition Times	6-23
Adding Extensibility To Symbols	6-24
Using Extension Points	6-25
Using Packages	6-26
Generating a Sequence Diagram from a Collaboration Diagram.....	6-27
Validating a Sequence Diagram.....	6-27
Checking a Diagram.....	6-27
Checking the StP Repository.....	6-28

Chapter 7 Creating a Collaboration Diagram

What Are Collaboration Diagrams?.....	7-1
Using the Collaboration Editor.....	7-2
Starting the Collaboration Editor	7-3
Navigating to Object References	7-3
Using the Collaboration Editor Symbol Toolbar.....	7-6
Using the UML Menu	7-6
Using Display Marks.....	7-7
Drawing Collaboration Diagrams.....	7-8
Representing and Labeling Collaborations	7-9
Expanding a Collaboration	7-10
Representing and Labeling Objects	7-11
Setting Class Names.....	7-11
Representing Composite Objects	7-12
Making Objects Active or Passive	7-13
Representing a MultiObject.....	7-13
Representing an Actor	7-14
Creating an Object Diagram.....	7-14
Creating Message Links.....	7-15

Types of Message Synchronization	7-15
Drawing Links and Message Passing	7-16
Adding Details to Messages	7-17
Adding Extensibility to Symbols	7-18
Using Extension Points	7-19
Using Packages	7-19
Generating a Collaboration Diagram from a Sequence Diagram	7-20
Validating a Collaboration Diagram	7-21
Checking a Diagram	7-21
Checking the StP Repository	7-21

Chapter 8 Creating a State Diagram

What Are State Diagrams?	8-1
Using the State Editor	8-2
Starting the State Editor	8-3
Navigating to Object References	8-3
Using the State Editor Symbol Toolbar	8-6
Using the UML Menu	8-6
Using Display Marks	8-7
State Machine	8-8
States	8-9
State Labels	8-9
Drawing and Labeling States	8-10
Drawing Initial States	8-10
Drawing Final States	8-10
State Transition Links	8-11
Drawing and Labeling State Transition Links	8-12
Capturing Other Information About States	8-13
Decomposing a State	8-14

Decomposing a State	8-14
Decomposing a State In a Composite State	8-15
Navigating to the Parent State	8-16
Modeling Concurrent Activities	8-17
Showing Synchronization of Control Flows	8-18
Indicating State History	8-20
Adding Extensibility Mechanisms	8-21
Validating a State Diagram	8-21
Checking a Diagram	8-22
Checking the StP Repository	8-22

Chapter 9 Defining States with the State Table Editor

What Is the State Table Editor?	9-1
Using the State Table Editor	9-2
Starting the State Table Editor	9-3
Navigating to Object References	9-3
Parts of the State Table Editor	9-5
Hiding and Showing Table Sections	9-6
Manipulating States	9-7
Defining a New State	9-7
Sorting a State Table	9-8
Validating a State Table	9-9
Deleting a State Table	9-10

Chapter 10 Creating an Activity Diagram

What Are Activity Diagrams?	10-1
Using the Activity Editor	10-2
Starting the Activity Editor	10-3
Navigating to Object References	10-3

Using the Activity Editor Symbol Toolbar	10-5
Using the UML Menu	10-5
Using Display Marks.....	10-6
State Machine	10-7
Action States	10-7
Drawing and Labeling Action States	10-8
Drawing Initial States.....	10-8
Drawing Final States	10-9
State Transition Links.....	10-9
Drawing and Labeling State Transition Links.....	10-9
Decisions	10-10
Using Objects with Action States.....	10-12
Modeling Synchronization	10-13
Adding Extensibility Mechanisms	10-15
Validating an Activity Diagram.....	10-15
Checking a Diagram.....	10-16
Checking the StP Repository.....	10-16

Chapter 11 Creating a Component Diagram

What Are Component Diagrams?	11-1
Using the Component Editor	11-2
Starting the Component Editor	11-3
Navigating to Object References	11-3
Using the Component Editor Symbol Toolbar	11-3
Using Display Marks.....	11-4
Drawing Software Components	11-5
Representing and Labeling Components.....	11-6
Adding Objects to a Component.....	11-7
Adding Interfaces to a Component.....	11-8

Drawing Software Dependencies	11-9
Representing and Labeling Dependencies.....	11-9
Adding Extensibility Mechanisms	11-10
Creating a Deployment Diagram	11-10
Validating a Component Diagram	11-11
Checking a Diagram.....	11-11
Checking the StP Repository.....	11-11

Chapter 12 Creating a Deployment Diagram

What Are Deployment Diagrams?	12-1
Using the Deployment Editor	12-2
Starting the Deployment Editor	12-3
Navigating to Object References	12-3
Using the Deployment Symbols List	12-3
Using the UML Menu	12-4
Using Display Marks.....	12-4
Drawing Hardware Components.....	12-5
Representing and Labeling Deployment Nodes.....	12-5
Adding Software Components	12-6
Drawing Hardware Dependencies.....	12-7
Representing Connections.....	12-7
Adding Extensibility Mechanisms	12-7
Deploying a Component Diagram.....	12-8
Validating a Deployment Diagram	12-9
Checking a Diagram.....	12-9
Checking the StP Repository.....	12-9

Chapter 13 Creating a Stereotype Diagram

What Are Stereotype Diagrams?	13-1
-------------------------------------	------

Using the Stereotype Editor	13-2
Starting the Stereotype Editor.....	13-3
Navigating to Object References	13-3
Using the Stereotype Symbol List	13-3
Using the UML Menu	13-4
Using Display Marks.....	13-4
Drawing a Stereotype Diagram	13-5
Representing and Labeling Stereotypes	13-5
Adding Extensibility Mechanisms	13-6
Pre-defined UML Stereotypes.....	13-6
Validating a Stereotype Diagram.....	13-10
Checking a Diagram.....	13-10
Checking the StP Repository.....	13-10

Chapter 14 Renaming Symbols

What Are the Options for Renaming Symbols?	14-1
Objects and Object References	14-2
Choosing an Option	14-3
Retrying a Label	14-4
Remapping to an Object without Annotations	14-5
Remapping to an Object with Annotations	14-6
Changing a Label.....	14-7
Deleting Unreferenced Objects	14-7
Cloning Objects by Relabeling Symbols.....	14-7
Renaming Objects Systemwide.....	14-8
Objects That Can Be Renamed.....	14-9
Renaming Instances in StP/UML.....	14-10
Renaming an Object Systemwide.....	14-10
Rename Options.....	14-11
Previewing Your Rename Selection	14-15

Global Renaming Errors	14-17
Effects on Dependent Objects	14-17
The Visual Effects of Object Rename	14-20

Chapter 15 Generating UML Reports

What Is in UML Reports?	15-1
Running UML Reports.....	15-2
Using External Variables in Reports	15-2
Generating Language-specific UML Reports	15-2
Running a Report from the Script Manager	15-3
Running a Report Using QRL.....	15-5
Format File, QRL Scripts, and Format Include Files	15-5
Viewing UML Reports	15-6
Changing External Variables.....	15-7
Setting Document Information External Variables.....	15-7
Setting Document Layout External Variables.....	15-8
Setting Document Chapter External Variables.....	15-10
Setting Limit Evaluation External Variables.....	15-14

Appendix A PDM Types and Application Types

Persistent Data Model Types.....	A-1
Application Types.....	A-2
Symbol Mappings.....	A-3
Use Case Diagram Mappings	A-3
Class Diagram Mappings	A-4
Class Table Mappings	A-5
Sequence Diagram Mappings.....	A-7
Collaboration Diagram Mappings	A-9
State Diagram Mappings.....	A-11

State Table Mappings	A-12
Activity Diagram Mappings	A-13
Component Diagram Mappings.....	A-14
Deployment Diagram Mappings	A-14
Stereotype Diagram Mappings.....	A-15

Appendix B Command Line Commands

Index

Preface

Creating UML Models presents a comprehensive, task-oriented approach to using Software through Pictures Unified Modeling Language (StP/UML), a multi-user development environment for:

- Analyzing and defining object-oriented models
- Creating requirement, logical, physical, dynamic, and static views of classes and objects
- Navigating between models
- Validating UML models for consistency and completeness
- Generating C++, Java, IDL, TOOL, or Ada_95 code

StP/UML uses notation that is described in *Unified Modeling Language User Guide* (Booch et al 1998) and *Unified Modeling Language Reference Manual* (Rumbaugh et al 1998).

Creating UML Models builds on information from these books, but does not fully duplicate their descriptions of UML notation and semantics. For complete information about UML, refer to these books.

This manual is part of a complete StP documentation set that also includes “Core” manuals: [Fundamentals of StP](#), [Customizing StP](#), [Query and Reporting System](#), [Object Management System](#) and [StP Administration](#). Basic information about the StP user interface is documented in [Fundamentals of StP](#).

Intended Audience

The audience for this manual includes analysts and developers who use StP/UML to create class, interaction, implementation, and use case model diagrams.

The information in this manual assumes that you are familiar with:

- Fundamentals of Software through Pictures, including using diagram editors, the Object Annotation Editor, and annotation templates
- Object-oriented concepts
- Object-oriented modeling, as documented in *Unified Modeling Language User Guide* (Booch et al 1998) and *Unified Modeling Language Reference Manual* (Rumbaugh et al 1998)
- Conventions of Windows NT

Refer to the appropriate documentation if you have any questions on these topics.

Typographical Conventions

This manual uses the following typographical conventions:

Table 1: Typographical Conventions

Convention	Meaning
Palatino bold	Identifies commands, buttons, and menu items.
Courier	Indicates system output and programming code.
Courier bold	Indicates information that must be typed exactly as shown, such as command syntax.
<i>italics</i>	Indicates pathnames, filenames, and ToolInfo variable names.

Table 1: Typographical Conventions (Continued)

Convention	Meaning
<angle brackets>	Surround variable information whose exact value you must supply.
[square brackets]	Surround optional information.

Contacting Aonix

You can contact Aonix using any of the following methods.

Technical Support

If you need to contact Aonix Technical Support, you can do so by using the following email aliases:

Table 2: Technical Support Email Aliases

Country	Email Alias
Canada	support@aonix.com
France	customer@aonix.fr
Germany	stp-support@aonix.de
United Kingdom	stp-support@aonix.co.uk
United States	support@aonix.com

Users in other countries should contact their StP distributor.

Websites

You can visit us at the following websites:

Table 3: Aonix Websites

Country	Website URL
Canada	http://www.aonix.com
France	http://www.aonix.fr
Germany	http://www.aonix.de
United Kingdom	http://www.aonix.co.uk
United States	http://www.aonix.com

Reader Comments

Aonix welcomes your comments about its documentation. If you have any suggestions for improving *Creating UML Models*, you can send email to docs@aonix.com.

Related Reading

Creating UML Models is part of a set of Software through Pictures documentation. For more information about StP/UML and related subjects, refer to the sources listed in Table 4.

Table 4: Further Reading

For Information About	Refer To
Using the StP Desktop	<i>Fundamentals of StP</i>
Using StP editors	<i>Fundamentals of StP</i>

Table 4: Further Reading (Continued)

For Information About	Refer To
StP/UML tutorial	<i>Getting Started with StP/UML</i>
Generating C++, Java, IDL, TOOL, or Ada_95 code from StP/UML models	<i>Generating and Reengineering Code</i>
Printing diagrams, tables, and reports	<i>Fundamentals of StP, Query and Reporting System</i>
Internal components of StP and how to customize an StP installation	<i>Customizing StP</i>
Using the Script Manager, writing Query and Reporting Language (QRL) scripts	<i>Query and Reporting System</i>
StP Object Management System (OMS)	<i>Object Management System</i>
Installing StP	<i>StP Installation Guide</i>
Interacting directly with the StP storage manager, Sybase SQL Server	<i>StP Guide to Sybase SQL Server</i>
Managing an StP installation	<i>StP Administration</i>
Annotation items	Object Annotation Editor Help menu

1 Introduction

This chapter provides an introduction to StP/UML and is intended for all users. Topics covered are as follows:

- [“What Are UML Models?” on page 1-2](#)
 - [“StP/UML Diagram Types” on page 1-2](#)
 - [“Generating Code” on page 1-6](#)
 - [“Capturing External Classes” on page 1-6](#)
 - [“Printing” on page 1-6](#)
 - [“Features of StP/UML” on page 1-7](#)
 - [“Using the Editors” on page 1-8](#)
 - [“Adding Annotations to a UML Model” on page 1-9](#)
 - [“How StP/UML Stores Information” on page 1-9](#)
 - [“StP/UML Summary” on page 1-12](#)
-

What Are UML Models?

UML models are representations of objects in the real world and their interrelationships. In this manual, the real-world domain of activity used for the UML model is an electronic mail (Email) system. Classes in the Email system include *MailSystem*, *User*, *Receiver*, *Mailbox*, *Conference*, and *Message*.

There are five major, overlapping components of UML models:

- A requirements model—A behavioral view of system requirements interacting with external agents
- A logical model—A conceptual view of the entire domain of activity that is not concerned with physical implementation
- A physical model—A material view of the details of implementation, such as the hardware and software components of a system
- A static model—A descriptive view of the characteristics of parts of the domain of activity
- A dynamic model—A behavioral view of the parts of the domain of activity

StP/UML Diagram Types

Each of the five components of a UML model represents a view of the entire model. In StP/UML, these views are represented through the following diagrams:

- Use case diagram
- Class diagram
- Interaction diagrams
- State diagram
- Activity diagram
- Implementation diagrams

In addition, StP/UML provides stereotype diagrams for creating UML stereotypes.

Use Case Diagram

A use case diagram models behavior scenarios of a system interacting with external agents. Use case diagrams provide analysis and overview information and help in the early phases of the system development life cycle.

For information about developing use case diagrams, see [Chapter 3, “Creating a Use Case Diagram.”](#)

Class Diagram

A class diagram is a graphical representation of the classes in the real-world domain and the relationships between the classes. Class diagrams can have associated class tables, which provide detailed information about the classes in the system. Class diagrams and class tables are components in the logical-static view of a system.

For information about developing a class diagram, see [Chapter 4, “Creating a Class Diagram,”](#) and [Chapter 5, “Defining Classes with the Class Table Editor.”](#)

Interaction Diagrams

Interaction diagrams contain views of objects and the message passing between them. Interaction diagrams emphasize message types, concurrency, visibility, and return values. The views are shown in two types of interaction diagrams: sequence and collaboration.

Sequence Diagram

A sequence diagram shows the order of messages sent between objects.

For information about developing sequence diagrams, see [Chapter 6, “Creating a Sequence Diagram.”](#)

Collaboration Diagram

A collaboration diagram is a graphical representation of objects and relationships between objects. Collaboration diagrams complement class diagrams in the logical-static view of a system.

For information about developing collaboration diagrams, see [Chapter 7, “Creating a Collaboration Diagram.”](#)

State Diagrams

A state diagram represents states of a single object in the modeling domain and the actions and activities that transform the object from one state to another. State diagrams can have associated state tables, which provide detailed information about the states of an object. State diagrams and state tables present a logical-dynamic view of the system.

For information about developing a state diagram, see [Chapter 8, “Creating a State Diagram.”](#) For information about developing a state table, see [Chapter 9, “Defining States with the State Table Editor.”](#)

Activity Diagrams

An activity diagram is a variation of a state diagram used for modeling the procedural flow of a task. Using activity diagrams, you can model the steps required to perform an operation, a use case, or a behavior spanning several use cases. Activity diagrams present a logical-dynamic view of the system.

For information about developing an activity diagram, see [Chapter 10, “Creating an Activity Diagram.”](#)

Implementation Diagrams

Implementation diagrams show views of implementation, including the physical structure and allocation of software to hardware. These views are shown in component and deployment diagrams.

Component Diagram

A component diagram shows the software dependencies of a system. Component diagrams complement deployment diagrams in presenting a physical-static view of a system.

For information about drawing component diagrams, see [Chapter 11, “Creating a Component Diagram.”](#)

Deployment Diagram

A deployment diagram shows the allocation of software to hardware. Deployment diagrams complement component diagrams in showing a physical-static view of a system.

For information about drawing deployment diagrams, see [Chapter 12, “Creating a Deployment Diagram.”](#)

Stereotype Diagrams

A stereotype diagram shows a user-defined stereotype. Stereotypes expand the semantics of an object (class, attribute, association, and so on) and are used throughout UML diagrams.

For information about drawing stereotype diagrams and for a list of pre-defined stereotypes, see [Chapter 13, “Creating a Stereotype Diagram.”](#)

Navigating Between Diagrams

StP/UML enables you to navigate between specific types of diagrams. For information about navigating between diagrams, see the chapters in this manual that describe the individual editors.

Generating Code

You can generate C++, Java, IDL, TOOL, and Ada_95 code from valid StP/UML class diagrams and tables.

For information about generating code from a UML model, see [*Generating and Reengineering Code*](#).

Capturing External Classes

Reverse engineering provides an automated method for capturing specification-level information about object classes in existing C++, Java, IDL, and TOOL libraries for reuse in StP/UML models.

For information about capturing external classes using reverse engineering, see [*Generating and Reengineering Code*](#).

Printing

StP provides capabilities for printing diagrams and tables from the StP Desktop, all UML diagram and table editors, and the command line.

You can choose to generate print jobs in Postscript, FrameMaker, HTML (for tables only), or RTF format. There are also features for creating page format settings and saving them for later use, and previewing multi-page diagram print jobs online.

For more information on printing capabilities, see [*Fundamentals of StP*](#).

Features of StP/UML

As described in Table 1, StP/UML provides a variety of editors and utilities that enable you to develop UML models. For a graphical representation of these editors and utilities, see [Figure 1 on page 1-12](#).

Table 1: StP/UML Editors and Utilities

Editor or Utility	Use for
StP Desktop	Starting all the StP/UML editors and utilities
Use Case Editor	Drawing use case diagrams
Class Editor	Drawing a logical-static view of classes in a system
Class Table Editor	Providing full definitions of class attributes and operations
Sequence Editor	Drawing a dynamic view of objects in a system and the messages sent between them
Collaboration Editor	Drawing a logical-static view of objects in a system and the messages sent between them
State Editor	Drawing a logical-dynamic view of states of objects in a system
State Table Editor	Providing full definitions of state actions
Activity Editor	Drawing a logical-dynamic view of a procedural flow of a task
Component Editor	Drawing a physical-static view of the structure of a system
Deployment Editor	Drawing a physical-static view of the hardware resource allocation of a system
Stereotype Editor	Defining stereotypes

Table 1: StP/UML Editors and Utilities (Continued)

Editor or Utility	Use for
Requirements Table Editor	Capturing system requirements (described in Fundamentals of StP)
Object Annotation Editor (OAE)	Adding information to all aspects of a UML model
Script Manager	Generating documents based on StP Repository information (described in Query and Reporting System)

Using the Editors

This manual contains instructions for using StP diagram and table editors to create UML models specifically. The manual does not contain general information about using the editors. For complete information about standard features of StP editors, see [Fundamentals of StP](#).

Table 2 provides an overview of frequently used functions and commands available from the StP standard editor menus.

Table 2: Selected StP Editor Functions

To	Use
Create a new diagram or table	Save As from the File menu
Open an existing diagram or table	Open from the File menu
Save a diagram or table to a file and to the repository	Save from the File menu
Print a diagram or table	Print from the File menu
Exit an editor	Exit from the File menu

Adding Annotations to a UML Model

Most annotations in a UML model are added through the Properties dialog box for a selected symbol. An annotation is a group of descriptive notes about an object in the system being modeled. You can annotate most elements of a UML model, including:

- Source code description
- Definitions for classes, attributes, and operations

For more information on adding annotations through the Properties dialog box, see the chapter in this manual that discusses the appropriate editor.

In some cases, you use the Object Annotation Editor (OAE) to annotate objects. Online descriptions of annotation notes and items are available through the OAE Help menu. To display descriptions, select a note or item and choose **On Selection**. For general information about using the OAE, see [Fundamentals of StP](#).

Several annotations support programming language code generation. For more information about annotating a diagram for code generation, see [Generating and Reengineering Code](#).

How StP/UML Stores Information

As you model data with StP/UML, you capture a concept in a diagram or table. In diagrams, the information is represented as symbols and labels. In tables, the information is represented as text in cells. Diagram and table elements can have annotations. (StP/UML treats text in some table cells as annotations.)

StP manages elements of a UML model by storing all information associated with the model both in files and as objects in the repository.

Diagram, Table, and Annotation Files

A diagram, table, or annotation file is an ASCII representation of information about a model. Each file contains information about a single diagram, table, or set of annotations. A diagram file stores information about the relationships among objects represented by symbols in the diagram. A table file stores information about the table cells in a table. An annotation file stores information about the annotations for a single element in a diagram or table.

Every occurrence of a symbol, table cell, or annotation within a system model is recorded in its related diagram, table, or annotation file. Thus, information about a single element can be stored repeatedly in several different ASCII files or more than once in the same file. For example, if a single class appears as a symbol twice in one diagram and once in another diagram, the information about the class symbol is recorded three times in the ASCII files—twice in one diagram file and once in the other diagram file.

The Repository

When you save your UML model, StP writes information (application types) from the UML editors to the repository, a central pool of data that StP can access in a consistent manner. Each object in a UML model is stored in the repository only once, regardless of the number of times that object is used in your model's diagrams or tables. Each appearance of an object in a diagram or table is a “reference” to the single object that is stored in the repository. The repository ensures that the information about an object is consistent across the entire model.

The repository stores application types as persistent objects. Each object in the repository is unique. An object contains all the information about a construct in the design domain.

Each application type maps to a persistent object type. The scheme that determines the meaning of objects and their inter-relationships is the “Persistent Data Model” (PDM).

For more information about persistent objects, the Persistent Data Model, and the schema for the repository, see [Object Management System](#).

Object Mapping

All symbols on diagrams and cells in tables in a particular system are associated, or “mapped,” to objects in the repository. The single object provides the identity for all occurrences (“references”) of the symbol or cell. For example, a class that appears five times in the model diagrams and files appears only once as an object in the repository, but with multiple references.

Consequently, you need to enter information about an object only once because StP/UML associates the information with each symbol or cell that represents the object. Also, because all related information is stored with the object, you cannot enter conflicting information about anything in the model.

For specific StP/UML-to-repository mappings, refer to [Appendix A, “PDM Types and Application Types.”](#)

Renaming an Object

StP provides two options for renaming a symbol on a diagram or text in a table. You can:

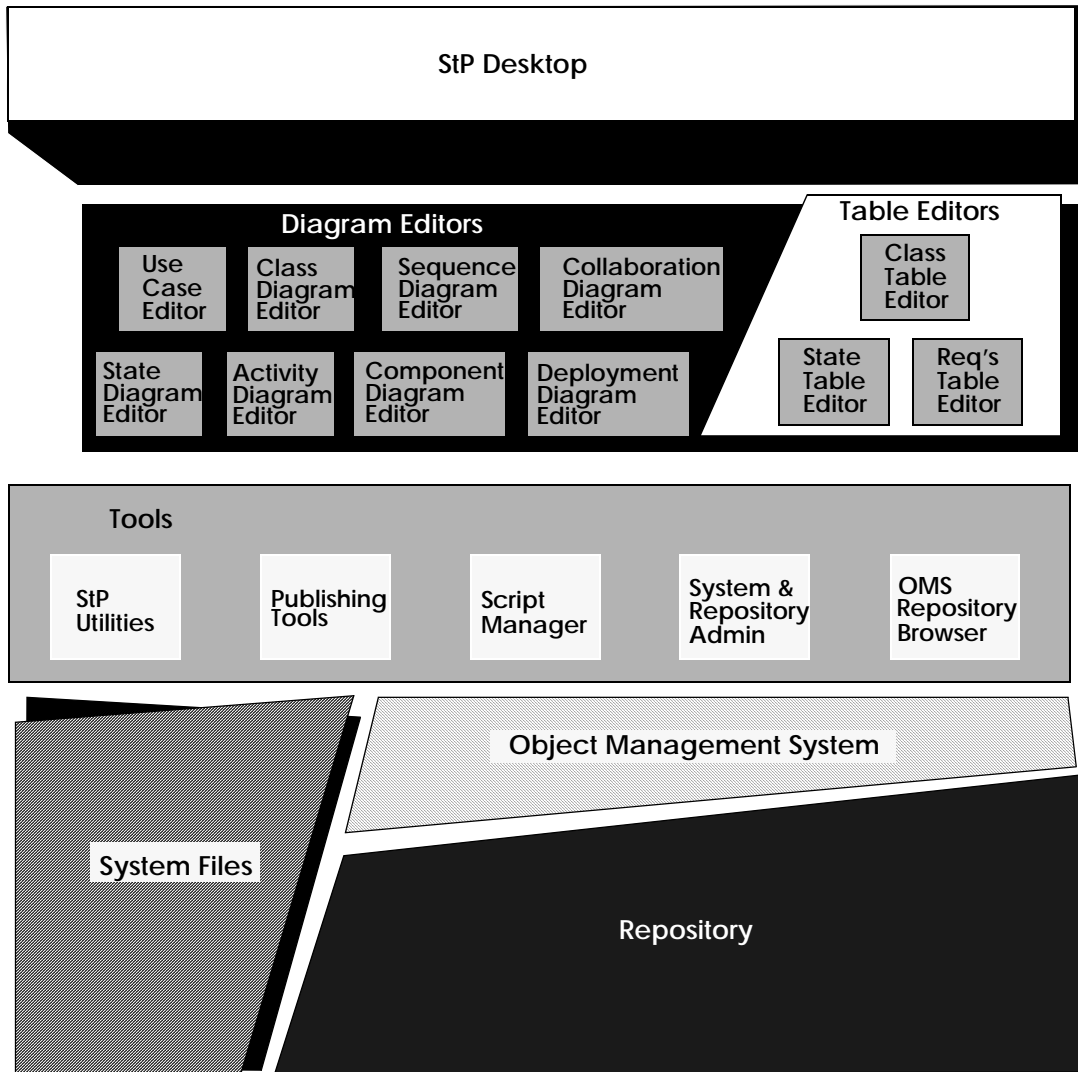
- Change the name of the symbol on the current diagram only
- Change the name of the symbol (or text in a table cell) in other diagrams and tables that contain it

For information on using the **Rename Object Systemwide** command, see [Chapter 14, “Renaming Symbols.”](#)

StP/UML Summary

Figure 1 provides a graphical representation of StP/UML.

Figure 1: Overview of StP/UML



2

Using the StP Desktop

The Software through Pictures Desktop (StP Desktop) provides access to all StP/UML editors and tools, as well as to all Core commands and utilities. Using the StP Desktop, you can perform a variety of tasks, including:

- Starting any StP/UML editor
- Generating code for all classes in the repository
- Printing a diagram, table, or report
- Checking consistency and completeness of diagrams or tables
- Managing locks for diagrams and tables

Topics covered in this chapter are as follows:

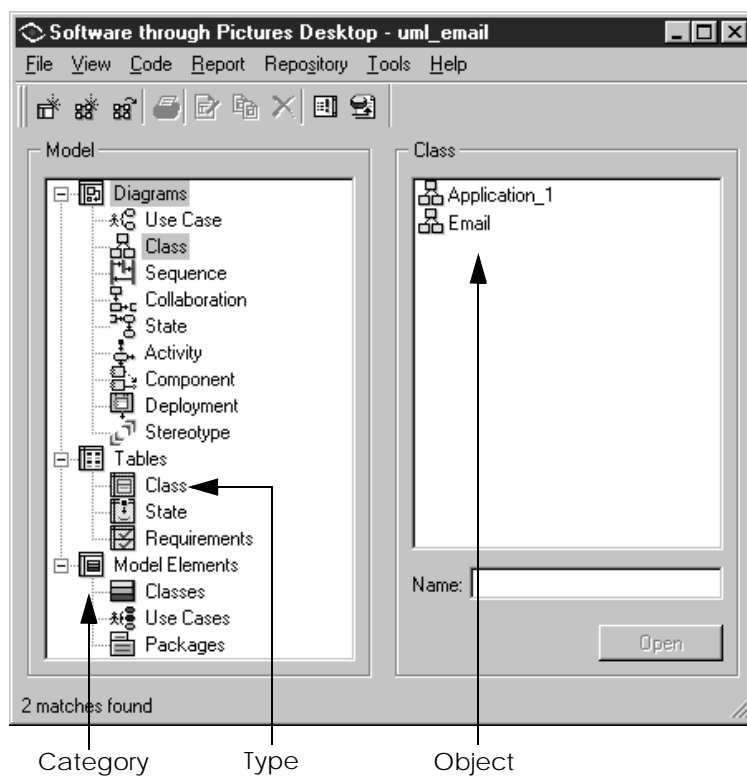
- [“Starting the StP Desktop” on page 2-2](#)
- [“Understanding the Model Pane” on page 2-3](#)
- [“Using StP/UML Desktop Commands” on page 2-5](#)

This chapter focuses on using the StP Desktop with StP/UML. For general information about the StP Desktop, see [Fundamentals of StP](#).

Starting the StP Desktop

This section tells you how to start the StP Desktop. The StP Desktop is shown in Figure 1.

Figure 1: Software through Pictures Desktop



Starting the StP Desktop

To start the StP Desktop for StP/UML:

1. Click the **Start** button.
2. Choose the **Programs** menu.
3. Choose **Aonix Software through Picture > StP UML**.

Understanding the Model Pane

The Model pane on the StP Desktop lists an icon for each category: Diagrams, Tables, and Model Elements. Each category contains subcategories for StP/UML. When you select a subcategory, any existing objects for that subcategory are listed in the Objects pane. Click on a category to open it and display the subcategories; click on the category again to close it.

Subcategories represent collections of objects (such as diagrams, tables, or classes) that have associated commands. [Table 1](#) provides an overview of StP/UML categories and their subcategories.

The subcategories that appear on the StP Desktop are determined by your installation configuration. Your desktop might include additional subcategories from other installed StP products.

Table 1: StP/UML Categories and Subcategories

Category	Types	Description
Diagrams	Use Case	Provides commands that operate on use case diagrams
	Class	Provides commands that operate on class diagrams
	Sequence	Provides commands that operate on sequence diagrams
	Collaboration	Provides commands that operate on collaboration diagrams
	State	Provides commands that operate on state diagrams
	Activity	Provides commands that operate on activity diagrams
	Component	Provides commands that operate on component diagrams
	Deployment	Provides commands that operate on deployment diagrams
	Stereotype	Provides commands that operate on stereotype diagrams
Tables	Class	Provides commands that operate on class tables
	State	Provides commands that operate on state tables
	Requirements	Provides commands that operate on the Requirements Table Editor

Table 1: StP/UML Categories and Subcategories (Continued)

Category	Types	Description
Model Elements	Classes	Provides code generation and validation commands that operate on classes or the whole StP/UML model in the repository
	Use Cases	Provides commands for editing use cases and related annotations, as well as commands that operate on the whole model in the repository
	Packages	Provides Ada_95 code generation commands that operate on packages; provides code generation commands for all supported languages that operate on the whole model in the repository

Using StP/UML Desktop Commands

Of the seven StP Desktop menus (**File**, **View**, **Code**, **Report**, **Repository**, **Tools**, and **Help**), only the **File** and **Code** menus contain functions specific to StP/UML.

The commands under the **File** and **Code** menus typically act on objects or perform tasks relevant to the selected subcategory. Each command starts a process, such as generating C++ code or opening a diagram.

Most commands are inactive (grayed out) until you select an object from the Objects pane. While some commands are available only when one or more objects have been selected, other commands, such as **File > New > Class Diagram**, do not require a selected object.

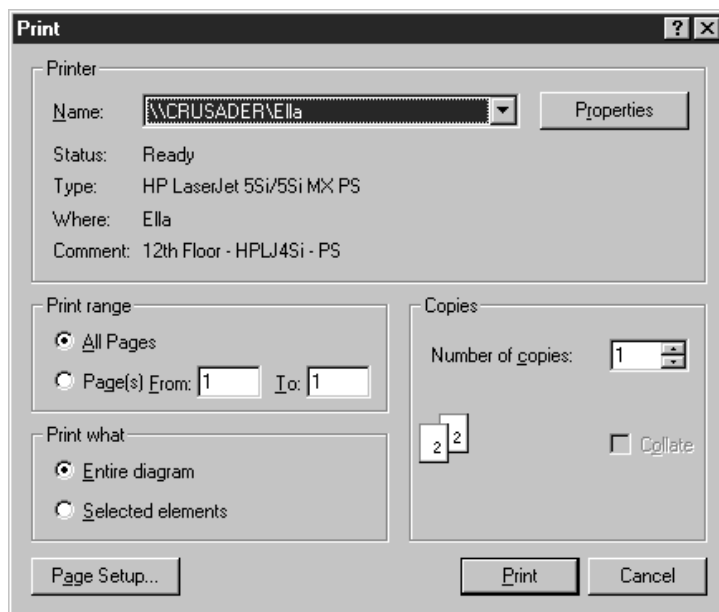
Commands common to all StP products are described in [Fundamentals of StP](#).

Invoking Commands

Except for commands that start editors (**File** > **New...** or **File** > **Open...**), each command specific to StP/UML displays a dialog box containing options for the selected command. Using the dialog box, you can change the command's options before executing it.

Figure 2 shows a typical command option dialog box, in this case, for the **Print Diagram** command.

Figure 2: Print Diagram Option Dialog



To invoke a command from the **File** or **Code** menu that requires a selected object:

1. Click on the Diagram, Table, or Model Element category in the Model pane.
2. Select a subcategory from the Model pane.
3. Select one or more object names from the Objects pane.
4. Select a command from the **File** or **Code** menu.

If you select **File > Open Diagram** or **File > Open Table**, the command is executed. Skip the remaining steps.

For all other commands, the command's option dialog box appears.

5. Adjust the command's options as desired.
6. Click **OK** or **Apply**.

OK executes the command and dismisses the option dialog; **Apply** executes the command but retains the option dialog for future use.

File Menu Summary

Table 2 describes the commands available from the **File** menu for each StP/UML component type in the Model pane. For information about commands for StP Core tools, see [Fundamentals of StP](#).

Table 2: StP/UML Commands Summary

Command	Description	For More Information, See
New > System	Creates a system	Fundamentals of StP
New > [Diagram/Table]	Creates a new diagram or table	Fundamentals of StP
Open Diagram/Table	Loads the selected diagram or table in an editor; starts the editor, if necessary	
Print Diagram/Table	Prints the selected diagram or table	Fundamentals of StP
Print Diagram/Table As	Prints the selected diagram or table to an output file based on selected output type	Fundamentals of StP
Copy Diagram/Table	Makes a copy of the selected diagrams or tables	Fundamentals of StP
Delete Diagram/Table	Deletes the selected diagrams or tables from the current system	Fundamentals of StP
Rename Diagram/Table	Renames the selected diagram or table	Fundamentals of StP
Open System	Opens an existing system	Fundamentals of StP
Copy System	Copies an existing system	Fundamentals of StP

Table 2: StP/UML Commands Summary (Continued)

Command	Description	For More Information, See
Destroy System	Destroys an existing system	Fundamentals of StP
Exit StP	Exits StP, including all editors and the StP Desktop	Fundamentals of StP
Exit Desktop	Exits StP Desktop only	

Code Menu Summary

Table 3 describes the commands available from the **Code** menu for each StP/UML component type in the Model pane.

Table 3: StP/UML Commands Summary

Command	Description	For More Information, See
C++ > Generate C++ for Diagram's Classes	Generates C++ code for classes in the selected class diagram or for the selected class	Generating and Reengineering Code
C++ > Generate C++ for Whole Model	Generates C++ for all the classes in the system	
Java > Generate Java for Diagram's Classes	Generates Java code for classes in the selected class diagram or for the selected class	Generating and Reengineering Code
Java > Generate Java for Whole Model	Generates Java for all the classes in the system	
IDL > Generate IDL for Diagram's Classes	Generates IDL code for classes in the selected class diagram or for the selected class	Generating and Reengineering Code
IDL > Generate IDL for Whole Model	Generates IDL for all the classes in the system	

Table 3: StP/UML Commands Summary (Continued)

Command	Description	For More Information, See
TOOL > Generate TOOL for Diagram's Classes	Generates TOOL code for classes in the selected class diagram or for the selected class	Generating and Reengineering Code
TOOL > Generate TOOL for Whole Model	Generates TOOL for all the classes in the system	
Ada 95 > Generate Ada 95 for Diagram's Packages	Generates Ada 95 code for packages in the selected class diagram or for the selected class	Generating and Reengineering Code
Ada 95 > Generate Ada 95 for Whole Model	Generates Ada 95 for all the classes in the system	
Reverse Engineering > Parse Source Code	Extracts all information needed to generate design documentation from the source code	Generating and Reengineering Code
Reverse Engineering > Extract Source Code Comments	Associates comments extracted by the parser	
Reverse Engineering > Generate Model from Parsed Source Files	Automatically creates a graphical model of the reverse engineered software	
Reverse Engineering > Check RE Semantic Model Locks	Checks to find out the lock status of a semantic model.	
Reverse Engineering > Check RE Semantic Model Consistency	Checks to find out if a semantic model has been corrupted.	
Reverse Engineering > Generate Model from TOOL Code	TOOL code only. Automatically creates a graphical model of the reverse engineered software, consisting of class diagrams, class tables, and annotations. This model is saved to the repository.	

3

Creating a Use Case Diagram

This chapter describes how to create use case diagrams. Topics covered are as follows:

- [“What Are Use Cases?” on page 3-1](#)
- [“Using the Use Case Editor” on page 3-2](#)
- [“Creating a Use Case Diagram” on page 3-6](#)
- [“Validating a Use Case Diagram” on page 3-20](#)

What Are Use Cases?

Use cases provide a means for expressing high-level interactions with the system. Typically, these interactions involve a sequence of transactions that are initiated by a system user (called an actor) when using the system.

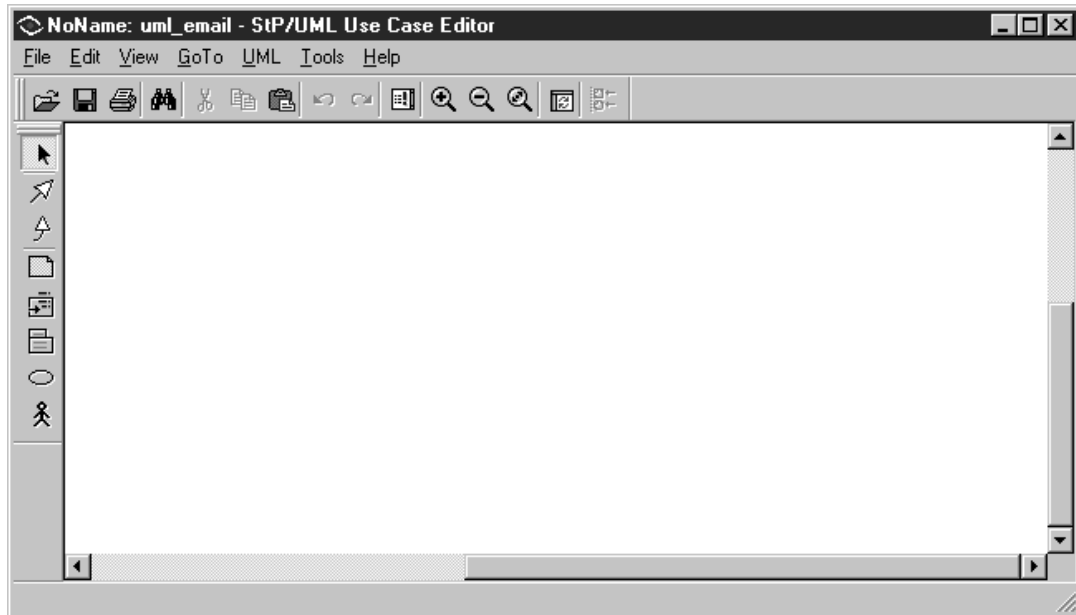
The notation associated with use cases allows end users and managers to model a system in the early stages of development. In this way, use cases can help you determine a system’s requirements.

After creating a use case diagram, you can decompose each use case into a set of associated scenarios. For more information, see [“Decomposing a Use Case” on page 3-18](#).

Using the Use Case Editor

You create use case diagrams using the Use Case Editor.

Figure 1: StP/UML Use Case Editor



The Use Case Editor provides all the functions and menu options of every StP diagram editor. For general information about using StP diagram editors, see [Fundamentals of StP](#).

For a tutorial on using the Use Case Editor, refer to [Getting Started with StP/UML](#).

Starting the Use Case Editor

StP/UML provides access to the Use Case Editor from:

- The StP Desktop
- Sequence Editor
- Collaboration Editor
- Activity Editor
- Stereotype Editor

For information about starting an editor from the StP Desktop, see [Chapter 2, “Using the StP Desktop.”](#)

For information about starting (navigating to) the Use Case Editor from another StP/UML editor, see the chapter in this manual that discusses the editor.

Navigating to Object References

“Navigation” is the use of the inherent relationships between repository objects and references in diagrams or tables to select an object and display another reference to it. This reference can be within the same diagram, a different use case diagram, or another StP/UML editor. When you navigate to another editor, the current session continues; in some cases, a predefined symbol appears in the editor.

You can navigate to these destinations from the Use Case Editor:

- Sequence Editor
- Collaboration Editor
- Activity Editor
- Requirements Table Editor

The **GoTo** Menu provides context-sensitive navigation choices for the selected symbol.

To navigate to a target:

1. Select an object.
2. From the **GoTo** menu, choose a command.

The navigation target appears.

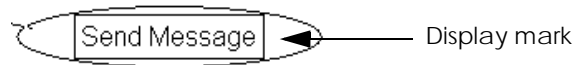
Table 1 provides a summary of navigations for the Use Case Editor.

Table 1: Use Case Editor Navigations

Navigate From	Navigate To	GoTo Menu Command
Actor	Scenario in sequence or collaboration diagram involving the actor	Use Case Scenario Involving Actor
	Another use case diagram involving the actor	Use Case Diagram Involving Actor
Use case	Scenario in sequence diagram involving the use case	Scenario in Sequence Diagram
	Scenario in collaboration diagram involving the use case	Scenario in Collaboration Diagram
	Scenario in activity diagram involving the use case	Scenario in Activity Diagram
Communicates link	Scenario in sequence or collaboration diagram involving the communicates link	Use Case Scenario Involving External Event
All symbols	Requirements table	Allocate Requirements

UseCaseScenariosExist Display Mark

When a use case has one or more sequence or collaboration scenarios associated with it, and the UseCaseScenariosExist display mark is set for display, the use case symbol appears with a box around its label, as shown in [Figure 2](#).

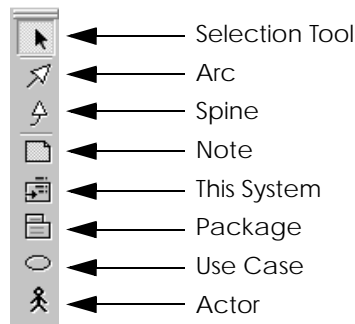
Figure 2: UseCaseScenariosExist Display Mark

For information about setting display marks, see [“Using Display Marks” on page 3-5](#).

For information on creating use case scenarios, see [“Decomposing a Use Case” on page 3-18](#).

Using the Use Case Symbol Toolbar

You select a symbol using the Symbol Toolbar.

Figure 3: Use Case Editor Symbol Toolbar

Procedures for using all the symbols on the Symbol Toolbar are described in this chapter.

Using Display Marks

Several use case annotations can cause display marks to appear in the diagram. For example, if you add a stereotype to a use case symbol, the stereotype may appear next to the symbol.

You control the behavior of display marks using the **Display Marks** tab in the **Options** dialog box.

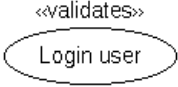
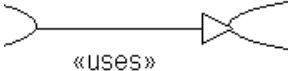

To display this tab:

1. From the **Tools** menu, select **Options**.
2. In the **Options** dialog box, choose the **Display Marks** tab.

For details about the Display Marks tab, see [Fundamentals of StP](#).

Table 2 lists all use case diagram display marks.

Table 2: UML Display Marks

Name	Display Mark	Description
Stereotype		A stereotype added to a symbol through the symbol's Properties dialog box. For details, see “Adding Extensibility Mechanisms” on page 3-12 .
UseCaseInfo		Built-in stereotypes attached to links. The stereotypes are «uses», «extends», and «inherits». For details, see “Drawing a Use Case Diagram” on page 3-7 .
UseCaseScenarios Exist		A scenario for the use case exists in either a sequence or collaboration diagram. For details, see “Decomposing a Use Case” on page 3-18 .

Creating a Use Case Diagram

A use case diagram is used to model the system as a whole. It shows high-level interactions between an actor and the system. Typically, these interactions involve a sequence of transactions that the system performs in response to an action by an actor.

You create use case analysis diagrams using the following symbols:

- **Actor**—Represents an outside entity, such as a human, machine, computer task, or another system, that interacts with the system.
An actor is a pre-defined UML stereotype.
- **Use case**—Represents a basic use of the system.
- **Package**—Represents a collection of use cases.
- **This System**—Represents the entire UML model and shows data flowing to and from external sources.

The symbols are connected to each other by the following relationships:

- **Communicates link**—Connects an actor to a use case.
- **Extends link**—Connects one use case to another in an inheritance-type relationship.
An extends link is a pre-defined UML stereotype.
- **Uses link**—Connects a base use case to another use case that represents a behavior subroutine of the base use case.
A uses link is a pre-defined UML stereotype.
- **Inherits link**—Connects two actors.

Drawing a Use Case Diagram

To create a use case diagram:

1. **Insert actor symbols into the diagram.**
To insert an actor symbol, select the symbol from the Symbol Toolbar and then click the mouse in the drawing area.
2. **Label the actors.**
By default, a text box appears when you insert a new symbol. If a text box is not visible, double-click the symbol to activate the text box.
3. **Insert use case symbols into the diagram.**
4. **Label the use cases.**
5. **Insert links between the symbols.**
To insert links, select the Arc symbol from the Symbol Toolbar, click the mouse on one symbol to initiate the link, drag the link to a second symbol, then click the mouse to terminate the link.

6. If desired, label the links.

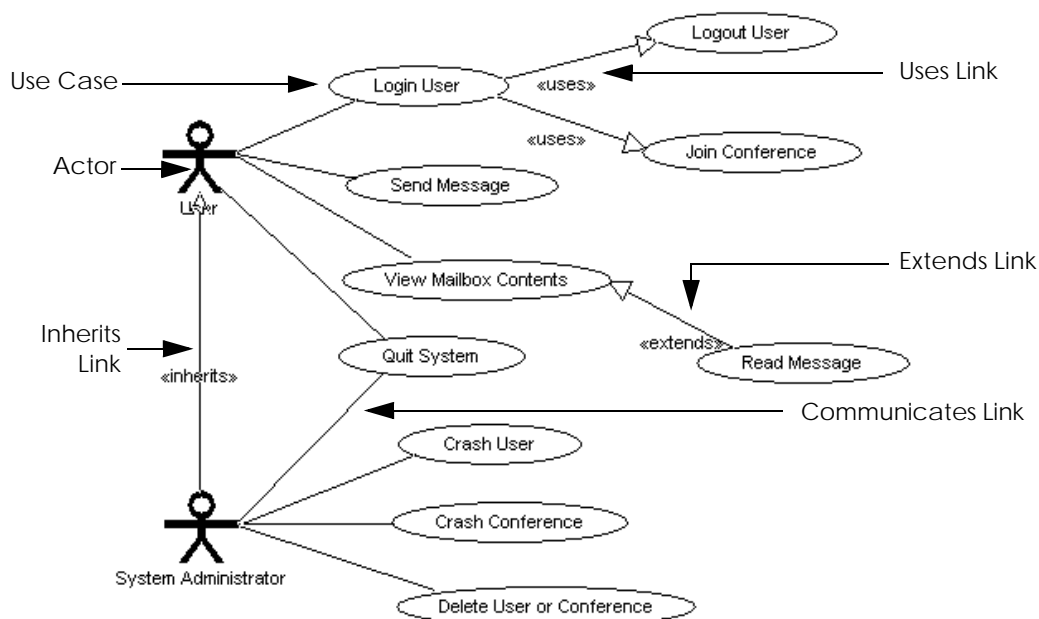
To label a link, double-click the link and enter the label in its text box. You can label uses and communicates links only.

For additional information on using links between use cases, see [“Using Extends and Uses Links” on page 3-9](#) and [“Changing Extends and Uses Links” on page 3-9](#).

7. Choose **File > Save**.

Figure 4 shows an example of a use case diagram for an electronic mail system.

Figure 4: Use Case Diagram



Using Extends and Uses Links

Use cases are connected to each other by either uses or extends relationships.

A uses relationship connects a base use case to another use case that represents a behavior subroutine of the base use case. An extends relationship provides an abstract use case to be used by other use cases.

Extends and uses relationships are displayed as links with attached stereotypes in the format «extends» and «uses». For information on other stereotypes, refer to [“Adding Extensibility Mechanisms” on page 3-12](#) and [Chapter 13, “Creating a Stereotype Diagram.”](#)

To insert a uses link, ensure that the default arc type is set to UseCaseUses. To insert an extends link, ensure that the default arc type is set to UseCaseExtends.

To set the default arc type:

1. Choose **Tools > Options**.
2. Select the **Default Arc** tab.
3. To change the default arc type, select the desired arc type from the options list.
4. Click **OK**.

Alternatively, you can change the arc type from extends to uses, and vice versa, by using the **Replace** command; for instructions, see the following section, “Changing Extends and Uses Links.”

Changing Extends and Uses Links

By using the **Replace** command, you can switch extends and uses links with each other. To make this change:

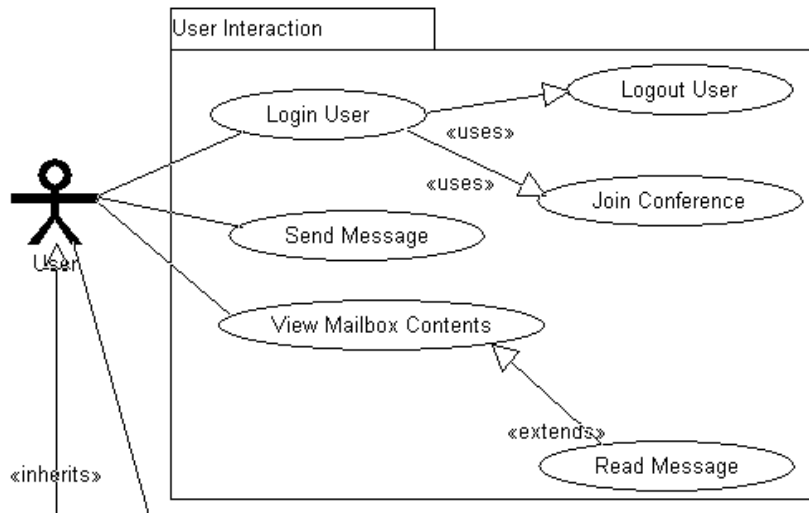
1. Select the link.
2. Choose **Edit > Replace**.
3. Choose **File > Save**.

Using Packages in a Use Case Diagram

A package represents an aggregate of use cases or packages (called “members” of the package) that are grouped together because they collaborate to provide a set of services.

Identifying packages is an activity that can take place at any time during object-based development. It is also a relatively subjective task. The use case diagram in [Figure 4](#) contains several use cases that could belong to various packages. One possibility for a use case package is shown in Figure 5.

Figure 5: Package Symbol



To insert a package:

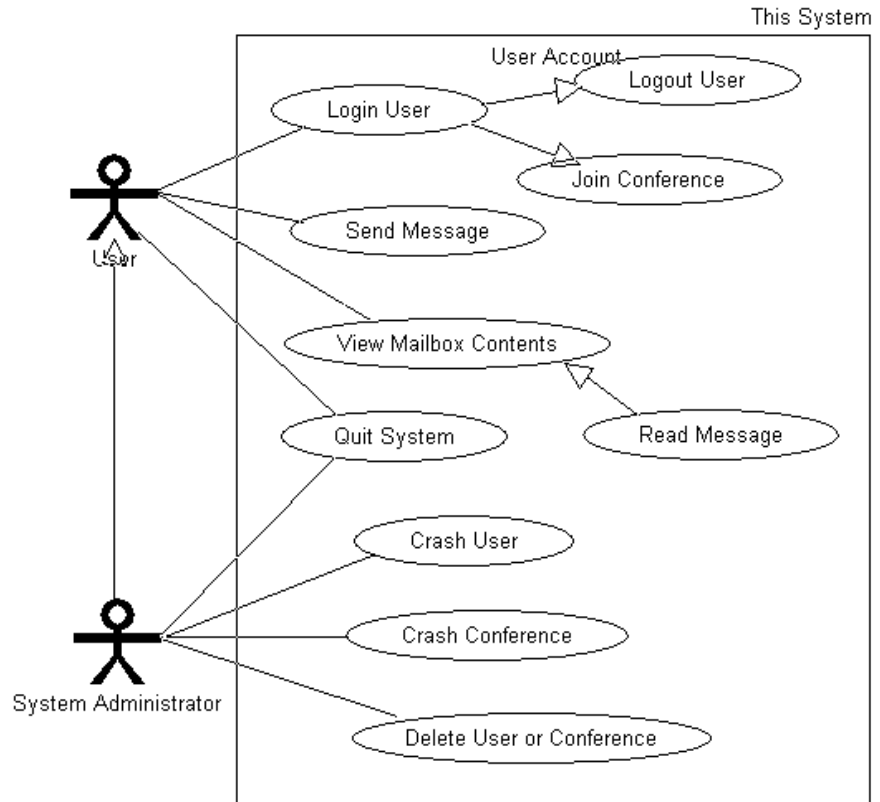
1. Insert a package symbol into the diagram.
2. Label the package.
3. Drag the boundaries of the package around the use case and package symbols that form the aggregate of the package.

If you move a package symbol, all use case and package symbols contained within the package move with it.

Using a This System Symbol

This System represents the entire UML model and shows data flowing to and from external sources. A system can contain use cases and packages. Figure 6 shows an example of a This System symbol.

Figure 6: This System Symbol



To insert a system symbol:

1. Insert a This System symbol into the diagram.
2. Label the system.
3. Drag the boundaries of the system symbol around the use case and package symbols that form the UML model.

Adding Extensibility Mechanisms

StP/UML supports three extensibility mechanisms for many diagram symbols. Extensibility mechanisms extend the information pertaining to a particular symbol.

The three extensibility mechanisms are:

- Stereotype
- Constraints
- Tagged Values

Extensibility mechanisms are defined using the Properties dialog box, described in [“Using the Properties Dialog Box” on page 3-13](#).

Stereotype

A stereotype extends the semantics of the symbol it is attached to. Each symbol can have one stereotype. UML supports pre-defined and user-defined stereotypes. (An example of a pre-defined stereotype is the «uses» link described in [“Drawing a Use Case Diagram” on page 3-7](#).)

StP checks the semantics of pre-defined stereotypes. It does not check user-defined stereotypes.

For more information on stereotypes, see [Chapter 13, “Creating a Stereotype Diagram.”](#)

Constraints

Constraints are conditions or restrictions that can apply to one or more symbols. There is no limit to the number of constraints associated with a symbol.

An example of a constraint for the *Join Conferences* use case in [Figure 6](#) is {requires password}.

Tagged Values

Tagged values describe explicit characteristics of a symbol. There is no limit to the number of tagged values associated with a symbol.

An example of tagged values for the Join Conferences use case in [Figure 6](#) are {author=larkin, status=analysis}.

Using the Properties Dialog Box

Most symbols in StP/UML have a Properties dialog box. The three extensibility mechanisms, along with other possible properties, are defined in the symbol's Properties dialog box.

To add an extensibility mechanism to a symbol:

1. Select a symbol.
2. Use one of the following methods to open the Properties dialog box:
 - From the **Edit** menu, choose **Properties**.
 - From the shortcut menu, choose **Properties**

The Properties dialog box for the symbol appears. [Figure 7](#) is an example of a properties dialog box for a use case symbol.

Figure 7: Properties Dialog Box

The image shows a 'Properties' dialog box with a title bar containing a close button (X). The dialog is divided into several sections. At the top, there is a 'Name:' label followed by a text field containing 'Login User'. Below this is an 'Extensibility' section, which is a container for other fields. Inside 'Extensibility', there is a 'Stereotype:' label followed by an empty text field. Below the 'Stereotype' field is the text 'One constraint value per line' followed by a large empty text area labeled 'Constraints:'. Below the 'Constraints' area is the text 'One tagged value per line' followed by another large empty text area labeled 'Tagged Values:'. At the bottom of the dialog, there are four buttons: 'Reset', 'OK', 'Apply', and 'Cancel'.

3. Add the extensibility mechanisms in the appropriate text fields.
For constraints and tagged values, add one constraint or tagged value per line. For a new line, press the Enter key.
4. Click **OK** or **Apply**.

The stereotype display mark appears near the symbol displaying the stereotype name within guillemets (for example, «validates»).

For more information on display marks, see [“Using Display Marks” on page 3-5](#). There are no display marks in the Use Case Editor for constraints and tagged values.

To edit an existing extensibility mechanism, open the Properties dialog box for the selected symbol and edit the existing information.

Annotating a Use Case Symbol

Using the Object Annotation Editor (OAE), you can place the following annotations on a use case:

- A description of the use case
- Preconditions
- Postconditions
- Exceptions
- Requirements
- Requirement allocation
- A note description
- Extensibility mechanisms

To annotate a use case symbol:

1. Select the use case symbol
2. Use one of the following methods to open the OAE:
 - Choose **Edit > Object Annotation**
 - Choose **Object Annotation** from the shortcut menu
 - Use the accelerator Ctrl+a

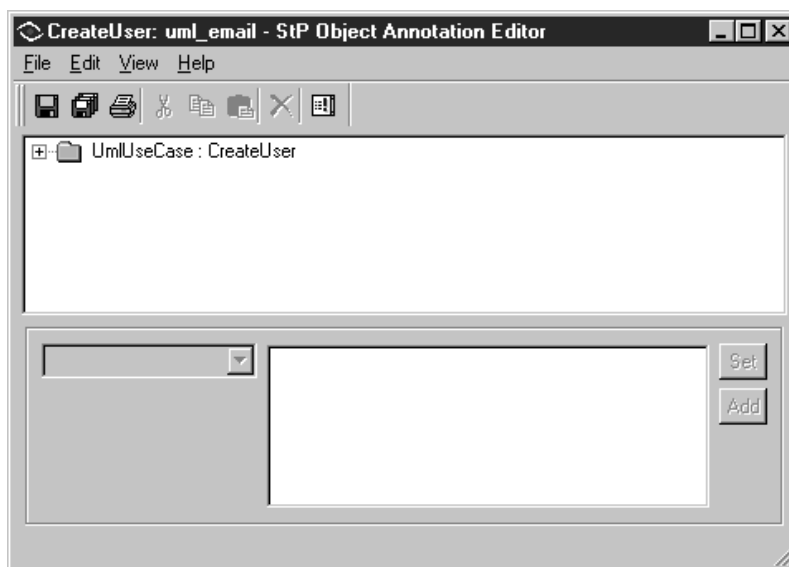
For complete instructions on using the OAE, see [Fundamentals of StP](#).

3. Add annotations as described in subsequent sections.
4. Choose **File > Save** to save the annotation.
5. Choose **File > Exit** to exit the OAE.

Adding a Use Case Description

When you start the OAE for a use case, the use case appears as a folder, as shown in [Figure 8](#).

Figure 8: OAE with Use Case Definition Note Highlighted



To add a description of a use case:

1. Open the UmlUseCase folder for the designated use case.
To open the folder, single-click the plus (+) sign or double-click the folder icon.
2. Select the Use Case Definition note.
3. Choose **Edit > Note Description**.
4. In the OAE Note Description dialog box, type the description.
5. Click **OK** or **Apply**.

Adding a Condition or Exception

You can annotate a use case with the following conditions and exceptions:

- Precondition—A condition that must be satisfied before the use case can be executed
- Postcondition—A condition that must be satisfied after the use case is executed
- Exception—A limitation on the use case's typical behavior

To add a condition or exception to a use case:

1. Open the Use Case Definition note.
2. Select the appropriate item.
3. Type the precondition, postcondition, or exception in the text box.
4. Click **Set**.

Allocating a Requirement

You can use the OAE to add a requirement associated with the selected use case. After you add the requirement, it appears in a requirements table.

To add a requirement associated with a use case:

1. Choose **Edit > Add Note > Requirement Allocation**.
2. Select the Requirement Allocation note.
3. Choose **Edit > Add Item**.
4. Select an item from the Add Item submenu.

The choices on this menu correspond to phases of a project used by the Requirements Table Editor (RTE). For complete information about the RTE, see [Fundamentals of StP](#).

5. For each item, type a value and click **Set**.

Adding References to Requirements Documentation

The Requirement note enables you to add references to requirements documentation not related to the StP Requirements Table Editor.

To add references to requirements documentation:

1. Choose **Edit > Add Note > Requirement**.

The Requirement note appears in the window, and the Requirement Name, Requirement Document, and Requirement Paragraph items appear in the Items window.

For descriptions of each of these items, choose **Help > On Selection**.

2. Select an item.
3. Type a value for the selected item in the text box.
4. Click **Set**.

5. Repeat the procedure for each item you want to add to the Requirement note.

Adding Extensibility Mechanisms

In addition to using the Properties dialog box, you can define UML extensibility mechanisms using the OAE.

To define an extensibility mechanism using the OAE:

1. With the Extensibility Definition note highlighted, choose **Edit > Add Item** and add the stereotype, constraint, or tagged value item.
2. Type the value in the text box.
If you are adding multiple constraints or tagged values, separate each one with a comma.
3. Click **Set**.
4. Repeat the procedure for each item you want to add to the Extensibility Definition note.

For more information on extensibility mechanisms, see [“Adding Extensibility Mechanisms” on page 3-12](#).

Decomposing a Use Case

You can decompose a use case into one or more scenarios that represent a major behavior associated with the use case. For example, a use case called *SendMessage* in an Email system could be decomposed into two scenarios: *ValidUser* and *InvalidUser*.

Each scenario is modeled in either a sequence, collaboration, or activity diagram. Sequence and collaboration diagrams are similar to each other and are collectively referred to as interaction diagrams.

Decomposing a Use Case into an Interaction Diagram

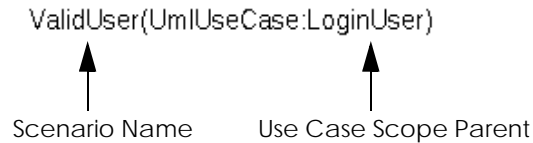
When you decompose a use case into a sequence or collaboration diagram, the scope parent appears on the new diagram to indicate the source of the decomposition.

To decompose a use case into a sequence or collaboration diagram:

1. Select a use case symbol.
2. Choose **GoTo > Scenario in Sequence Diagram** or **GoTo > Scenario in Collaboration Diagram**.
3. In the Create Scenario dialog box, type the scenario name in the Scenario Name field.
4. Click **OK**.

A new sequence or collaboration diagram appears with the use case scope parent flashing.

Figure 9: Use Case Scope Parent



The new diagram is automatically named `<use case>__<number>`, where `<use case>` is the name of the use case and `<number>` is 1 for the first scenario you create, 2 for the second, and so on.

5. Develop the new sequence or collaboration diagram, as described in either [Chapter 6, “Creating a Sequence Diagram”](#) or [Chapter 7, “Creating a Collaboration Diagram.”](#)

Decomposing a Use Case into an Activity Diagram

To decompose a use case into an activity diagram:

1. Select a use case symbol.
2. Choose **GoTo > Scenario in Activity Diagram**.

The activity diagram appears. The name of the use case is displayed as the diagram’s state machine.

The new diagram is automatically named `<use case>__1`, where `<use case>` is the name of the use case. You can create one activity scenario for each use case.

3. Develop the new activity diagram, as described in [Chapter 10, “Creating an Activity Diagram.”](#)

Validating a Use Case Diagram

StP/UML provides two options for checking the correctness and consistency of a use case analysis diagram:

- Checking the current diagram
- Checking the repository for the current system

To check a diagram, choose **Tools > Check Syntax**.

To check the repository, choose **Tools > Check Semantics**. When you check the diagram's semantics, StP checks for the following:

- Does each use case have at least one use case scenario?
- Does each communicates link have a corresponding external event in a use case scenario?
- Does each actor appear in a use case scenario with the use case it communicates with?
- Does each extension use case have an extension point defined in a use case scenario?

When you check your model, the Message Log appears listing the errors. For information about using the Message Log, see [Fundamentals of StP](#).

4

Creating a Class Diagram

This chapter describes how to create and use class diagrams. Topics covered are as follows:

- [“What Are Class Diagrams?” on page 4-2](#)
- [“Using the Class Editor” on page 4-2](#)
- [“Classes” on page 4-12](#)
- [“Adding Details about Classes” on page 4-22](#)
- [“Representing Objects \(Class Instances\)” on page 4-29](#)
- [“Creating Relationships” on page 4-31](#)
- [“Adding Details about Associations” on page 4-39](#)
- [“Using Specialized Relationships” on page 4-49](#)
- [“Creating Class Packages” on page 4-55](#)
- [“Designating View Points” on page 4-57](#)
- [“Validating a Class Diagram” on page 4-60](#)
- [“Updating a Class Diagram” on page 4-61](#)

This chapter provides brief descriptions of the elements of object-oriented modeling. For a complete description of object-oriented modeling and the UML, see *Unified Modeling Language User Guide* (Booch et al 1998) and *Unified Modeling Language Reference Manual* (Rumbaugh et al 1998).

For a tutorial that guides you through each step in the process of creating a UML model, see [Getting Started with StP/UML](#).

What Are Class Diagrams?

One key component of a complete UML model is the class diagram, supported by a class table (described in [Chapter 5, “Defining Classes with the Class Table Editor”](#)). A class diagram is a representation of the static structure of a system. You create a class diagram using the symbols and display marks provided by UML notation. The notation also provides semantics and guidelines for each symbol and diagram.

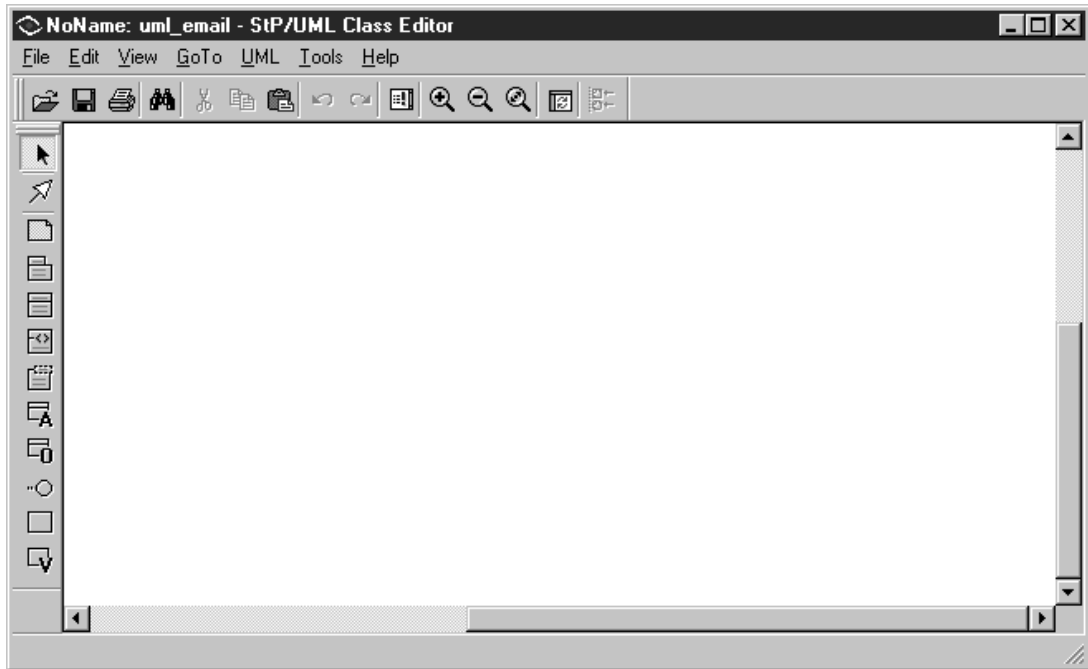
Using examples primarily from the electronic mail system, this chapter describes:

- Using the Class Editor
- Representing classes, packages, attributes, and operations
- Representing object instances of classes
- Representing associations and specialized relationships

Using the Class Editor

You create a class diagram using the Class Editor.

Figure 1: StP/UML Class Editor



The Class Editor provides all the functions and menu options of every StP diagram editor. For general information about using StP diagram editors, see [Fundamentals of StP](#).

Starting the Class Editor

StP/UML provides access to the Class Editor from:

- The StP Desktop
- Class Table Editor
- Sequence Editor
- Collaboration Editor
- State Editor
- State Table Editor
- Activity Editor
- Stereotype Editor

For information about starting an editor from the StP Desktop, see [Chapter 2, “Using the StP Desktop.”](#)

For information about starting (navigating to) the Class Editor from another StP/UML editor, see the chapter in this manual that discusses the editor.

Navigating to Object References

“Navigation” is the use of the inherent relationships between repository objects and references in diagrams or tables to select an object and display another reference to it. This reference can be within the same diagram, a different class diagram, or another StP/UML editor. When you navigate to another editor, the current session continues; in some cases, a predefined symbol appears in the editor.

You can navigate to these destinations from the Class Editor:

- State Editor
- Class Table Editor
- Activity Editor
- Stereotype Editor
- StP Requirements Table Editor
- Source code files, displayed in your source code viewer or standard editor window (for information about navigating to source code files, see [Generating and Reengineering Code](#))

The **GoTo** Menu provides context-sensitive navigation choices for the selected symbol.

To navigate to a target:

1. Select an object.
2. From the **GoTo** menu, choose a command.

The navigation target appears.

Table 1 provides a summary of navigations for the Class Editor.

Table 1: Class Editor Navigations

Navigate From	Navigate To	GoTo Menu Command
Class, parameterized class, interface	Class table for selected class	Class Table
	If the symbol includes a stereotype, the stereotype's definition in a stereotype diagram	Stereotype Definition
	Class diagram with the All Aggregation ViewPoint set for the class	All Aggregations View
	Class diagram with the All Generalization ViewPoint set for the class	All Generalizations View
	State diagram with the class represented as the state machine	State Diagram for Class
	Activity diagram with the class represented as the state machine	Activity Diagram for Class
Class, parameterized class, interface, package	Class diagram with the All Association ViewPoint set for the class	All Associations View
	Class diagram with the All Dependency ViewPoint set for the class	All Dependencies View
	Class diagram with the All Members ViewPoint set for the class	All Members View

Table 1: Class Editor Navigations (Continued)

Navigate From	Navigate To	GoTo Menu Command
Class, parameterized class, interface, package, attribute	Viewer or editor with the source code loaded for the selected object	Source Code
Operation	State diagram for the selected operation	State Diagram for Operation
	Activity diagram for the selected operation	Activity Diagram for Operation
	Viewer or editor with the source code declaration loaded for the selected operation.	Source Code Declaration
	Viewer or editor with the source code definition loaded for the selected operation. Appears on the menu only if a semantic model, created with the Reverse Engineering tool, is present for the current system. For information about the Reverse Engineering tool, see Generating and Reengineering Code .	Source Code Definition
Any symbol	Requirements table with the object id for the object loaded	Allocate Requirements

ClassInfo Display Mark

If a class table exists for any type of class symbol and the ClassInfo display mark is set to display, a display mark appears on the class symbol in the diagram, as shown in Figure 2.

Figure 2: ClassInfo Display Mark for Tables

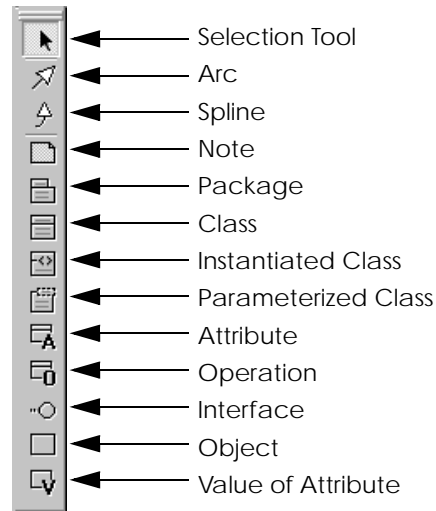


For information about setting display marks, see [“Using Display Marks” on page 4-9](#).

Using the Class Editor Symbol Toolbar

You select a symbol using the Symbol Toolbar.

Figure 3: Class Editor Symbol Toolbar



To detach the toolbar from the editor and make it a separate window, refer to [Fundamentals of StP](#).

Procedures for using all symbols on the Symbol Toolbar are described in this chapter.

Using the UML Menu

In addition to the standard diagram menu options, the Class Editor also provides the UML Menu. This menu lists commands specific to class diagrams.

[Table 2](#) describes the commands available from the UML menu.

Table 2: UML Menu Commands

Command	Description	For Details, See
Create Association Roles	Adds explicit role names to an association.	“Designating Roles” on page 4-38
Create Association Class	Creates an association class link from an association to a class.	“Creating Association Classes” on page 4-36
Create N-ary Association	Creates an n-ary association between two or more classes.	“Creating N-ary Associations” on page 4-34
Create Or Association	Creates an “Or” link between two association links.	“Creating Or Associations” on page 4-37
Generalization Hierarchy	Provides submenu options to create a generalization hierarchy from an existing semantic model created from Reverse Engineering.	Generating and Reengineering Code
Attributes and Operations	Provides submenu options to build a class’s attributes and operations from information in a class table, the repository, or from reverse engineering source code files.	“Updating a Class Diagram” on page 4-61
Packages and Classes	Provides submenu options for rebuilding a class package symbol from either member or non-member packages and classes.	“Updating a Class Diagram” on page 4-61

In addition, the **Edit > Properties** command provides a dialog box for adding details to class diagram objects. For more information, see:

- [“Adding Details about Classes” on page 4-22](#)
- [“Adding Details about Associations” on page 4-39](#)
- [“Adding Details to Specialized Relationships” on page 4-53](#)

Using Display Marks

Several class diagram annotations can cause display marks to appear in the diagram. For example, if you annotate a relationship for multiplicity, the value may appear on the relationship arc.

You control the behavior of display marks using the **Display Marks** tab in the **Options** dialog box. To display this tab:

1. From the **Tools** menu, select **Options**.
2. In the **Options** dialog box, choose the **Display Marks** tab.

For details about the Display Mark Options dialog, see [Fundamentals of StP](#).

Table 3 lists all class diagram display marks. The names include any associated item names within parentheses.

Table 3: UML Display Marks

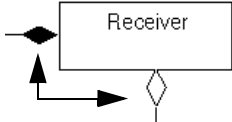
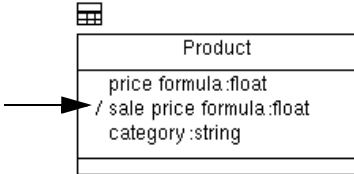
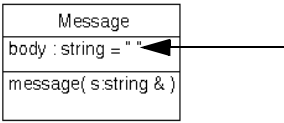
Name	Display Mark	Description
Aggregation Indicators (UmlAggregationType)		An aggregation (hollow diamond) or composite aggregation (solid diamond). For details, see “Setting Aggregation” on page 4-44 .
AttributeIsDerived (UmlAttributeIsDerived)		An attribute that can be determined from the values of other attributes (the base attributes). For details, see “Showing Derived Attributes” on page 4-16 .
AttributeTypeAndValue (UmlAttributeType, UmlAttributeDefaultValue)		The attribute type and default value as specified in the Class Table Editor. For details, see Chapter 5, “Defining Classes with the Class Table Editor.”

Table 3: UML Display Marks (Continued)

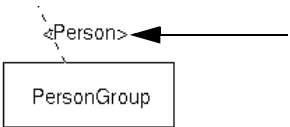
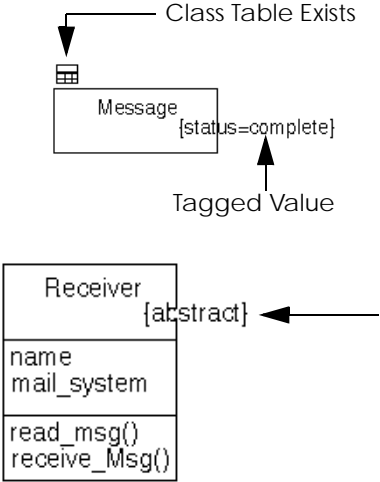
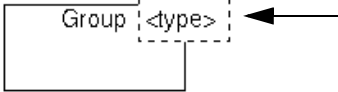
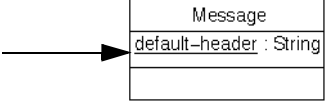
Name	Display Mark	Description
BindsArguments (UmlRefineArgs)		Actual parameters assigned to an instantiated class (connected by a binds link). For details, see “Using Binds Relationships” on page 4-51 .
ClassInfo (UmlTaggedValue, UmlClassIsAbstract)		The Class Info display mark can show the following information for a class: its tagged values, if a class table exists, and if the class is abstract. For details, see “ClassInfo Display Mark” on page 4-6 , “Adding Tagged Values” on page 4-29 , and “Designating Abstract Classes” on page 4-26 .
ClassParameters (UmlClassParameters)		List of parameters for a parameterized class. For details, see “Representing Parameterized and Instantiated Classes” on page 4-18 .
ClassScoping (UmlMemberIsClass)		The attribute or operation is scoped to the class. For details, see “Inserting and Labeling Attributes” on page 4-14 or “Inserting and Labeling Operations” on page 4-16 .

Table 3: UML Display Marks (Continued)

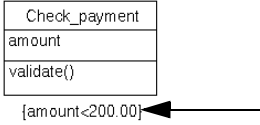
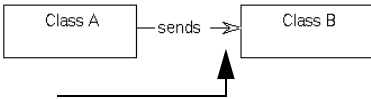

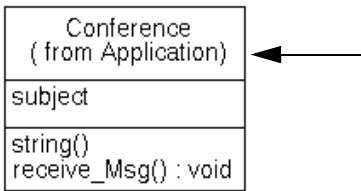
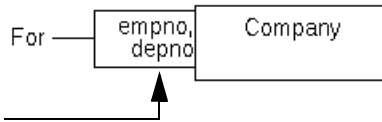
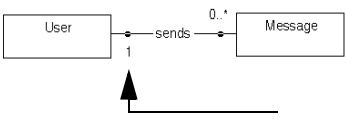
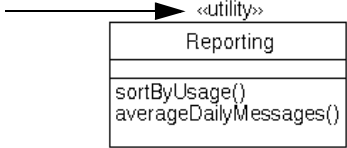
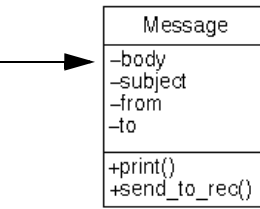
Name	Display Mark	Description
Constraints (UmlConstraint Item)	 <p>A UML class diagram for a class named 'Check_payment'. It has two attributes: 'amount' and 'validate()'. Below the class box, there is a constraint box containing the expression '{amount < 200.00}'. An arrow points from the constraint box to the class box.</p>	<p>A restriction on the value a class can have.</p> <p>For details, see “Specifying Constraints on Classes” on page 4-28.</p>
Navigability (UmlRole Navigability)	 <p>A UML class diagram showing an association between 'Class A' and 'Class B'. The association is labeled 'sends'. An arrow points from 'Class A' to 'Class B', indicating the direction of the association.</p>	<p>Indicates that the association has a direction.</p> <p>For details, see “Adding Directionality to Roles” on page 4-46.</p>
OperationReturn Type (UmlOperation ReturnType)	 <p>A UML class diagram for a class named 'Message'. It has two attributes: 'body : string' and 'message(s:string &)'. An arrow points from the return type 's:string' in the message signature to the class box.</p>	<p>Indicates the return type of an operation as indicated in the Class Table Editor.</p> <p>For details, see “Parts of the Class Table Editor” on page 5-6.</p>
PackageScope (UmlPackageScope)	 <p>A UML class diagram for a class named 'Conference'. It has two attributes: 'subject' and 'string()'. Below the class box, there is a package scope box containing the text '(from Application)'. An arrow points from the package scope box to the class box.</p>	<p>Designates a class or package as a member of the parent package.</p> <p>For details, see “Package Scope Display Mark” on page 4-57.</p>
Qualifiers (UmlRole Qualifier)	 <p>A UML class diagram showing an association between 'Company' and 'empno, depno'. The association is labeled 'For'. An arrow points from 'Company' to 'empno, depno', indicating the direction of the association.</p>	<p>An attribute that modifies one end of an association and reduces the multiplicity of the association.</p> <p>For details, see “Adding Qualifiers” on page 4-43.</p>
RoleMultiplicity (UmlMultiplicity)	 <p>A UML class diagram showing an association between 'User' and 'Message'. The association is labeled 'sends'. There is a multiplicity of '1' on the 'User' end and '0..*' on the 'Message' end. An arrow points from the multiplicity '1' to the 'User' class box.</p>	<p>The number of instances of a class that may participate in an association.</p> <p>For details, see “Setting Association Multiplicity” on page 4-42.</p>

Table 3: UML Display Marks (Continued)

Name	Display Mark	Description
Stereotype (UmlStereotype)		The stereotype of the model element. For details, see “Designating a Stereotype” on page 4-28 .
Visibility (UmlMember Visibility)		The attribute or operation member visibility, which is one of the following: + public # protected - private For details, see “Inserting and Labeling Attributes” on page 4-14 and “Inserting and Labeling Operations” on page 4-16 .

Classes

The essential pieces of a class diagram are various types of classes and their relationships. A class is a group of things, abstractions, or concepts that share similar or identical characteristics (attributes), behavior (operations), and relationships to other classes. An example of a class from the mail system model is *Message*, a group of things that have similar attributes (such as *header* and *body*), and operations (such as *print* and *sendTo*).

An element that shows a grouping of classes is called a “package,” which is further described in [“Creating Class Packages” on page 4-55](#).

This section describes how to draw classes, attributes, and operations. To represent parameterized classes and instantiated classes, see [“Representing Parameterized and Instantiated Classes” on page 4-18](#).

Class Symbols

In a class diagram, a class symbol comprises three parts:

- Class name part with the class label
- Class attribute part, containing individual attributes
- Class operation part, containing individual operations

Class Names

Each part of a class symbol is labeled with a name. When labeling classes, follow these rules:

- Use alphanumeric characters only.
- Do not use colons, commas, backward slashes, or forward slashes.
- Make the label between one and 220 characters long.

The maximum length of a label is a Sybase limitation of 220 characters. However, you should keep labels short, because characters are frequently added to names (for example, a class name becomes a class table filename, and decomposing various objects creates filenames built on object names).

- Ensure that labels adhere to the naming rules of your programming language.

Since StP/UML uses labels to generate identifiers in some programming languages, class labels should adhere to the naming rules of the language. Avoid using single quotes, double quotes, or angle brackets.

For instructions on adding and labeling attributes, see [“Attribute Labels” on page 4-14](#) and [“Operation Labels” on page 4-16](#). You add details about attributes and operations with the Class Table Editor and State Table Editor, described in their chapters in this manual.

Inserting and Labeling Classes

To insert a new class into a diagram:

1. Insert a class symbol from the Symbol Toolbar into the drawing area.

2. Label the class.



Check that you have typed the name correctly, and verify that it is the name you want to use. When you specify all the parts of the class (such as attributes, operations, and annotations), many references to this class are created in the repository, which complicate the process of renaming it later. For further discussion, see [Chapter 14, “Renaming Symbols.”](#)

Inserting and Labeling Attributes

An attribute is a characteristic of a class with a set of data values held by each instance of the class. For example, the *Message* class has an attribute, *header*. An attribute may have default values. When an instance of a class is created, the attribute’s value in the object instance is the default value, unless otherwise specified. An attribute is scoped to the class where it appears and cannot be overloaded.

Attribute Labels

In a class diagram, you label an attribute with its name only. For labeling rules, see [“Class Names” on page 4-13.](#)

The label may have multiple lines. To start a new line, press Shift+Enter.

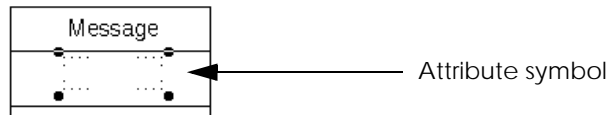
To provide a complete description of an attribute, including its data type and initial value, use the Class Table Editor, described in [Chapter 5, “Defining Classes with the Class Table Editor.”](#)

If a class table already exists for a class, you can have its attributes inserted from its table. For information, see [“Updating a Class Diagram” on page 4-61.](#)

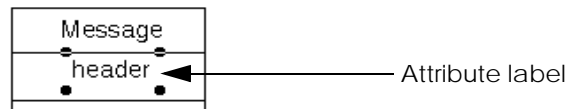
Adding Attributes

To add an attribute to a class:

1. Insert an attribute symbol into the class symbol in the drawing area.
The class symbol automatically resizes to fit the attribute. The attribute symbol is selected.



2. Label the attribute symbol.

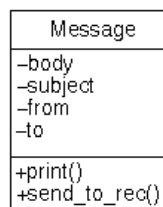


To add more than one attribute, repeat the procedure.

Showing Member Visibility

Visibility indicates the access to the attribute by other classes. The types and display marks for member visibility are public (+), protected (#), and private (-). The display mark for member visibility is the attribute name preceded by the visibility marker.

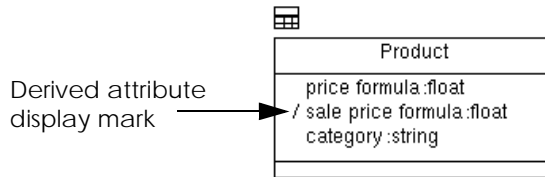
Figure 4: Visibility



To define member visibility, add information to the Visibility column in the Attribute section of the Class Table Editor (described in [Chapter 5. “Defining Classes with the Class Table Editor”](#)).

Showing Derived Attributes

A derived attribute is one that can be determined from the values of other attributes (the base attributes). The display mark for a derived attribute is the attribute name preceded by a slash.



To define a derived attribute, add information to the Derived? column in the Attribute section of the Class Table Editor (described in [Chapter 5, “Defining Classes with the Class Table Editor”](#)).

Inserting and Labeling Operations

An operation (sometimes called a method) is a function or transformation that can be applied to or by objects in a class. All objects in a class have the same operations. Examples of operations for the *User* class include *send_mail*, *read_mail*, and *delete_mail*.

The implicit target of an operation is the object that invoked it.

Operations can be overloaded. That is, the same operation may apply to different classes, but behave differently depending on the class of its target object. The signature of an operation is its name followed by a parenthesized comma-separated list of types of the arguments.

An operation is scoped to the class where it appears.

Operation Labels

In a class diagram, you label an operation with its name and signature. For labelling rules, see [“Class Names” on page 4-13](#).

An operation label may have multiple lines. To start a new line, press Shift+Enter.

To provide a complete description of an operation, including its arguments and return types, use the Class Table Editor, described in [Chapter 5, “Defining Classes with the Class Table Editor.”](#)

If a class table already exists for a class, you can have its operations inserted from its table. For information, see [“Updating a Class Diagram” on page 4-61.](#)

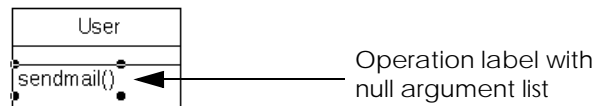
Adding Operations

To add an operation to a class:

1. Insert an operation symbol into the class symbol in the drawing area.
The class symbol automatically resizes to fit the operation. The operation symbol remains selected.



2. Label the operation.
StP automatically appends a null argument list to the end of the operation label.



To add more than one operation, repeat the procedure.

Showing Member Visibility

Visibility indicates the access to the operation by other classes. The types and display marks for member visibility are public (+), protected (#), and

private (-). The display mark for member visibility is the operation name preceded by a visibility marker (see [Figure 4 on page 4-15](#)).

To define member visibility, add information to the Visibility column in the Operation section of the Class Table Editor (described in [Chapter 5, “Defining Classes with the Class Table Editor”](#)).

Designating Abstract Operations

An abstract operation is one that defines the form of an operation; each concrete subclass must provide its own implementation.

To define an abstract operation, add information to the Abstract? column in the Operation section of the Class Table Editor (described in [Chapter 5, “Defining Classes with the Class Table Editor”](#)).

Abstract operations in a superclass can be automatically included as concrete operations in the class table of an inheriting subclass. See [Chapter 5, “Defining Classes with the Class Table Editor”](#) for instructions.

There is no display mark for abstract operations.

Representing Parameterized and Instantiated Classes

A parameterized class provides a template of the structure and behavior of a class independently of the types of objects that it manipulates. Its parameters are indicated by formal parameters, located in a dashed box in the upper-right side of the symbol, which are placeholders in the class definition.

An example of a parameterized class might be a class *Group*, which has a parameter *Member*. The *Group* class may support operations such as *AddMember*, *RemoveMember*, *CountMember*, *DisplayMember*. A parameterized class does not support instances.

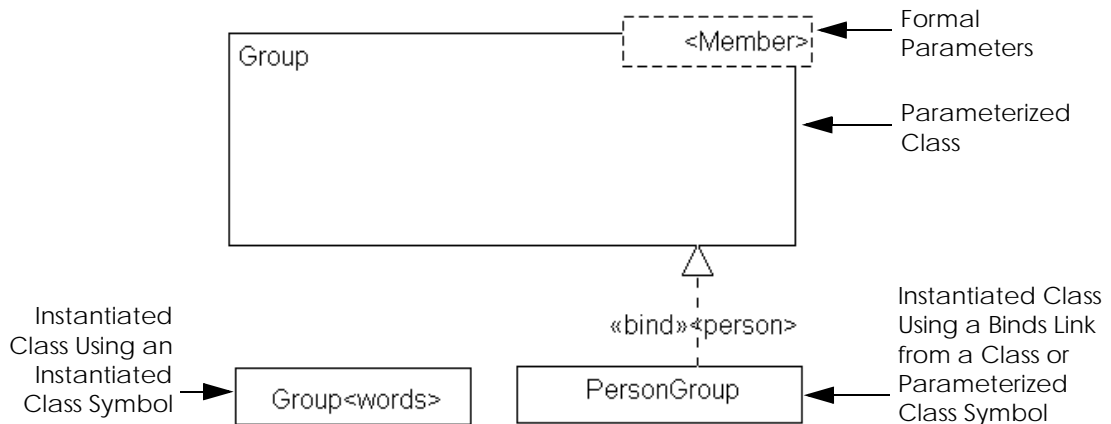
When a parameterized class is instantiated, an instantiated class is created in which the formal parameters are replaced by actual parameters. For example, from the *Group* parameterized class, you could create:

- An instantiated class symbol with the same name, *Group*, in which the placeholder *Member* is replaced by the type “words” in the class label
- A class or parameterized class symbol with a different name, *PersonGroup*, in which *Member* is replaced by the argument “Person” on a binds link

An instantiated class supports instances. Instantiation of a parameterized class allows for reuse of the operations defined in the parameterized class definition.

Figure 5 displays the two types of instantiated classes. In this example, the formal parameter, *<Member>*, is replaced with actual parameters, *<words>* and *<person>*.

Figure 5: Parameterized and Instantiated Classes



Inserting and Labeling a Parameterized Class

To add a parameterized class to a diagram:

1. Insert a parameterized class symbol into the drawing area.
2. Label the parameterized class.

To add formal parameters to the parameterized class, see [“Adding Parameters to a Parameterized Class” on page 4-27](#).

Inserting and Labeling an Instantiated Class

You create instantiated classes in one of two ways:

- Using an instantiated class symbol
- Using a class or parameterized class symbol with a binds link

To label an instantiated class using an instantiated class symbol:

1. Insert an instantiated class symbol into the drawing area.
2. Label the instantiated class using the following syntax:

ParameterizedClassName<ActualArguments>

To label an instantiated class using a class or parameterized class symbol:

1. Insert a class or parameterized class symbol into the drawing area.
2. Label the symbol with a name that is different than the parameterized class being instantiated.
3. Add a binds link that has the stereotype «bind» and includes the actual parameters of the parameterized class being instantiated.

For information on creating a binds link and adding actual parameters, refer to [“Using Binds Relationships” on page 4-51](#).

Setting Role Navigability for Associations

Associations are supported between parameterized and instantiated classes. If you set role navigability, the direction must be towards the instantiated class.

For more information, refer to [“Adding Directionality to Roles” on page 4-46](#).

Representing Interfaces

An interface specifies the externally visible behavior of a class. It declares a set of operations for use by other classes, but it does not specify the implementation of the operations. The class that supports the interface provides the methods that implement the operations declared in the interface.

You model an interface by assigning the interface stereotype to a class symbol that contains abstract operations. After creating the interface using a class symbol, you use a special interface symbol to designate the interface.

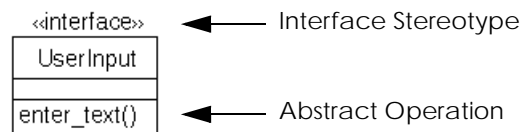
Creating an Interface Using a Class Symbol

To create an interface using a class symbol:

1. Insert a class symbol into the drawing area.
2. Label the class symbol, as described in [“Inserting and Labeling Classes” on page 4-13](#).
3. Add attributes, as described in [“Inserting and Labeling Attributes” on page 4-14](#).
4. Add abstract operations, as described in [“Inserting and Labeling Operations” on page 4-16](#) and [“Designating Abstract Operations” on page 4-18](#).
5. Add the «interface» stereotype to the class, as described in [“Designating a Stereotype” on page 4-28](#).

An example of a class with the «interface» stereotype is shown in Figure 6.

Figure 6: Interface



Using the Interface Symbol

After creating the interface using a class symbol, you can use an interface symbol to designate the interface.

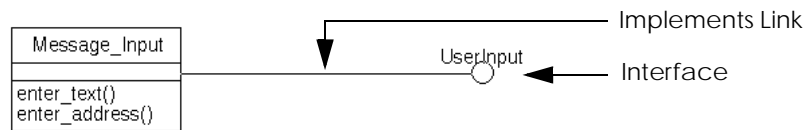
To represent an interface using the interface symbol:

1. Insert an interface symbol into the drawing area.
2. Label the interface with the same name as the class symbol created in [“Creating an Interface Using a Class Symbol”](#) above.

3. Draw a connection from the class that implements the interface to the interface symbol.

The connection is represented by an implements link, shown as a solid line. If necessary, select the connection and choose **Edit > Replace** to change the connection to an implements link.

Figure 7: Interface Symbol



For information on using a dependency relationship with an interface, see [“Using Dependency Relationships” on page 4-52](#).

Adding Details about Classes

StP/UML provides features for adding details about classes, such as:

- A class is external
- A class is abstract
- Class visibility
- Class parameters

In addition, classes support the three UML extensibility mechanisms: stereotypes, constraints, and tagged values.

Several details about classes have display marks associated with them. For information about controlling the behavior of display marks, see [“Using Display Marks” on page 4-9](#).

Using the Class Properties Dialog Box

The Properties dialog box enables you to change the characteristics of a selected class and apply all the changes at once. [Figure 8](#) displays the **Class Properties** dialog box. The **Parameterized Class Properties** dialog box also includes a Parameters field.

Figure 8: Class Properties Dialog Box

To use the Properties dialog box for a class:

1. Select a class.
2. Use one of the following methods to open the Properties dialog box:

- From the **Edit** menu, choose **Properties**.
- From the shortcut menu, choose **Properties**

The Properties dialog box for the symbol appears.

3. Change all of the desired settings for the selected class.

For details about each property, see [Table 4](#) and subsequent sections.

4. Click **OK** or **Apply**.

The **OK** button applies the changes and dismisses the dialog. The **Apply** button applies the changes without dismissing the dialog.

5. To show display marks on your diagram, choose **View > Refresh Display Marks**.

Summary of Class Properties

Table 4 is a summary of the various parts of the Properties dialog box for classes.

Table 4: Class Properties Summary

Property	Description	Value	For Details, See
Class Name or Param Class Name	The name of the selected class.	Retrieved from the information in the diagram.	“Inserting and Labeling Classes” on page 4-13
Visibility	Access to the interface of a class.	Public, Private, Protected	“Specifying Class Visibility” on page 4-25
Multiplicity	How many instances of a class may exist.	User input	“Setting Class Multiplicity” on page 4-25
Abstract	The class cannot be instantiated.	On/Off	“Designating Abstract Classes” on page 4-26
External	The class has an External annotation, assumed to be defined in another system.	On/Off	“Designating External Classes” on page 4-27
Parameters (parameterized class only)	List of parameters for a parameterized class.	User input	“Adding Parameters to a Parameterized Class” on page 4-27
Stereotype	Extends the semantics of the class.	User input based on pre-defined and user-defined stereotypes	“Designating a Stereotype” on page 4-28
Constraints	The constraints on the class, restricting the values that its attributes can have.	User input—one constraint per line	“Specifying Constraints on Classes” on page 4-28
Tagged Values	Explicit characteristics of the class.	User input—one tagged value per line	“Adding Tagged Values” on page 4-29

Specifying Class Visibility

Class visibility indicates the access to the interface of a class. The types of visibility are:

- Public—the interface is accessible to other classes
- Protected—the interface is accessible to subclasses but not client classes
- Private—the interface is accessible only by its own class

To designate class visibility:

1. Use the Properties dialog box, as described in [“Using the Class Properties Dialog Box” on page 4-22](#).
2. Select the visibility type from the Visibility pull-down menu.

There is no display mark for class visibility.

Setting Class Multiplicity

In some cases, there is a limit on the number of instances of a class that can be created from a class; this is the multiplicity of a class. You set multiplicity using the format:

`[lower bound..upper bound]`

in which lower bound and upper bound are integer values, specifying the range of the multiplicity. The star character (*) can be used as an upper bound value, indicating an unlimited upper bound.

Table 5 illustrates some examples for multiplicity.

Table 5: Class Multiplicity Examples

Description	Example
The number of instances is not specified	
A single instance of a class can be created from the class	1
Zero or more instances of a class can be created from the class	*

Table 5: Class Multiplicity Examples (Continued)

Description	Example
A single instance of a class may or may not be created from the class	0..1
A single instance or more than one instance of a class may be created from the class	1..*
A specific number of instances of a class may be created from the class	1..3,6,9..12

To indicate multiplicity on a class:

1. Use the Properties dialog box, as described in [“Using the Class Properties Dialog Box” on page 4-22.](#)
2. Type the multiplicity value in the Multiplicity text field.

There is no display mark for class multiplicity.

Designating Abstract Classes

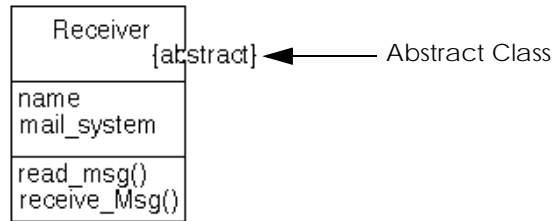
An abstract class is one for which no instances may be created. In the Email system example, the *Receiver* class provides traits that can be inherited by *User* and *Conference* classes, but there would not be any instances of a receiver in a system (there would be instances of users and conferences). The *Receiver* class is an abstract class.

To designate an abstract class:

1. Use the Properties dialog box, as described in [“Using the Class Properties Dialog Box” on page 4-22.](#)
2. Click the Abstract button.

The abstract class display mark appears in curly braces by the class name. [Figure 9](#) shows an abstract class.

Figure 9: Abstract Class



Designating External Classes

An external class is a class with the External annotation. You can specify any class as external, but this annotation is designated by default on classes brought into the system with the Reverse Engineering utility. (For information on reverse engineering, see [Generating and Reengineering Code](#).)

A typical external class is one that comes from an existing library. Since StP assumes that the complete definition of these classes is not in the system, StP treats them differently from other classes. For example, they are excluded from semantic checks.

To designate an external class:

1. Use the Properties dialog box, as described in [“Using the Class Properties Dialog Box” on page 4-22](#).
2. Click the External button.

There is no display mark for an external class.

Adding Parameters to a Parameterized Class

To add parameters to parameterized classes:

1. Use the Properties dialog box, as described in [“Using the Class Properties Dialog Box” on page 4-22](#).
2. Type the parameters in the Parameters text field.

For more information, see [“Representing Parameterized and Instantiated Classes” on page 4-18](#).

Designating a Stereotype

A stereotype extends the semantics of a model element (class, association, and so on). A stereotype is one of three extensibility mechanisms in UML (the others are constraints and tagged values).

You can attach a pre-defined or user-defined stereotype. For a list of the pre-defined stereotypes and information on defining additional stereotypes, see [Chapter 13, “Creating a Stereotype Diagram.”](#)

To designate a stereotype for a class:

1. Use the Properties dialog box, as described in [“Using the Class Properties Dialog Box” on page 4-22.](#)
2. Type the stereotype name in the Stereotype text field.

If the Stereotype display mark is set to display, the stereotype name appears over the class symbol within guillemets (for example, «interface»).

Specifying Constraints on Classes

A constraint on a class restricts the values of the class's attributes. A constraint is one of three extensibility mechanisms in UML (the others are stereotypes and tagged values).

For example, given a *Check payment* class, a class constraint can prevent the *amount* attribute from exceeding \$199.

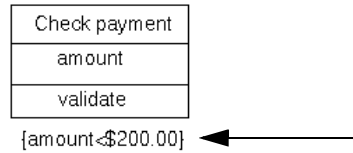
To designate a constraint for a class:

1. Use the Properties dialog box, as described in [“Using the Class Properties Dialog Box” on page 4-22.](#)
2. Type the constraint in the Constraints text field.

You can add multiple constraints; add each constraint on a new line.

The constraint display mark is the constraint's value in curly brackets, which appears below the class symbol, as shown in [Figure 10.](#)

Figure 10: Class Constraint Display Mark



Adding Tagged Values

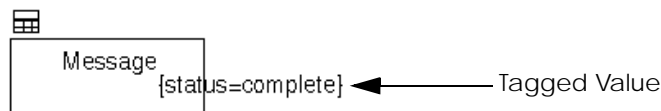
A tagged value is an explicit characteristic attached to a model element (class, association, and so on). A tagged value is one of three extensibility mechanisms in UML (the others are stereotypes and constraints).

To designate a tagged value for a class:

1. Use the Properties dialog box, as described in [“Using the Class Properties Dialog Box” on page 4-22](#).
2. Type the tagged value in the Tagged Values text field.
You can add multiple tagged values; add each tagged value on a new line.

The tagged value display mark is a subset of the ClassInfo display mark. The tagged value appears between curly braces and appears below and to the right of the class name, as shown in Figure 11.

Figure 11: ClassInfo Display Mark Showing a Tagged Value



Representing Objects (Class Instances)

Individual instances of classes are objects. Each instance of an object is distinguishable from other instances of the same object. Examples of objects of the *Message* class include message systems of different users.

Similar or identical objects can be grouped into object classes. Each object's class is an implicit property of the object.

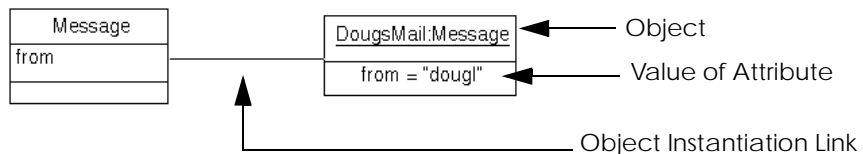
The scope of an object is the class it is instantiated from.

An object belongs to a class and is defined by the values of its attributes. For example, an object of the *Message* class may be defined by the value of the *from* attribute.

An object has no semantics; it is modeled to show its relationship to the class.

An object's name and the class it belongs to are automatically underlined. An object is related to its class by an object instantiation link. Two objects are connected by an object association link.

Figure 12: An Object of the Message Class



Before creating an object, ensure that the class that the object belongs exists.

Representing and Labeling an Object Instance

To model an object instance:

1. Insert an object symbol into the diagram.
2. Type the object and class name in the object symbol, as described in [“Setting Class Names” on page 7-11](#) in [Chapter 7, “Creating a Collaboration Diagram.”](#)
3. Drag a Value of Attribute symbol into the object symbol.
The Value of Attribute symbol specifies a value for an attribute of an object instance.
4. Label the Value of Attribute symbol.
Use this syntax:
`<attribute>=<value>`

5. Repeat steps 3 and 4 to indicate values of all distinct attributes.
6. Insert a connection from the object symbol to the class symbol.

Creating an Object Diagram

UML supports the notion of object diagrams. An object diagram represents a static instance of a class diagram. To draw an object diagram using the Class Editor, create a diagram using object symbols. An “object diagram” is a class diagram without any class symbols.

An object diagram can also be created using the Collaboration Editor. For information, see [“Creating an Object Diagram” on page 7-14](#).

Creating Relationships

Classes and objects in a class diagram are connected to each other through a relationship, represented by an arc. There are several relationship types, as shown in Table 6.

Table 6: Class Editor Relationships

Relationship	Description	For Details, See
Association	Connects two independent classes	“Creating Associations” on page 4-33
Generalization	Connects a specialized subclass to a general superclass	“Using Generalization Relationships” on page 4-50
Binds	Connects a class, instantiated class, or parameterized class to a parameterized class	“Using Binds Relationships” on page 4-51

Table 6: Class Editor Relationships (Continued)

Relationship	Description	For Details, See
Dependency	Connects a class, parameterized class, or package to a class, parameterized class, package, or interface, in which one element is dependent on the other	“Using Dependency Relationships” on page 4-52
Implements	Connects a class to an interface	“Using Implements Relationships” on page 4-52
Object Association	Connects two objects	“Representing Objects (Class Instances)” on page 4-29
Object Instantiation	Connects an object to its class	

Changing Relationship Types

When you create a relationship, StP designates the arc's type based on the default arc type setting. To set the default arc type:

1. Choose **Tools > Options**.
2. Select the **Default Arc** tab.
3. To change the default arc type, select the desired arc from the options list.
4. Click **OK**.

You can also change the type of the existing arc by using the **Replace** command. To change an arc:

1. Select the link.
2. Choose **Edit > Replace**.
3. Select the desired arc from the options list.
4. Click **OK**.

By default, most connections between one class and another class are drawn as associations. Connections between instantiated classes,

parameterized classes, and interfaces are drawn with the appropriate type of link.

Creating Associations

An association is a connection of two or more independent classes or objects. The signature of an association is the combination of the roles for the association. (For more information, see [“Adding Details about Associations” on page 4-39](#).) The scope of an association and the connecting objects or classes is global.

Associations can have attributes. For details, see [“Creating Association Classes” on page 4-36](#).

Association Labels

Although association labels are interpreted in two ways, they are written for only one direction; the reverse direction is implicit in the label. For example, in Figure 13, the *User Sends the Message*; the *Message* is sent by the *User*.

Figure 13: Association Label



Inserting and Labeling Associations

To draw an association between classes:

1. Insert a connection from one class to another.
2. Label the association (double-click the label to activate a text box).

To break to a new line, press Shift+Enter.

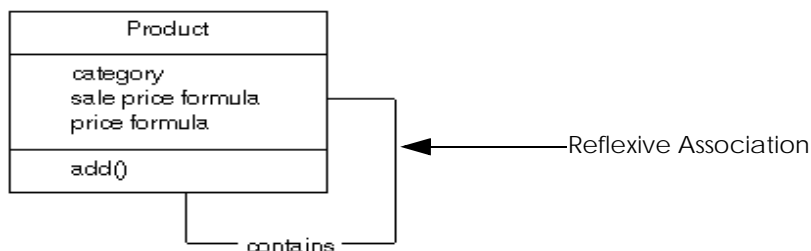
You can also use the Current Symbol Options dialog box to label the association. To display this dialog, select the association and choose **Edit > Current Symbol Options**.

For more information about the Current Symbol Options dialog, see [Fundamentals of StP](#).

Inserting Reflexive Associations

An association that involves instances of a single class only is a reflexive association.

Figure 14: Reflexive Association



Reflexive associations must have roles (see [“Adding Details about Associations” on page 4-39](#)) to distinguish one end of the association from the other.

To draw a reflexive association:

1. Insert a connection in the class.
The association is attached to the class.
2. Move the mouse into the drawing area and click the mouse button.
A vertex appears on the association.
3. Move the mouse to a second point in the drawing area and click the mouse button (optional).
Another vertex appears.
4. Move the association back into the class and click the mouse button.

Creating N-ary Associations

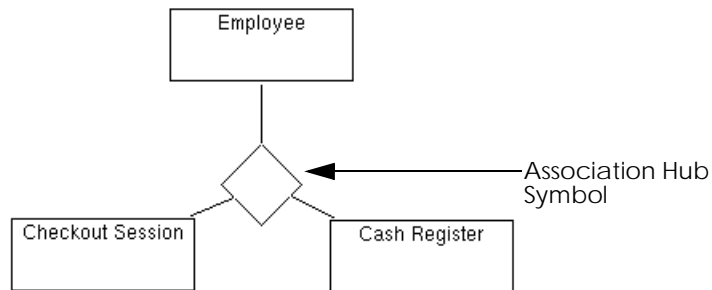
N-ary associations relate three or more classes.

Connections in n-ary associations do not have labels; the n-ary association hub symbol has the label for the relationship.

To draw an n-ary association:

1. Insert three or more classes into the drawing area.
2. Select the classes.
Press the Shift key to select multiple items.
3. Choose **UML > Create N-ary Association**.
The association hub symbol appears.

Figure 15: N-ary Association



4. Label the association hub by double-clicking the symbol.

For information on adding properties to an n-ary association, see [“Adding Details to N-ary Associations” on page 4-49](#).

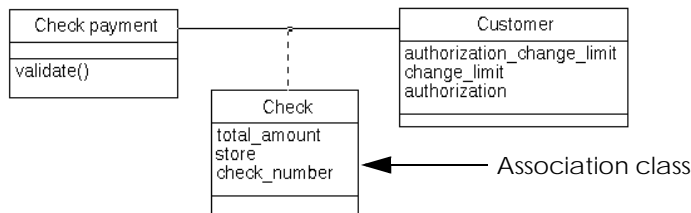
N-ary associations support association classes. For details, see [“Adding Association Classes to N-ary Associations” on page 4-37](#).

Creating Association Classes

An association can have characteristics with data values that cannot be attached to the class or object at either end without losing information. These characteristics are modeled as attributes or operations of an “association class,” which is a class belonging to the link or association. When attributes are modeled on links, they are called “link attributes.”

For example, in a supermarket model, there is an association between the classes *Check Payment* and *Customer*. An association class called *Check* can be added to this association. The attributes of *Check* can be “total amount,” “store,” and “check number.”

Figure 16: Association Class



Link attributes are useful in clarifying many-to-many associations. Also, a link attribute can have relationships with multiple classes.

To create an association class

1. Insert a class symbol onto the diagram.
This is your association class.
2. Label the class symbol.
3. Select the association that owns the association class.
4. Choose **UML > Create Association Class**.
One end of an Association Class Link is attached to the association.
5. Using the mouse, move the other end of the association into the association class and click the mouse button.

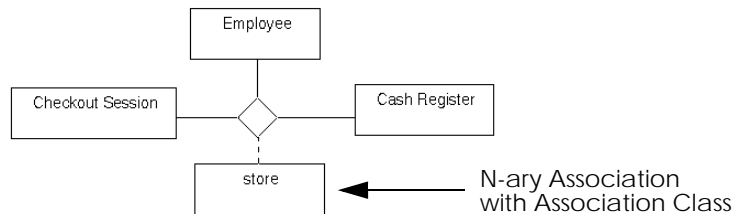
Adding Association Classes to N-ary Associations

An n-ary association can have an association class; the association class is attached to the n-ary association by an association class link.

To draw an n-ary association with an association class:

1. Draw an n-ary association (as described in [“Creating N-ary Associations” on page 4-34](#)).
 2. Insert a class symbol onto the diagram.
This is your association class.
 3. Label the class symbol.
 4. Insert an arc from the n-ary association hub symbol to the association class.
- An n-ary association link appears.
5. Change the n-ary association link you just inserted to an association class link by choosing **Edit > Replace**.

Figure 17: Labeled Association Class



Creating Or Associations

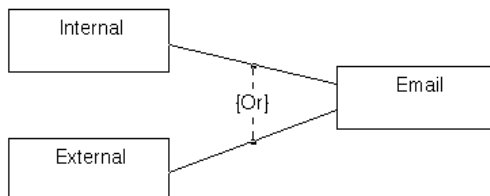
An Or association designates that only one of two or more associations may be instantiated for a particular object. The Or association is represented by a dashed line and the word “Or” displayed between the specified associations.

To create an Or association:

1. Select the associations connected to the class (use the Shift key to select multiple items).
2. Choose **UML > Create Or Association**.

A dashed line appears between the associations with “Or” displayed in the middle.

Figure 18: Or Association



Designating Roles

By default, each end of an association is a role. Optionally, each role can have an explicit role name, which serves to clarify the meaning of the relationship. Each role name appears next to the class it distinguishes.

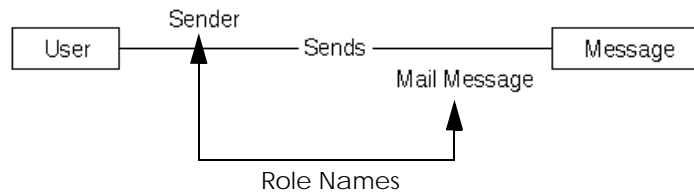
For example, in the mail system model, there is an association where the *User* class *Sends* to the *Message* class (see [Figure 19](#)). The *User* role name is *sender*, and the *Message* role name is *mail message*. That is, a user assumes the role of a sender with respect to a message; a message assumes the role of a mail message with respect to a user.

A role name cannot contain a colon or comma.

In reflexive associations, role names are necessary to distinguish objects.

To add role names to an association:

1. Select an association.
2. Choose **UML > Create Association Roles** or use the accelerator Alt+Ctrl+L.
Label handles appear in the diagram for each role.
3. Double-click each role to activate the text field and type the labels.

Figure 19: Role Names

Adding Details about Associations

In addition to the different types of associations, each association can have various characteristics, including:

- Multiplicity
- Role Navigability
- Qualifiers
- Ordering
- Constraints

In addition, associations support the three UML extensibility mechanisms—stereotypes, constraints, and tagged values.

Several details about associations cause display marks to appear in the class diagram, if the display marks are set to display. For information on display marks, see [“Using Display Marks” on page 4-9](#).

You add details to n-ary associations using the Object Annotation Editor. For more information, see [“Adding Details to N-ary Associations” on page 4-49](#).

Using the Association Properties Dialog Box

The Association Properties dialog box enables you to change the characteristics of a selected association and apply all the changes at once.

[Figure 20](#) shows the Association Properties dialog box.

Figure 20: Association Properties Dialog Box

Properties

Association Name:

Associated Classes

Class:

Role:

Stereotype:

Multiplicity:

Qualifiers:

AggregationType:

Role Navigability: ☐ ☐

Ordered: ☐ ☐

Extensibility

Stereotype:

One constraint value per line

Constraints:

One tagged value per line

Tagged Values:

To use the Association Properties dialog box:

1. Select an association.
2. Use one of the following methods to open the Properties dialog box:
 - From the **Edit** menu, choose **Properties**.
 - From the shortcut menu, choose **Properties**

The Association Properties dialog box appears.

3. Choose or enter all the desired settings for the selected association.
For details about each property, see Table 7 and subsequent sections.
4. Click **OK** or **Apply**.
The **OK** button applies the changes and dismisses the dialog. The **Apply** button applies the changes without dismissing the dialog.
5. To show display marks on your diagram, choose **View > Refresh Display Marks**.

Summary of Association Relationship Properties

Table 7 is a summary of the various parts of the Properties dialog box for associations.

Table 7: Association Properties Summary

Property	Description	Settings	For Details, See
Association Name	The text string that provides the label for the relationship.	User input or derived from the information in the diagram	“Inserting and Labeling Associations” on page 4-33
Class	The labels of the classes at the two ends of the relationship.	Derived from the information in the diagram (read-only)	“Inserting and Labeling Classes” on page 4-13
Role	The labels of the role symbols at each end of the association relationship.	Derived from the information in the diagram (read-only)	“Designating Roles” on page 4-38
Stereotype (in Associated Classes group)	Extends the semantics of each association role.	User input based on pre-defined and user-defined stereotypes	“Adding Stereotypes” on page 4-47
Multiplicity	The multiplicity of the role.	User input based on supported notation	“Setting Association Multiplicity” on page 4-42

Table 7: Association Properties Summary (Continued)

Property	Description	Settings	For Details, See
Qualifiers	The qualifier on the role.	User input	“Adding Qualifiers” on page 4-43
Aggregation Type	The containment for an association.	None, Composition, Aggregation	“Setting Aggregation” on page 4-44
Role Navigability	Indicates whether or not the role is traversed for purposes of code generation. If on, a display mark appears on the diagram.	On/Off	“Adding Directionality to Roles” on page 4-46
Ordered	Indicates whether or not the multiplicity on the role is ordered.	On/Off	“Adding Ordering to Multiplicity” on page 4-47
Stereotype (in Extensibility group)	Extends the semantics of the association.	User input based on pre-defined and user-defined stereotypes	“Adding Stereotypes” on page 4-47
Constraints	The constraints on the relationship.	User input—one constraint per line	“Adding Constraints” on page 4-48
Tagged Values	Explicit characteristics of the relationship.	User input—one tagged value per line	“Adding Tagged Values” on page 4-29

Setting Association Multiplicity

In an association, a single instance of one class can relate to instances of the other class one or more times. The number of instances that can relate at one time is the multiplicity of the association. For example, in the Email object model, there is a relationship between the classes *User* and *Conference*. There can be many instances of a user to one instance of a conference. That is, one conference can support many users.

You set multiplicity using the format:

```
[lower bound..upper bound]
```

in which lower bound and upper bound are integer values, specifying the range of the multiplicity. The star character (*) can be used as an upper bound value, indicating an unlimited upper bound.

Each end of the relationship has multiplicity. Table 8 illustrates some examples for multiplicity.

Table 8: Multiplicity Examples

Description	Example
The association is not specified	
A single instance of a class participates in the association	1
Zero or more instances of a class participates in the association	*
A single instance of a class may or may not participate in the association	0..1
A single instance or more than one instance of a class may participate in the association	1..*
A specific number of instances of a class may participate in the association	1..3,6,9..12

To indicate multiplicity on an association role:

1. Use the Properties dialog box, as described in [“Using the Association Properties Dialog Box” on page 4-39](#).
2. Type the multiplicity value in the Multiplicity text field.

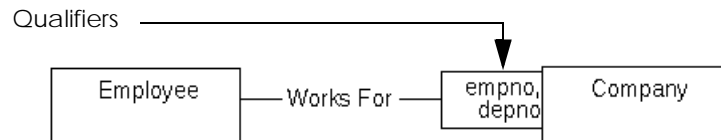
Adding Qualifiers

An association with multiplicity can be further defined through qualification, which reduces multiplicity. A qualified association contains an attribute (qualifier) that modifies one end of an association. One-to-many and many-to-many associations can be constrained through qualification.

For example, a company can employ many people. Within a company, each employee can be assigned a unique identification number. The class *Employee* plus a specific employee number (*empno*) can yield a specific employee. Alternatively, you can identify employees by their department numbers (*depno*).

Figure 21 shows how these two qualifiers are shown on the *Works For* association between the classes *Employee* and *Company*. Qualifiers are always shown attached to the class opposite the one they belong to.

Figure 21: Qualified Association



To add a qualifier to an association role:

1. Use the Properties dialog box, as described in [“Using the Association Properties Dialog Box” on page 4-39](#).
2. Type the qualifier in the Qualifiers text field.
You can add multiple qualifiers to each association role.

If you are generating code from your model and the qualifier is not an attribute of the class on the other end of the association, use the syntax `<name>:<type>` for the qualifier.

Modeling Association Qualifiers As Class Attributes

You may choose to store association qualifiers as attributes of the class they belong to. You can do this automatically using the Class Table Editor. For more information, see [Chapter 5, “Defining Classes with the Class Table Editor.”](#)

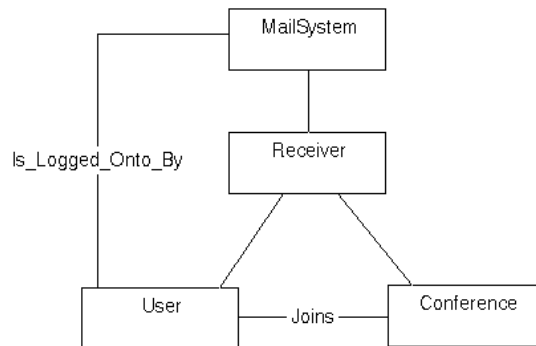
Setting Aggregation

An aggregation association represents various classes as components of another class. Instances of the component (“part of”) classes can be

assembled into a single instance of the assembly class. Aggregations also show relationships between classes and packages.

An example of an aggregation in the Email system consists of users and conferences. In the Email model, *User* and *Conference* are subclasses of *Receiver*. Therefore, you model the relationship between the *MailSystem* class (the assembly object), and the *Receiver* class (the components) as an aggregation. Figure 22 illustrates this relationship.

Figure 22: Aggregation



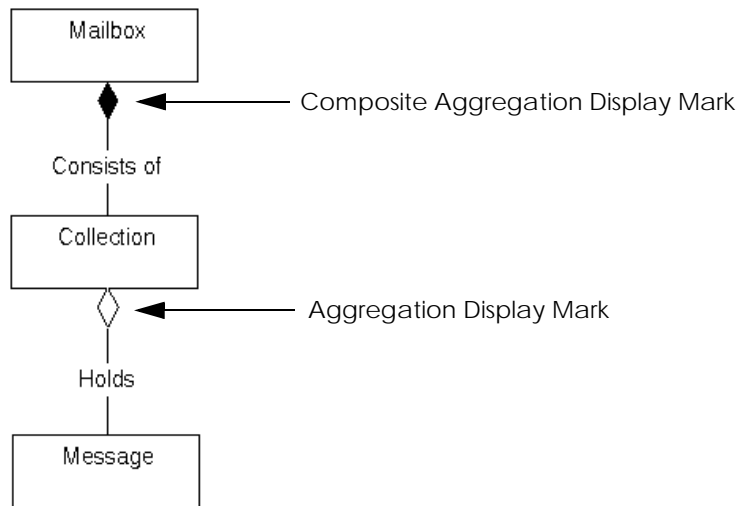
Aggregation associations incorporate:

- **Transitivity**—If a low-level object (A) is incorporated in a higher-level object (B), and B is incorporated in a higher-level object (C), then A is incorporated in C.
- **Antisymmetry**—If one object (A) is incorporated in another object (B), then B cannot be incorporated in A.

In some aggregation relationships, a class can contain another class. This is a composite aggregation. In the example shown in [Figure 23](#), the *Mailbox* class physically contains the *Collection* class. If the *Mailbox* class is destroyed, the *Collection* class is destroyed also. The display mark for composite aggregation is a solid diamond.

In contrast, the *Collection* class contains the *Message* class by reference. If the *Collection* class is destroyed, this does not affect the *Message* class. The display mark for standard aggregation is a hollow diamond.

Figure 23: Aggregation Display Marks



To indicate aggregation:

1. Open the Properties dialog box, as described in [“Using the Association Properties Dialog Box” on page 4-39](#).
2. Select the aggregation value (composition or aggregation) from the Aggregation Type pull-down menu.

Adding Directionality to Roles

If an association is traversed in a single direction, the association acts like a pointer. This is a function of the role of one of the connected classes.

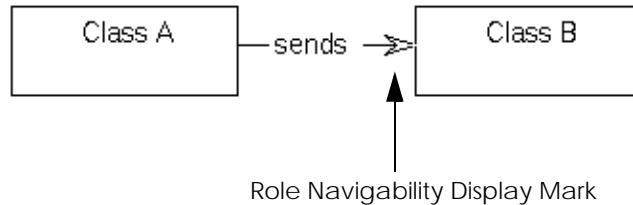
For code generation, if you do not set role navigability, the association is not implemented in that direction. For more information, see [Generating and Reengineering Code](#).

To add directionality to a role:

1. Use the Properties dialog box, as described in [“Using the Association Properties Dialog Box” on page 4-39](#).
2. Click the Role Navigability button.

The Role Navigability display mark is shown in [Figure 24](#).

Figure 24: Role Navigability Directionality



Adding Ordering to Multiplicity

In some cases, objects with multiplicity of many are explicitly ordered.

To indicate ordering of objects:

1. Use the Properties dialog box, as described in [“Using the Association Properties Dialog Box” on page 4-39](#).
2. Click the Ordered button.

There is no display mark for ordering.

Adding Stereotypes

A stereotype is an explicit characteristic attached to a model element (class, association, association role, and so on). A stereotype is one of three extensibility mechanisms in UML (the others are tagged values and constraints).

You can attach a pre-defined or user-defined stereotype. For a list of the pre-defined stereotypes and information on defining additional stereotypes, see [Chapter 13, “Creating a Stereotype Diagram.”](#)

To designate a stereotype for a relationship:

1. Use the Properties dialog box, as described in [“Using the Association Properties Dialog Box” on page 4-39](#).
2. Type the stereotype name in the Stereotype text field.

For associations, stereotypes can be entered for association roles (directly beneath the role names) and for the association (in the Association Extensibility section).

The name of the stereotype appears near the relationship or role name within guillemets (for example, «bind»).

Adding Constraints

A constraint is an explicit characteristic attached to a model element (class, association, and so on). A constraint is one of three extensibility mechanisms in UML (the others are stereotypes and tagged values).

A constraint on an association is a restriction on the values an association can have. The constraint can add information about the association.

To add relationship constraints:

1. Use the Properties dialog box, as described in [“Using the Association Properties Dialog Box” on page 4-39](#).
2. Type the constraint in the Constraints text field.
You can add multiple constraints to each association role; add each constraint on a new line.

There is no display mark for constraints on relationships.

Adding Tagged Values

A tagged value is an explicit characteristic attached to a model element (class, association, and so on). A tagged value is one of three extensibility mechanisms in UML (the others are stereotypes and constraints).

To designate a tagged value for a relationship:

1. Use the Properties dialog box, as described in [“Using the Association Properties Dialog Box” on page 4-39](#).
2. Type the tagged value in the Tagged Values text field.
You can add multiple tagged values; add each tagged value on a new line.

There is no display mark for tagged values on relationships.

Adding Details to N-ary Associations

You add details to n-ary associations through the Object Annotation Editor (OAE). You can add the following details to n-ary associations:

- Multiplicity
- Role Navigability
- Qualifiers
- Ordering
- Aggregation Type

To add details to n-ary associations:

1. Select the appropriate association link between the class and the n-ary hub symbol.
2. Use one of the following methods to open the OAE:
 - Choose **Edit > Object Annotation**
 - Choose **Object Annotation** from the shortcut menu
 - Use the accelerator Ctrl+a

For complete instructions on using the OAE, see [Fundamentals of StP](#).

3. Open the UmlRole folder in the OAE.
4. With the Role Definition note highlighted, choose **Edit > Add Item** and add the appropriate items.

For a description of the items, refer to [Table 7 on page 4-41](#).

5. For each added item, select the item and add or change the value; then click **Set**.
6. Choose **File > Save** to save the annotation.
7. Choose **File > Exit** to exit the OAE.

Using Specialized Relationships

In addition to associations, class diagrams support the following specialized relationships:

- Generalization

- Binds
- Implements
- Dependency

Specialized relationships support the three UML extensibility mechanisms—stereotypes, constrains, and tagged values—described in [“Adding Details to Specialized Relationships” on page 4-53](#).

Several details about relationships cause display marks to appear in the class diagram, if the display marks are set to display. For information on display marks, see [“Using Display Marks” on page 4-9](#).

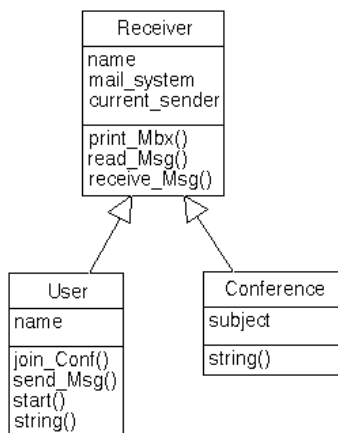
Using Generalization Relationships

In a generalization, various subclasses represent specialized versions of the superclass, each with its own attributes and operations that build on those inherited from the superclass.

An example of a generalization from the Email object model is formed by the superclass *Receiver* and its subclasses *User* and *Conference*. That is, the receiver of a message can be either a user or a conference. Figure 25 illustrates this generalization.

The generalization link is represented by a solid line with a hollow triangle pointing towards the superclass.

Figure 25: Generalization



Subclasses can:

- Inherit from superclasses over an arbitrary number of levels
- Extend the characteristics of the superclass by adding attributes or operations
- Have attributes or operations that override an attribute or operation of a superclass.

Classes in a generalization association must have names. The scope of a generalization is global. The signature of the generalization is the combination of the superclass with the sorted names of the subclasses.

To create a generalization, refer to [“Changing Relationship Types” on page 4-32](#). To add details to a generalization relationship, see [“Adding Details to Specialized Relationships” on page 4-53](#).

For information about creating generalizations from Reverse Engineering, see [Generating and Reengineering Code](#).

Using Binds Relationships

A binds link connects any class symbol to a parameterized class; it represents a refinement of the parameterized class.

The binds link is represented by a dashed line with a hollow triangle pointing towards the parameterized class. Each binds link must include bind arguments and the stereotype «bind».

Bind arguments represent the actual parameters of the parameterized class.

To draw a binds link:

1. Connect a class, instantiated class, or parameterized class to a parameterized class.
If necessary, select the link and choose **Edit > Replace** and change the link to Binds.
2. Add the stereotype «bind».
For information on attaching a stereotype, see [“Adding Stereotypes” on page 4-47](#).
3. Add bind arguments.

For information on adding bind arguments, see [“Adding Details to Specialized Relationships” on page 4-53](#).

The Binds Arguments display mark appears near the binds link, as shown in [Figure 5 on page 4-19](#).

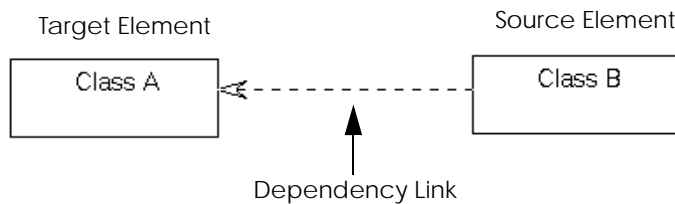
For more information on parameterized and instantiated classes, see [“Representing Parameterized and Instantiated Classes” on page 4-18](#).

Using Dependency Relationships

A dependency link connects a class or a package to another class, package, or interface. It indicates that a change in one element (the target element) may require changes in the second element (the source element).

The dependency link is represented by a dashed line with an arrow pointing towards the target element.

Figure 26: Dependency



To create a dependency, refer to [“Changing Relationship Types” on page 4-32](#). To add details to a dependency relationship, see [“Adding Details to Specialized Relationships” on page 4-53](#).

Using Implements Relationships

An implements relationship connects a class to the interface it implements. An interface represents the externally visible behavior of a class.

The implements relationship is represented by a solid line between a class and an interface. For an example, see [Figure 7 on page 4-22](#).

For more information on interfaces, see [“Representing Interfaces” on page 4-20](#).

To create an implements relationship, refer to [“Changing Relationship Types” on page 4-32](#). To add details to an implements relationship, see [“Adding Details to Specialized Relationships” on page 4-53](#).

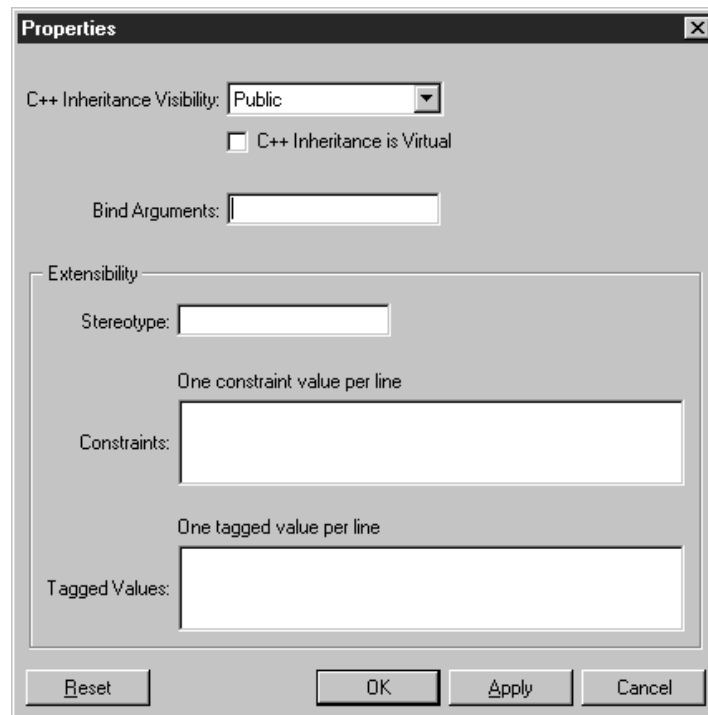
Adding Details to Specialized Relationships

The Properties dialog box enables you to change the characteristics of a selected specialized relationship and apply all the changes at once.

Figure 27 shows the Properties dialog box for binds relationships. The Property dialog boxes for generalization, dependency, and implements relationships are similar.

See [Table 9](#) for a description of the properties available for each specialized relationship.

Figure 27: Properties Dialog Box for Specialized Relationships



The image shows a 'Properties' dialog box with a title bar containing a close button. The dialog is divided into several sections. The top section has 'C++ Inheritance Visibility:' with a dropdown menu set to 'Public' and a checkbox for 'C++ Inheritance is Virtual' which is unchecked. Below this is a 'Bind Arguments:' text box. The 'Extensibility' section is enclosed in a rounded rectangle and contains a 'Stereotype:' text box, a label 'One constraint value per line' above a 'Constraints:' text box, and a label 'One tagged value per line' above a 'Tagged Values:' text box. At the bottom of the dialog are four buttons: 'Reset', 'OK', 'Apply', and 'Cancel'.

To use the Properties dialog box for specialized relationships:

1. Select a specialized relationship.
2. Use one of the following methods to open the Properties dialog box:
 - From the **Edit** menu, choose **Properties**.
 - From the shortcut menu, choose **Properties**The Properties dialog box for the symbol appears.
3. Choose or enter all the desired settings for the selected relationship.
For details about each property, see Table 9.
4. Click **OK** or **Apply**.
5. To show display marks on your diagram, choose **View > Refresh Display Marks**.

Summary of Specialized Relationship Properties

Table 9 is a summary of the various parts of the Properties dialog box for generalization, binds, implements, and dependency relationships.

Table 9: Specialized Relationship Properties Summary

Property	Description	Settings	For Details, See
C++ Inheritance Visibility	Indicates if the C++ inheritance is public, protected, or private. Not available for dependencies.	Public Private Protected	Generating and Reengineering Code
C++ Inheritance is Virtual	Indicates if the C++ inheritance is virtual. Not available for dependencies.	On/Off	
Bind Arguments	The actual parameters of an instantiated class. Available for binds only.	User input	“Using Binds Relationships” on page 4-51
Stereotype	Extends the semantics of the relationship.	User input based on pre-defined and user-defined stereotypes	“Adding Stereotypes” on page 4-47

Table 9: Specialized Relationship Properties Summary (Continued)

Property	Description	Settings	For Details, See
Constraint	The constraints on the relationship.	User input—one constraint per line	“Adding Constraints” on page 4-48
Tagged Values	Explicit characteristics of the relationship.	User input—one tagged value per line	“Adding Tagged Values” on page 4-48

Creating Class Packages

A class package represents an aggregate of classes or class packages (called “members” of the package) that are grouped together because they collaborate to provide a set of services.

Labeling Class Packages

A class package name must be unique within a system.

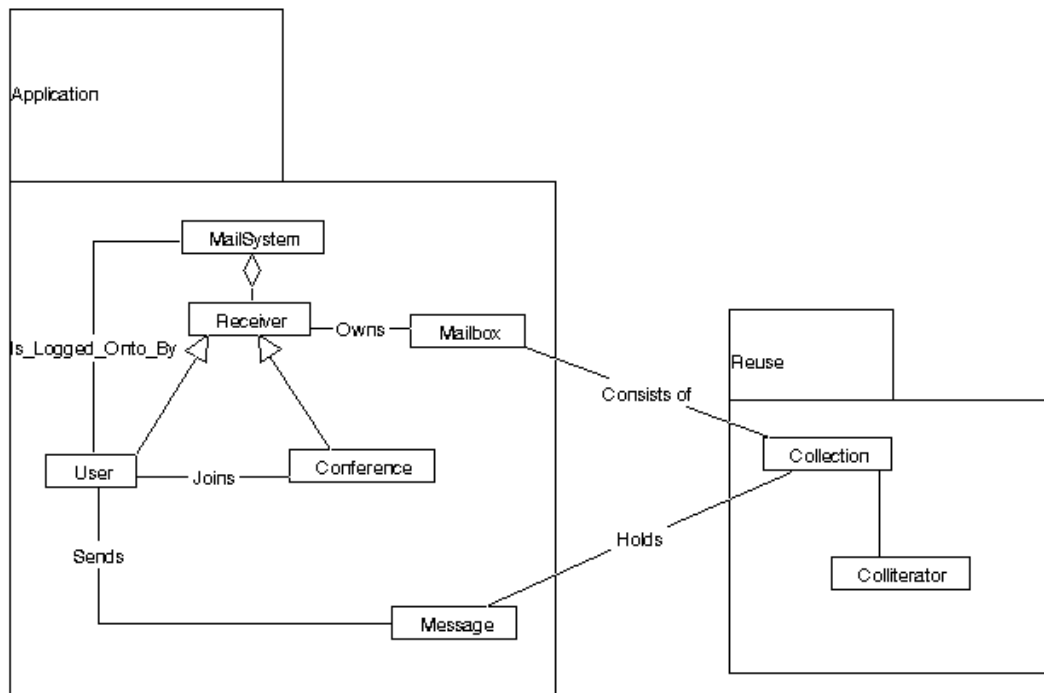
When labeling class packages, follow the rules for labeling classes in [“Inserting and Labeling Classes” on page 4-13](#).

Identifying Class Packages

Identifying class packages is an activity that can take place at any time during object-based development. It is also a relatively subjective task. The *MailSystem* diagram, shown in [Figure 28](#), contains several classes that could belong to various packages. For example, the following two class packages can be derived from the classes:

- *Application*—Contains classes that are pertinent to the mail system application (such as *MailSystem*, *User*, and *Mailbox*)
- *Reuse*—Contains classes that come from a reuse library (*Collection* and *Colliterator*)

Figure 28: Candidates for Class Package



To designate the members of a class package, you encompass the class or package members within the parent package symbol. If a package symbol is moved, all symbols contained within the package also move.

Creating a Class Package

To draw a class package:

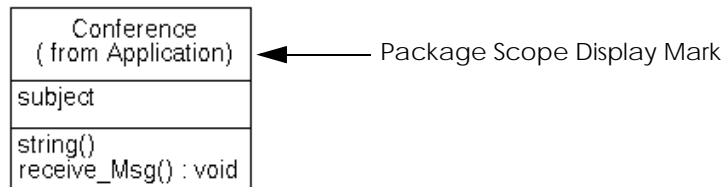
1. Insert a class package symbol into the drawing area.
The class package symbol is selected.
2. Label the class package symbol.
3. Encompass the members of the package by enlarging the package symbol.

Package Scope Display Mark

The Package Scope display mark designates a class or package as a member of the parent package. The display mark is in the format *(from <parentpackage name>)*.

In Figure 29, the *Conference* class is enclosed by the *Application* package. The actual diagram that shows the containment could be the same diagram or a different class diagram.

Figure 29: Package Scope Display Mark



Public Classes in a Class Package

If a class package contains public classes (described in [“Specifying Class Visibility” on page 4-25](#)), the public classes can be used by other classes in a system.

Non-public classes in a package cannot be used outside the class package.

Designating View Points

In StP, an object can have several references (symbols) in one or more diagrams for a system. In a large model it is possible to have several references that show different information about an object, such as its relationships to other objects. StP enables you to centralize some information about the object by designating one reference as a View Point for the object.

For example, in StP/UML, you can have several references to a class, and each reference can show a subset of all the class’s operations. This

decentralization of operations could lead to confusion. To avoid this, you can draw one reference to the class, attach all the class's operations to the reference, and designate it as the All Members View Point. This means that even though there are several references to the class that have operations, only the designated View Point shows all the operations.

After a View Point exists in a system, StP can navigate to it from various places in the same system.

StP/UML provides the following View Points for classes only:

- All Aggregations—The aggregate (whole) symbol which shows all aggregations (parts) attached
- All Generalizations—The generalization (parent) symbol that has all generalization subclasses attached

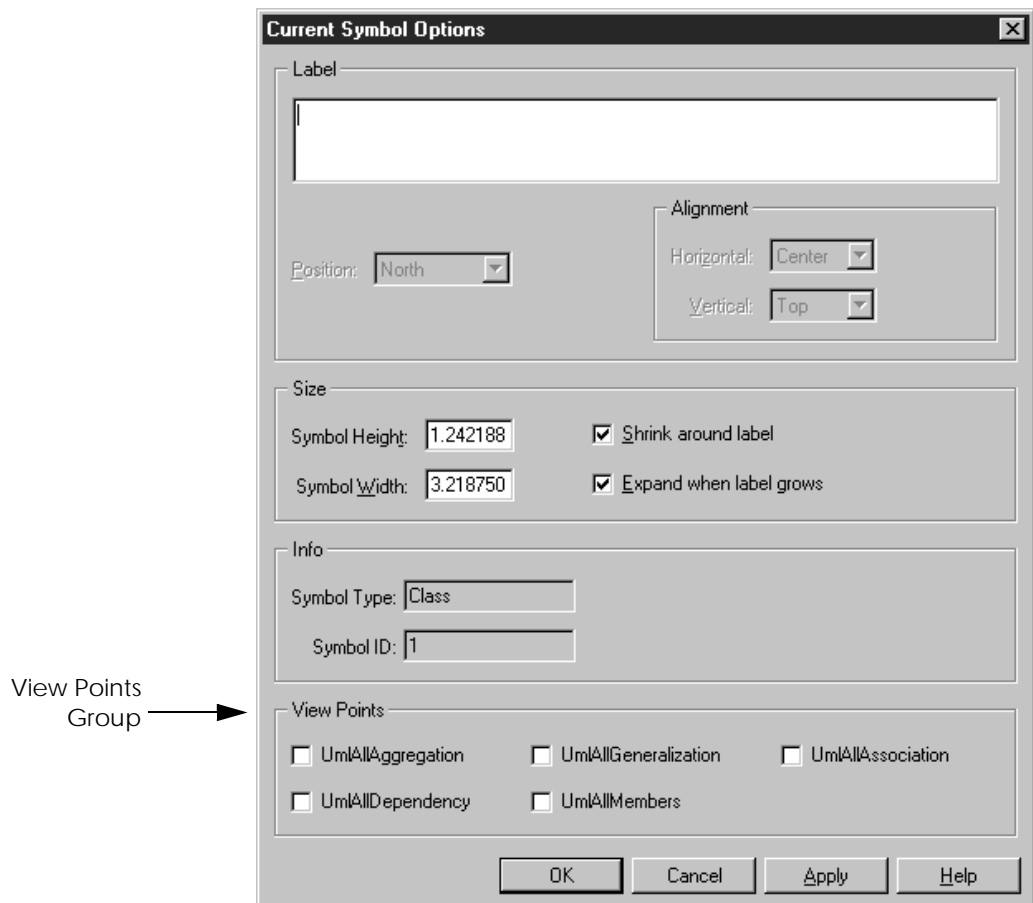
StP/UML provides the following View Points for classes and packages:

- All Associations—The class or package that shows all associations to and from the class or package
- All Dependencies—The class or package that shows all entities on which the class or package is dependent
- All Members—The class or package that shows all contained members (for a class, the contained members are attributes and operations; for a package, the contained members are classes and other packages)

To designate a View Point:

1. Select a class or package symbol.
2. Choose **Edit > Current Symbol Options**.
3. In the Current Symbol Options dialog box, check a View Point.

Figure 30: Current Symbol Options Dialog



4. Click **OK** or **Apply**.

Navigating to ViewPoints

To navigate to an existing View Point:

1. Select a class or package in the diagram.
2. Choose **GoTo > All Aggregations View, All Generalizations View, All Associations View, All Dependencies View, or All Members View**.

The symbol designated as the View Point appears and is blinking.

Changing View Points

To change or remove a View Point, use the Current Symbol Options dialog box to click the appropriate View Points check box on or off.

Validating a Class Diagram

StP/UML provides two options for checking the correctness and consistency of a model from the Class Editor:

- Checking the current diagram
- Checking the repository for the current system

When you check your model, the Message Log appears listing the errors. For information about using the Message Log, see [Fundamentals of StP](#).

Checking a Diagram

You can check a diagram to validate that the drawing is correct. Checking the diagram does not check the contents of the repository. Checks of a diagram include:

- Are all classes that participate in a generalization relationship or association named?
- Do any class names contain a slash or a comma?
- Do any role names contain a comma?
- Are the links legal for the symbol types?
- Are required links present?
- Are there too many of certain types of links in the diagram?
- Are there any vertexes with only one in link or one out link?
- Are labels missing from the diagram?

To check a diagram, choose **Tools > Check Syntax**.

Checking the Repository

You can check the repository for a system to validate that objects are properly defined and correct. Checks of the repository include:

- Do instantiated classes have any attributes or operations (they should not)?
- Do the actual parameters on a binds link match the formal parameters of the parameterized class in type and number?
- Do bind links have the stereotype «bind»?
- Are any classes or packages members of more than one class or package?

To check the repository, choose **Tools > Check Semantics**.

Updating a Class Diagram

If you make changes to a package or class anywhere in a model, you can update the diagram with this information automatically. You can update the package or class based on either a table definition or repository information.

To update the diagram, choose **UML > Attributes and Operations** or **Packages and Classes**.

[Table 10](#) lists the submenu commands that are available from these commands.

Table 10: UML Submenu Commands

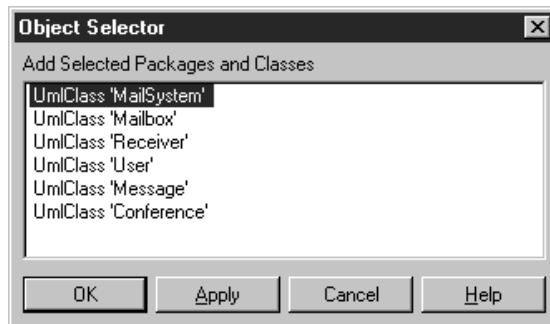
UML Menu Command	Submenu Command	Description
Attributes and Operations	Construct Class by Adding Selectively	Displays a dialog with all the attributes and operations that are currently contained in the class; lets you show selected ones on the symbol.
	Construct Class from Class Table Definitions	Rebuilds the class from a class table.
	Construct Class from All Definitions	Rebuilds the class symbol with all attributes and operations in the repository.
	Construct from Reverse Engineering	Gets the definition of the selected class from the source code files and rebuilds this reference to it. Available if a semantic model, created with Reverse Engineering, is present for the current system. For more information, see Generating and Reengineering Code .
Packages and Classes	Construct Package by Adding Selectively	Uses the repository to display a dialog with classes and packages that are contained within the package; lets you show selected ones on the symbol.
	Construct Package by Adding Selectively (Global)	Uses the repository to display a dialog with all the classes and packages that do not exist as part of any package (global classes); lets you show selected ones on the symbol.
	Construct Package from All Components	Uses the repository to rebuild the package from all components (classes and packages) of the selected package.

To update the diagram:

1. Select a class or package.
2. Choose a command from the **Attributes and Operations** or **Packages and Classes** cascade menu (see [Table 10](#)).

If you choose **Construct Package By Adding Selectively** or **Construct Package By Adding Selectively (Global)**, the Object Selector appears, as shown in Figure 31.

Figure 31: Selecting an Object



3. Select an object from the list.
4. Click **OK**.

The updated information appears in the diagram.

You can also use keyboard accelerators:

- For constructing selected classes from table definitions, use **Alt+Ctrl+C**.
- For constructing selected classes from all definitions, use **Ctrl+Shift+A**.

Updating a Class Table

You can update a class table based on information in a class diagram. For more information, see [Chapter 5, "Defining Classes with the Class Table Editor."](#)

5

Defining Classes with the Class Table Editor

This chapter describes how to define classes by using the class table editor. Topics covered are as follows:

- [“What Are Class Tables?” on page 5-1](#)
- [“Using the Class Table Editor” on page 5-2](#)
- [“Defining and Modifying Classes” on page 5-12](#)
- [“Displaying Inherited Operations” on page 5-16](#)
- [“Validating a Class Table” on page 5-18](#)
- [“Updating a Class Table” on page 5-19](#)
- [“Deleting a Class From the Repository” on page 5-19](#)

What Are Class Tables?

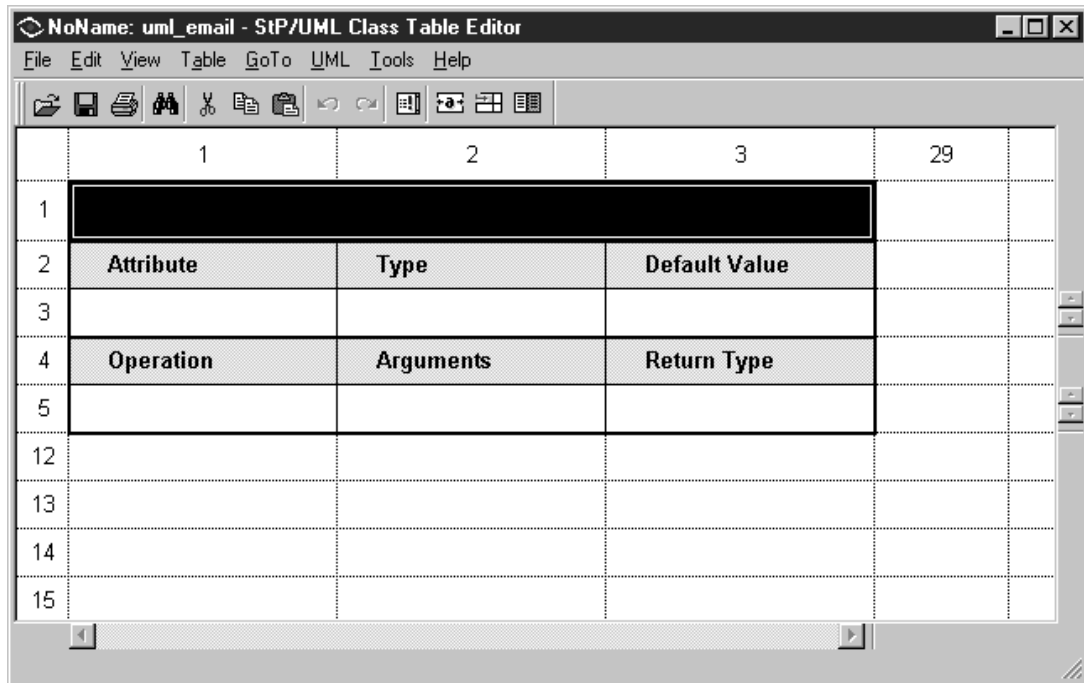
In StP/UML, you show the relationships between classes on a class diagram. On the diagram, you can also indicate the names of a class's attributes and operations. However, you capture the full definitions of a class's attributes and operations in a class table, which provides capabilities for defining extensive information for a given class.

The Class Table Editor uses the same elements as the Class Editor. For general information about these elements, see [Chapter 4, “Creating a Class Diagram.”](#)

Using the Class Table Editor

With the Class Table Editor, you define all the attributes and operations for a single class.

Figure 1: The Class Table Editor



The Class Table Editor provides all the functions and menu options of every StP table editor. For general information about using StP table editors, see [Fundamentals of StP](#).

When you start the editor, some sections may be hidden by default. For information about showing hidden sections, see [“Hiding and Showing Table Sections” on page 5-8](#).

Starting the Class Table Editor

You can start the Class Table Editor from:

- A class symbol in a class diagram
- Sequence Editor
- Collaboration Editor
- State Editor
- Activity Editor
- The StP Desktop

To create a class table, start the Class Table Editor by navigating from a class symbol in a class diagram (described in [Chapter 4, “Creating a Class Diagram”](#)). The class table corresponds to the class that is the source of the navigation.

If you navigate to the Class Table Editor, the name in the Class section is that specified in the source editor.

For more information about navigating to the Class Table Editor from another StP/UML editor, see the chapter in this manual that discusses the editor.

For instructions on starting the Class Table Editor from the StP Desktop, see [Chapter 2, “Using the StP Desktop.”](#)

Navigating to Object References

If you are viewing the table for a class, you can navigate to another diagram where the class appears. When you navigate to another editor, the current session continues. You can navigate to the following locations from the Class Table Editor:

- Class Editor
- State Editor
- Activity Editor
- Source code files, displayed in your source code viewer or standard editor window (for information about navigating to source code files, see [Generating and Reengineering Code](#))

The **GoTo** Menu provides dynamic navigation choices for the selected symbol.

To navigate to a target:

1. Select a cell.
2. From the **GoTo** menu, choose a command.

The navigation target appears.

Table 1 provides a summary of navigations for the Class Table Editor.

Table 1: Class Table Editor Navigations

Navigate From	Navigate To	GoTo Menu Command
Class name	Class symbol in a class diagram	Class Diagram
	State diagram with the class represented as the state machine	State Diagram for Class
	Activity diagram with the class represented as the state machine	Activity Diagram for Class
Class name, attribute	Source code for the selected object	Source Code
Operation	State diagram with the operation's class represented as the state machine	State Diagram for Operation
	Activity diagram with the operation's class represented as the state machine	Activity Diagram for Operation
	Viewer or editor with the source code declaration loaded for the selected operation.	Source Code Declaration

Table 1: Class Table Editor Navigations (Continued)

Navigate From	Navigate To	GoTo Menu Command
Operation	Viewer or editor with the source code definition loaded for the selected operation. Appears on the menu only if a semantic model, created with the Reverse Engineering tool, is present for the current system. For information about the Reverse Engineering tool, see Generating and Reengineering Code .	Source Code Definition
Class name, attribute, operation, signal sent, signal received	Requirement in a requirement table	Allocate Requirements

Using the UML Menu

In addition to the standard table menu options, the Class Table Editor also provides the UML Menu. This menu lists commands specific to class tables.

Table 2 describes the commands available from the UML menu.

Table 2: UML Menu Commands

Command	Description	For Details, See
Construct Class from Diagram Definitions	Provides a submenu for updating the class table with information from the class diagram.	“Constructing a Class From the Repository” on page 5-14 and “Updating a Class Table” on page 5-19
Construct Class from All Definitions	Provides a submenu for updating the class table with information from the StP repository.	

Table 2: UML Menu Commands (Continued)

Command	Description	For Details, See
Update from Reverse Engineering	Updates the class table with attributes and operations captured with the Reverse Engineering tool. Appears on the menu if a semantic model, created with the Reverse Engineering tool, is present for the current system.	Generating and Reengineering Code
Refresh Inherited Operations Section	Updates the inherited operations section of the table.	“Displaying Inherited Operations” on page 5-16

Parts of the Class Table Editor

This section describes the parts of the Class Table Editor.

Class

The class section contains the name of the class described in the table.

Attribute

Each row of cells in the Attribute section contains information that describes one attribute of the class. This section corresponds to the attributes part of a class symbol on a class diagram.

You can add an unlimited number of attributes using **Table > Insert Rows Before** or **Table > Insert Rows After** (described in [Fundamentals of StP](#)). [Table 3](#) lists and describes columns in the Attribute and Analysis Items sections.

Table 3: Generic Attribute Information

Header	Description
Attribute	Name of the attribute
Type	Data type of the attribute
Default Value	Initial value of the attribute; for empty strings, you must enter “ “
Visibility	The visibility of the attribute (private, protected, or public)
Class Attr?	Only a single value of the attribute exists, which is common to all class or object instances (True/False)
Derived?	The attribute value is calculated from other attributes (True/False)

Operation

Each row of cells in the Operation section contains information that describes one operation of the class. This section corresponds to the operations part of a class symbol on a class diagram.

You can add an unlimited number of operations using **Table > Insert Rows Before** or **Table > Insert Rows After** (described in [Fundamentals of STP](#)). Table 4 lists and describes columns in the Operation and Analysis Items sections.

Table 4: Generic Operation Information

Header	Description
Operation	Name of the operation
Arguments	Parameters of the operation
Return Type	Return type for the operation

Table 4: Generic Operation Information (Continued)

Header	Description
Visibility	The visibility of the attribute (private, protected, or public)
Class Op?	The operation is a class operation rather than a class instance operation (True/False)
Abstract?	The operation is abstract (cannot be implemented in an instance of the class) (True/False)
Throws	Exception types that can be thrown by the operation (separated by commas)

Inherited Operations

The Inherited Operations section of the table contains read-only information about inherited operations.

The columns include the operation name (in the format `class::operation`), its arguments, and the return type. By default, this section is hidden.

Signals Received and Sent

A signal is a named event. The Signals Received and Signals Sent sections of the table contain events that a class can receive and send. Each event can include arguments, which are the parameters of the signal event.

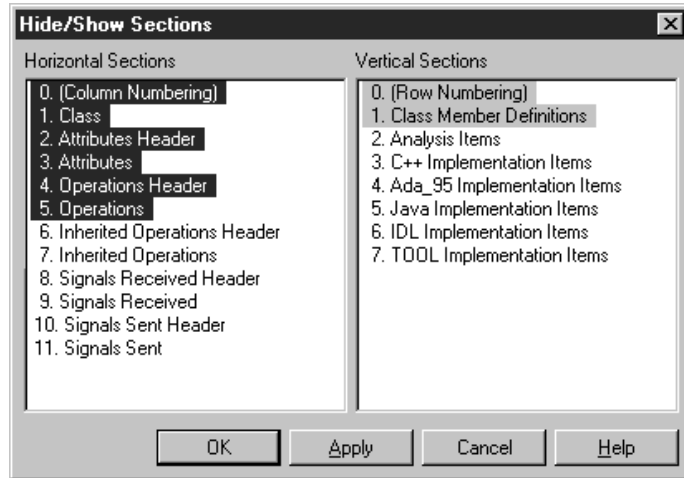
You can add an unlimited number of signals received and sent using **Table > Insert Rows Before** or **Table > Insert Rows After** (described in [Fundamentals of StP](#)).

Hiding and Showing Table Sections

The Class Table Editor is divided into meaningful sections that you can hide from view or show. There are horizontal sections consisting of rows and vertical sections consisting of columns; some of the sections overlap.

To hide or show a table section, choose **View > Hide/Show**. The Hide/Show dialog box appears.

Figure 2: Class Table Hide/Show Dialog Box



Highlighted items are displayed in the table editor. Click on an item to change its status. Use the Shift key to select a list of items. Use the Ctrl key to select items that are not adjacent.

Table 5 lists the generic horizontal sections of the Class Table Editor that you can hide or show.

Table 5: Class Table Generic Horizontal Sections

Section	Description
Class	See “Class” on page 5-6
Attributes Header	Attribute section header (read-only)
Attributes	See “Attribute” on page 5-6
Operations Header	Operation section header (read-only)
Operations	See “Operation” on page 5-7

Table 5: Class Table Generic Horizontal Sections (Continued)

Section	Description
Inherited Operations Header	Inherited operations section header (read-only)
Inherited Operations	See “Displaying Inherited Operations” on page 5-16
Signals Received Header	Signals received section header (read-only)
Signals Received	See “Signals Received and Sent” on page 5-8
Signals Sent Header	Signals sent section header (read-only)
Signals Sent	See “Signals Received and Sent” on page 5-8

Table 6 lists the sections of the Class Table Editor that hold information important to specific programming languages.

Table 6: Class Table Vertical Sections

Section	Description	Attribute Contents	Operation Contents
Class Member Definitions	Information about the class's attributes and operations.	Attribute	Operation
		Type	Arguments
		Default Value	Return Type
Analysis Items	Information about the class's attributes and operations. For a description of these sections, see “Attribute” on page 5-6 and “Operation” on page 5-7 .	Visibility	Visibility
		Class Attr?	Class Op?
		Derived?	Abstract?
			Throws

Table 6: Class Table Vertical Sections (Continued)

Section	Description	Attribute Contents	Operation Contents
C++ Implementation Items	Information for generating C++ code for the class. For a description of these sections, see Generating and Reengineering Code .	Const?	Const?
		Visibility	Visibility
		Volatile?	Virtual?
			Inline?
			Ctor Init List
Ada_95 Implementation Items	Information for generating Ada_95 code for the class. For a description of these sections, see Generating and Reengineering Code .	Aliased?	Visibility
		CA Visibility	Subunit?
			Inline?
Java Implementation Items	Information for generating Java code for the class. For a description of these sections, see Generating and Reengineering Code .	Visibility	Visibility
		Final?	Final?
		Transient?	Native?
		Volatile?	Synchronized?
IDL Implementation Items	Information for generating IDL code for the class. For a description of these sections, see Generating and Reengineering Code .	Type	Arguments
		Read Only?	Return Type
			Attribute
			Context
TOOL Implementation Items	Information for generating TOOL code for the class. For a description of these sections, see Generating and Reengineering Code .	Privilege	Privilege
		Set Expr	Return Event
		Get Expr	Exception Event
			Copy Return?
			Kind?

Choosing Valid Values for Cells

The Class Table Editor provides lists of valid values for annotations in the Analysis Items and programming language sections of the table. You can select a value from the list.

To set a valid value:

1. Select a cell in the Analysis Items programming language sections.
2. Choose **Edit > Set Label**.
An options list appears with a list of values appropriate for the cell.
3. Choose a value from the Set Label menu.
The value appears in the selected cell.

Adding Annotations for Member Functions

You can add member functions for code generation by annotating an operation cell in a class table. To add a member function, use the source code note for the appropriate language. For more information about annotating an operation row for code generation, see [Generating and Reengineering Code](#).

Defining and Modifying Classes

This section describes:

- Defining and modifying a class with a class table
- Constructing a class from the repository

Defining a Class With a Class Table

You define a class by using the Class Table Editor to provide the class's attribute and operation information. Before defining a class, you should draw it in a class diagram (as described in [Chapter 4, "Creating a Class Diagram"](#)), and then navigate from the class in the diagram to the Class Table Editor. The class name appears in the Class section of the table.

Alternatively, you can define a new class in the class table by choosing **New > Class Table** from the Stp Desktop.

To define a class:

1. Start the Class Table Editor.

For instructions, see [“Starting the Class Table Editor” on page 5-3](#).

2. If necessary, type the name of the new class in the class section.

You can also select the name of an existing class from a scrolling list of available classes by highlighting the class section and using the accelerator Ctrl+Tab.

Figure 3: Class Name in Class Section

Message			← Class Section
Attribute	Type	Default Value	
Operation	Arguments	Return Type	

3. Hide or display the desired table sections, as described in [“Hiding and Showing Table Sections” on page 5-8](#).
4. Complete the Attributes section of the table by adding attributes, types, default values, Analysis Items, and language-specific items, as required.

Figure 4: Attributes Section of the Class Table

Message			Analysis Items	
Attribute	Type	Default Value	Visibility	Class Attr?
body	String	" "	private	False
header	String	" "	private	False
from	String	" "	private	False
to	Receiver		private	False

5. Complete the Operations section of the table by adding operations, arguments, return types, Analysis Items, and language-specific items, as required.

To enter more than one argument for an operation, separate them with commas.

Figure 5: Operations Section of the Class Table

Operation	Arguments	Return Type	Visibility	Class Op?
Message	s:String &		public	False
fromTerminal		void	public	False

6. Choose **File > Save**.

Modifying a Class

You can modify or delete an attribute or operation in a table at any time using commands from the Edit menu. Be sure to save the table and update the repository after making any changes.

Constructing a Class From the Repository

You can use the UML commands of the Class Table Editor to define a class using information from a diagram or the repository. Besides filling the class table with information from these sources, you can perform several specialized functions:

- Copy abstract operations from superclasses for concrete definition in subclasses
- Model association qualifiers or indexes as class attributes

To define a class using repository commands:

1. Start the Class Table Editor.

For instructions, see [“Starting the Class Table Editor” on page 5-3](#).

Ensure that the name of the class appears in the Class section. To add a class name, go to Step 2. If the class name is correct, go to [Step 4](#).

2. If no class name appears, or if the class name is incorrect, select the class section and press Ctrl+Tab to bring up a scrolling list of available classes.
3. Select the appropriate class and click **OK**.
The class name appears in the class section.
4. Choose either **UML > Construct Class from Diagram Definitions** or **UML > Construct Class from All Definitions**.
For information about these commands, see [“Using the UML Menu” on page 5-5](#).
In either case, choose **Local Operations Only** from the command’s submenu.
The class table is filled with information from the designated source (diagram or repository definition) without consideration for any superclasses that the class inherits from.
5. Choose **File > Save**.

Inheriting Abstract Operations

Abstract operations in superclasses are placeholders for concrete operations that are implemented in inheriting classes.

To create a class where the abstract operations of the class’s superclass are inherited as concrete operations:

1. Start the class table for a class that is in a generalization relationship with a superclass.
2. Choose either **UML > Construct Class from Diagram Definitions** or **UML > Construct Class from All Definitions**.
In either case, choose **Inherit Abstract Operations** from the command’s submenu.
[Figure 6](#) is based on the generalization example in [Figure 25 on page 4-50](#). The *receive_Msg* abstract operation of the *Receiver* superclass appears as a concrete operation in the *Conference* class table.

Figure 6: Class Table With Concrete Operations

Conference			Analysis Items		C++ Items
Attribute	Type	Default Value	Visibility	Class Attr?	Const?
subject			private	False	False
Operation	Arguments	Return Type	Visibility	Class Op?	Const?
string			public	False	False
→ receive_Msg		void	public	False	False

3. Choose **File > Save**.

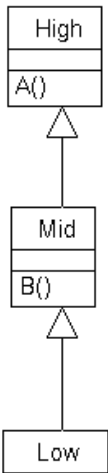
Displaying Inherited Operations

A class can inherit operations from other classes, both directly, and indirectly (through layers of generalizations). StP/UML enables you to display all operations in the repository that the current class inherits from superclasses.

[Figure 7](#) illustrates a generalization relationship where the inherited operation list for class *Low* would be:

- *High::A()*
- *Mid::B()*

Figure 7: Inherited Operations Example



To display inherited operations:

1. Show the Inherited Operations horizontal section of the table by using the Hide/Show dialog (described in [“Hiding and Showing Table Sections” on page 5-8](#)).
2. Choose **UML > Refresh Inherited Operations Section**.
The inherited operations appear in the table.

Figure 8: Class Table with Inherited Operations

Low		
Attribute	Type	Default Value
Operation	Arguments	Return Type
Inherited Class::Operation	Arguments	Return Type
High::A		
Mid::B		

3. Choose **File > Save**.

If the structure of a class in an inheritance relationship changes, you must use the **Refresh Inherited Operations Section** command to update the table.

Validating a Class Table

StP/UML provides two options for checking the correctness and consistency of a model from the Class Table Editor:

- Checking the current diagram
- Checking the repository for the current system

When you check your model, the Message Log appears listing the errors. For information about using the Message Log, see [Fundamentals of StP](#).

Checking a Table

StP/UML provides options for checking the correctness and consistency of the current table's contents, including checks for duplicate attribute and operation names, and for valid values in cells that can be validated. Checking the table does not check the contents of the repository.

To check a table's syntax, choose **Tools > Check Syntax**.

Checking the StP Repository

You can check the StP repository for a system to validate that objects are properly defined and correct.

To check the StP repository, choose **Tools > Check Semantics**.

Updating a Class Table

If you make changes to a class's attributes or operations anywhere in a class diagram, you can update the class table with this information automatically. You can update the class table based on a class diagram definition, repository information, or existing code. The class diagram must pass syntax checking before you can update the table.

To update the table, choose one of these commands from the UML menu in the Class Table Editor:

- **Construct Class from Diagram Definitions**
- **Construct Class from All Definitions**

The updated information appears in the table. For more information, see [“Constructing a Class From the Repository” on page 5-14.](#)

Updating a Class Diagram

You can update a class diagram based on information in a class table. For more information, see [Chapter 4, “Creating a Class Diagram.”](#)

Deleting a Class From the Repository

You delete a class from the repository by deleting the class symbol from all diagrams it appears in and then deleting its table.

To delete a class table:

1. From the Model pane on the StP Desktop, select the **Tables** category.
A list of subcategories for **Tables** appears.
2. Select **Class**.
A list of the class tables in the system appears in the Class pane.
3. Select the class to be deleted from the list.
4. Choose **File > Delete Table**.

6

Creating a Sequence Diagram

This chapter describes how to create and use sequence diagrams. Topics covered are as follows:

- [“What Are Sequence Diagrams?” on page 6-1](#)
- [“Using the Sequence Editor” on page 6-2](#)
- [“Drawing Sequence Diagrams” on page 6-9](#)
- [“Adding Details to Messages” on page 6-19](#)
- [“Adding Extensibility To Symbols” on page 6-24](#)
- [“Using Extension Points” on page 6-25](#)
- [“Using Packages” on page 6-26](#)
- [“Generating a Sequence Diagram from a Collaboration Diagram” on page 6-27](#)
- [“Validating a Sequence Diagram” on page 6-27](#)

This chapter provides brief descriptions of the elements of sequence diagrams.

What Are Sequence Diagrams?

A sequence diagram shows objects and their relationships in terms of the order of message passing.

Sequence diagrams are similar to collaboration diagrams. However, a collaboration diagram emphasizes message types, concurrency, visibility, and return values.

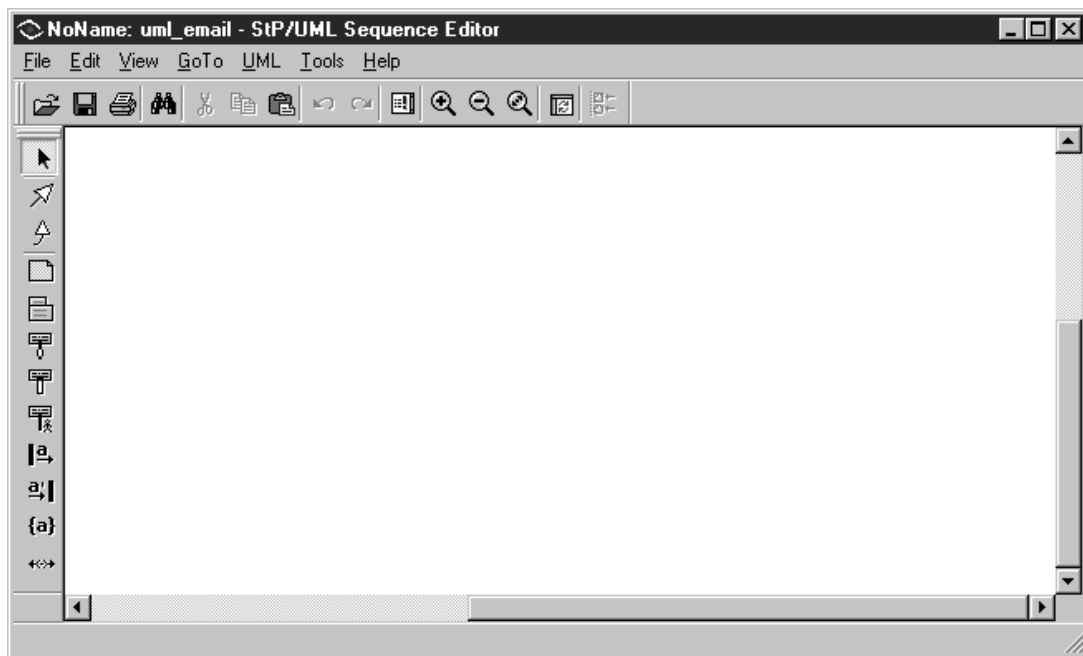
Together, sequence and collaboration diagrams are called interaction diagrams because they show the behavior (interaction) between objects.

For information on collaboration diagrams, see [Chapter 7, “Creating a Collaboration Diagram.”](#)

Using the Sequence Editor

You create a sequence diagram using the Sequence Editor.

Figure 1: Sequence Editor



The Sequence Editor provides all the functions and menu options of every StP diagram editor. For general information about using StP diagram editors, see [Fundamentals of StP](#).

Starting the Sequence Editor

StP/UML provides access to the Sequence Editor from:

- The StP Desktop
- Use Case Editor
- Collaboration Editor
- Stereotype Editor

For information about starting an editor from the StP Desktop, see [Chapter 2, “Using the StP Desktop.”](#)

For more information about starting (navigating to) the Sequence Editor from another StP/UML editor, see the chapter in this manual that discusses the editor.

Navigating to Object References

“Navigation” is the use of the inherent relationships between repository objects and references in diagrams or tables to select an object and display another reference to it. This reference can be within the same diagram, a different sequence diagram, or another StP/UML editor. When you navigate to another editor, the current session continues; in some cases, a predefined symbol appears in the editor.

You can navigate to these destinations from the Sequence Editor:

- Use Case Editor
- Class Editor
- Class Table Editor
- Collaboration Editor
- State Editor
- Activity Editor
- Requirements Table Editor

The **GoTo** Menu provides context-sensitive navigation choices for the selected symbol.

To navigate to a target:

1. Select a symbol.

2. Choose a command from the **GoTo** menu.
The navigation target appears.

Table 1 provides a summary of navigations for the Sequence Editor.

Table 1: Sequence Editor Navigations

Navigate From	Navigate To	GoTo Menu Command
Active Object, Passive Object	Class diagram with a reference to the class from which the object is instantiated	Class Diagram Where Class Is Referenced
	Class table with a reference to the class from which the object is instantiated	Class Table Where Class Is Referenced
	State diagram with a reference to the class from which the object is instantiated	State Diagram Where Class Is Referenced
	Activity diagram with a reference to the class from which the object is instantiated	Activity Diagram Where Class Is Referenced
Actor	Another scenario involving the same actor	Scenario Involving Actor
	The parent use case diagram	Use Case Diagram Involving Actor
Scenario name	Use case parent	Parent Use Case
	Another sequence or collaboration diagram scenario with the same use case parent	Sibling Use Case Scenario
Diagram	Corresponding collaboration diagram—if it does not exist, creates one	Collaboration Diagram

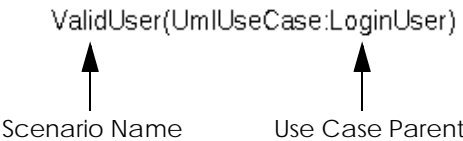
Table 1: Sequence Editor Navigations (Continued)

Navigate From	Navigate To	GoTo Menu Command
Package, active object, passive object, actor, start time, end time, constraint, extension point	Requirements table with the object id for the object loaded	Allocate Requirements

Creating a Scenario from a Use Case Diagram

A scenario is created when you navigate from a use case diagram to a sequence or collaboration diagram. The scenario name, along with the use case parent, appears in the sequence or collaboration diagram, as shown in Figure 2.

Figure 2: Scenario Name

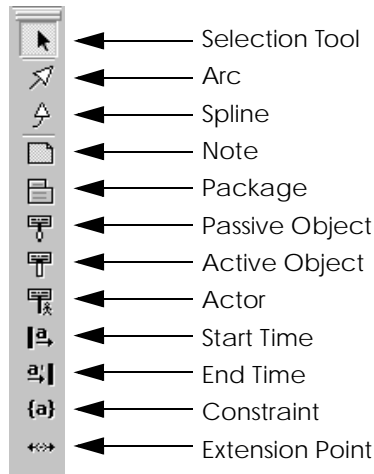


For information on the Use Case Editor, see [Chapter 3, “Creating a Use Case Diagram.”](#)

Using the Sequence Editor Symbol Toolbar

You select a symbol using the Symbol Toolbar.

Figure 3: Sequence Editor Symbol Toolbar



To detach the toolbar from the editor and make it a separate window, refer to [Fundamentals of StP](#).

Procedures for using all symbols on the Symbol Toolbar are described in this chapter.

Using the UML Menu

In addition to the standard diagram menu options, the Sequence Editor also provides the UML Menu. This menu lists commands specific to sequence diagrams.

Table 2 describes the commands available from the UML menu.

Table 2: UML Menu Commands

Command	Description	For Details, See
Generate Sequence from Collaboration	Creates a new sequence diagram from an existing collaboration diagram.	“Generating a Sequence Diagram from a Collaboration Diagram” on page 6-27

Table 2: UML Menu Commands (Continued)

Command	Description	For Details, See
Turn Auto Alignment On/Off	If on, automatically aligns object and actor symbols.	“Setting Automatic Alignment” on page 6-17
Align All Objects	Aligns all actor and object symbols in the diagram.	“Aligning All Objects and Actors” on page 6-18
Expand All Objects 50%/to Selected Size	If no actor or object is selected, stretches the symbols by 50%. If an actor or object is selected, resizes all actors and objects to the length of the selected symbol.	“Expanding Objects and Actors” on page 6-18
Distribute All Arcs	Evenly distributes arcs along the height of the actor and object symbols.	“Distributing All Arcs” on page 6-18
Shift Messages Down from Selected	Shifts messages down to create space for inserting a new message.	“Inserting Additional Messages” on page 6-16
Resequence All Messages	Resequences messages based on their position in the diagram.	“Resequencing Messages” on page 6-16
Sort Messages Based on Sequence	Places messages in order based on their sequence numbers.	“Reordering Messages” on page 6-17
Set Message Name	Sets the name of the message.	“Labeling a Message Using Set Message Name” on page 6-15

In addition, the **Edit > Properties** command provides a dialog box for adding details to sequence diagram objects. For more information, see:

- [“Adding Details to Messages” on page 6-19](#)
- [“Adding Extensibility To Symbols” on page 6-24](#)

Using Display Marks

Several object annotations cause display marks to appear in the diagram. For example, there is a display mark for activation of an object.

You control the behavior of display marks using the **Display Marks** tab in the **Options** dialog box. To display this tab:

1. From the **Tools** menu, select **Options**.
2. In the **Options** dialog box, choose the **Display Marks** tab.

For details about the Display Mark Options dialog box, see [Fundamentals of StP](#).

Table 3 lists the sequence diagram display marks. The names include any associated item names within parentheses.

Table 3: UML Display Marks

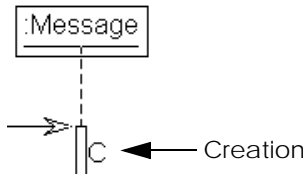
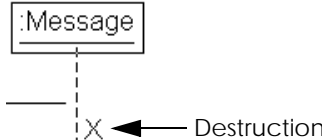
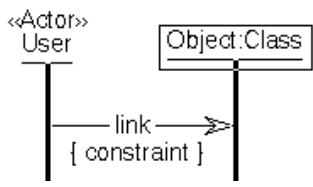
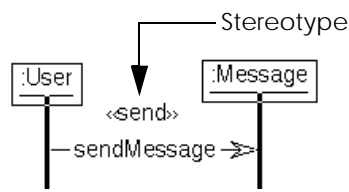
Name	Display Mark	Description
Creation (UmlMessageIs Creation)		<p>An incoming message that creates an object and/or activates an object.</p> <p>For details, see “Setting Creation” on page 6-22.</p>
Destruction (UmlMessageIs Destruction)		<p>An outgoing message that destroys an object.</p> <p>For details, see “Setting Destruction” on page 6-22.</p>

Table 3: UML Display Marks (Continued)

Name	Display Mark	Description
SequenceInfo (UmlStereotype)		The SequenceInfo display mark: <ul style="list-style-type: none">- Underlines the object name- Creates a box around the object name- Adds braces to constraints- Adds the «actor» stereotype to an actor For details, see “Drawing Sequence Diagrams” on page 6-9 and “Setting Transition Times” on page 6-23 .
Stereotype, StereotypeLink (UmlStereotype)		The stereotype of the model element. For details, see “Adding Extensibility To Symbols” on page 6-24 .

Drawing Sequence Diagrams

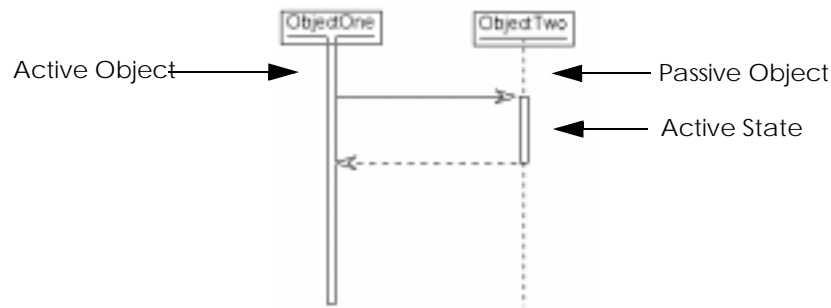
Individual instances of classes are objects. Each instance of an object is distinguishable from other instances of the same object. Examples of object instances in the conference class would be computer games, movies, and job listings.

Sequence diagrams show objects and the messages that pass between them, based on time. Sequence diagrams emphasize the order of message passing, as well as synchronization and constraints. In addition to drawing a sequence diagram, you can also create one automatically from an existing collaboration diagram as discussed in [“Generating a Sequence Diagram from a Collaboration Diagram” on page 6-27](#).

Representing Passive and Active Objects

The passive and active objects represent the existence of an object during a span of time. The passive object, a dashed vertical line, indicates that the object is in a passive state, awaiting activation. The active object, a solid vertical rectangle, indicates that the object has its own thread of control. You set the passive and active states between two objects by using message links. Figure 4 shows how the two states work.

Figure 4: Passive and Active Objects



To insert a new passive or active object into a sequence diagram:

1. Insert a Passive or Active Object symbol from the Symbol Toolbar into the drawing area.
2. With the object selected, label the object symbol as described in [“Setting Class Names”](#) below.

If the Sequence Info display mark is set to display, the object label is automatically underlined (see [Figure 4](#)).

Setting Class Names

In an object, the object label reflects the relationship between an object and its class. The label syntax for an object is:

[Object Name][:Class Name]

An object can:

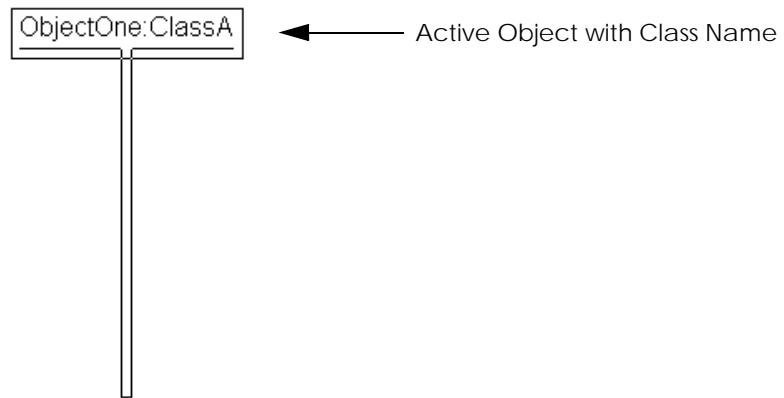
- Have a fully qualified name, as in *The System:MailSystem*

- Be an anonymous object of a class, as in :User
- Be an object of an anonymous class, as in Colliterator (in this case, no checking against the class diagram is possible)

To add an object's class name to the object label:

1. Select the object.
2. Label the object's class as described above.

Figure 5: Object with Class Name



If two or more fully qualified objects with the same name appear in the same diagram or diagrams, they are the same object.

Representing an Actor

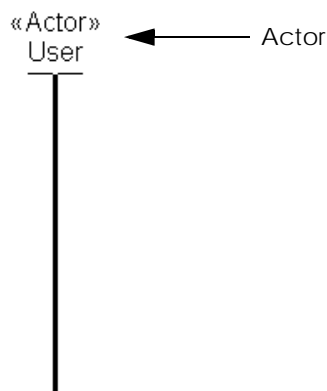
An actor represents an outside entity, such as a human, machine, computer task, or another system, that interacts with the system.

Actors are used in sequence diagrams, as well as use case and collaboration diagrams, to show external behavior. The actor invokes an interaction of an object.

To add an actor to a sequence diagram:

1. Insert an actor symbol into the diagram.
2. Label the actor.
3. Connect the actor to an object.

Figure 6: Actor in Sequence Diagram

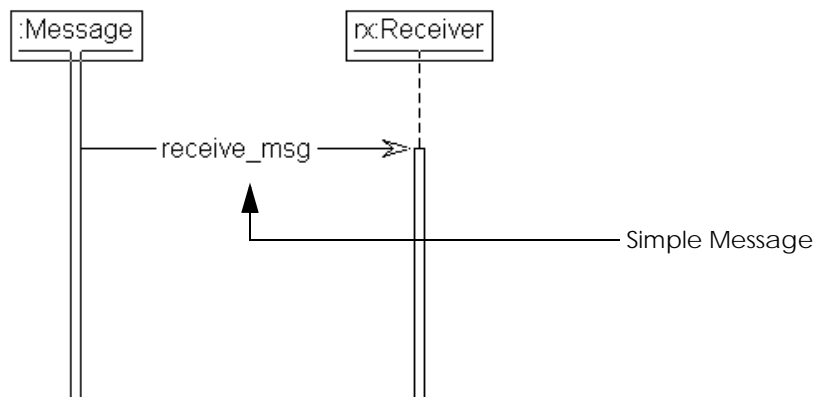


Understanding Message Links

Links between objects are instances of relationships between the object's classes. For example, the relationship between *Message* and *Receiver* (described in [Chapter 4, "Creating a Class Diagram"](#)) appears in the sequence diagram as a link.

Objects pass messages to other objects through links. In message passing, one object requests a service (client object) and the other object supplies a service (supplier). For example, *Message* sends a simple message to the *Receiver* to be the receiver of an email message (see Figure 7).

Figure 7: Simple Message Passing






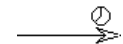


Types of Message Synchronization

The different behaviors of clients and suppliers can be differentiated by the type of message synchronization used in passing:

- Simple
- Synchronous
- Balking
- Timeout
- Asynchronous
- Return

Table 4 contains the symbols used for message synchronization.

Table 4: Message Synchronization

Type	Description	Symbol
Simple	There is a single thread of control.	
Synchronous	The client object waits for the supplier object to accept the message.	
Balking	If the supplier object cannot immediately service the message, the client object discards the message.	
Timeout	If the supplier object cannot service the message within a given amount of time, the client object discards the message.	
Asynchronous	The client object does not wait after sending the message; the supplier object queues the message for processing.	
Return	The supplier object returns a message to the client object.	

Drawing and Labeling Links between Objects

Although you can type message names directly on links, StP/UML enables you to choose message names from the available operation names in the repository using the Set Message Name dialog (see Figure 8).

Figure 8: Set Message Name Dialog

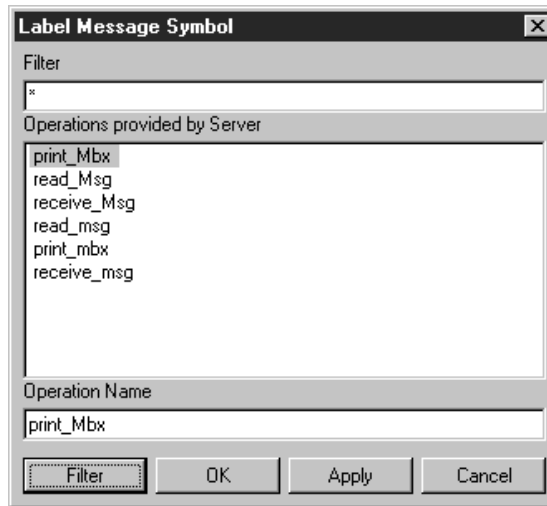


Table 5 is a summary of settings for the Set Message Name dialog box.

Table 5: Set Message Name Dialog Summary

Part	Description
Filter	Contains a string or wild card character that limits the entries in the scrolling list to those that match the entered pattern.
Operations provided by Server	Lists the operations on the supplying class. These operations can be directly implemented by the server or inherited from superclasses.
Operation Name	The selected operation. This name is used to label the message.

Labeling a Message Using Set Message Name

To use the Set Message Name dialog box:

1. Insert a connection from one object to another object.
2. Select the message.
3. Choose **Set Message Name** from the **UML** menu.
The **Set Message Name** dialog box appears.
4. Choose the appropriate settings.
5. Click **OK** or **Apply**.

The message name appears in the diagram. The full name of the operation is not copied to the message. The parameters, return type, and superclass (if any) are stripped out.

To add details to a message label, use the Properties dialog box, described in [“Adding Details to Messages” on page 6-19](#).

Labeling a Message Directly

To draw a link between objects and label it directly:

1. Insert a connection from one object to another object.
2. Label the message.

Label the message with an existing operation name. The complete format for a message label is:

```
predecessor guard/ sequence number[recurrence]:  
return type:= message name (arguments)
```

To add details to a message label, use the Properties dialog box, described in [“Adding Details to Messages” on page 6-19](#).

Alternatively, you can add details by entering them directly in the message label on the diagram.

For descriptions of the message label syntax, see [Table 6 on page 6-21](#).

An example message label is `1.2.1:p=find(files)`.

Changing the Type of Message Synchronization

To change the type of message synchronization:

1. Select the message.
2. Choose **Edit > Replace**.
3. Select the desired message type from the options list.
4. Click **OK**.

Alternatively, you can change the default arc type prior to inserting a message:

1. Choose **Tools > Options**.
2. Select the **Default Arc** tab.
3. Select the arc from the options list and click **OK**.

For more information, see [Fundamentals of StP](#).

Inserting Additional Messages

At times, you may need to insert a new message between existing messages.

To create space for the new message:

1. Select the existing message below the area you will be inserting the new message.
2. Choose **Shift Messages Down from Selected** from the UML menu.
The selected message and all messages below it shift down to create space for inserting a new message.
3. Insert the new message.

Resequencing Messages

To resequence all messages in a sequence diagram based on their vertical position in the diagram, choose **Resequence All Messages** from the UML menu.

This command reorders messages in sequential order from top to bottom using integers starting with zero (0).

Reordering Messages

To automatically reorder messages based on their sequence numbers, choose **Sort Messages Based on Sequence** from the UML menu.

Messages are repositioned in sequential order from top to bottom.

Adjusting Symbols in a Sequence Diagram

The following sections describe commands from the UML menu that adjust and align symbols in a sequence diagram.

Setting Automatic Alignment

Automatic alignment positions object and actor symbols evenly across the sequence diagram. If you turn on automatic alignment:

- If object or actor symbols currently exist, distributes the symbols evenly, resizes the symbols to match the length of the longest symbol, and aligns the symbols so that the tops and bottoms are aligned horizontally.
- Places new symbols to the far left or far right of the diagram depending on where the symbol is inserted.

When you turn automatic alignment on, the first symbol on the left side of the diagram is used as the base symbol. New symbols inserted anywhere to the left of the base symbol are positioned at the far left of the diagram. New symbols placed anywhere to the right of the base symbol are positioned at the far right of the diagram.

To align all existing and new object and actor symbols:

1. Choose **Turn Auto Alignment On** from the UML menu.
If there are existing symbols, a confirmation box appears.
2. Click **OK**.

Turn Auto Alignment On toggles with **Turn Auto Alignment Off**. To turn automatic alignment off, choose **Turn Auto Alignment Off** from the UML menu.

Aligning All Objects and Actors

To align all object and actor symbols in a sequence diagram so that the tops are aligned horizontally, choose **Align All Objects**. The objects are also stretched to match the length of the longest symbol.

Expanding Objects and Actors

To expand all object and actor symbols in a sequence diagram, choose one of the following:

- To expand all object and actor symbols by 50%, verify that no actor or object symbol is selected and choose **UML > Expand All Objects 50%**.
- To resize all object and actor symbols to the length of the selected object or actor symbol, select the object or actor and choose **UML > Expand All Objects to Selected Size**.

The **Expand All Objects 50%** command toggles with the **Expand All Objects to Selected Size** command.

Distributing All Arcs

To evenly distribute all messages along the height of the object and actor symbols, choose **Distribute All Arcs**.

Adding Details to Messages

StP/UML provides features for adding details to messages, such as:

- Return type
- Message arguments
- Predecessor guard
- Message sequence
- Message recurrence
- Creation
- Destruction
- Transition times

Messages also support the three UML extensibility mechanisms—stereotype, constraints, and tagged values—described in [“Adding Extensibility To Symbols” on page 6-24](#).

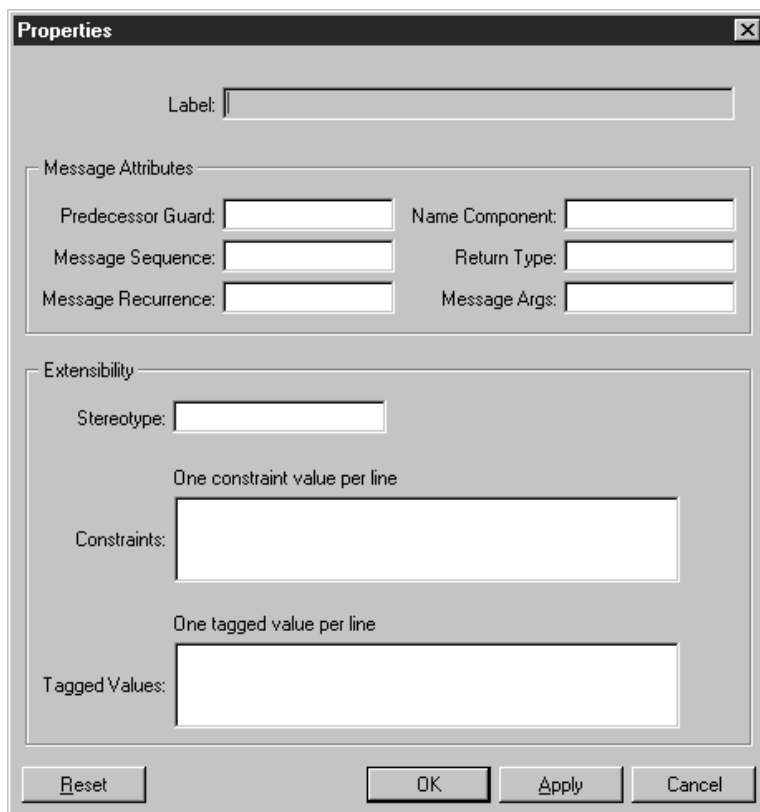
Several details about objects have display marks associated with them. For information about controlling the behavior of display marks, see [“Using Display Marks” on page 6-8](#).

Most details are added through the Properties dialog box or the Object Annotation Editor (OAE) and are described in this section.

Using the Properties Dialog Box

The Properties dialog box enables you to change most characteristics of a selected message label and apply all the changes at once.

Figure 9: Properties Dialog Box for Messages



The image shows a 'Properties' dialog box with a title bar containing a close button. The dialog is divided into several sections. At the top is a 'Label:' text field. Below it is a 'Message Attributes' section containing six fields: 'Predecessor Guard:', 'Name Component:', 'Message Sequence:', 'Return Type:', 'Message Recurrence:', and 'Message Args:'. The bottom section is 'Extensibility', which includes a 'Stereotype:' field, a 'Constraints:' field with the instruction 'One constraint value per line' above it, and a 'Tagged Values:' field with the instruction 'One tagged value per line' above it. At the bottom of the dialog are four buttons: 'Reset', 'OK', 'Apply', and 'Cancel'.

To use the Properties dialog box for a message:

1. Select a message.
2. Use one of the following methods to open the Properties dialog box:
 - From the **Edit** menu, choose **Properties**.
 - From the shortcut menu, choose **Properties**The Properties dialog box for the symbol appears.
3. Change all of the desired settings for the selected message.
For details about each property, see [Table 6](#).
4. Click **OK** or **Apply**.
5. To show display marks on your diagram, choose **View > Refresh Display Marks**.

Summary of Message Properties

Table 6 is a summary of the various parts of the Properties dialog box for messages.

Table 6: Message Properties Summary

Group	Property	Description
	Name	The complete label of the selected message; a read-only field based on the name appearing in the sequence diagram.
Message Attributes	Predecessor Guard	A guard condition in which listed message sequences must be satisfied before the message is enabled.
	Name Component	The name of the message corresponding to an operation in a class.
	Message Sequence	A dot-separated list of sequence terms. Each term designates a level of nesting and has the syntax: integer name Examples include (in sequential order): 1.2.1 1.2.2 1.2.3a 1.2.3b
	Return Type	The value returned by the message. If no value is returned, this field is blank.
	Message Recurrence	The conditional or iterative execution of the message.
	Message Args	A list of arguments (actual parameters) separated by commas.
Extensibility	Stereotype	Extends the semantics of the class.
	Constraints	The constraints on the class, restricting the values that its attributes can have.
	Tagged Values	Explicit characteristics of the class.

For more information on labeling messages, including the complete syntax, see [“Drawing and Labeling Links between Objects” on page 6-14](#).

Setting Creation

Adding a creation annotation to a message signifies that an object is created.

To create an object based on an incoming message, you use the Object Annotation Editor (OAE):

1. Select the incoming message that will create the object.
2. Open the OAE using one of the following methods:
 - Choose **Edit > Object Annotation**
 - Choose **Object Annotation** from the shortcut menu
 - Use the accelerator Ctrl+a

For detailed information on using the OAE, refer to [Fundamentals of StP](#)

3. Open the folder for the selected message.
4. Open the Message Definition note and select the Creation item.
5. From the pull-down menu, choose **True**.
6. Choose **File > Save**.
7. To show display marks on your diagram, choose **View > Refresh Display Marks**.

The creation display mark is described in [“Using Display Marks” on page 6-8](#).

Setting Destruction

Adding a destruction annotation to a message signifies that an object is destroyed.

To destroy an object based on its outgoing message:

1. Select the outgoing message that will destroy the object.
2. Open the OAE using one of the following methods:
 - Choose **Edit > Object Annotation**

- Choose **Object Annotation** from the shortcut menu
- Use the accelerator Ctrl+a

For detailed information on using the OAE, refer to [Fundamentals of StP](#)

3. Open the folder for the selected message.
4. Open the Message Definition note and select the Destruction item.
5. From the pull-down menu, choose **True**.
6. Choose **File > Save**.
7. To show display marks on your diagram, choose **View > Refresh Display Marks**.

The destruction display mark is described in [“Using Display Marks” on page 6-8](#).

Setting Transition Times

You can designate start and end times on messages. These transition instances are useful in modeling real-time applications.

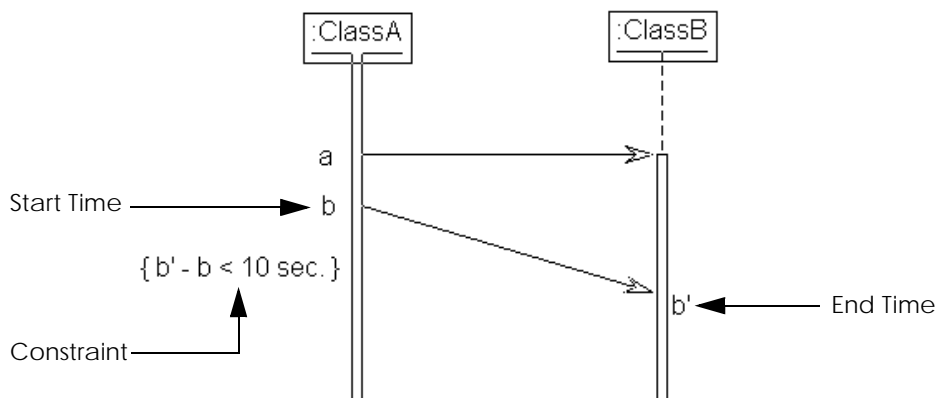
Constraints on time instances can be added to the diagram (these are separate from constraints added through the Properties dialog box, described in [“Adding Extensibility To Symbols” on page 6-24](#)).

Time constraints are not mapped to the repository and checking cannot be performed on them.

To place time transitions in a sequence diagram:

1. Drag a start time or end time symbol onto the drawing area and drop it next to the message.
The labeling area is selected.
2. Label the start time or end time.
If you are labeling an end time, include a prime sign (') at the end of the label (see [Figure 10](#)).
3. Reposition the label, if necessary.
If you move the object, the symbol also moves.

Figure 10: Transition Times and Constraints



To add a constraint to the transition times:

1. Drag a constraint symbol onto the drawing area.
2. Label the constraint.

Adding Extensibility To Symbols

You can add extensibility mechanisms—stereotypes, constraints, and tagged values—to the following Sequence Editor symbols:

- Package
- Actor
- Active Object
- Passive Object

To add extensibility mechanisms to a symbol:

1. Select the symbol.
2. Use one of the following methods to open the Properties dialog box:
 - From the **Edit** menu, choose **Properties**.
 - From the shortcut menu, choose **Properties**

The Properties dialog box for the symbol appears.

3. Add the desired extensibility mechanisms.
4. Click **OK** or **Apply**.
5. To show display marks on your diagram, choose **View > Refresh Display Marks**.

For information on adding stereotypes, constraints, and tagged values to sequence diagram messages, see [“Adding Details to Messages” on page 6-19](#).

For more information on stereotypes, constraints, and tagged values, see [“Adding Extensibility Mechanisms” on page 3-12](#).

Using Extension Points

An extension point is a text-only symbol that is a reference to the extend use case associated with the use case scenario. The extend use case is defined in a use case diagram and is the source of the «extends» link.

To add an extension point:

1. Insert an extension point symbol into the drawing area.
2. Label the extension point with the extension use case name by entering the name or choosing **Edit > List Labels**.
If you choose **List Labels**, the **List Labels** dialog box appears. Go to step 3.
3. Select the extend use case from the scrolling list.
4. Click **OK** or **Apply**.
The use case name appears in the extension point symbol.

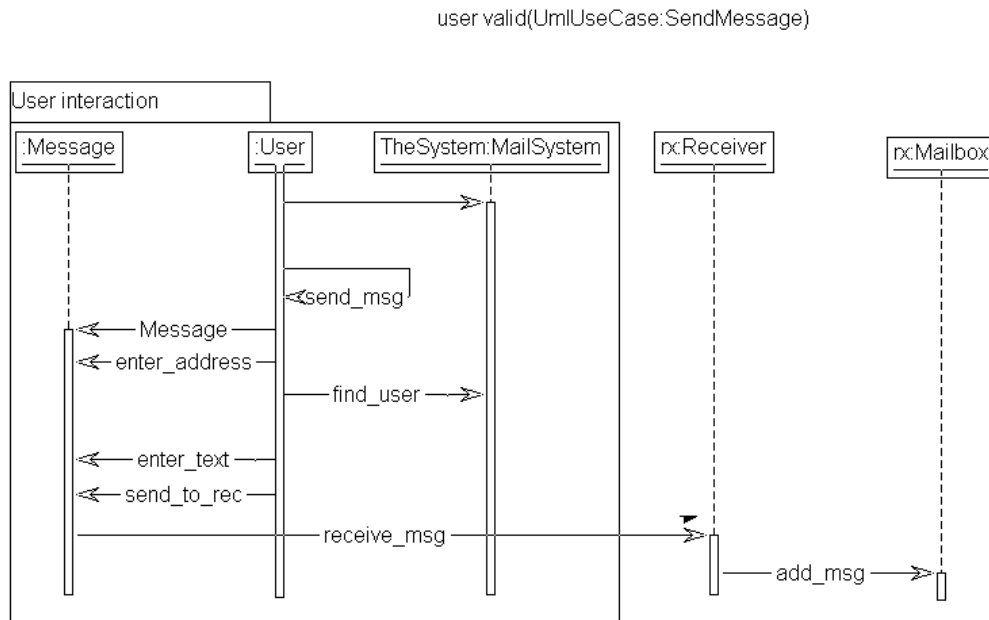
Using Packages

A package represents a collection of objects.

To add a package to a sequence diagram:

1. Insert a package symbol into the diagram.
2. Label the package.
3. Drag the boundaries of the package around the objects and packages that form the aggregate of the package.

Figure 11: Package in Sequence Diagram



Generating a Sequence Diagram from a Collaboration Diagram

You can create or replace a sequence diagram from an existing collaboration diagram. If you are replacing an existing sequence diagram, the entire diagram is overwritten by the new data from the collaboration diagram.

To create or replace a sequence diagram from a collaboration diagram:

1. Choose **UML > Generate Sequence from Collaboration**.
The Select Collaboration Diagram dialog box appears.
2. Select the collaboration diagram from the scrolling list.
3. Click **Apply** or **OK**.
The information from the collaboration diagram appears in the Sequence Editor.

Validating a Sequence Diagram

StP/UML provides two options for checking the correctness and consistency of a model from the Sequence Editor:

- Checking the current diagram
- Checking the repository for the current system

When you check your model, the Message Log appears listing the errors. For information about using the Message Log, see [Fundamentals of StP](#).

Checking a Diagram

You can check a diagram to validate that all objects and messages have labels. Checking the diagram does not check the contents of the repository.

To check a diagram, choose **Tools > Check Syntax**.

Checking the StP Repository

You can check the StP repository for a system to validate that objects are properly defined and correct. Checks of the StP repository include:

- Do all objects come from an existing class (that is, have references in class diagrams)?
- Are all messages represented as operations in a class?
- Does each actor appear in at least one use case diagram?
- Does each external event in a scenario have a corresponding communicates link in a use case diagram?
- Does each extension point in a scenario have a corresponding extend use case in a use case diagram?

To check the StP repository, choose **Tools > Check Semantics**.

7

Creating a Collaboration Diagram

This chapter describes how to create and use collaboration diagrams. Topics covered are as follows:

- [“What Are Collaboration Diagrams?” on page 7-1](#)
- [“Using the Collaboration Editor” on page 7-2](#)
- [“Drawing Collaboration Diagrams” on page 7-8](#)
- [“Creating Message Links” on page 7-15](#)
- [“Adding Extensibility to Symbols” on page 7-18](#)
- [“Using Extension Points” on page 7-19](#)
- [“Using Packages” on page 7-19](#)
- [“Generating a Collaboration Diagram from a Sequence Diagram” on page 7-20](#)
- [“Validating a Collaboration Diagram” on page 7-21](#)

This chapter provides brief descriptions of the elements of collaboration diagrams.

What Are Collaboration Diagrams?

Collaboration diagrams complement class diagrams in the logical-static view of a system. A collaboration diagram shows objects and their relationships in terms of passing messages.

A sequence diagram shows essentially the same information as a collaboration diagram, but with an emphasis on the order of message passing.

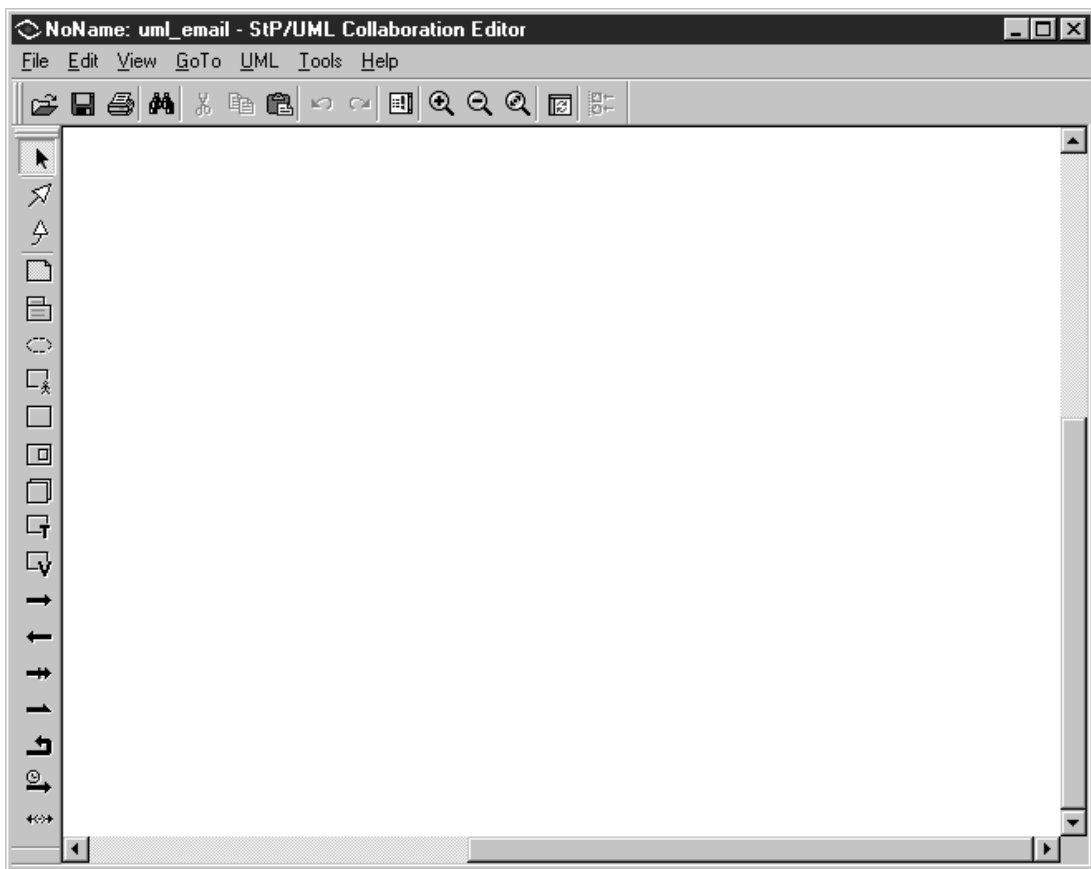
Together, sequence and collaboration diagrams are called interaction diagrams because they show the behavior (interaction) between objects.

For information on sequence diagrams, see [Chapter 6, “Creating a Sequence Diagram.”](#)

Using the Collaboration Editor

You create a collaboration diagram using the Collaboration Editor.

Figure 1: StP/UML Collaboration Editor



The StP/UML Collaboration Editor provides all the functions and menu options of every StP diagram editor. For general information about using StP diagram editors, see [Fundamentals of StP](#).

Starting the Collaboration Editor

StP/UML provides access to the Collaboration Editor from:

- The StP Desktop
- Use Case Editor
- Sequence Editor
- Activity Editor
- Stereotype Editor

For information about starting an editor from the StP Desktop, see [Chapter 2, “Using the StP Desktop.”](#)

For more information about starting (navigating to) the Collaboration Editor from another StP/UML editor, see the chapter in this manual that discusses the editor.

Navigating to Object References

“Navigation” is the use of the inherent relationships between repository objects and references in diagrams or tables to select an object and display another reference to it. This reference can be within the same diagram, a different collaboration diagram, or another StP/UML editor. When you navigate to another editor, the current session continues; in some cases, a predefined symbol appears in the editor.

You can navigate to these destinations from the Collaboration Editor:

- Use Case Editor
- Class Editor
- Class Table Editor
- Sequence Editor
- State Editor
- Activity Editor
- Requirements Table Editor

The **GoTo** Menu provides context-sensitive navigation choices for the selected symbol.

To navigate to a target:

1. Select a symbol.
2. Choose a command from the **GoTo** menu.

The navigation target appears.

Table 1 provides a summary of navigations for the Collaboration Editor.

Table 1: Collaboration Editor Navigations

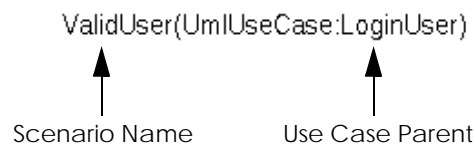
Navigate From	Navigate To	GoTo Menu Command
Object, Composite object, Active object, Active composite object, MultiObject	Class diagram with a reference to the class from which the object is instantiated	Class Diagram Where Class Is Referenced
	Class table with a reference to the class from which the object is instantiated	Class Table Where Class Is Referenced
	State diagram with a reference to the class from which the object is instantiated	State Diagram Where Class Is Referenced
	Activity diagram with a reference to the class from which the object is instantiated	Activity Diagram Where Class Is Referenced
Actor	Another scenario involving the same actor	Scenario Involving Actor
	The parent use case diagram	Use Case Diagram Involving Actor
Scenario name	Use case parent	Parent Use Case
	Another sequence or collaboration diagram scenario with the same use case parent	Sibling Use Case Scenario

Table 1: Collaboration Editor Navigations (Continued)

Navigate From	Navigate To	GoTo Menu Command
Diagram	Corresponding sequence diagram—if it does not exist, creates one	Sequence Diagram
Collaboration, Collaboration link	Go to the context view of the collaboration	Collaboration Context
Collaboration context	Collaboration parent	Parent Collaboration
	Another collaboration diagram scenario with the same collaboration parent	Sibling Collaboration Scenario
Type	Class diagram if the stereotype is defined	Class Diagram
	State diagram describing the semantics of the type or operation represented by the collaboration semantics	State Diagram
Any symbol	Requirements table with the object id for the object loaded	Allocate Requirements

Creating a Scenario from a Use Case Diagram

A scenario is created when you navigate from a use case diagram to a sequence or collaboration diagram. The scenario name, along with the use case parent, appears in the sequence or collaboration diagram, as shown in Figure 2.

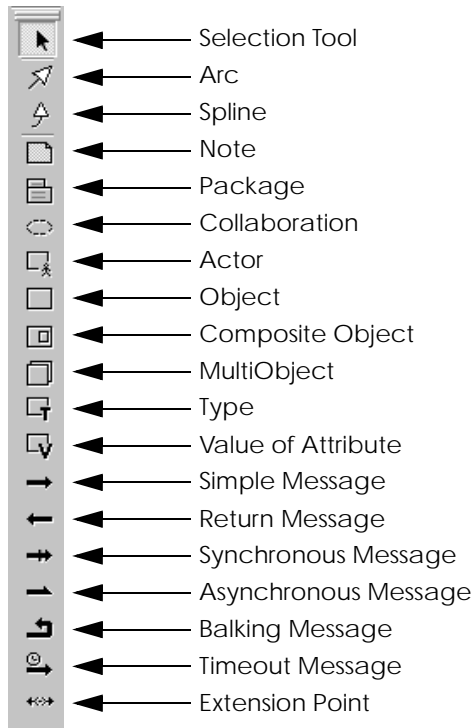
Figure 2: Scenario Name

For information on the Use Case Editor, see [Chapter 3, “Creating a Use Case Diagram.”](#)

Using the Collaboration Editor Symbol Toolbar

You select a symbol using the Symbol Toolbar.

Figure 3: Collaboration Editor Symbol Toolbar



Procedures for using all symbols on the Symbol Toolbar are described in this chapter.

Using the UML Menu

In addition to the standard diagram menu options, the Collaboration Editor also provides the UML Menu. This menu lists commands specific to collaboration diagrams.

[Table 2](#) describes the commands available from the UML menu.

Table 2: UML Menu Commands

Command	Description	For Details, See
Generate Collaboration from Sequence	Generates a collaboration diagram from a selected sequence diagram.	“Generating a Collaboration Diagram from a Sequence Diagram” on page 7-20
Reverse Direction	Switches the direction of message passing between objects.	“Reversing Message Direction” on page 7-17
Make Object Active/Passive	Makes the selected object active or passive. The command toggles based on the object’s status.	“Making Objects Active or Passive” on page 7-13
Set Message Name	Sets the name of the message.	“Drawing Links and Message Passing” on page 7-16

In addition, the **Edit > Properties** command provides a dialog box for adding details to collaboration diagram objects. For more information, see [“Adding Extensibility to Symbols” on page 7-18](#).

Using Display Marks

Several object annotations cause display marks to appear in the diagram. For example, there is a display mark for a stereotype of an object.

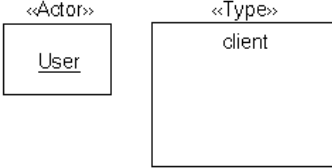
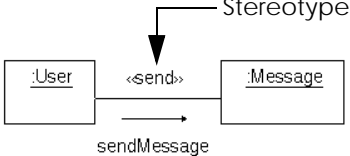
You control the behavior of display marks using the **Display Marks** tab in the **Options** dialog box. To display this tab:

1. From the **Tools** menu, select **Options**.
2. In the **Options** dialog box, choose the **Display Marks** tab.

For details about the Display Mark Options dialog, see [Fundamentals of StP](#).

[Table 3](#) lists the collaboration diagram display marks. The names include any associated item names within parentheses.

Table 3: UML Display Marks

Name	Display Mark	Description
CollaborationInfo (UmlStereotype)		<p>The CollaborationInfo display mark:</p> <ul style="list-style-type: none"> - Underlines the object or actor name - Adds the «type» stereotype to a type - Adds the «actor» stereotype to an actor <p>For details, see “Drawing Collaboration Diagrams” on page 7-8.</p>
Stereotype, StereotypeLink, Stereotype Message (UmlStereotype)		<p>The stereotype of the model element.</p> <p>For details, see “Adding Extensibility to Symbols” on page 7-18.</p>

Drawing Collaboration Diagrams

Collaboration diagrams are similar to sequence diagrams—both show the messages that pass between objects. Sequence diagrams emphasize the order of message passing; collaboration diagrams emphasize patterns, message types, concurrency, visibility, and return values.

In addition to drawing a collaboration diagram, you can also create one automatically from an existing sequence diagram. For information, see [“Generating a Collaboration Diagram from a Sequence Diagram” on page 7-20](#).

You create collaboration diagrams using the following symbols:

- Package—A collection of objects.
- Collaboration—A design pattern.
- Type—A set of instances that share operations, abstract attributes, abstract relationships, and semantics.

- Actor—An outside entity that interacts with the system.
- Object—An instance of a class.
- Composite Object—An instance of a class that is in a composite aggregation relationship.
- MultiObject—A set of objects that are at a “many” end of an association relationship

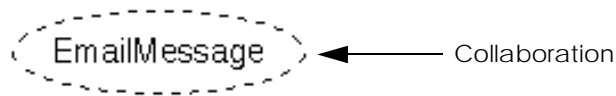
Representing and Labeling Collaborations

A collaboration describes a set of interactions (or patterns) between objects. The context of the collaboration describes the static structure of the objects including attributes and relationships.

To represent patterns using the collaboration symbol:

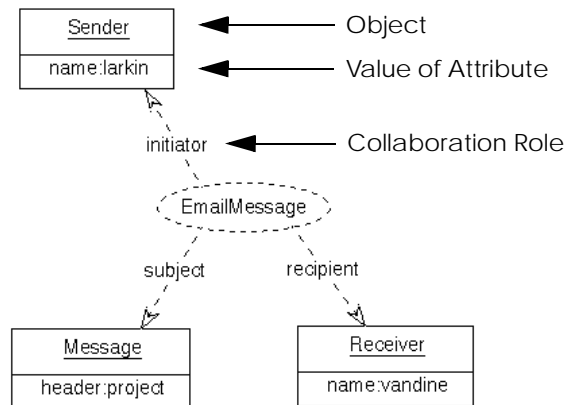
1. Insert a collaboration symbol onto the diagram.
2. Label the collaboration with the name of the pattern.

Figure 4: Collaboration



3. Insert object symbols into the diagram.
4. Label the objects.
For information on labeling object names, see [“Representing and Labeling Objects” on page 7-11](#).
5. If required, insert a Value of Attribute symbol onto the object symbol and label the value.
For information on labeling attribute values, see [“Representing and Labeling an Object Instance” on page 4-30](#).
6. Construct links from the collaboration to the object symbols.
7. Label the links.
Each link can be labeled with the role of the participant.
8. Save the diagram.

Figure 5: Collaboration Pattern



After creating the collaboration pattern, you create a collaboration scenario and elaborate on the message passing.

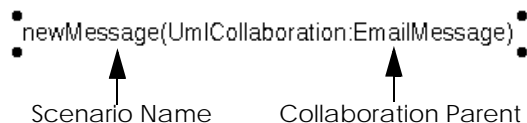
Expanding a Collaboration

To create a scenario with the collaboration symbol name as its context:

1. Select the collaboration or the collaboration role.
2. Choose **GoTo > Collaboration Context**.
3. In the Create New Scenario dialog box, enter a scenario name.
4. Click **OK**.

A new collaboration diagram appears with the collaboration context flashing.

Figure 6: Collaboration Context



To return to the diagram containing the parent collaboration symbol, select the collaboration context symbol and choose **GoTo > Parent Collaboration**.

Representing and Labeling Objects

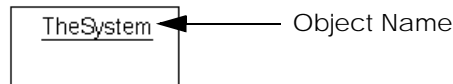
The interactions of a collaboration are represented by objects and the messages passed between them. An object in a collaboration diagram corresponds to an object in a sequence diagram.

To create a collaboration diagram showing objects and their message passing:

1. Insert an object symbol into the drawing area.
2. With the object selected, label the object symbol.

If the CollaborationInfo display mark is set to display, the object label is automatically underlined.

Figure 7: Labeled Object Symbol



Setting Class Names

In an object, the object label reflects the relationship between an object and its class. The label syntax for an object is:

[Object Name][:Class Name]

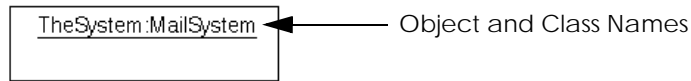
An object can:

- Have a fully qualified name, as in *The System:MailSystem*
- Be an anonymous object of a class, as in *:User*
- Be an object of an anonymous class, as in *Colliterator* (in this case, no checking against the class diagram is possible)

To add an object's class name to the object label:

1. Select the object.
2. Label the object's class as described above.

Figure 8: Object with Class Name



If two or more fully qualified objects with the same name appear in the same diagram or diagrams, they are the same object.

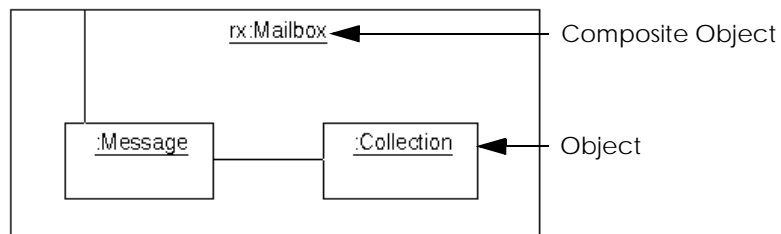
Representing Composite Objects

A composite object is an instance of a composite class and contains other objects participating in the composite aggregation relationship.

To draw a composite object:

1. Insert a composite object into the drawing area.
2. Label the composite object.
The class name must match a composite class that participates in a composite aggregation relationship.
3. Insert object symbols into the composite object symbol (resize the composite object, if necessary).
The objects represent instances of the class parts participating in the composite aggregation relationship.
4. Label the objects.
5. Construct links between the object and composite object symbols.

Figure 9: Composite Object



Making Objects Active or Passive

Objects and composite objects can be either active or passive. An active object owns a thread of control and may initiate control activity.

To designate an object or composite object as active:

1. Select the object or composite object.
2. Choose **UML > Make Object Active**.

The symbol border changes from a thin line to a thick line.

Figure 10: Active Object



After an object or composite object becomes active, the command toggles to **Make Object Passive**. To change an active object to passive, repeat the steps using the new command.

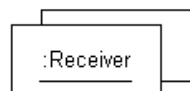
Representing a MultiObject

A MultiObject represents a set of objects that are at the “many” end of an association relationship. MultiObjects show messages that affect an entire set of objects, rather than just a single object.

To add a MultiObject to a collaboration diagram:

1. Insert a MultiObject symbol into the diagram.
2. Label the MultiObject.
3. Connect the MultiObject to an object.

Figure 11: MultiObject in Collaboration Diagram



For information on setting a “many” end of an association, see [“Setting Association Multiplicity” on page 4-42](#).

Representing an Actor

An actor represents an outside entity, such as a human, machine, computer task, or another system, that interacts with the system.

Actors are used in collaboration diagrams, as well as use case and sequence diagrams, to show external behavior. The actor invokes the interaction of an object.

To add an actor to a collaboration diagram:

1. Insert an actor symbol into the diagram.
2. Label the actor.
3. Connect the actor to an object.

Figure 12: Actor in Collaboration Diagram



Creating an Object Diagram

UML supports the notion of object diagrams. Object diagrams represent a static instance of a collaboration diagram. To draw an object diagram in the Collaboration Editor, place object symbols in your diagram. Whereas a collaboration diagram includes messages, an object diagram contains no messages.

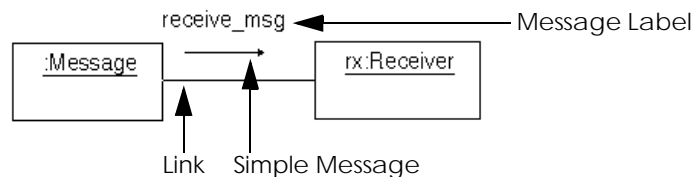
An object diagram can also be created using the Class Editor. For information, see [“Creating an Object Diagram” on page 4-31](#).

Creating Message Links

Links between objects are instances of relationships between the object's classes. For example, the relationship between *MailSystem* and *User* (described in [Chapter 4, "Creating a Class Diagram"](#)) appears in the object diagram as a link.

Objects pass messages to other objects through links. In message passing, one object requests a service (client object) and the other object supplies a service (supplier). For example, *User* sends a simple message to the *System* to be the receiver of an email message (see Figure 13).

Figure 13: Simple Message Passing









Types of Message Synchronization

The different behaviors of clients and suppliers can be differentiated by the type of message synchronization used in passing:

- Simple
- Synchronous
- Balking
- Timeout
- Asynchronous
- Return

[Table 4](#) contains the descriptions and symbols used for message synchronization.

Table 4: Message Synchronization

Type	Description	Symbol
Simple	There is a single thread of control.	
Synchronous	The client object waits for the supplier object to accept the message.	
Balking	If the supplier object cannot immediately service the message, the client object discards the message.	
Timeout	If the supplier object cannot service the message within a given amount of time, the client object discards the message.	
Asynchronous	The client object does not wait after sending the message; the supplier object queues the message for processing.	
Return	The supplier object returns a message to the client object.	

Drawing Links and Message Passing

Message names, in both collaboration and sequence diagrams, correspond to operations in classes.

Although you can type message names directly on links, StP/UML enables you to choose message names from the available operation names in the repository using the **Set Message Name** dialog box.

To draw and label a link between objects:

1. Insert a connection from one object to another object.
2. Select the link.

3. Choose **UML > Set Message Name**.

The **Set Message Name** dialog box appears.

For detailed information on the **Set Message Name** dialog box, refer to [“Drawing and Labeling Links between Objects” on page 6-14](#)

4. Choose the settings.
5. Click **OK** or **Apply**.

The message name appears in the diagram. The full name of the operation is not copied to the message. The parameters, return type, and superclass (if any) are stripped out.

For information on typing the message name directly on a message, see [“Labeling a Message Directly” on page 6-15](#).

You add details to a message using the Properties dialog box. For more information, see [“Using the Properties Dialog Box” on page 6-19](#).

Reversing Message Direction

To reverse the direction of a message:

1. Select the message.
2. Choose **UML > Reverse Direction**.

The keyboard accelerator for this command is Alt+Ctrl+t.

Adding Details to Messages

StP/UML provides features for adding details to messages, such as:

- Return type
- Message arguments
- Predecessor guard
- Message sequence
- Message recurrence
- Creation
- Destruction

- Transition times

Messages also support the three UML extensibility mechanisms: stereotype, constraints, and tagged values.

Most details are added through the Properties dialog box or the Object Annotation Editor (OAE).

For descriptions and procedures for adding details to messages, see the Sequence Editor section, [“Adding Details to Messages” on page 6-19](#).

Adding Extensibility to Symbols

You can add extensibility mechanisms—stereotypes, constraints, and tagged values—to the following Collaboration Editor symbols:

- Package
- Collaboration
- Collaboration link
- Object
- Composite Object
- Actor
- Message

To add extensibility mechanisms to a symbol:

1. Select the symbol.
2. Use one of the following methods to open the Properties dialog box:
 - From the **Edit** menu, choose **Properties**.
 - From the shortcut menu, choose **Properties**The **Properties** dialog box for the symbol appears.
3. Add the desired extensibility mechanisms.
4. Click **OK** or **Apply**.
5. To show display marks on your diagram, choose **View > Refresh Display Marks**.

For more detailed information on stereotypes, constraints, and tagged values, see [“Adding Extensibility Mechanisms” on page 3-12.](#)

Using Extension Points

An extension point is a text-only symbol that is a reference to the extend use case associated with the use case scenario. The extend use case is defined in a use case diagram and is the source of the «extends» link.

To add an extension point:

1. Insert an extension point symbol into the drawing area.
2. Label the extension point with the extension use case name by entering the name or choosing **Edit > List Labels**.
If you choose List Labels, the Name Completion dialog box appears. Go to step 3.
3. Select the extend use case from the scrolling list.
4. Click **OK** or **Apply**.
The use case name appears in the extension point symbol.

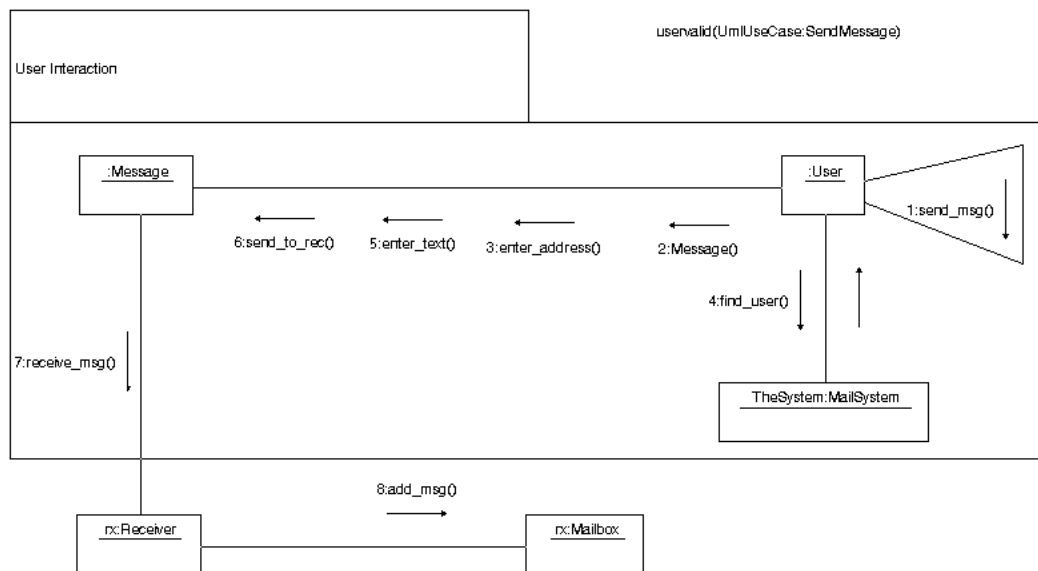
Using Packages

A package represents a collection of objects.

To add a package to a collaboration diagram:

1. Insert a package symbol into the diagram.
2. Label the package.
3. Drag the boundaries of the package around the objects and packages that form the aggregate of the package.

Figure 14: Package in Collaboration Diagram



Generating a Collaboration Diagram from a Sequence Diagram

You can create or replace a collaboration diagram from an existing sequence diagram. If you are replacing an existing collaboration diagram, the entire diagram is overwritten by the new data from the sequence diagram.

To automatically create or replace a collaboration diagram from a sequence diagram:

1. Choose **UML > Generate Collaboration from Sequence**.
2. In the **Select Sequence Diagram** dialog box, select the sequence diagram from the scrolling list.
3. Click **Apply** or **OK**.

The information from the sequence diagram appears in the Collaboration Editor.

Validating a Collaboration Diagram

StP/UML provides two options for checking the correctness and consistency of an information model from the Collaboration Editor:

- Checking the current diagram
- Checking the repository for the current system

When you check your model, the Message Log appears listing the errors. For information about using the Message Log, see [Fundamentals of StP](#).

Checking a Diagram

You can check a diagram to validate that all objects and messages have labels. Checking the diagram does not check the contents of the repository.

To check a diagram, choose **Tools > Check Syntax**.

Checking the StP Repository

You can check the StP repository for a system to validate that objects are properly defined and correct. Checks of the StP repository include:

- Do all objects come from an existing class (that is, have references in class diagrams)?
- Are all messages represented also as operations?
- Does each actor appear in at least one use case diagram?
- Does each external event in a scenario have a corresponding communicates link in a use case diagram?
- Does each extension point in a scenario have a corresponding extend use case in a use case diagram?

To check the StP repository, choose **Tools > Check Semantics**.

8

Creating a State Diagram

This chapter describes how to create and use state diagrams. Topics covered are as follows:

- [“What Are State Diagrams?” on page 8-1](#)
- [“Using the State Editor” on page 8-2](#)
- [“State Machine” on page 8-8](#)
- [“States” on page 8-9](#)
- [“State Transition Links” on page 8-11](#)
- [“Decomposing a State” on page 8-14](#)
- [“Modeling Concurrent Activities” on page 8-17](#)
- [“Indicating State History” on page 8-20](#)
- [“Adding Extensibility Mechanisms” on page 8-21](#)
- [“Validating a State Diagram” on page 8-21](#)

This chapter includes brief descriptions of the notation and procedures for drawing state diagrams. For a complete description of state transitions, see *Unified Modeling Language User Guide* (Booch et al 1998) and *Unified Modeling Language Reference Manual* (Rumbaugh et al 1998).

What Are State Diagrams?

State diagrams, supported by state tables, show the behavior of a single class or operation in a UML model. For information about state tables, see [Chapter 9, “Defining States with the State Table Editor.”](#)

There is one state diagram for each major class in a class diagram.

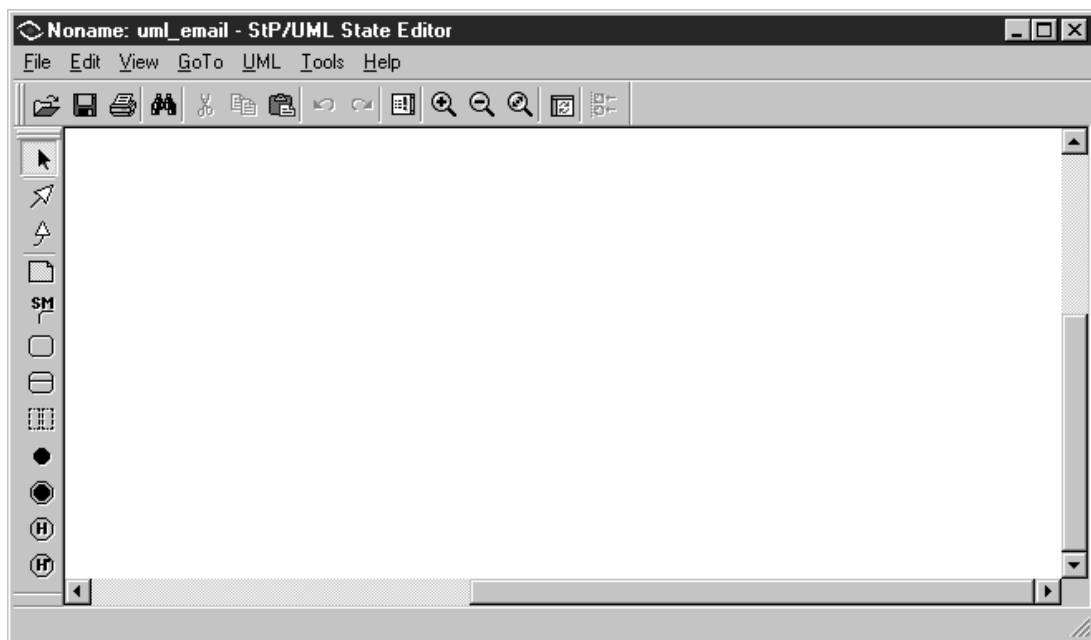
Using examples primarily from the electronic mail system, this chapter describes:

- Using the State Editor
- Validating a state diagram

Using the State Editor

You create a state diagram using the State Editor.

Figure 1: State Editor



The State Editor provides all the functions and menu options of every StP diagram editor. For general information about using StP diagram editors, see [Fundamentals of StP](#).

Starting the State Editor

StP/UML provides access to the State Editor from:

- The StP Desktop
- Class Editor
- Class Table Editor
- Collaboration Editor
- State Table Editor
- Stereotype Editor

For information about starting an editor from the StP Desktop, see [Chapter 2, “Using the StP Desktop.”](#)

For more information about starting (navigating to) the State Editor from another StP/UML editor, see the chapter in this manual that discusses the editor.

Navigating to Object References

“Navigation” is the use of the inherent relationships between repository objects and references in diagrams or tables to select an object and display another reference to it. This reference can be within the same diagram, a different state diagram, or another StP/UML editor. When you navigate to another editor, the current session continues; in some cases, a predefined symbol appears in the editor.

You can navigate to these destinations from the State Editor:

- Class Editor
- Class Table Editor
- Use Case Editor
- Sequence Editor
- Collaboration Editor
- Decomposed state diagram
- State Table Editor
- Requirements Table Editor

The **GoTo** Menu provides navigation context-sensitive commands for the selected symbol.

To navigate to a target:

1. Select a symbol.
2. Choose a command from the **GoTo** menu.

The navigation target appears.

Table 1 provides a summary of navigations for the State Editor.

Table 1: State Editor Navigations

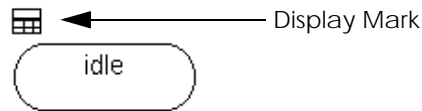
Navigate From	Navigate To	GoTo Menu Command
State Machine	Class table attached to state machine	Class Table for Class
	Class diagram attached to state machine	Class Diagram for Class
	Operation attached to the state machine	Operation
	Sequence diagram where class attached to state machine is used	Sequence Diagram Where Class Is Used
	Collaboration diagram where class attached to state machine is used	Collaboration Diagram Where Class Is Used
	A use case in a use case diagram that has the same name as the state machine	Use Case with Same Name
State, Concurrent Subregion	State diagram where selected state is the composite state	Refine
State, CompositeState, Concurrent Subregion	State table for state	State Table

Table 1: State Editor Navigations (Continued)

Navigate From	Navigate To	GoTo Menu Command
State (Parent)	State diagram in which the parent state is the state or composite state	Parent
Any symbol	Requirements table with the object id for the object loaded	Allocate Requirements

State Table Exists Display Mark

If a state or composite state has an associated state table, and the State Table Exists display mark is set for display, the display mark appears near the symbol, as shown in Figure 2.

Figure 2: State Table Exists Display Mark

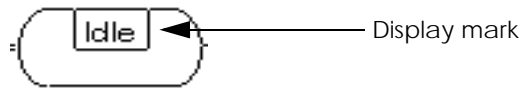
Refer to [“Using Display Marks” on page 8-7](#) for information on controlling the behavior of display marks.

To a Decomposed State Diagram

If you want to decompose a state, you can navigate to a lower-level state diagram that shows a substate. You can also navigate back to a top-level state using the scope parent. For more information, see [“Decomposing a State” on page 8-14](#).

If a state has an associated substate diagram, and the Refined State Exists display mark is set for display, a box appears around the state’s label in the higher-level diagram, as shown in [Figure 3](#).

Figure 3: Refined State Exists Display Mark

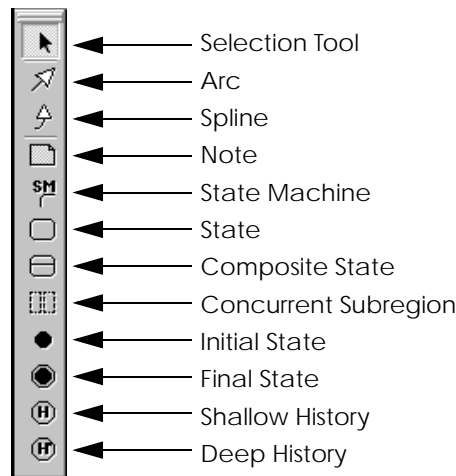


Refer to [“Using Display Marks” on page 8-7](#) for information on controlling the behavior of display marks.

Using the State Editor Symbol Toolbar

You select a symbol using the Symbol Toolbar.

Figure 4: State Diagram Symbol Toolbar



Procedures for using all the symbols on the Symbol Toolbar are described in this chapter.

Using the UML Menu

In addition to the standard diagram menu options, the State Editor also provides a UML Menu. This menu lists choices specific to state diagrams.

[Table 2](#) describes the commands available from the UML menu.

Table 2: UML Menu Commands

Command	Description	For Details, See
Split Control	Splits a single control flow into two or more flows.	“Splitting a Control Flow” on page 8-18
Merge Control	Merges two or more control flows into a single flow.	“Merging Control Flows” on page 8-19

In addition, the **Edit > Properties** command provides a dialog box for adding details to state diagram objects. For more information, see [“Adding Extensibility Mechanisms” on page 8-21](#).

Using Display Marks

Several state annotations can cause display marks to appear in the diagram. For example, if there is a state table for a state in the diagram, a display mark appears near the state symbol.

You control the behavior of display marks using the **Display Marks** tab in the **Options** dialog box. To display this tab:

1. From the **Tools** menu, select **Options**.
2. In the **Options** dialog box, choose the **Display Marks** tab.

For details about the Display Mark Options dialog box, see [Fundamentals of StP](#).

Table 3 lists state diagram display marks.

Table 3: UML Display Marks

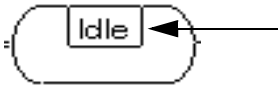
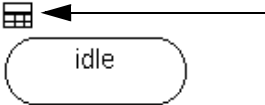
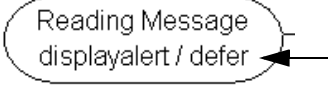

Name	Display Mark	Description
RefinedStateExists		A refined state diagram exists for the state. “Decomposing a State” on page 8-14

Table 3: UML Display Marks (Continued)

Name	Display Mark	Description
StateTableExists		A state table exists for the state. For details, see Chapter 9, “Defining States with the State Table Editor”
DeferredEvents		The state specifies a deferred event. For details, see Chapter 9, “Defining States with the State Table Editor”
Stereotype, StereotypeLink		A stereotype added to a symbol through the symbol's Properties dialog box. For details, see “Adding Extensibility Mechanisms” on page 8-21 .

State Machine

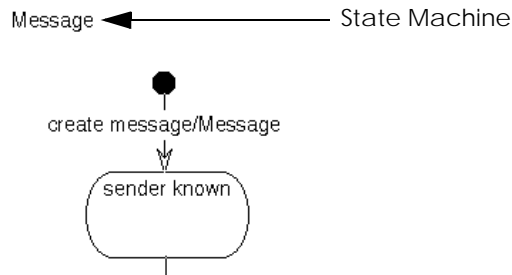
The context of a state diagram is displayed in a state machine symbol, which references a class or an operation. The state machine is represented by a text-only symbol, designating the class or operation which the state diagram belongs to.

When you create a state diagram from an existing class diagram, the state machine appears in the state diagram displaying the class or operation name.

To add a state machine to a state diagram:

1. Insert a state machine symbol into the diagram.
2. Label the state machine with a class or operation name.

Each state diagram must have a state machine. [Figure 5](#) shows a state machine, *Message*, which corresponds to a *Message* class in a class diagram.

Figure 5: State Machine

States

The fundamental symbol in a state diagram is the state. A state is:

- The condition of an object as determined by its attribute values and links
- A response of an object to an event
- The interval between two events received by an object

Where an event represents a point in time, a state represents a duration of time. A state is defined by grouping together all combinations of attribute values and links that have the same response to events.

Examples of states in the *Message* state transition include *user known*, *wait for enter body*, and *message in receiver mailbox*.

A state can be decomposed. For information on decomposing a state and using composite states, see [“Decomposing a State” on page 8-14](#).

Each state that is not part of a composite state is scoped to the state machine.

State Labels

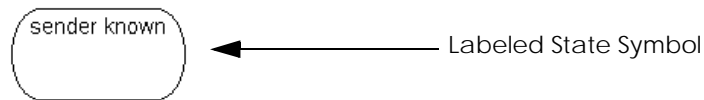
In a state diagram, a state is labeled with a short descriptive name. To provide a full definition of the state, use the State Table Editor (described in [Chapter 9, “Defining States with the State Table Editor”](#)).

Drawing and Labeling States

To insert a state in a diagram:

1. Insert a state symbol from the Symbol Toolbar.
2. Label the state symbol.

Figure 6: State Label



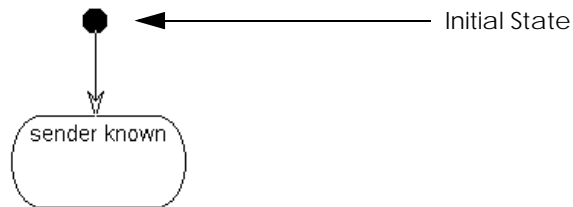
Drawing Initial States

An initial state indicates the creation of an object. To draw an initial state:

1. Insert an initial state symbol into the diagram.
2. Insert a connection from the initial state symbol to a state.

An arrow on the connection indicates a transition to the first state after the object is created.

Figure 7: Initial State



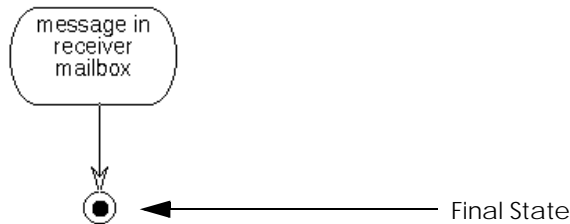
Drawing Final States

A final state indicates the ultimate condition of the state. To draw a final state:

1. Insert a final state symbol into the diagram.
2. Insert a connection from a state to the final state.

An arrow indicates the transition to the final state of the object.

Figure 8: Final State



State Transition Links

In a state diagram, events flow from state to state, causing a transition in the state with the incoming flow. Links between states are called state transition links. A state transition link can be split or merged to show concurrent activities (as described in [“Modeling Concurrent Activities” on page 8-17](#)).

The state transition link label is the name of the event causing the transition followed by the name of the action performed during the transition, with the two names separated by a slash (/). Guard conditions and send clauses may also be shown. A complete labeling notation for a transition is:

```
event_signature [guard_condition]/action_expression ^ send_clause
```

An automatic transition, which is a transition with no event, has no event label. If there is no activity, there is no label at all, unless a guard condition exists, in which case, the label is just the guard condition in brackets.

In the *Message* state transition in [Figure 9](#), there are four state transition links:

- The event between the initial state and the *sender known* state
- The event between the *sender known* and *body complete* states

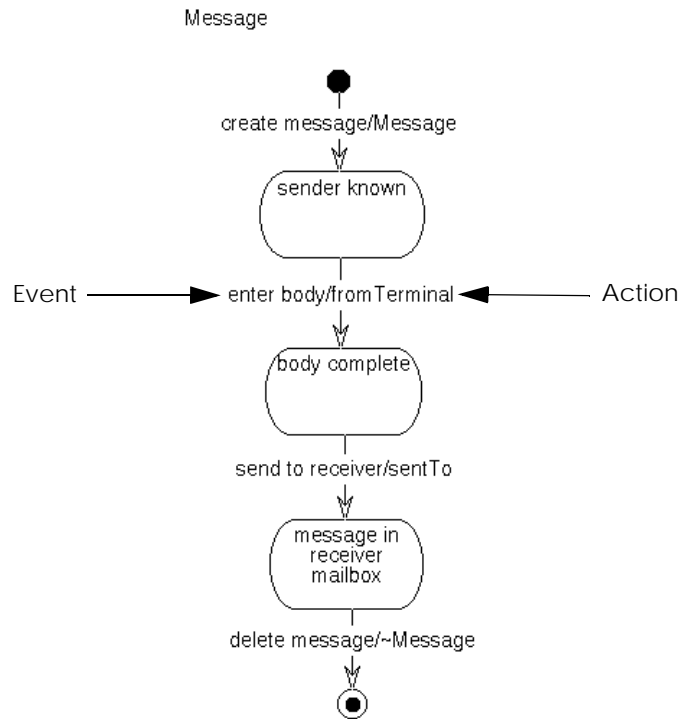
- The event between the *body complete* and *message in receiver mailbox* states
- The event between the *message in receiver mailbox* state and the final state

Drawing and Labeling State Transition Links

To draw and label a state transition link, insert a connection from one state to another. Then label each state transition link with its events and actions.

To draw and label a state transition link:

1. Insert a connection from one state to another.
2. Select the link.
Type in an appropriate event on the link's label.
3. Type a slash (/) after the event.
4. Type in the action.
[Figure 9](#) highlights an event and action for a state diagram link.

Figure 9: State Diagram Link

5. Choose **File > Save**.

To square or straighten the arcs, choose **Align All Links** from the shortcut menu (available by clicking the right mouse button inside the drawing area).

Capturing Other Information About States

An action is an operation that has no duration. An activity is an operation that happens over time; activities can be continuous or self-terminating. In StP/UML, you capture actions and activities in a state table.

1. Select a state in the diagram.
2. Choose **GoTo > State Table**.

The State Table Editor appears.

For information about the State Table Editor, see [Chapter 9, “Defining States with the State Table Editor.”](#)

Decomposing a State

Decomposing a state creates a lower-level state diagram where you can show more detail about the state. You can decompose a state in a state diagram to several levels, thereby creating a hierarchy of decompositions.

When you decompose a state, a composite state for the selected state appears to indicate the source of the decomposition. The corresponding state symbol in the parent diagram shows a box around the label, as in [Figure 3 on page 8-6](#).

Decomposing a State

To decompose a state:

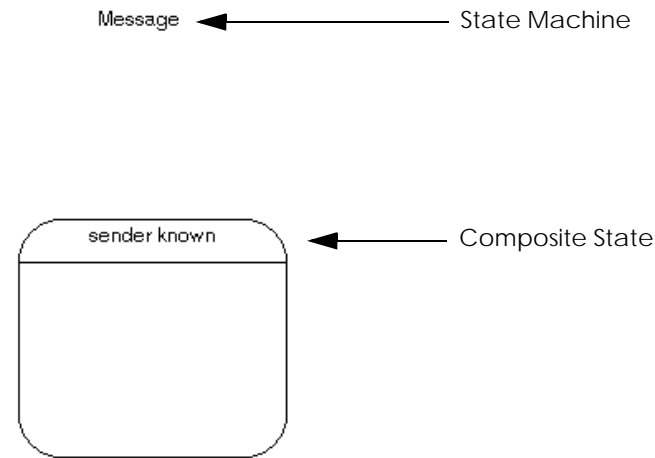
1. Select a state in the diagram.
2. Choose **GoTo > Refine**.

A confirmation box asking if you want to create the lower-level diagram appears.

3. Click **Yes**.

The lower-level diagram appears with the name of the state in the composite state symbol. The state machine from the top-level diagram appears in the decomposed diagram.

[Figure 10](#) shows a decomposed state, *sender known*, which is associated with the *Message* class.

Figure 10: Decomposed State

Create the state transition of the decomposed state within the new composite state symbol.

All states within a composite state are scoped to the composite state.

4. Choose **File > Save**.

Decomposing a State In a Composite State

To decompose a state from within a composite state:

1. Select a state that is inside a composite state symbol.
2. Choose **GoTo > Refine**.

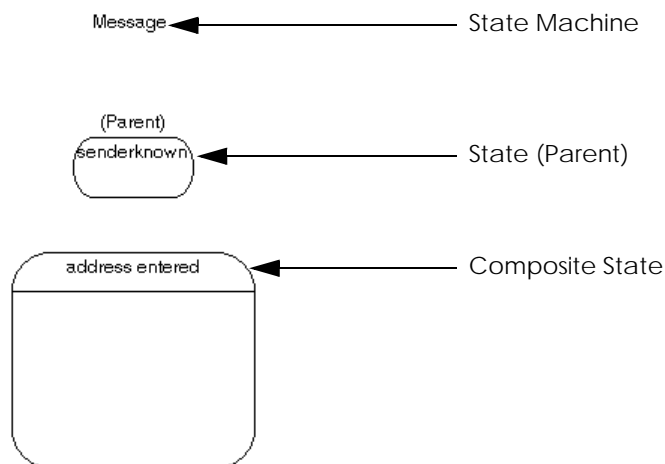
A confirmation box asking if you want to create the lower-level diagram appears.

3. Click **Yes**.

The lower-level diagram appears with the name of the state in the composite state symbol and the name of the parent composite state in the parent state symbol. The state machine from the top-level diagram also appears in the decomposed diagram.

[Figure 11](#) shows a decomposed state, *address entered*, which is a substate of the state, *sender known*.

Figure 11: Decomposed State with State Parent



Create the state transition of the decomposed state within the new composite state symbol.

4. Choose **File > Save**.

Navigating to the Parent State

You can navigate from the state (parent) symbol to a parent diagram containing a state or composite state with the same name.

To display the parent diagram:

1. Select the state (parent) symbol.
2. Choose **GoTo > Parent**.

If the state (parent) exists as a state or composite state in more than one state diagram, the Object Selector appears.

3. Select the diagram.
4. Click **OK** or **Apply**.

The diagram appears in the editor window.

Modeling Concurrent Activities

Concurrent activities are two or more sets of transitions that take place simultaneously within a composite state. The transitions may eventually join in a common state, or they may remain completely disconnected.

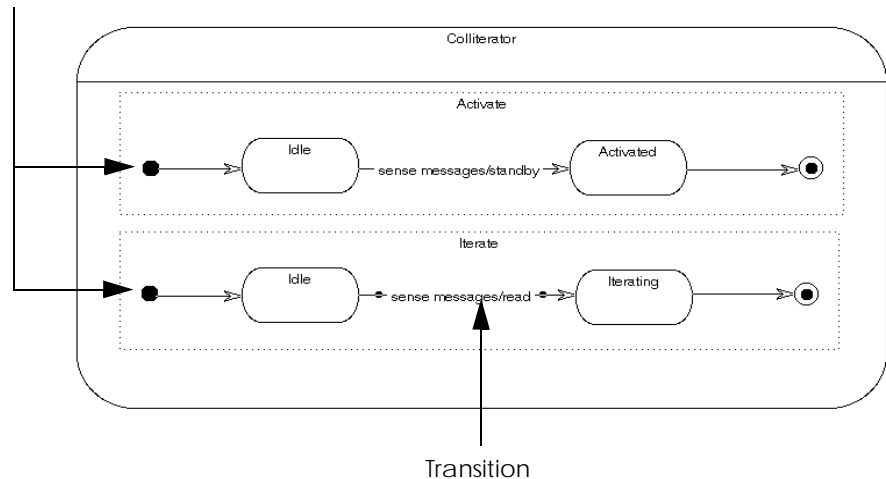
Before modeling concurrent activities, ensure that there is a composite state in the diagram.

To model concurrent activities:

1. Insert two concurrent subregion symbols into the composite state.
2. Resize the concurrent subregion symbols.
3. Model the transitions within each concurrent subregion symbol.
4. Label the concurrent subregion symbols.

Figure 12: Concurrent Activities

Concurrent Activities

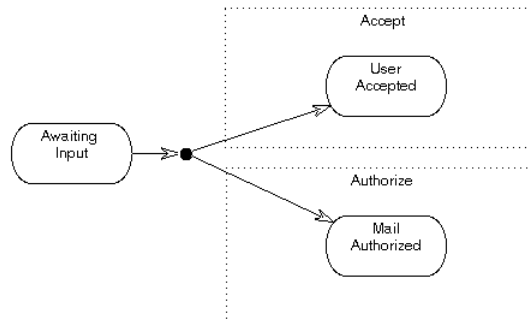


You can also model the concurrent transitions within the composite state before inserting the concurrent subregion symbols. In this case, resize the concurrent subregion symbol around the transition.

StP/UML enables you to model synchronization of concurrent states by splitting a single flow into two or more branches that terminate independently. The outgoing split flows can be merged back into a single flow.

Before splitting a flow, insert the concurrent states, and draw a link from the symbol with the outgoing flow to one of the concurrent states.

1. Select the flow.
2. With the flow selected, select the other concurrent state(s).
Press the Shift key to select multiple items.
3. Choose **UML > Split Control**.
The split flow appears.

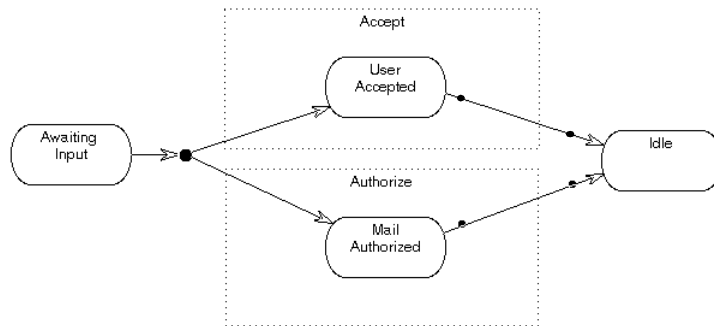


Merging Control Flows

Before merging flows, draw flows from each of the concurrent states to another state symbol.

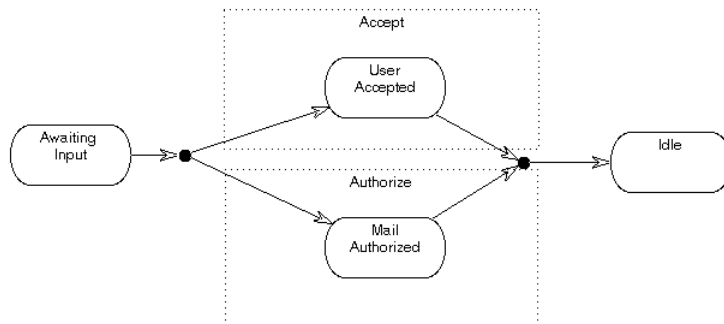
To merge the flows:

1. Select the flows.



2. Choose **UML > Merge Control**.

The selected flows are merged at a vertex in the outgoing flow.



Indicating State History

A history symbol is added to a state diagram to receive transitions from outside states. When a history symbol receives a transition, the object resumes the state it was last in.

Two types of history symbols are available in a state diagram:

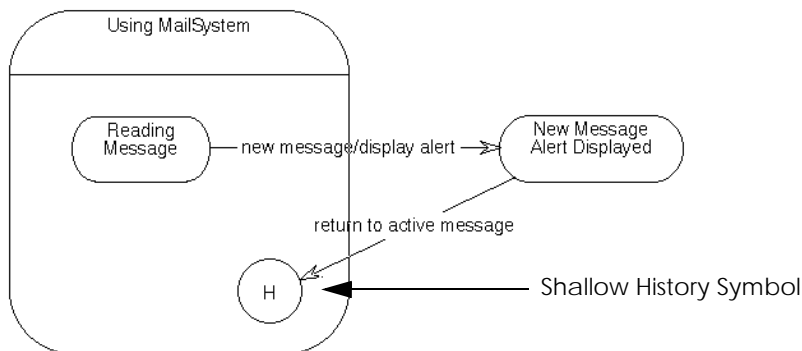
- **Shallow History**—Resumes the object's state at the same level as the Shallow History symbol.

Shallow History is represented by "H" surrounded by a circle.

- **Deep History**—Resumes the object's state at the appropriate level (but not limited to the same level as the Deep History symbol).

Deep History is represented by "H*" surrounded by a circle.

Figure 13: History Symbol



To indicate state history:

1. Insert a Shallow History or Deep History symbol into a state diagram.
2. Draw a state transition link from an outside state to the history symbol.

Adding Extensibility Mechanisms

You can add extensibility mechanisms—stereotypes, constraints, and tagged values—to most symbols in the State Editor.

To add extensibility mechanisms to a symbol:

1. Select the symbol.
2. Use one of the following methods to open the Properties dialog box:
 - From the **Edit** menu, choose **Properties**.
 - From the shortcut menu, choose **Properties**

The Properties dialog box for the symbol appears.

3. Add the desired extensibility mechanisms.
4. Click **OK** or **Apply**.
5. To show display marks on your diagram, choose **View > Refresh Display Marks**.

For more information on stereotypes, constraints, and tagged values, see [“Adding Extensibility Mechanisms” on page 3-12](#).

Validating a State Diagram

StP/UML provides two options for checking the correctness and consistency of a model from the State Editor:

- Checking the current diagram
- Checking the repository for the current system

When you check your model, the Message Log appears listing the errors. For information about using the Message Log, see [Fundamentals of StP](#).

Checking a Diagram

You can check a diagram to validate that the drawing is correct. Checking the diagram does not check the contents of the repository. Checks of a diagram include:

- Are the links legal for the symbol types?
- Are required links present?
- Are there too many of certain types of links in the diagram?
- Are there any vertices with only one in link or one out link?
- Are labels missing from the diagram?
- Are transitions deterministic, that is, are multiple transitions from a single state unambiguous?
- Are the destinations of split flows in concurrent states?
- Are the sources of merged flows in concurrent states?
- Is there one state machine?

To check a diagram, choose **Tools > Check Syntax**.

Checking the StP Repository

You can check the StP repository for a system to validate that objects are properly defined and correct. Checks of the repository include:

- Does the state machine correspond to a class or operation?
- Does the composite state have any activities (it should not)?

To check the StP repository, choose **Tools > Check Semantics**.

9

Defining States with the State Table Editor

This chapter describes how to define states by using the state table editor. Topics covered are as follows:

- [“What Is the State Table Editor?” on page 9-1](#)
- [“Using the State Table Editor” on page 9-2](#)
- [“Manipulating States” on page 9-7](#)
- [“Validating a State Table” on page 9-9](#)
- [“Deleting a State Table” on page 9-10](#)

What Is the State Table Editor?

State tables complement state transition diagrams in presenting a logical-dynamic view of a system. In StP/UML, you show the network of states and events on a state diagram. However, you capture the full definition of a state's events and actions using the State Table Editor.

The State Table Editor uses the same elements as the State Editor. For general information about these elements, see [Chapter 8, “Creating a State Diagram.”](#)

Using the State Table Editor

With the State Table Editor, you define initial states, final states, actions, internal activities, state variables, and deferred events for a state. Each state table describes one state.

Figure 1: The State Table Editor

The screenshot shows a window titled "NoName: uml_email - StP/UML State Editor". It has a menu bar (File, Edit, View, Table, GoTo, Tools, Help) and a toolbar with icons for file operations, editing, and state table manipulation. The main area is a table with 14 rows and 3 columns, numbered 1 to 14 in the first column. The table is used to define state components.

	1	2	3
1	State Machine		
2	State		
3	Entry Action		
4			
5	Activity		
6			
7	Exit Action		
8			
9	Internal Event	Internal Action	
10			
11	State Variable	Type	Default Value
12			
13	Deferred Events		
14			

At the bottom left, there is a "Scaling column width" button.

The State Table Editor provides all the functions and menu options of every StP table editor. For general information about using StP table editors, see [Fundamentals of StP](#).

Starting the State Table Editor

To create a state table, start the State Table Editor by navigating from a state symbol in a state transition diagram (described in [Chapter 8, “Creating a State Diagram”](#)). The state table corresponds to the state that is the source of the navigation.

You can also start the State Table Editor from the StP Desktop. For information about starting the State Table Editor from the StP Desktop, see [Chapter 2, “Using the StP Desktop.”](#)

Navigating to Object References

“Navigation” is the use of the inherent relationships between repository objects and references in diagrams or tables to select an object and display another reference to it. When you navigate to another editor, the current session continues; in some cases, a predefined symbol appears in the editor.

You can navigate to these destinations from the State Table Editor:

- State Editor
- Class Editor
- Class Table Editor
- Sequence Editor
- Collaboration Editor
- Requirements Table Editor

The **GoTo** Menu provides navigation context-sensitive commands for the selected symbol.

To navigate to a target:

1. Select a table cell.
2. Choose a command from the **GoTo** menu.

The navigation target appears.

[Table 1](#) provides a summary of navigations for the State Table Editor.

Table 1: State Table Editor Navigations

Navigate From	Navigate To	Navigation Command
State Machine	State diagram where the state machine appears	State Diagram
	Class table attached to state machine	Class Table for Class
	Class diagram attached to state machine	Class Diagram for Class
	Sequence diagram where class attached to state machine is used	Sequence Diagram Where Class Is Used
	Collaboration diagram where class attached to state machine is used	Collaboration Diagram Where Class Is Used
State	State diagram for the selected state	State Diagram
State Machine, State, Entry Action, Activity, Exit Action, Internal Event, State Variable, Deferred Events	Requirements table with the object id for the object loaded	Allocate Requirements

Navigating to the State Editor

To navigate to a state in a state diagram:

1. Select the state name or state machine name in the table.
If necessary, expand the table cells to the right one column to view the state name or state machine name.
2. Choose **GoTo > State Diagram**.

If the symbol appears in only one state diagram, it appears with the selected symbol flashing.

If the symbol appears in more than one state diagram, the Object Selector appears. Continue with step 3.

3. Select the object from the Object Selector list and click **OK**.

The state diagram appears with the selected symbol flashing.

Parts of the State Table Editor

This section describes the parts of the state table.

For all parts except the State Machine and State, you can add an unlimited number of entry actions using the **Table > Insert Rows Before** or **Table > Insert Rows After** commands (described in [Fundamentals of StP](#)).

State Machine

The State Machine section is read-only; it contains the name of the state's class or operation described in the table. If you start the table by navigating from a state in the State Editor, the state machine name is the same as the one in the state diagram.

This section corresponds to a state machine symbol on a state diagram.

State

The State section is read-only; it contains the name of the state described in the table. If you start the table by navigating from a state in the State Editor, the state name is the same as the one in the state diagram.

If the state is embedded in a composite state, or if there is a concurrent subregion, these names are also listed in the State section.

This section corresponds to a state symbol on a state diagram.

Entry Action

Each row of cells in the Entry Action section contains information that describes the initial action when control enters the state; an action is instantaneous.

This section corresponds to an initial state symbol on a state diagram.

Activity

Each row of cells in the Activity section contains information that describes an operation that happens while control is in the state; an activity has duration. An activity in a state is continuous within the state.

Exit Action

Each row of cells in the Exit Action section contains information that describes the final action when control leaves a state.

Internal Event

Each row of cells in the Internal Event section contains the event that triggers an internal action.

State Variables

Each row of cells in the State Variables section contains variables in the form of attributes. State variables are attributes of the state's class.

Deferred Events

Each row of cells in the Deferred Events section contains events that are deferred during the state.

Hiding and Showing Table Sections

You can hide from view or show each of the sections of the State Table Editor. To hide or show a table section, choose **View > Hide/Show**. For

information about using the Hide/Show dialog box, see [Fundamentals of StP](#).

Manipulating States

This section describes:

- Defining a state
- Sorting a state table

Defining a New State

You define a new state by drawing it in a state diagram (as described in [Chapter 8, “Creating a State Diagram”](#)), then navigating from the state in the diagram to the State Table Editor. The state name appears in the State section of the table. Using the State Table Editor, you can provide the information that describes a state.

To define a new state:

1. Ensure that the name of the state and state machine appear in the appropriate section.

Figure 2: State Machine and State Names

State Machine	Message
State	sender known

If not, quit the editor and restart it by navigation or the command line.

2. Type the state’s entry actions in the Entry Action section.
3. Type the state’s activities in the Activity section.
4. Type the state’s exit actions in the Exit Action section.
5. Type the state’s internal activities in the Internal Event and Internal Action sections.

6. Type the state's variables in the State Variable section. Add the variable's type and default value in the appropriate sections.
7. Type the deferred events in the Deferred Events section.

Figure 3 is an example of a completed state table.

Figure 3: Completed State Table

State Machine	Message	
State	sender known	
Entry Action		
receive message request		
Activity		
recognize sender		
Exit Action		
prepare for body		
Internal Event	Internal Action	
sender not recognized	send message "sender unknown"	
State Variable	Type	Default Value
body	string	" "
Deferred Events		
save copy		

8. Choose **Save** from the File menu.

Sorting a State Table

StP/UML provides sorting options for most writable rows in a state table. [Table 2](#) shows the sorting options available in the State Table Editor.

Table 2: State Table Sort Options

State Table Section	Sort Option
Entry Action	By Name
Activity	
Exit Action	
Internal Event	By Event, Action
	By Action, Event
State Variable, Deferred Events	Not available

To sort a state table:

1. Select the row number to be sorted.
2. Choose **Tools > Sort** and select a sort option.
The sorted rows appear in the table.

Validating a State Table

StP/UML provides options for checking the correctness and consistency of the current table's contents; checking the table does not check the contents of the repository.

To check a table, choose **Tools > Check Syntax**.

When you check your model, the Message Log appears listing the errors. For information about using the Message Log, see [Fundamentals of StP](#).

Deleting a State Table

To delete a state table:

1. From the Model pane on the StP Desktop, select the **Tables** category.
A list of subcategories for **Tables** appears.
2. Select **State**.
A list of the state tables in the system appears in the State pane.
3. Select the state to be deleted from the list.
4. Choose **File > Delete Table**.

This chapter describes how to create and use activity diagrams. Topics covered are as follows:

- [“What Are Activity Diagrams?” on page 10-1](#)
- [“Using the Activity Editor” on page 10-2](#)
- [“State Machine” on page 10-7](#)
- [“Action States” on page 10-7](#)
- [“State Transition Links” on page 10-9](#)
- [“Decisions” on page 10-10](#)
- [“Using Objects with Action States” on page 10-12](#)
- [“Modeling Synchronization” on page 10-13](#)
- [“Adding Extensibility Mechanisms” on page 10-15](#)
- [“Validating an Activity Diagram” on page 10-15](#)

This chapter provides brief descriptions of the elements of activity diagrams. For a complete description of activity diagrams and the UML, see *Unified Modeling Language User Guide* (Booch et al 1998) and *Unified Modeling Language Reference Manual* (Rumbaugh et al 1998).

What Are Activity Diagrams?

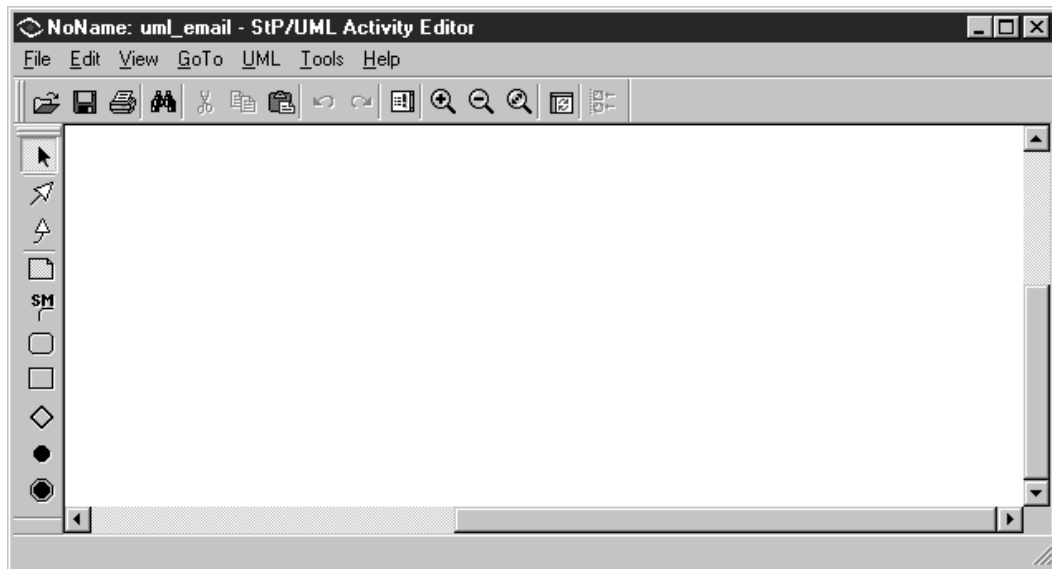
An activity diagram is a variation of a state diagram used for modeling the procedural flow of a task.

Using activity diagrams, you can model the steps required to perform an operation, a use case, or a behavior spanning several use cases. Activity diagrams present a logical-dynamic view of the system.

Using the Activity Editor

You create an activity diagram using the Activity Editor.

Figure 1: Activity Editor



The Activity Editor provides all the functions and menu options of every StP diagram editor. For general information about using StP diagram editors, see [Fundamentals of StP](#).

Starting the Activity Editor

StP/UML provides access to the Activity Editor from:

- The StP Desktop
- Class Editor
- Class Table Editor
- Sequence Editor
- Collaboration Editor
- Stereotype Editor

For information about starting an editor from the StP Desktop, see [Chapter 2, “Using the StP Desktop.”](#)

For more information about starting (navigating to) the Activity Editor from another StP/UML editor, see the chapter in this manual that discusses the editor.

Navigating to Object References

“Navigation” is the use of the inherent relationships between repository objects and references in diagrams or tables to select an object and display another reference to it. This reference can be within the same diagram, a different activity diagram, or another StP/UML editor. When you navigate to another editor, the current session continues; in some cases, a predefined symbol appears in the editor.

You can navigate to these destinations from the Activity Editor:

- Class Editor
- Class Table Editor
- Sequence Editor
- Collaboration Editor
- Use Case Editor
- Requirements Table Editor

The **GoTo** Menu provides navigation context-sensitive commands for the selected symbol.

To navigate to a target:

1. Select a symbol.
2. From the **GoTo** menu, choose a command.

The navigation target appears.

Table 1 provides a summary of navigations for the Activity Editor.

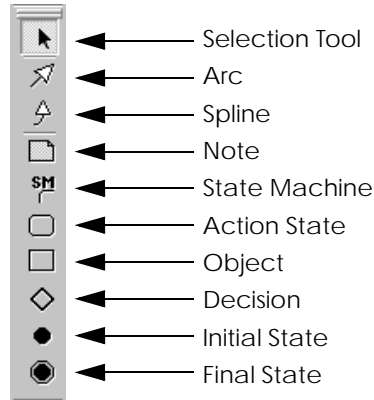
Table 1: Activity Editor Navigations

Navigate From	Navigate To	GoTo Menu Command
State Machine	Class table attached to the state machine	Class Table for Class
	Class diagram attached to state machine	Class Diagram for Class
	Operation attached to the state machine	Operation
	Sequence Diagram where class attached to state machine is used	Sequence Diagram Where Class Is Used
	Collaboration diagram where class attached to state machine is used	Collaboration Diagram Where Class Is Used
	A use case in a use case diagram that has the same name as the state machine	Use Case with Same Name
Any symbol	Requirements table with the object id for the object loaded	Allocate Requirements

Using the Activity Editor Symbol Toolbar

You select a symbol using the Symbol Toolbar.

Figure 2: Activity Diagram Symbol Toolbar



Procedures for using all the symbols on the Symbol Toolbar are described in this chapter.

Using the UML Menu

In addition to the standard diagram menu options, the Activity Editor also provides a UML Menu. This menu lists choices specific to activity diagrams.

Table 2 describes the commands available from the UML menu.

Table 2: UML Menu Commands

Command	Description	For Details, See
Split Control	Splits a single control flow into two or more flows.	“Splitting a Transition” on page 10-13
Merge Control	Merges two or more control flows into a single flow.	“Merging Control Flows” on page 10-14

In addition, the **Edit > Properties** command provides a dialog box for adding details to state diagram objects. For more information, see [“Adding Extensibility Mechanisms” on page 10-15](#).

Using Display Marks

Several activity annotations can cause display marks to appear in the diagram. For example, if there is a deferred event for a state, a display mark appears near the state symbol.



You control the behavior of display marks using the **Display Marks** tab in the **Options** dialog box. To display this tab:

1. From the **Tools** menu, select **Options**.
2. In the **Options** dialog box, choose the **Display Marks** tab.

For details about the Display Mark Options dialog box, see [Fundamentals of StP](#).

Table 3 lists activity diagram display marks.

Table 3: UML Display Marks

Name	Display Mark	Description
DeferredEvents		The state specifies a deferred event. For details, see Chapter 9, “Defining States with the State Table Editor”
Stereotype, StereotypeLink		A stereotype added to a symbol through the symbol's Properties dialog box. For details, see “Adding Extensibility Mechanisms” on page 10-15 .

State Machine

The context of an activity diagram is displayed in a state machine symbol, which references a class or an operation. The state machine is represented by a text-only symbol, designating the class or operation which the activity diagram belongs to.

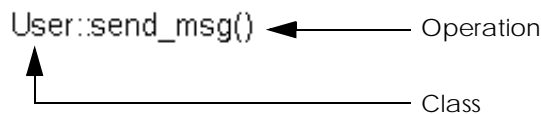
When you create an activity diagram from an existing class diagram, the state machine appears automatically in the activity diagram displaying the class or operation name.

State machine names for operations are qualified with the containing class name and are written in the format `class::operation()`.

Each activity diagram must have a state machine.

Figure 3 shows a state machine, `User::send_msg()`, which corresponds to the `send_msg` operation in the `User` class.

Figure 3: State Machine



To add a state machine to an activity diagram:

1. Insert a state machine symbol into the diagram.
2. Label the state machine.

Action States

The fundamental symbol in an activity diagram is the action state. An action state is identical to a state in a state diagram, with the following exception: an action state has at least one outgoing transition based on the completion of the internal event.

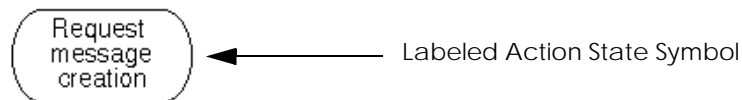
For more information on states, see [Chapter 8, “Creating a State Diagram.”](#)

Drawing and Labeling Action States

To insert an action state in a diagram:

1. Insert an action state symbol into the diagram.
2. Label the action state symbol.

Figure 4: Action State Label

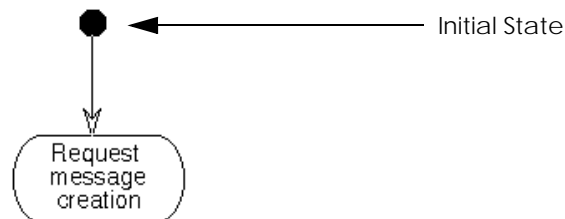


Drawing Initial States

An initial state indicates the creation of a task. To draw an initial state:

1. Insert an initial state symbol into the diagram.
2. Insert a connection from the initial state symbol to an action state.
An arrow on the connection indicates a transition to the first action state after the process is created.

Figure 5: Initial State

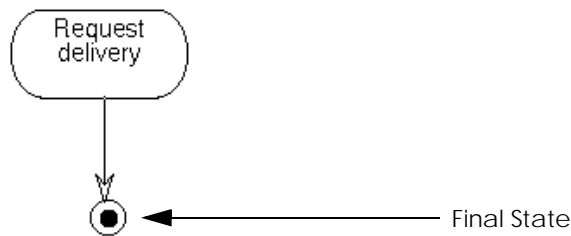


Drawing Final States

A final state indicates the ultimate condition of the state. To draw a final state:

1. Insert a final state symbol into the diagram.
2. Insert a connection from an action state to the final state.
An arrow indicates the transition to the final action state of the task.

Figure 6: Final State



State Transition Links

State transition links between action states show guard conditions and the action performed during the transition. A complete labeling notation for a transition is:

`[guard_condition]/action`

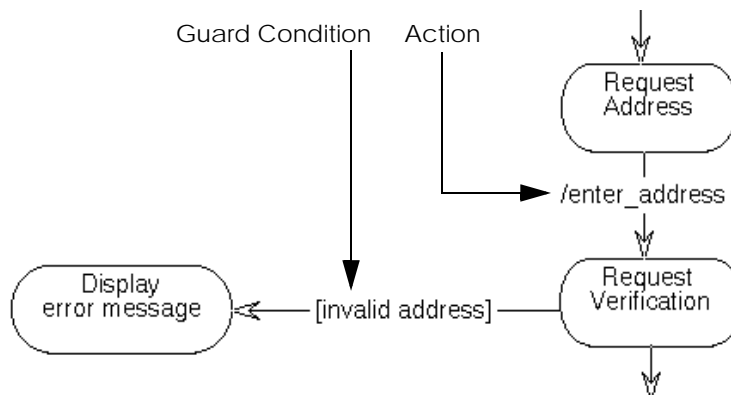
Drawing and Labeling State Transition Links

To draw and label a state transition link:

1. Insert a connection from one action state to another.
2. Select the link.
3. Type in the guard condition between brackets.
4. Type in the action. Precede the action with a forward slash (/).

Figure 7 highlights an activity diagram with a guard condition and action.

Figure 7: Activity Diagram with Guard Condition and Action



5. Choose **Save** from the File menu.

To square or straighten the arcs, choose **Tools > Align Links** or choose **Align Links** from the shortcut menu (available by clicking the right mouse button inside the drawing area).

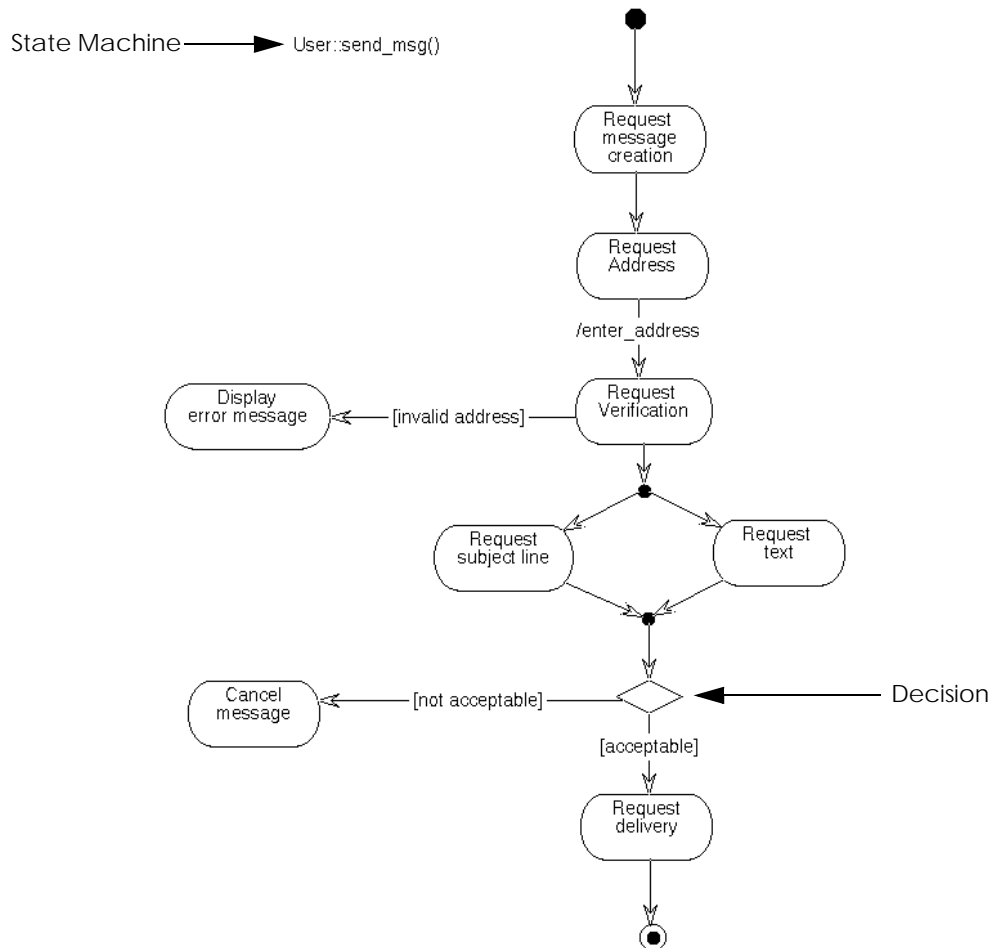
Decisions

Decisions indicate different possible transitions between action states. Decisions are represented by a diamond shape and have at least one incoming state transition link and at least two outgoing state transition links.

To add a decision to an activity diagram:

1. Insert a decision symbol into the diagram.
2. Insert a connection from at least one action state to the decision symbol.
3. Insert connections from the decision symbol to at least two action states.

Figure 8 shows an activity diagram with a decision.

Figure 8: Activity Diagram with Decision

To designate guard conditions for the different transitions, see [“State Transition Links” on page 10-9](#).

Using Objects with Action States

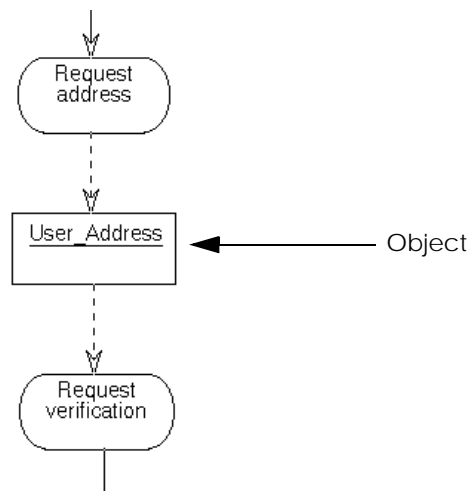
Objects interact with action states by receiving or providing values. To indicate a relationship between an object and an action state:

1. Insert an object symbol into the diagram.
2. Label the object diagram as described in [“Representing and Labeling Objects” on page 7-11](#) in [Chapter 7, “Creating a Collaboration Diagram.”](#)

3. If the action state provides values to the object, draw a link from the action state to the object. This is an output link.

If the object provides values to the action state, draw a link from the object to the action state. This is an input link.

Figure 9: Object Connected to Action States



Modeling Synchronization

StP/UML enables you to model synchronization of transitions by splitting a single flow into two or more branches. The branches can be merged back into a single flow.

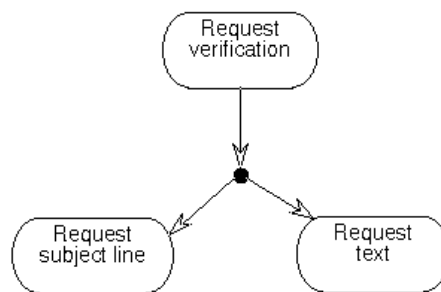
Splitting a Transition

Before splitting a flow, insert the action states, and draw a flow from the outgoing action state to one of the remaining action states.

To split the flow:

1. Select the flow.
2. With the flow selected, select the other action state(s).
Press the shift key to select multiple items.
3. Choose **UML > Split Control**.

The split flow appears at the synchronization vertex.

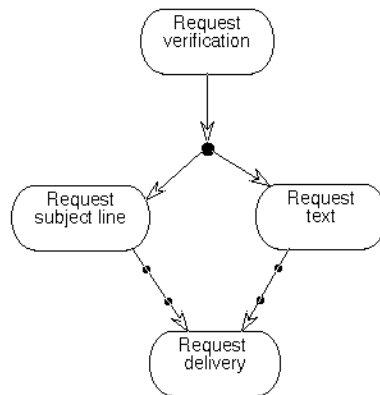


Merging Control Flows

Before merging flows, draw flows from each of the action states to another action state symbol.

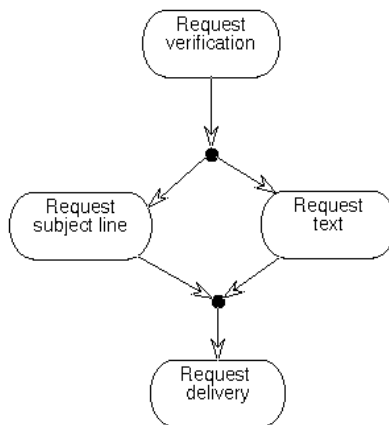
To merge the flows:

1. Select the flows.



2. Choose **UML > Merge Control**.

The two flows are merged at a synchronization vertex in the outgoing flow.



Adding Extensibility Mechanisms

You can add extensibility mechanisms—stereotypes, constraints, and tagged values—to most symbols in the Activity Editor.

To add extensibility mechanisms to a symbol:

1. Select the symbol.
2. Use one of the following methods to open the Properties dialog box:
 - From the **Edit** menu, choose **Properties**.
 - From the shortcut menu, choose **Properties**The Properties dialog box for the symbol appears.
3. Add the desired extensibility mechanisms.
4. Click **OK** or **Apply**.
5. To show display marks on your diagram, choose **View > Refresh Display Marks**.

For more information on stereotypes, constraints, and tagged values, see [“Adding Extensibility Mechanisms” on page 3-12](#).

Validating an Activity Diagram

StP/UML provides two options for checking the correctness and consistency of a model from the Activity Editor:

- Checking the current diagram
- Checking the repository for the current system

When you check your model, the Message Log appears listing the errors. For information about using the Message Log, see [Fundamentals of StP](#).

Checking a Diagram

You can check a diagram to validate that the drawing is correct. Checking the diagram does not check the contents of the repository. Checks of a diagram include:

- Are the links legal for the symbol types?
- Are required links present?
- Are there too many of certain types of links in the diagram?
- Are there any vertices with only one in link or one out link?
- Are labels missing from the diagram?
- Are transitions deterministic, that is, are multiple transitions from a single state unambiguous?
- Is there one state machine?

To check a diagram, choose **Tools > Check Syntax**.

Checking the StP Repository

You can check the StP repository for a system to validate that the state machine corresponds to a class or operation in the repository.

To check the StP repository, choose **Tools > Check Semantics**.

11 Creating a Component Diagram

This chapter describes how to create and use component diagrams. Topics covered are as follows:

- [“What Are Component Diagrams?” on page 11-1](#)
- [“Using the Component Editor” on page 11-2](#)
- [“Drawing Software Components” on page 11-5](#)
- [“Adding Extensibility Mechanisms” on page 11-10](#)
- [“Creating a Deployment Diagram” on page 11-10](#)
- [“Validating a Component Diagram” on page 11-11](#)

For more information on components, see *Unified Modeling Language User Guide* (Booch et al 1998) and *Unified Modeling Language Reference Manual* (Rumbaugh et al 1998).

What Are Component Diagrams?

Component diagrams show the allocation of classes and objects to software components in a system. A component consists of either procedures, data definitions and declarations, or both.

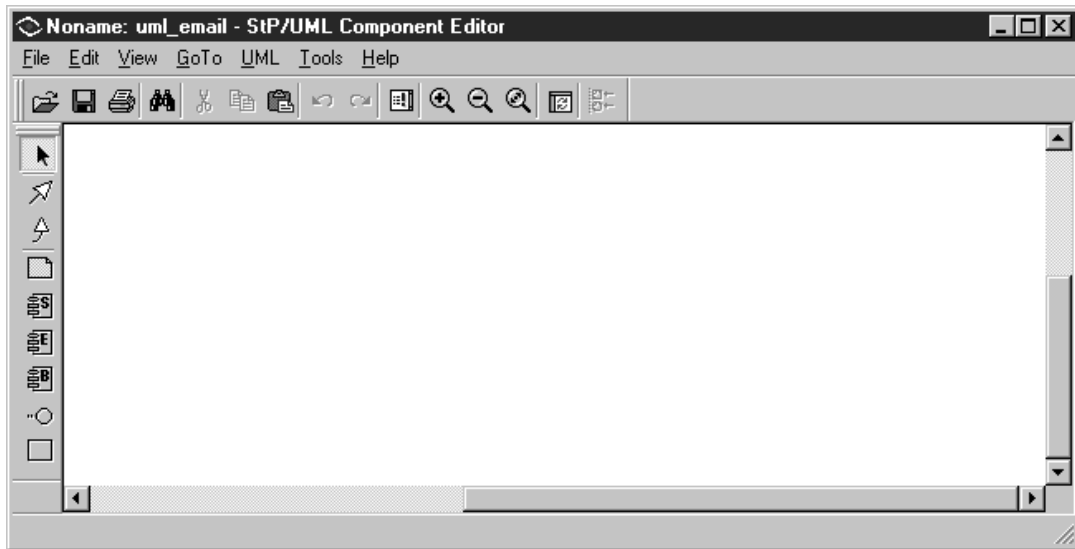
Component diagrams, in combination with deployment diagrams (described in [Chapter 12, “Creating a Deployment Diagram”](#)), present a physical-static view of a system. Collectively, component and deployment diagrams are referred to as implementation diagrams.

Using a component diagram, you can model the physical architecture of the software components and their dependencies.

Using the Component Editor

You create a component diagram using the Component Editor.

Figure 1: Component Editor



The Component Editor provides all the functions and menu options of every StP diagram editor. For general information about using StP diagram editors, see [Fundamentals of StP](#).

Starting the Component Editor

StP/UML provides access to the Component Editor from:

- The StP Desktop
- Deployment Editor
- Stereotype Editor

For information about starting an editor from the StP Desktop, see [Chapter 2, “Using the StP Desktop.”](#)

For more information about starting (navigating to) the Component Editor from another StP/UML editor, see the chapter in this manual that discusses the editor.

Navigating to Object References

You can navigate from the Component Editor to the Deployment Editor or Requirements Table Editor.

To navigate to a deployment diagram:

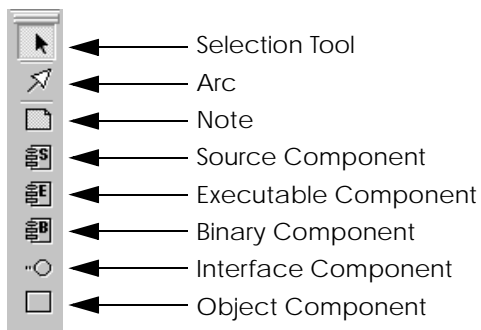
1. Click the mouse in an empty area of the diagram to deselect any symbol.
2. Choose **GoTo > Deployment Diagram**.
If the diagram already exists, the navigation target appears. If no diagram exists, a dialog box appears asking if you want to create a one.
3. Click **OK**.

To navigate to a requirements table, select any symbol in the diagram and choose **GoTo > Allocate Requirements**.

Using the Component Editor Symbol Toolbar

You select a symbol using the Symbol Toolbar.

Figure 2: Component Editor Symbol Toolbar



Procedures for using all symbols on the Symbol Toolbar are described in this chapter.

Using Display Marks

Stereotype annotations can cause display marks to appear in a component diagram.

You control the behavior of display marks using the **Display Marks** tab in the **Options** dialog box. To display this tab:

1. From the **Tools** menu, select **Options**.
2. In the **Options** dialog box, choose the **Display Marks** tab.

For details about the Display Mark Options dialog box, see [Fundamentals of StP](#).

For information on stereotypes, see [“Adding Extensibility Mechanisms” on page 11-10](#).

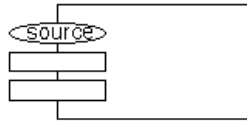
Drawing Software Components

The Component Editor provides the following software components:

- Source
- Executable
- Binary
- Interface
- Object

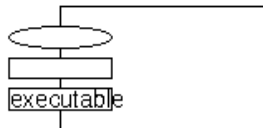
Source Component

The source component symbol represents the file containing the root of a program. An example is the file containing the *main()* function in a C++ application.



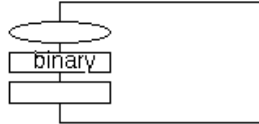
Executable Component

The executable component symbol represents executable files that are not binary.



Binary Component

The binary component symbol represents a compiled binary file that is usually an executable. In most instances, a binary component is a compiled source component.



Interface Component

The interface component symbol represents the visible behavior of a binary, executable, or source component, such as a function called by other components.

Object Component

An object component represents an object that is important to the implementation of the model but is not defined as a binary, executable, or source component.

An example of an object component is a remote database necessary to the implementation.

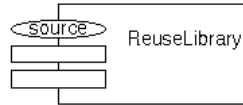
Representing and Labeling Components

Each component symbol in a component diagram must have a name. Typically, the name of the component is the same as the file portion of the name of the physical file that it represents. Component names should adhere to the file naming conventions of the operating system.

To add a component into a diagram:

1. Insert a component symbol from the symbol toolbar.
The symbol remains selected.

2. Label the component symbol.



Adding Objects to a Component

An object component lives on the host binary or executable component. The host component calls the object to perform a particular function. A host component can contain multiple objects.

There are two methods for adding an object to a host component: using a relationship or by containment.

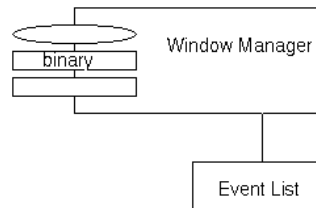
Adding Objects Using Relationships

To add an object using a relationship arc:

1. Insert an object symbol onto the diagram.
2. Label the object.
3. Connect an arc from the host component to the object.

The object is connected to its host component by an “Is Component Of” link, represented by a solid line.

Figure 3: Object Linked to a Host Component

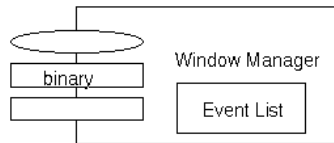


Adding Objects By Containment

To add an object by containment:

1. Insert an object symbol into a host component symbol.
2. Label the object symbol.

Figure 4: Object Contained in a Host Component



You can add multiple objects by containment. Resize the symbols when necessary.

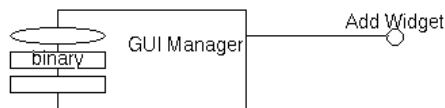
Adding Interfaces to a Component

Each binary, executable, or source component can have multiple interfaces.

To connect an interface to its host component, add an interface symbol and connect an arc from the interface to the component.

The interface is connected to its host component by an interface link, represented by a solid line.

Figure 5: Interface Linked to a Host Component



Connecting an arc from a component to the interface creates a dependency, shown as a dashed line with an open triangle pointing towards the interface. For more information on software dependencies, see the next section.

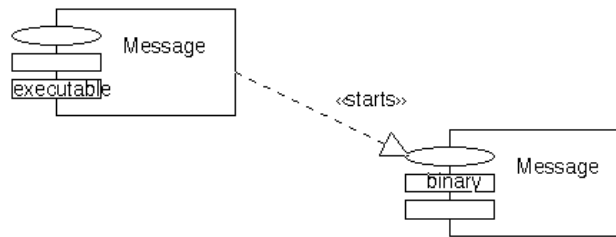
Drawing Software Dependencies

You indicate dependencies among software components by drawing connections between pairs of components. A dependency is indicated by a connection from the dependent component to the component on which the dependency exists.

In some cases, the dependency is connected to an interface belonging to another component.

Figure 6 indicates that the executable component, *Message*, is dependent on the binary component, *Message*.

Figure 6: Dependency Relationship



The manner in which the dependency is implemented is language-dependent.

Representing and Labeling Dependencies

To draw a dependency connection between software components, draw a connection from the dependent component to the component or component's interface on which the dependency exists.

Dependencies do not have labels but support extensibility mechanisms described in the next section.

Adding Extensibility Mechanisms

Each component and dependency supports the three UML extensibility mechanisms—stereotype, constraints, and tagged values.

To add extensibility mechanisms to a symbol:

1. Select the symbol.
2. Use one of the following methods to open the Properties dialog box:
 - From the **Edit** menu, choose **Properties**.
 - From the shortcut menu, choose **Properties**The Properties dialog box for the symbol appears.
3. Add the desired extensibility mechanisms.
4. Click **OK** or **Apply**.
5. To show display marks on your diagram, choose **View > Refresh Display Marks**.

For more information on stereotypes, constraints, and tagged values, see [“Adding Extensibility Mechanisms” on page 3-12](#).

Creating a Deployment Diagram

You can create a deployment diagram based on an existing component diagram. After creating the deployment diagram, you assign components to deployment nodes, which represent hardware devices in your model.

To create a deployment diagram from an existing component diagram, see [“Navigating to Object References” on page 11-3](#).

You can also generate a deployment diagram from an existing component diagram using the Deployment Editor. For more information, see [“Deploying a Component Diagram” on page 12-8](#).

For information on drawing a deployment diagram, refer to [Chapter 12, “Creating a Deployment Diagram.”](#)

Validating a Component Diagram

StP/UML provides two options for checking the correctness and consistency of a model from the Component Editor:

- Checking the current diagram
- Checking the repository for the current system

When you check your model, the Message Log appears listing the errors. For information about using the Message Log, see [Fundamentals of StP](#).

Checking a Diagram

You can check a diagram to validate that all symbols are properly labeled. Checking the diagram does not check the contents of the repository.

To check a diagram, choose **Tools > Check Syntax**.

Checking the StP Repository

You can check the StP repository for a system to validate that each interface has exactly one interface link to a component.

To check the StP repository, choose **Tools > Check Semantics**.

12

Creating a Deployment Diagram

This chapter describes how to create and use deployment diagrams. Topics covered are as follows:

- [“What Are Deployment Diagrams?” on page 12-1](#)
- [“Using the Deployment Editor” on page 12-2](#)
- [“Drawing Hardware Components” on page 12-5](#)
- [“Drawing Hardware Dependencies” on page 12-7](#)
- [“Adding Extensibility Mechanisms” on page 12-7](#)
- [“Deploying a Component Diagram” on page 12-8](#)
- [“Validating a Deployment Diagram” on page 12-9](#)

For more information on deployment diagrams, see *Unified Modeling Language User Guide* (Booch et al 1998) and *Unified Modeling Language Reference Manual* (Rumbaugh et al 1998).

What Are Deployment Diagrams?

Deployment diagrams show the allocation of processors and devices in a system. A processor is hardware capable of executing an application, such as a PC or a workstation. A device is hardware that is not capable of executing an application, such as a disk or a terminal. Each processor and device is represented as a deployment node.

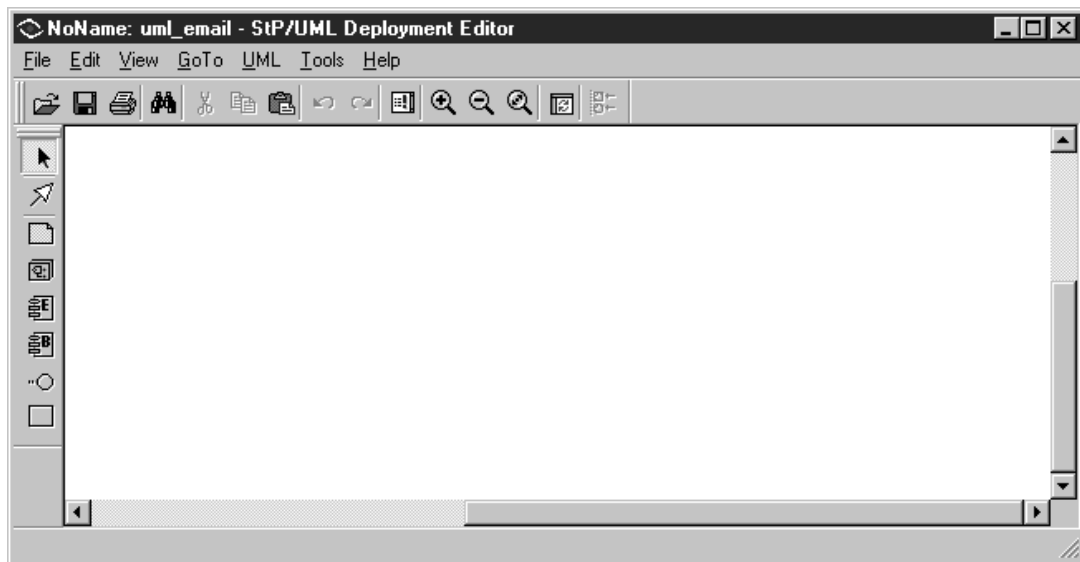
Deployment diagrams, in combination with component diagrams (described in [Chapter 11, “Creating a Component Diagram”](#)), present a physical-static view of a system. Collectively, component and deployment diagrams are referred to as implementation diagrams.

Using a deployment diagram, you can model the allocation of software to hardware.

Using the Deployment Editor

You create a deployment diagram using the Deployment Editor.

Figure 1: Deployment Editor



The Deployment Editor provides all the functions and menu options of every StP diagram editor. For general information about using StP diagram editors, see [Fundamentals of StP](#).

Starting the Deployment Editor

StP/UML provides access to the Deployment Editor from:

- The StP Desktop
- Component Editor
- Stereotype Editor

For information about starting an editor from the StP Desktop, see [Chapter 2, “Using the StP Desktop.”](#)

For more information about starting (navigating to) the Deployment Editor from another StP/UML editor, see the chapter in this manual that discusses the editor.

Navigating to Object References

You can navigate from the Deployment Editor to the Component Editor and Requirements Table Editor.

To navigate to a component diagram:

1. Click the mouse in an empty area of the diagram to deselect any symbol.
2. Choose **GoTo > Component Diagram**.

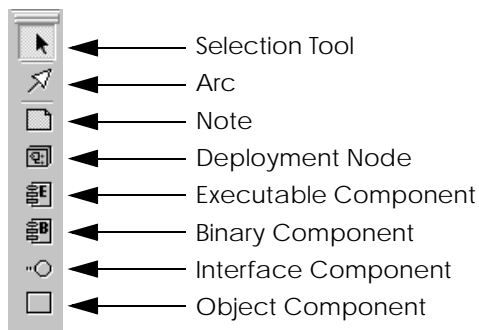
If a component diagram with the same name exists, the navigation target appears.

To navigate to a requirements table, select any symbol in the diagram and choose **GoTo > Allocate Requirements**.

Using the Deployment Symbols List

You select a symbol using the Symbol Toolbar.

Figure 2: Deployment Editor Symbol Toolbar



Procedures for using all symbols on the Symbol Toolbar are described in this chapter.

Using the UML Menu

In addition to the standard diagram menu options, the Deployment Editor also provides a UML Menu.

This menu lists the **Deploy Component Diagram** command, described in [“Deploying a Component Diagram” on page 12-8](#).

Using Display Marks

Stereotype annotations can cause display marks to appear in the diagram.

You control the behavior of display marks using the **Display Marks** tab in the **Options** dialog box. To display this tab:

1. From the **Tools** menu, select **Options**.
2. In the **Options** dialog box, choose the **Display Marks** tab.

For details about the Display Mark Options dialog box, see [Fundamentals of StP](#).

For information on stereotypes, see [“Adding Extensibility Mechanisms” on page 12-7](#).

Drawing Hardware Components

The Deployment Editor provides the same symbols as the Component Editor, with the following exceptions:

- The deployment node symbol, shown as a three-dimensional rectangle, represents hardware devices. Software components are contained within the node.
- The source executable symbol is not supported in the Deployment Editor.

You can label the deployment node as well as all components contained within the deployment node.

For information on shared symbols between the Component Editor and Deployment Editor, see [Chapter 11, “Creating a Component Diagram.”](#)

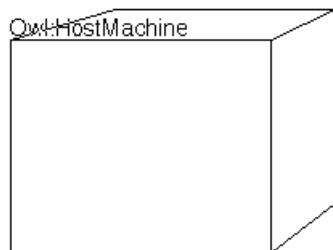
Representing and Labeling Deployment Nodes

To add a new deployment node into a diagram:

1. Insert a deployment node symbol into the drawing area.
The symbol remains selected.
2. Label the deployment node symbol, using either of the following formats:
 - [node type], such as `HostMachine`
 - [node name:node type], such as `Owl:HostMachine`

[Figure 3](#) illustrates a labeled deployment node.

Figure 3: Labeled Deployment Node



Adding Software Components

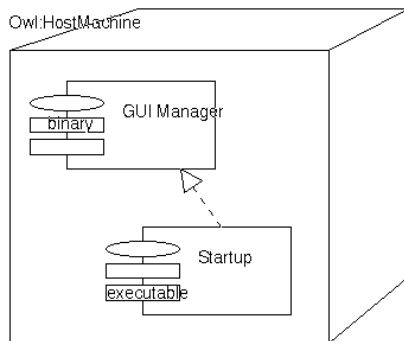
You add software components to a deployment node. All software components must be labeled.

For a description of software components and their relationships, refer to [Chapter 11, “Creating a Component Diagram.”](#)

To add a software component to a deployment node:

1. Insert a software component symbol into a deployment node.
2. Label the symbol.
3. If necessary, connect the component to another component.

Figure 4: Deployment Node with Software Components



For information on drawing relationships between deployment nodes, refer to the next section, [“Drawing Hardware Dependencies.”](#)

Drawing Hardware Dependencies

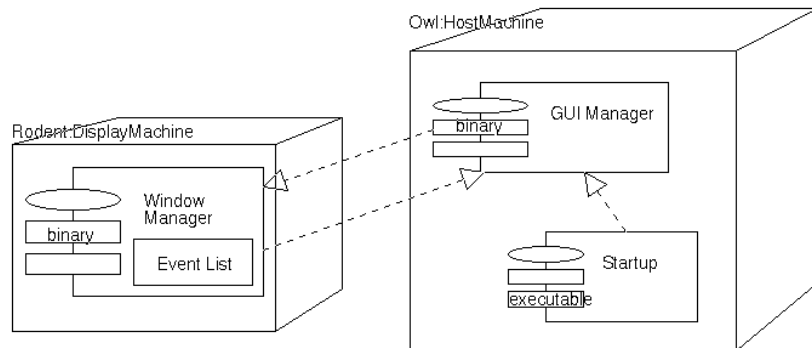
You indicate dependencies between deployment nodes by drawing connections between components in one deployment node with components in another deployment node.

Representing Connections

To draw a connection between deployment nodes, insert a connection from one software component to another. You can connect a component to:

- Another component
- An object
- An interface

Figure 5: Deployment Node Dependency



Adding Extensibility Mechanisms

Each component and dependency supports the three UML extensibility mechanisms—stereotype, constraints, and tagged values.

To add extensibility mechanisms to a symbol:

1. Select the symbol.
2. Use one of the following methods to open the Properties dialog box:
 - From the **Edit** menu, choose **Properties**.
 - From the shortcut menu, choose **Properties**The Properties dialog box for the symbol appears.
3. Add the desired extensibility mechanisms.
4. Click **OK** or **Apply**.
5. To show display marks on your diagram, choose **View > Refresh Display Marks**.

For more information on stereotypes, constraints, and tagged values, see [“Adding Extensibility Mechanisms” on page 3-12](#).

Deploying a Component Diagram

You can deploy information from an existing component diagram into a deployment diagram.

To deploy a component diagram in the Deployment Editor:

1. Choose **UML > Deploy Component Diagram**.
The Select Component Diagram to Deploy dialog box appears.
2. Select the component diagram.
3. Click **OK** or **Apply**.
The deployment diagram appears containing information from the component diagram.

Validating a Deployment Diagram

StP/UML provides two options for checking the correctness and consistency of an information model from the Deployment Editor:

- Checking the current diagram
- Checking the repository for the current system

When you check your model, the Message Log appears listing the errors. For information about using the Message Log, see [Fundamentals of StP](#).

Checking a Diagram

You can check a diagram to validate that all symbols are properly labeled. Checking the diagram does not check the contents of the repository.

To check a diagram, choose **Tools > Check Syntax**.

Checking the StP Repository

You can check the StP repository for a system to validate that each interface has exactly one interface link to a component.

To check the StP repository, choose **Tools > Check Semantics**.

13 Creating a Stereotype Diagram

This chapter describes how to create and use stereotype diagrams. Topics covered are as follows:

- [“What Are Stereotype Diagrams?” on page 13-1](#)
- [“Using the Stereotype Editor” on page 13-2](#)
- [“Drawing a Stereotype Diagram” on page 13-5](#)
- [“Pre-defined UML Stereotypes” on page 13-6](#)
- [“Validating a Stereotype Diagram” on page 13-10](#)

What Are Stereotype Diagrams?

A stereotype expands the semantics of the element it is attached to and can be used in all StP/UML diagram editors. Stereotype diagrams define UML stereotypes.

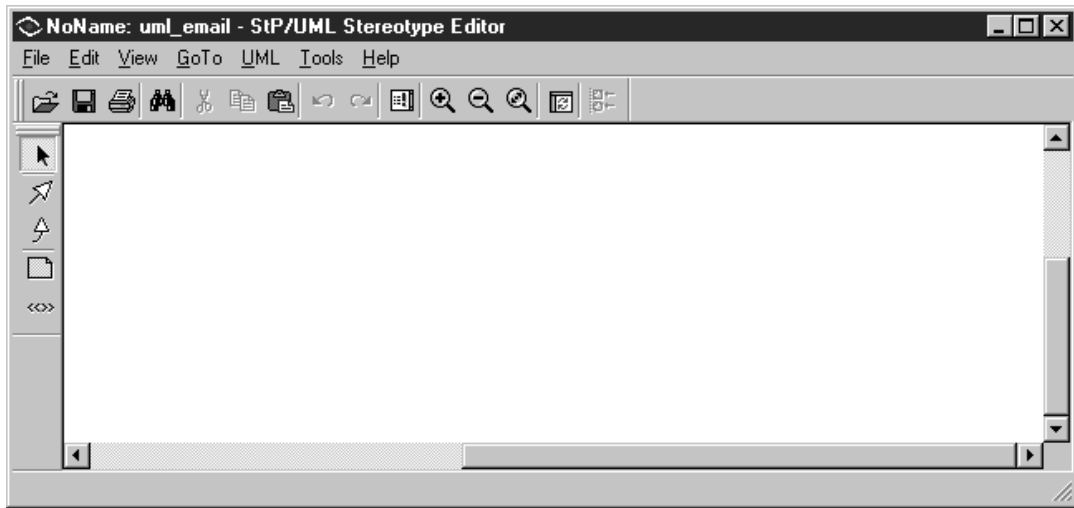
UML includes pre-defined stereotypes. These stereotypes are listed in [Table 1 on page 13-7](#).

In addition, UML supports user-defined stereotypes. You create these stereotypes in a stereotype diagram.

Using the Stereotype Editor

You create a stereotype diagram using the Stereotype Editor.

Figure 1: Stereotype Editor



The Stereotype Editor provides all the functions and menu options of every StP diagram editor. For general information about using StP diagram editors, see [Fundamentals of StP](#).

Starting the Stereotype Editor

StP/UML provides access to the Stereotype Editor from:

- The StP Desktop
- Class Editor

For information about starting an editor from the StP Desktop, see [Chapter 2, “Using the StP Desktop.”](#)

For information about starting (navigating to) the Stereotype Editor from the Class Editor, see [Chapter 4, “Creating a Class Diagram.”](#)

Navigating to Object References

You can navigate from a selected stereotype in a stereotype diagram to any element that reference the stereotype. You can also navigate to the Requirements Table Editor.

To navigate to an element that references the stereotype:

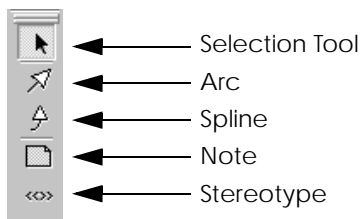
1. Select the stereotype.
2. Choose **GoTo > Where Stereotype Is Used**.
If only one reference exists, the navigation target appears.
If more than one reference exists for the stereotype, an Object Selector dialog box appears.
3. Select the correct element from the Object Selector dialog box.
4. Click **OK**.
The navigation target appears.

To navigate to a requirements table, select a stereotype symbol and choose **GoTo > Allocate Requirements**.

Using the Stereotype Symbol List

You select a symbol using the Symbol Toolbar.

Figure 2: Stereotype Diagram Symbol Toolbar



Procedures for using all symbols on the Symbol Toolbar are described in this chapter.

Using the UML Menu

In addition to the standard diagram menu options, the Stereotype Editor also provides a UML Menu.

This menu lists the **Label with Predefined Stereotype** command, described in [“Drawing a Stereotype Diagram” on page 13-5](#).

Using Display Marks

Stereotype annotations added through the Properties dialog box can cause display marks to appear in the diagram.

You control the behavior of display marks using the **Display Marks** tab in the **Options** dialog box. To display this tab:

1. From the **Tools** menu, select **Options**.
2. In the **Options** dialog box, choose the **Display Marks** tab.

For details about the Display Mark Options dialog box, see [Fundamentals of StP](#).

For information on adding a stereotype to a stereotype, see [“Adding Extensibility Mechanisms” on page 13-6](#).

Drawing a Stereotype Diagram

Stereotype diagrams represent a hierarchical description of a stereotype based on other stereotypes.

The root (top-level) stereotype is defined by the stereotypes below it. The root stereotype can be a pre-defined or user-defined stereotype. All lower-level stereotypes must be user-defined stereotypes.

StP checks the semantics of pre-defined stereotypes. It does not check user-defined stereotypes.

Representing and Labeling Stereotypes

To create a stereotype diagram:

1. Insert stereotype symbols into the drawing area to form a hierarchy.
2. Label the stereotypes.

The root stereotype can be either a pre-defined or user-defined stereotype. All lower-level stereotypes, which define the root stereotype, must be user-defined.

To label the root stereotype with a pre-defined stereotype, you can choose **Label with Predefined Stereotype** and select a stereotype from the scrolling list. The accelerator for this command is Ctrl+L.

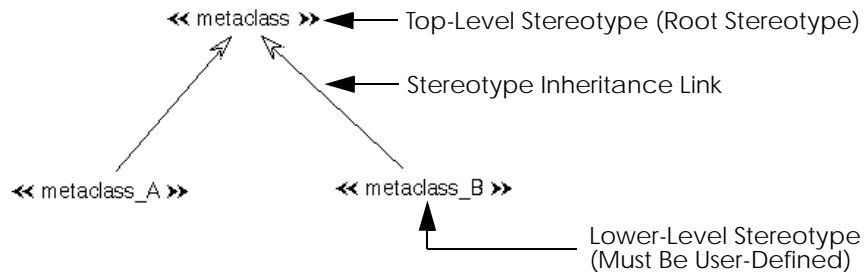
For a list of pre-defined stereotypes, see [Table 1 on page 13-7](#).

3. Add links to the stereotypes.

Links are drawn from the lower-level stereotype to the higher-level stereotype.

Relationships are represented by stereotype inheritance links which have an arrow pointing towards the higher-level stereotype.

Figure 3: Stereotype Diagram



4. Choose **File > Save**.

To navigate from a stereotype in a stereotype diagram to other StP/UML editors where the stereotype appears, select the stereotype and choose **GoTo > Where Stereotype Is Used**. For more information, see [“Navigating to Object References” on page 13-3](#).

Adding Extensibility Mechanisms

Each stereotype supports the three UML extensibility mechanisms—stereotype, constraints, and tagged values. To add an extensibility mechanism, select the stereotype and choose **Edit > Properties**.

For more detailed information on extensibility mechanisms, refer to [“Adding Extensibility Mechanisms” on page 3-12](#).

Pre-defined UML Stereotypes

Pre-defined stereotypes add specific semantics to the elements they are attached to. Consequently, pre-defined stereotypes have an impact on code generation (user-defined stereotypes do not).

StP/UML supports the pre-defined stereotypes listed in [Table 1](#).

Table 1: Pre-defined Stereotypes

Stereotype Name	Attached To	Describes
actor	Type	An external element that interacts with the system
application	Component	An executable program
becomes	Dependency	An element that is both the source and the target of the dependency
bind	Dependency	A target type or collaboration that is bound to the source type or collaboration
call	Dependency	A source operation that invokes a target operation
constraint	Note	A semantic condition or restriction
copy	Dependency	Two elements that are identical
derived	Dependency	One element is derived from another
document	Component	A document
enumeration	PrimitiveType	A domain consisting of a set of identifiers
extends	Generalization	An extension of behavior from the source element to the target element—applies to ‘type to type’ or ‘use case to use case’ relationships
facade	Package	A package that references but never owns other elements

Table 1: Pre-defined Stereotypes (Continued)

Stereotype Name	Attached To	Describes
file	Component	A document containing source code
friend	Dependency	Extends the visibility of the import dependency between two elements
import	Dependency	A source element that can reference the public contents of the target element
instance	Dependency	A target element that is an instance of the source element (for example, an object is an instance of a class)
interface	Type	An interface of the element
library	Component	A static or dynamic library
metaclass	Dependency	A target element is a metaclass of the source element (type)
	Type	A type that is a metaclass
page	Component	A page associated with the World Wide Web
powertype	Dependency	A target element (type) is a powertype of the source element (generalization)
	Type	A type that is a powertype of a generalization
process	ActiveClass	A heavy-weight flow of control

Table 1: Pre-defined Stereotypes (Continued)

Stereotype Name	Attached To	Describes
refinement	Dependency	A target element that expands the specifications of a source element (for example, an instantiated class is a refinement of a parameterized class)
requirement	Note	A responsibility or obligation
role	Dependency	An association role
send	Dependency	A source operation that sends a signal to the target operation
signal	Class	A named event
stub	Package	A package that is not completely transferred
subclass	Generalization	A subclass that inherits from a supertype (may be a different type)
subtype	Generalization	A subtype that inherits from a supertype (and is the same type)
table	Component	A database table
thread	ActiveClass	A light-weight flow of control
trace	Dependency	A dependency between a source element in one model and a target element in the same or different model
uses	Dependency	A source use case that includes the behavior of the target use case

Table 1: Pre-defined Stereotypes (Continued)

Stereotype Name	Attached To	Describes
utility	Type	A type that has no instances

Validating a Stereotype Diagram

StP/UML provides two options for checking the correctness and consistency of information in the Stereotype Editor:

- Checking the current diagram
- Checking the repository for the current system

When you check your model, the Message Log appears listing the errors. For information about using the Message Log, see [Fundamentals of StP](#).

Checking a Diagram

You can check a diagram to validate that all symbols are properly labeled. Checking the diagram does not check the contents of the repository.

To check a diagram, choose **Tools > Check Syntax**.

Checking the StP Repository

You can check the StP repository for a system to validate that stereotypes are properly defined and correct. Checks of the StP repository include:

- Are any lower-level stereotypes labeled with pre-defined stereotypes?
- Does the stereotype hierarchy have any circularity?

To check the StP repository, choose **Tools > Check Semantics**.

14 Renaming Symbols

This chapter describes how to rename a diagram symbol or table cell. Topics covered are as follows:

- [“What Are the Options for Renaming Symbols?” on page 14-1](#)
- [“Objects and Object References” on page 14-2](#)
- [“Retyping a Label” on page 14-4](#)
- [“Renaming Objects Systemwide” on page 14-8](#)

What Are the Options for Renaming Symbols?

StP provides two options for renaming a diagram symbol or table cell. You can:

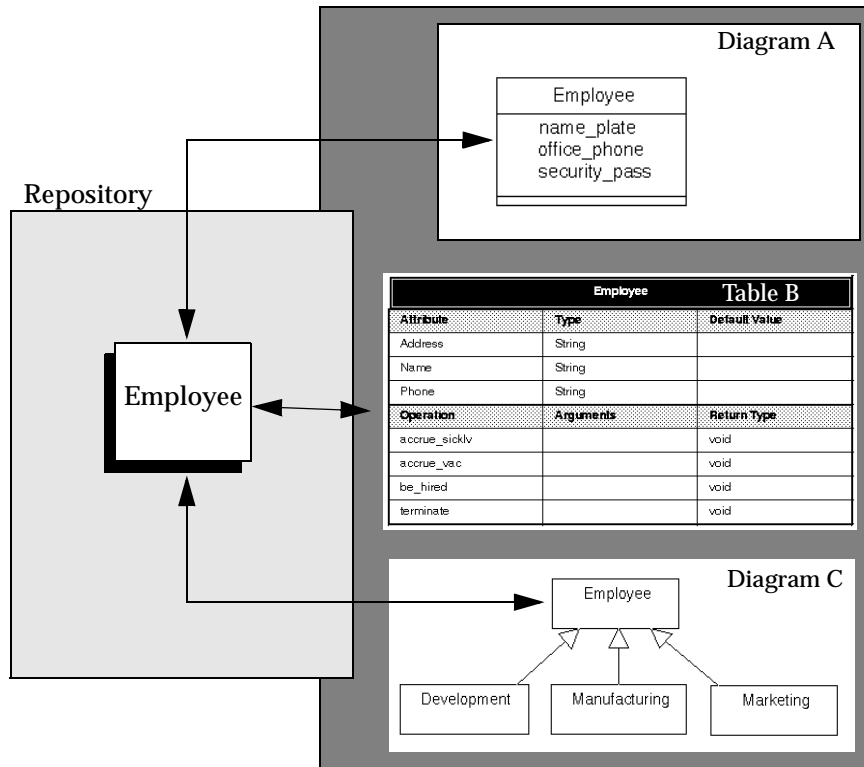
- Change the name of a symbol by retyping its label.
This change affects only the current diagram.
 - Change the name of the repository object using the **Edit > Rename Object Systemwide** command.
This ensures that the name is changed in certain diagrams and tables in which it occurs.
-

Objects and Object References

This section briefly reviews the way that StP stores information in order to clarify the difference between relabeling a symbol and renaming it using the **Rename Object Systemwide** command. This command affects objects patterned after the Persistent Data Model (PDM). For detailed information about the PDM, see [Object Management System](#).

Each diagram symbol and table cell in the UML system is a symbolic reference to an object in the repository. A single repository object can be represented in any number of diagrams or tables. Each symbolic reference is a particular view of the object and contributes defining information to the object in the form of annotations and relationships to other objects. Any defining information that a particular diagram symbol or table cell contributes to its corresponding repository object automatically applies to all other symbols and table cells that represent the object. [Figure 1](#) shows a repository object (*Employee*) that has references in two diagrams and a table.

Figure 1: Repository Object and Symbolic References



The repository object and its symbolic references are associated by their common name. When you rename a reference on a diagram or table, you can choose either to break or to preserve the association to the underlying repository object. Renaming the reference by retyping its label breaks the association, since the name of the reference and the name of the object no longer correspond. Renaming the reference using the **Rename Object Systemwide** command actually renames the repository object and all of its symbolic references, so the association is preserved.

Choosing an Option

Use these guidelines to decide whether to rename a symbol or cell by retyping its label or by using the **Rename Object Systemwide** command.

Retype the label when:

- You have made a typing error or otherwise mislabeled a symbol.
- No other diagrams or tables in the system contain a symbol of the same name.
- You want to create a new symbol with the same annotations and relationships as the current symbol.

Use **Rename Object Systemwide** when you want:

- The information defined in other diagrams and tables to be preserved for the current symbol.
- The name change to be reflected in other parts of the system.

Retyping a Label

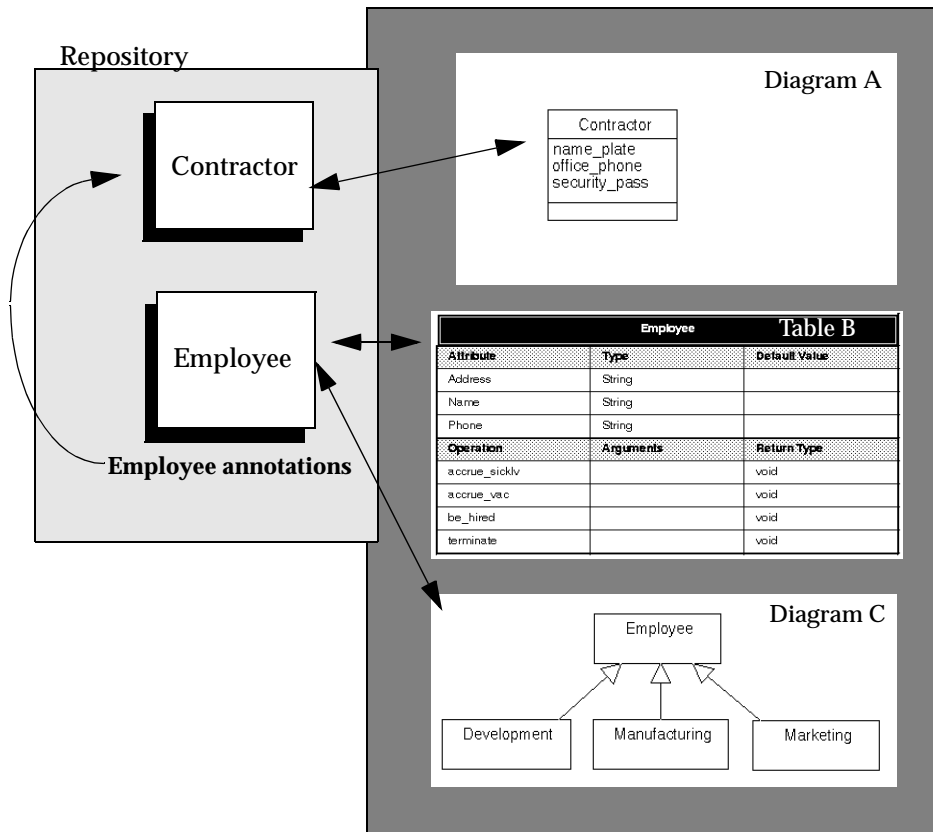
When you retype the label of a symbol, you map the symbol to a different repository object corresponding to the new name. Any relationships defined in the current diagram or table are transferred to the new object. Because the scope of retyping a label is limited to the current diagram or table, relationships defined in other diagrams and tables are severed along with the mapping to the original object.

If the new label is not already in use within the system, a new repository object is created, and all of the annotations associated with the old object are copied to the new object. If the new label corresponds to an existing repository object, that object does not inherit annotations unless it has none of its own.

Remapping to an Object without Annotations

Figure 2 illustrates what happens when the new label maps to a newly-created repository object or an existing object that has no annotations. In this example, the *Employee* class in Diagram A is relabeled *Contractor*.

Figure 2: Remapping to an Object without Annotations

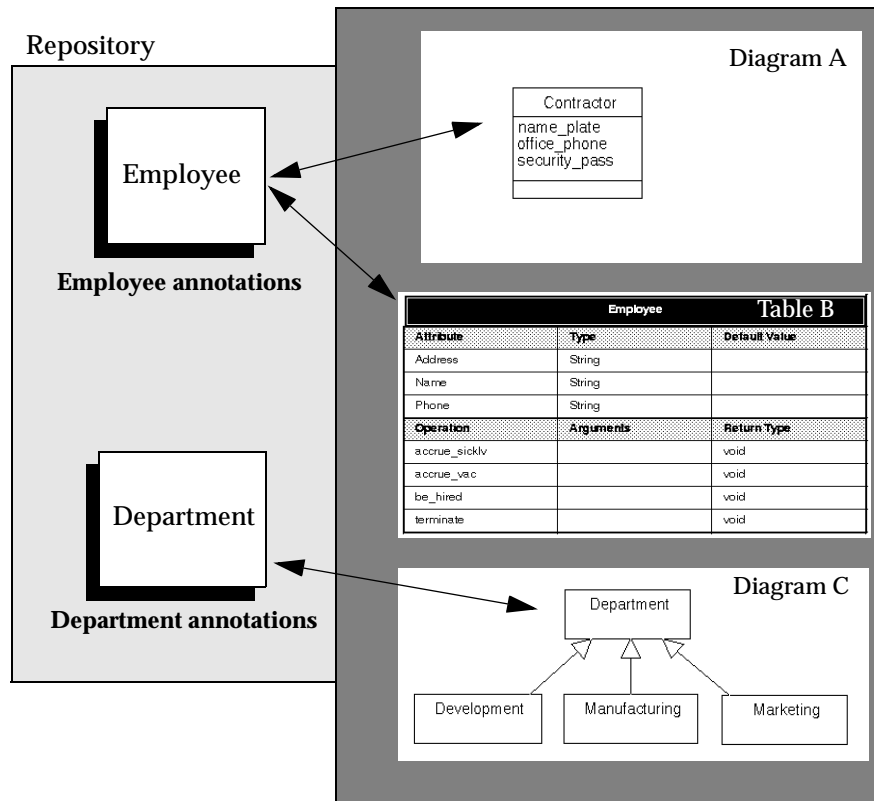


All of the annotations associated with the *Employee* object are copied over to the *Contractor* object. The attributes defined in Diagram A now belong exclusively to the *Contractor* object. The *Employee* object loses its attributes *name_plate*, *office_phone* and *security_pass*. In turn, the *Contractor* object loses the attributes and operations in Table B and the exclusive subtypes *R&D*, *Manufacturing*, and *Marketing* in Diagram C.

Remapping to an Object with Annotations

Figure 3 shows what happens when the new label maps to an existing object that already has annotations. In this example, the *Employee* class in Diagram C is relabeled *Department*. The relationships defined in Diagram C are transferred to *Department*. Consequently, *Employee* loses its exclusive subtypes *R&D*, *Manufacturing*, and *Marketing*. Since *Department* already has annotations, it does not inherit any annotations from *Employee*.

Figure 3: Remapping to an Object with Annotations



Changing a Label

To relabel a symbol or table cell:

1. Select the symbol or table cell.
2. Type the new label.
3. Save the diagram or table.

You can also change a label using the Current Symbol/Cell Options dialog box (as described in [Fundamentals of StP](#)).

Deleting Unreferenced Objects

When you relabel (and in consequence remap) the sole symbolic reference to a repository object, the object becomes “unreferenced.” It exists in the repository even though it is not depicted anywhere in the UML model.

You should delete unreferenced objects regularly. To do this, use **Repository > Maintain Systems > Delete Unreferenced Objects in Current System Repository** from the StP Desktop. For more information, see [StP Administration](#).

Cloning Objects by Relabeling Symbols

You can take advantage of symbol relabeling to quickly “clone” similar objects. This method avoids the repeated use of the Object Annotation Editor, table editors, and dialogs to annotate symbols with identical data.

For example, you wish to draw a diagram in which three personnel forms, *EmpHire*, *EmpEval* and *EmpTerm*, have the same multiplicity constraint of 45 lines. Rather than recreate this information three times, you can draw and annotate one class and use symbol relabeling to create the others.

To clone an object:

1. Insert, label, and annotate one class (for example, *EmpHire*).
You specify the constraint by typing “0..45” in the Constraints text field in the Properties dialog box.
2. Save the diagram.

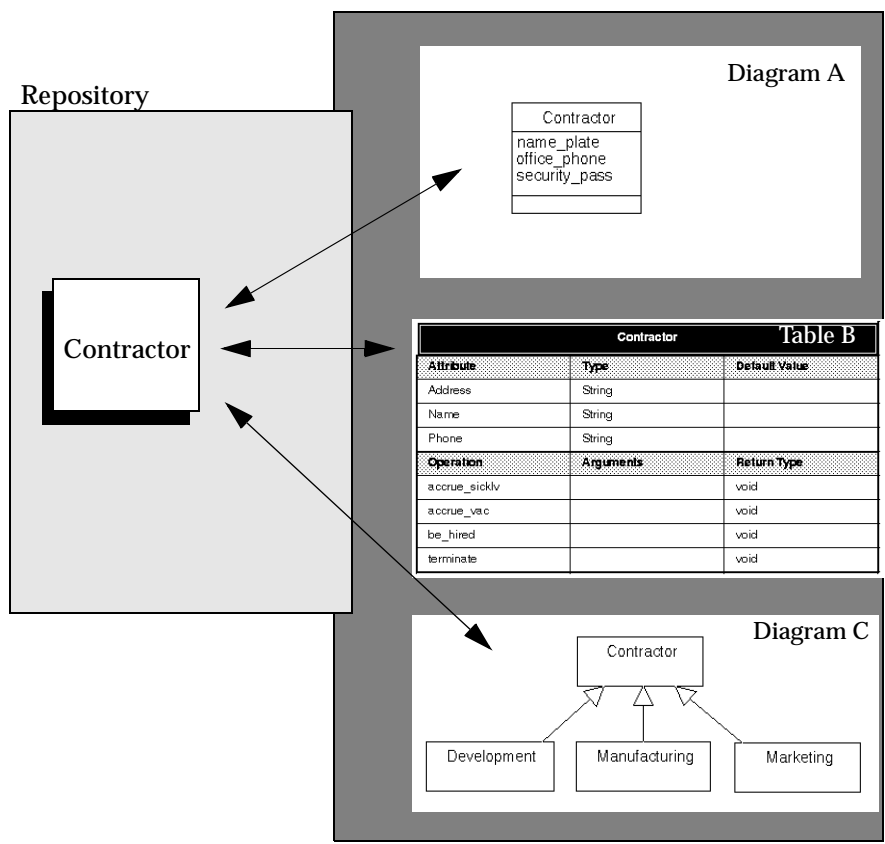
3. Copy the class.
4. Relabel the duplicate class (for example, to *EmpEval*).
5. Save the diagram.
6. Repeat steps 3 to 5 to create the class *EmpTerm*.

Renaming Objects Systemwide

You can propagate a name change throughout a system using the **Rename Object Systemwide** command. When you use this command, the repository object is updated, and the name change is reflected in every diagram and table in which the object is represented.

[Figure 4](#) shows what happens when the *Employee* class in Diagram A is renamed to *Contractor* using the **Rename Object Systemwide** command.

Figure 4: Rename Object Systemwide



Objects That Can Be Renamed

Any element that can be labeled in a diagram or table can be renamed. For special circumstances, see [“Renaming Instances in StP/UML” on page 14-10](#).

Any element that derives its name from the names of related elements cannot be renamed directly. However, StP automatically adjusts these elements to reflect name changes in the objects on which they depend. For more information, see [“Effects on Dependent Objects” on page 14-17](#).

Renaming Instances in StP/UML

Renaming an instance of an element (such as an object or message) in StP/UML updates the repository in the following ways:

- For objects, all instances with the same name are updated.
- For messages, all instances with the same signature and connected between the same objects are updated. (Anonymous objects with the same name are considered different unless they are created by generating a sequence or collaboration diagram.)
- The reference to the source element (such as the class or operation) is also updated to reflect the new source element name.

The original source element (such as the class or operation) is not renamed in the repository.

For example, if you rename an object from *instance:ClassA* to *instance:ClassB*, all objects currently labeled with *:ClassA* are updated to *:ClassB*. However, no changes are applied to *ClassA* in the repository. To update *ClassA* (the source element) systemwide, you must rename the class in a class symbol or class table.

Renaming an Object Systemwide

To rename an object systemwide:

1. Select the symbol or table cell.
2. Choose **Edit > Rename Object Systemwide**.
The Rename dialog box appears, and the name of the selected symbol is displayed in the New Label text field.
3. Change the label to the new name.
4. Click on those options you wish to activate.
5. Click **Check Impact**.
The StP Message Log appears listing the diagrams and tables that will be updated if you execute the rename operation. If there is a conflict with an existing object or file name, a warning message is displayed.
6. Click **Rename**.

The object and all of its references are renamed. The names and/or signatures of dependent objects and files are adjusted to reflect the change.

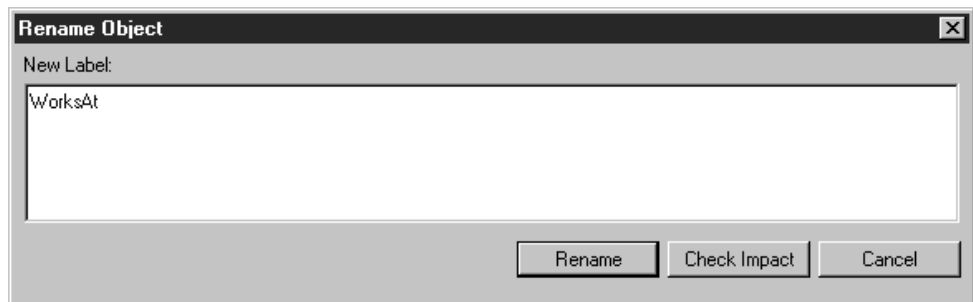
The rename operation is confirmed in the Message Area. In addition, a broadcast message is sent out to any user who is editing a diagram or table containing references to the renamed object.

The rename operation is final, even if the diagram or table is not saved. To change back to the object's original name, you must repeat the global rename procedure.

Rename Options

The Rename dialog box is the graphical interface for running the **Rename Object Systemwide** command. The basic Rename dialog box is shown in Figure 5.

Figure 5: Rename Object Dialog Box



There are three other versions of the Rename dialog box. One of these versions appears if you have selected a:

- Class
- Operation
- Message
- Use Case

Special options are added to the dialog boxes for these objects. The following figures show the Rename Class and Rename Operation dialog boxes.

Figure 6: Dialog box for Renaming Classes

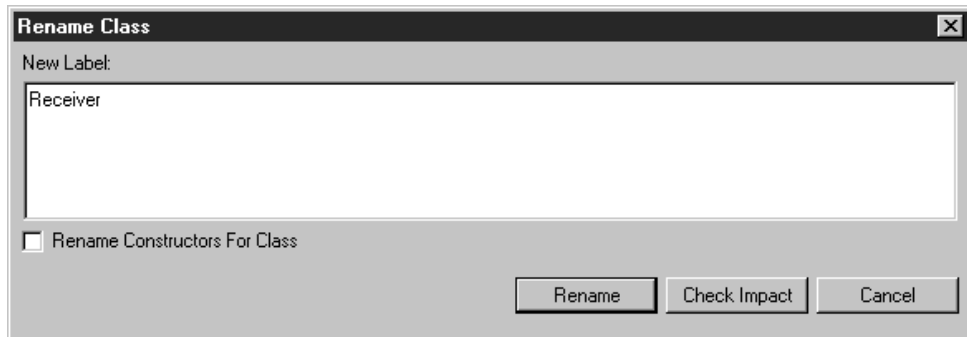


Figure 7: Dialog for Renaming Operations

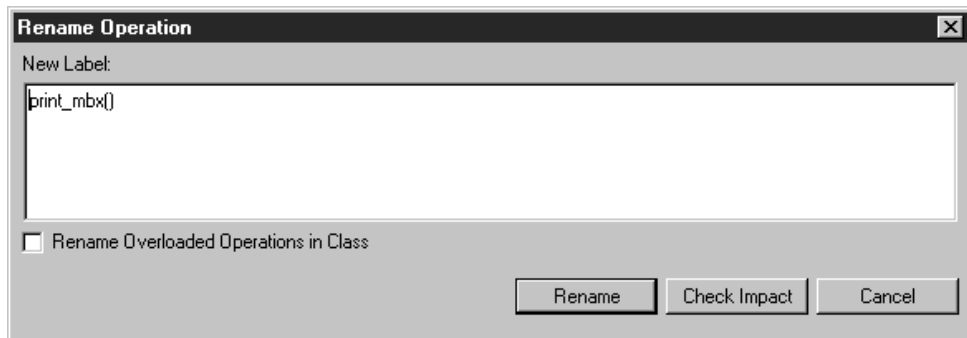
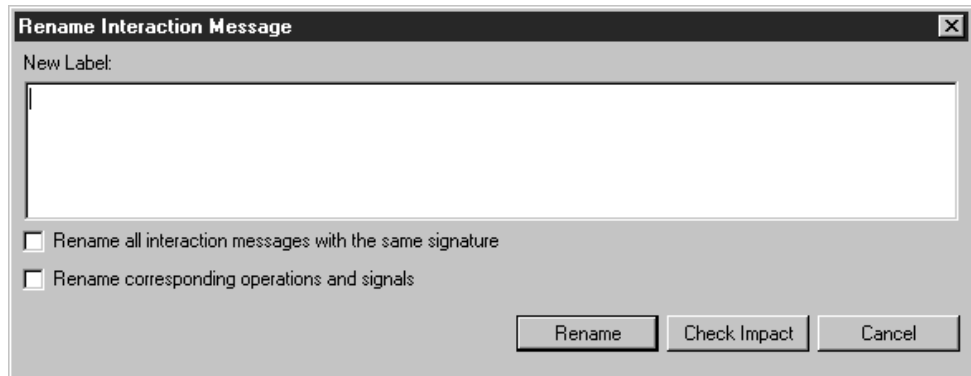
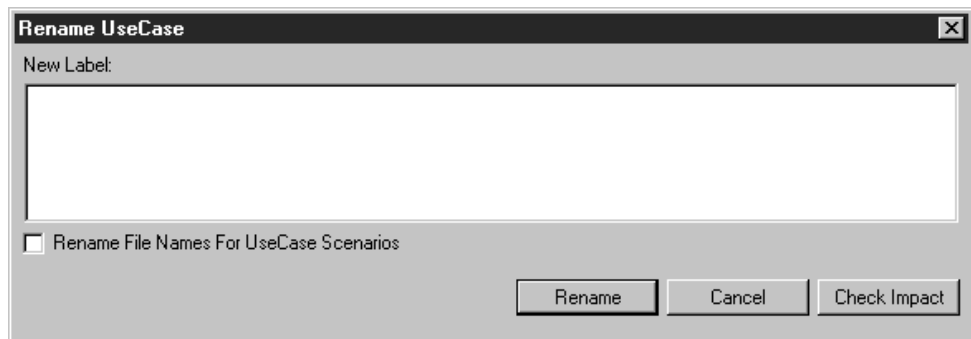


Figure 8: Dialog for Renaming Interaction Messages

The dialog box titled "Rename Interaction Message" features a "New Label:" text input field. Below the input field are two unchecked checkboxes: "Rename all interaction messages with the same signature" and "Rename corresponding operations and signals". At the bottom right are three buttons: "Rename", "Check Impact", and "Cancel".

Figure 9: Dialog for Renaming Use Cases

The dialog box titled "Rename UseCase" features a "New Label:" text input field. Below the input field is one unchecked checkbox: "Rename File Names For UseCase Scenarios". At the bottom right are three buttons: "Rename", "Cancel", and "Check Impact".

Table 1 describes the options available from all the Rename dialog boxes.

Table 1: Rename Class/Object Options

Option	Description
New Label	Input field for the new name for the selected object. The command fails if you have not changed the label.
Check Impact	Generates messages that indicate the impact of renaming the object if you decide to run it on the selected object.

Table 1: Rename Class/Object Options (Continued)

Option	Description
Class Option	
Rename Constructors For Class	If on, causes all the constructors and destructors for the class to be renamed with the class. This option is relevant only if you will be generating C++ code with your models.
Operation Options	
Rename Overloaded Operations in Class	If on, renames all overloaded operations in the class. Operation signatures remain unaffected. If off, renames only the selected operation. See “Renaming Overloaded Operations” on page 14-14 .
Interaction Message Options	
Rename all interaction messages with the same signature	If on, renames all interaction messages that have the same return type, message name, arguments, and server class (messages that are received by an object of the same class).
Rename corresponding operations and signals	If on, renames all operations and signals with the same name, return type, and number of arguments belonging to the server class.
Use Case Options	
Rename File Names for UseCase Scenarios	If on, renames all corresponding scenario file names in the sequence and collaboration editors.

Renaming Overloaded Operations

If you are renaming a class operation, you can choose whether or not to rename all overloaded operations for the class. An overloaded operation is an operation that occurs with more than one signature.

For example, assume class *Document* has two print operations: *print(page)* and *print(range)*. You select *print(page)* to be renamed to “printit.” If you turn on the Rename Overloaded Operations in Class option, both *print* operations will be renamed. This will not affect their signatures. The result is:

printit(page), printit(range)

If you do not turn on the Rename Overloaded Operations in Class option, the result is:

printit(page), print(range)

Previewing Your Rename Selection

You may want to preview your rename selection to check:

- How many objects (diagrams, tables) will be affected
- Whether your Rename operation will succeed

You can accomplish both goals by running a Check Impact report.

Rename Object Systemwide does not allow you to change the name of an object to one that already exists in the system. Run a Check Impact report before you rename an object to make sure there are no naming conflicts. The contents of the report are displayed in the StP Message Log. Figure 10 shows messages of a rename that would fail.

Figure 10: Report of an Object Rename That Would Fail

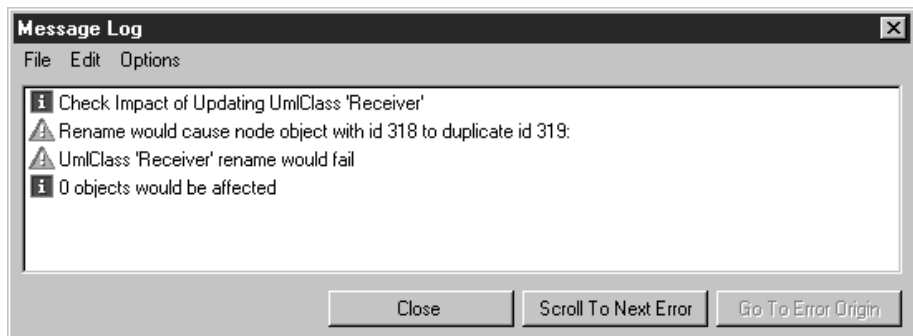
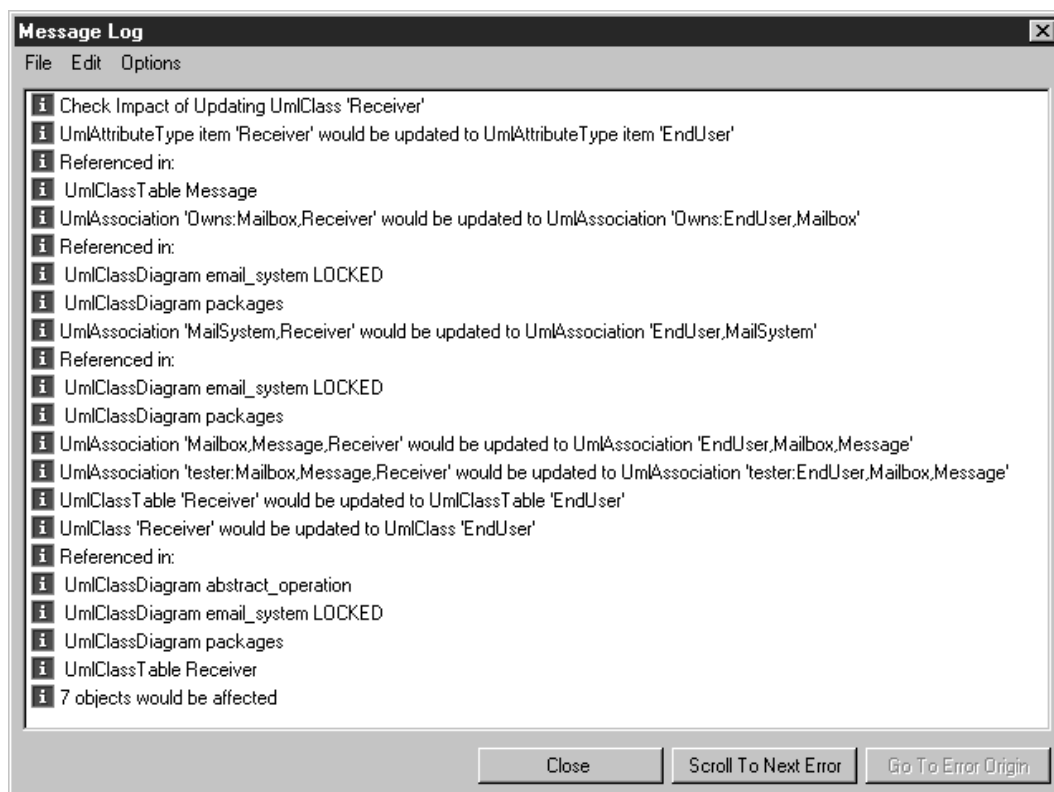


Figure 11 shows a report of a rename operation that would succeed. This report shows the impact of renaming a class that is referenced in several diagrams. The report shows which objects will be renamed and which diagrams (references) will have changes.

For a discussion of what to expect to be renamed for each object, see [“Effects on Dependent Objects” on page 14-17](#). For a discussion of possible conflicts, see [“Global Renaming Errors” on page 14-17](#).

Figure 11: Report of a Successful Object Rename



Global Renaming Errors

There are two instances in which system-wide renaming does not work:

- When the new name is identical to the name of an existing object that has annotations or references.
- When two users simultaneously rename either the same object or different objects with the same dependent object.

If the new name already exists in the repository, the Message Log displays an error message and the rename operation fails. If you cannot find the object in any diagram or table, it is probably an “unreferenced object.” For information on fixing this, see [“Deleting Unreferenced Objects” on page 14-7](#).

When two users attempt to rename the same object (or dependent object), the first rename operation to get to the object is applied. Subsequent rename operations can no longer find the target object, and therefore fail. An error message is displayed.

Effects on Dependent Objects

Some types of objects have names or signatures that are derived from the names of related objects. For example, the name of a class table or state table corresponds to the class it describes. When an object is renamed, StP automatically updates the names of any dependent objects.

Table 3 lists these automatic updates.

All the potential changes to an object’s dependencies appear in the Check Impact report, where each object is referred to by its application type. For the correspondences between symbols and application types, see [Appendix A, “PDM Types and Application Types.”](#)

Table 2: Rename Actions for UML Objects

Source Symbol (or Object)	Dependent Objects	Editor	What Happens to Dependent Objects upon Rename
Attribute	Object instance attribute value	Class Editor	Renames the “Value of” instance attributes for objects instantiated from the class containing the attribute in the repository and in the diagrams.
Class (in Class Editor or Class Table Editor)	Class table	Class Table Editor	Renames the file in the repository and in the file system.
Class (in Class Editor or Class Table Editor)	State diagram	State Editor	Renames the class portion of the name in the repository and renames the file in the file system. Renames the State Machine with the new class name.
	State table	State Table Editor	
	Association	Class Editor	Renames the association in the repository, but no change is visible in the diagrams.
	Operation	Class Editor Class Table Editor	Renames the class’s constructor and destructor functions in the repository and in the diagrams and tables.
	Operation’s return type	Class Table Editor	Renames the return type to match the new class name in the repository and in the tables.
Class (in an object symbol)	Object Class Scope	Class Editor Sequence Editor Collaboration Editor	Renames UmlObjectClassScope to reflect the reference to the new class. Does not change the original class name in the repository.

Table 2: Rename Actions for UML Objects (Continued)

Source Symbol (or Object)	Dependent Objects	Editor	What Happens to Dependent Objects upon Rename
Operation	Overloaded operation (differing from source operation only by signature)	Class Editor Class Table Editor	If Rename Overloaded Operations in Class is on, renames all matching operation names in the repository and references; the signatures are not affected. Renames all destructor operation names if the source operation happens to be a constructor.
RequirementCell Type (the cell containing the unique id of a requirement)	Requirement Allocation note, Analysis Requirement Id item	Annotation for a variety of objects	Changes the value of the note item to the new requirement id.
	Requirement Allocation note, Design Requirement Id item		
	Requirement Allocation note, Implementation Requirement Id item		
	Requirement Allocation note, Test Requirement Id item		
Use Case	Use case parent	Use Case Editor	Renames the parent in the repository and the diagram.
	Use case diagram		Renames the use case portion of the diagram name in the repository and renames the file in the file system.
	Extension Point	Sequence Diagram Collaboration Diagram	Renames the extension point to match the use case name.

The Visual Effects of Object Rename

When you run **Rename Object Systemwide** successfully, all references to the object in any diagram or table that is being edited are changed immediately. If several users are editing diagrams and tables, the rename is propagated to any open diagram or table with a reference to the object. This is true whether it is locked or not. It is not possible to lock a diagram or table against the rename operation.

The rename operation sends an alert message about the rename to each open editor containing a reference to the object on the current diagram or table. The message has the format:

```
<apptype> renamed from <oldname> to <newname> by <userid>
```

The message appears in the message area of the editor's window. For the correspondence of the application type <apptype> to the symbol, refer to [Appendix A, "PDM Types and Application Types."](#)

15 Generating UML Reports

This chapter describes reports you can generate for StP/UML. Topics covered are as follows:

- [“What Is in UML Reports?” on page 15-1](#)
- [“Running UML Reports” on page 15-2](#)
- [“Viewing UML Reports” on page 15-6](#)
- [“Changing External Variables” on page 15-7](#)

For complete instructions on running reports, including using the Script Manager, refer to [Query and Reporting System](#).

What Is in UML Reports?

Using the StP Query and Reporting System (QRS), you can generate UML reports. UML reports provide the following information:

- Requirements
 - Users View (Use Case Analysis)
 - System Architecture
 - Package Report
 - Class Report
 - Stereotype Definition
 - Glossary
-

Running UML Reports

You run UML reports:

- From the Script Manager, which is the main graphical user interface for the QRS
- Using a script created with the StP Query and Reporting Language (QRL)

Using External Variables in Reports

When running a report, you can use external variables to customize its scope. For a list of external variables associated with the UML report, see [“Changing External Variables” on page 15-7](#). For complete instructions on using external variables in reports, see [Query and Reporting System](#).

Using String Type External Variables

The default value for string type external variables is usually an empty string. The empty string means all, which refers to all of the objects of the selected type in the system. For example, if the string for classes is empty when you run a Class report, the report is for all classes in the system.

Generating Language-specific UML Reports

You can generate UML reports in English, French, or German. To view external variables in a specific language, choose the appropriate report from the Product Scripts folder in the Script Manager.

The available reports are:

- *UML_Report_D* (German version)
- *UML_Report_E* (English version)
- *UML_Report_F* (French version)

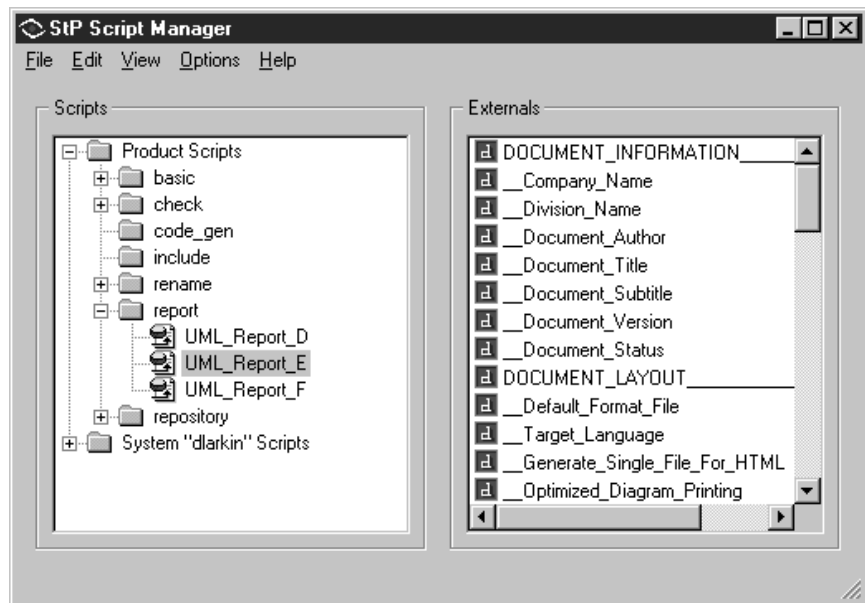
Running a Report from the Script Manager

To run a report from the Script Manager:

1. From the StP Desktop, choose **Report > Start Script Manager**.
2. In the Script Manager dialog box, select the Product Scripts folder.
3. Select the report folder.
4. Select the desired report (see [“Generating Language-specific UML Reports” on page 15-2](#) for more information).

The external variables associated with the UML report appear in the Externals window, as shown in Figure 12.

Figure 12: UML Report External Variables

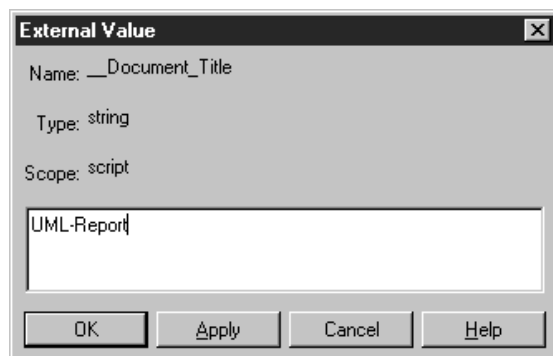


For information on external variables, see [“Changing External Variables” on page 15-7](#).

5. To change the default value of an external variable, click on the desired variable.

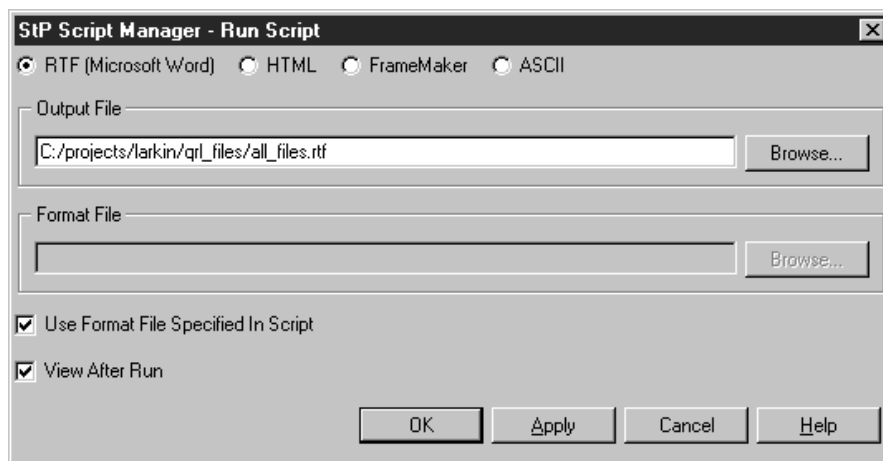
An External Value dialog box appears. The appearance of this dialog box depends on the selected variable.

Figure 13: External Value Dialog Example



6. Enter the value for the external variable, and click **OK** or **Apply**.
For additional details on setting external variables, see [Query and Reporting System](#).
7. Choose **File > Run Script**.
The Run Script dialog appears.

Figure 14: Run Script Dialog



8. Select settings and click **OK**.

Running a Report Using QRL

To run a report using QRL:

1. Create a script with the QRL.
2. Run the script with the **qrp** command.

For complete instructions on creating QRL scripts and using the **qrp** command, see [Query and Reporting System](#).

Format File, QRL Scripts, and Format Include Files

Table 1 lists the format file, qrl scripts, and format include files used in generating language-specific UML reports.

Table 1: StP-Supplied Files for Generating Reports

Language	Format File	QRL Script	Include File
English	<i>stpreport</i>	<i>UML_Report_E.qrl</i>	<i>ALL_Formats.inc</i>
German		<i>UML_Report_D.qrl</i>	
French		<i>UML_Report_F.qrl</i>	

These files are located in the StP installation directory in the following folders:

- Format file—*/templates/ct/print_format/<publishing tool>*
- QRL script—*/templates/uml/qrl/report*
- Format include file—*/templates/ct/qrl/include*

For more information on format files, qrl scripts, and format include files, see [Query and Reporting System](#).

Viewing UML Reports

The UML report provides information pertaining to an entire UML model. The report includes several chapters, described in Table 2.

To customize the scope of the UML report, use external variables, described in [“Changing External Variables” on page 15-7](#).

Table 2: UML Report Chapters

Chapter Name	Description
Introduction	Includes the front page, table of contents (RTF only), and introduction. To view the table of contents in RTF, update the table of contents in Microsoft Word.
Requirements	Displays the contents of the requirements table in alphabetical order.
Users View (Use Case Analysis)	Displays the Use Case diagrams in alphabetical order. Includes detailed information on actors, packages, use cases, and use case scenarios.
System Architecture	Displays the system architecture in logical (package hierarchy) and physical (component and deployment) implementations.
Package Report	Displays information pertaining to packages, including package hierarchy, contents, and connections to collaboration and sequence diagrams.
Classes	Displays classes in alphabetical order and includes class table descriptions and table diagrams. Includes related class information in sequence, collaboration, and activity diagrams.
Stereotype Definition	Displays stereotype diagrams in alphabetical order. Includes detailed information from stereotype annotations.
Glossary	Includes all glossary items from the system.

Changing External Variables

The external variables for UML reports are divided into four groups:

- Document information
- Document layout
- Document chapters
- Limiting the scope of evaluation

The subsequent sections describe the different groups, listing the external variables, including their type, scope, and default values.

Setting Document Information External Variables

Table 3 lists the external variables for displaying general information in a UML report.

Table 3: Document Information External Variables

Name	Description	Type	Scope	Default
DOCUMENT_INFORMATION__	Header only. Is not evaluated in a UML report.	Not applicable		
__Company_Name	The company name. If you want to use your company logo in the page headers, replace the AONIX logo in your format files.	string	Front page	AONIX Inc.
__Division_Name	The division name.			Development
__Document_Author	The author of the document.		Front page and page footers	Customization Team
__Document_Title	The document title.		Front page and page headers	UML-Report

Table 3: Document Information External Variables (Continued)

Name	Description	Type	Scope	Default
__Document_Subtitle	The document subtitle. The default is “for <StP-system>,” where <StP-system> is replaced by the name or your StP system. If you change this value, no substitution is performed and you will see only the text you entered.	string	Front page and page headers	for <StP - system>
__Document_Version	The version of the document.		Front page and page footers	V1.0
__Document_Status	The status of the document.	boolean	Front page	Analysis

Setting Document Layout External Variables

Table 4 lists the external variables for document layouts in a UML report.

Table 4: Document Layout External Variables

Name	Description	Type	Scope	Default
DOCUMENT_LAYOUT__	Header only. Is not evaluated in a UML report.	Not applicable		
__Default_Format_File	Name of the format file used for the document generation. See “Changing the Default Format File” on page 15-10 for more information.		Generated document	stpreport
__Target_Language	Language of the generated document. Supported languages are English (“E”), German (“D”), or French (“F”).	Enum: E/D/F	Generated document	E, D, or F (based on selected UML report)

Table 4: Document Layout External Variables (Continued)

Name	Description	Type	Scope	Default
__Generate_Single_File_For_HTML	For HTML, specifies whether to generate the report into a single file, or into multiple files. Multiple files correspond to pages in other publishing targets.	boolean	Generated document	False
__Optimized_Diagram_Printing	If True, uses optimized diagram printing. Otherwise, each diagram is printed on its own page.	boolean	All diagrams	True
__Use_Print_Setting	If you want to use your own print setting for diagrams and tables (use one name for all settings).	string	All diagrams and tables	Empty string
__Suppress_Displaymarks	You can suppress specified display marks. Use a blank space as a delimiter between the display mark names to be suppressed.	string	All diagrams	See External Value dialog box
__Order_Symbols_Alphabetically	If True, prints symbols in alphabetical order. If False, prints according to their vertical (diagram) and horizontal (table) position.	boolean	Generated document	False
__Include_Introductions	If True, prints an introduction paragraph at the beginning of each chapter (the text strings are pre-defined in <i>UML_Strings.inc</i>).			True
__Include_Descriptions	If True, prints the diagram, table, and object descriptions.			
__Notify_Missing_Descriptions	If True, prints remarks for missing diagram, table, or object descriptions.			

Table 4: Document Layout External Variables (Continued)

Name	Description	Type	Scope	Default
__Print_Multiple_Descriptions	If True, prints the description for every occurrence of an object.	boolean	Generated document	True
__Include_Generation_Information	If True, prints the author and date of generation (entered via the Object Annotation Editor) for use cases, classes, and diagrams.			False

Changing the Default Format File

You can replace the default format file with the name of any other existing format file (for the chosen format). If your format file contains other paragraph and character formats then *stpreport*, you have to adjust their names in the template file *ALL_Formats.inc*. For detailed information on format files, see [Query and Reporting System](#).

Setting Document Chapter External Variables

Table 5 lists the external variables for document chapters in a UML report.

Table 5: Document Chapter External Variables

Name	Description	Type	Scope	Default
DOCUMENT_CHAPTERS__	Header only. Is not evaluated in a UML report.	Not applicable		
__Generate_the_Requirement_Chapter	If True, generates the Requirements chapter.	boolean	script	True
__In_Requirement_Table_Form	If True, prints the requirements in table form. If False, prints in the form of structured text.			False

Table 5: Document Chapter External Variables (Continued)

Name	Description	Type	Scope	Default
__Show_ Unallocated_ Requirements	If True, marks unallocated requirements with a question mark (“?”). Available if output is in structured text format only.	boolean	script	False
__Include_ Superseded_ Requirements	If True, prints superseded requirements.			
__Generate_the_ Users_View_Chapter	If True, generates the Users View chapter.			True
__With_Detailed_ Use_Case_ Description	If True, prints detailed information about the symbols in the use case diagrams and use case scenarios.			False
__Include_Use_ Case_Scenarios	If True, prints sequence, collaboration, and activity scenarios for each use case.			True
__Generate_the_ System_Architecture_ Chapter	If True, generates the System Architecture chapter.			
__Include_logical_ Architecture	If True, generates a subchapter with the package structure.			
__Include_physical_ Architecture	If True, generates a subchapter with component and deployment information.			
__With_detailed_ Architecture_ description	If True, prints detailed information about the modules and processes.			
__Generate_the_ Package_Report_ Chapter	If True, generates the package report chapter.			

Table 5: Document Chapter External Variables (Continued)

Name	Description	Type	Scope	Default
___With_detailed_Package_Description	If True, prints detailed information about the packages.	boolean	script	False
___Include_Package_and_Class_Diagrams	If True, prints package and class diagrams.			True
__Generate_the_Class_Report_Chapter	If True, generates the Class Report chapter.			
___With_detailed_Class_Description	If True, prints detailed information about the class, including attributes, operations, associations, and related diagrams.			
___Include_Attributes_and_Operations	If True, prints information about the attributes and operations of the class.			
___In_Class_Table_Form	If True, prints attributes and operations in table format. If False, prints more detailed information in structured text format.			False
___With_Analysis_Information	If True, prints existing analysis information for attributes and operations.			
___With_Cxx_Information	If True, prints existing C++ information for attributes and operations.			
___With_Java_Information	If True, prints existing Java information for attributes and operations.			
___With_IDL_Information	If True, prints existing IDL information for attributes and operations.			

Table 5: Document Chapter External Variables (Continued)

Name	Description	Type	Scope	Default
___With_Ada95_Information	If True, prints existing Ada_95 information for attributes and operations.	boolean	script	False
___Include_Interaction_Diagrams	If True, prints sequence and collaboration diagrams with references to the class.			True
___Include_Behavior_Diagrams	If True, prints state and activity diagrams scoped to the class.			
___Include_the_Association_Summary_Table	If True, generates the association summary table, which lists class relationships, multiplicity, and descriptions.			
___Generate_the_Stereotype_Definition_Chapter	If True, generates the Stereotype Definition Chapter.			
___Include_Stereotype_Diagrams	If True, prints the stereotype diagrams at the beginning of the chapter.			False
___Generate_Glossary_Chapter	If True, generates the Glossary chapter.			
___For_Scope_of_Report_Only	If True, reduces the glossary to items defined for objects in the report.			

Setting Limit Evaluation External Variables

Table 6 lists the external variables for limiting the scope of the evaluation in a UML report.

Table 6: Limit Evaluation External Variables

Name	Description	Type	Scope	Default
LIMIT_EVALUATION__	Header only. Is not evaluated in a UML report.	Not applicable		
__To_Scope_of_Prefix	Limits the report to files and modeling elements matching the given prefix.	string	script	Empty string
__To_Packages	Limits the report to files containing the given packages and model elements contained in the given packages.			
__To_Classes	Limits the Class Report chapter to the given list of classes.			
__To_Stereotypes	Limits the Stereotype Definition chapter to the given stereotypes.			
__To_Requirement_Tables	Limits the Requirements chapter to the given list of requirement table files.	string	script	Empty string
__To_Use_Case_Diagrams	Limits the Users View chapter to the given list of use case diagram files.			
__To_Implementation_Diagrams	Limits the System Architecture chapter to the given list of component and deployment diagram files.			

A PDM Types and Application Types

This appendix describes how the StP/UML elements map to an StP application type and a Persistent Data Model type. These types determine how project information is stored in the StP Repository. Topics covered are as follows:

- [“Persistent Data Model Types” on page A-1](#)
- [“Application Types” on page A-2](#)
- [“Symbol Mappings” on page A-3](#)

For information about the Persistent Data Model types, see [Object Management System](#).

Persistent Data Model Types

The Persistent Data Model (PDM) is a scheme that defines the characteristics and behavior of objects in the StP repository in terms of 16 abstract, persistent data types. An abstract data type consists of a data structure and associated functions. Persistent data is any data that is stored in the StP repository or files and can outlive the process that produced it.

All data created by StP applications is stored in the repository as “CASE objects.” Each CASE object belongs to a PDM type and inherits attributes from the data structure of the corresponding PDM type.

For complete details about the PDM, see the [Object Management System](#).

Application Types

StP applications, such as StP/UML editors, provide capabilities for capturing information literally in table and text editors or symbolically in diagram editors. Each editor is an application. Each editor views information and accesses information in the repository as “application types.”

An application type is a collection of objects of the same type. All application types map to PDM Software Engineering type objects, such as:

- Node type
- Link type
- Cntx type
- File type
- Note type
- Item type

For example, class and attributes both map to the PDM node type. Some symbols, such as associations, map to more than one application and PDM type.

Each application type is unique within a PDM type and inherits attributes from the data structure of the corresponding PDM type. The application type’s name can be used in OMS queries, which can be used to populate the StP Repository Browser. For information on OMS queries, see [Object Management System](#).

The mapping between application types and the objects in the repository is provided by the *app.types* file.

The application type’s name appears as objects to be renamed in the Check Impact report of the **Rename Object Systemwide** command.

Symbol Mappings

This section lists all file, node, link, and cntx types accessible to StP/UML editors. For note, item, and viewpoint application types, see the StP/UML *app.types* file.

Use Case Diagram Mappings

Table 1 lists Use Case Editor application types.

Table 1: Use Case Editor Symbol Mappings

UML Construct	PDM Type	Application Type
Use Case	node	UmlUseCase
Package	node	UmlPackage
Actor	node	UmlActor
System	node	UmlSystem
Communicates	link	UmlUseCaseInteraction
Extends	link	UmlUseCaseExtends
Uses	link	UmlUseCaseUses
Inherits	link	UmlGeneralization

Class Diagram Mappings

Table 2 lists Class Editor application types.

Table 2: Class Editor Symbol Mappings

UML Construct	PDM Type	Application Type
Class	node	UmlClass
Package	node	UmlPackage
Attribute	node	UmlAttribute
Operation	node	UmlOperation
Parameterized Class	node	UmlClass
	node	UmlParameterizedClass
Instantiated Class	node	UmlClass
	node	UmlInstantiatedClass
Interface	node	UmlClass
Object	node	UmlObjectClassScope
	node	UmlObjectInstance
	node	UmlObjectInState
Instance Attribute	node	UmlObjectInstanceAttribute
Association	node	UmlAssociation
	link	UmlRole
Association Class Link	link	UmlAssociationClassLink
Generalization	link	UmlGeneralization
Dependency	link	UmlDependency
Implements	link	UmlImplements
Binds	link	UmlRefines

Table 2: Class Editor Symbol Mappings (Continued)

UML Construct	PDM Type	Application Type
Contains	link	UmlContains

Class Table Mappings

Table 3 lists Class Table Editor application types.

Table 3: Class Table Editor Symbol Mappings

Table Section	PDM Type	Application Type
Class	node	UmlClass
Attribute	node	UmlAttribute
	note	UmlAttributeDefinition
	item	UmlAttributeType
	item	UmlAttributeDefaultValue
	item	UmlMemberVisibility
	item	UmlMemberIsClass
	item	UmlAttributeIsDerived
	note	UmlAttributeCxxDefinition
	item	UmlMemberCxxIsConst
	item	UmlAttributeCxxIsVolatile
	item	UmlMemberCxxVisibility
	note	UmlAttributeJavaDefinition
	item	UmlMemberJavaVisibility
	item	UmlMemberJavaIsFinal
	item	UmlAttributeJavaIsVolatile

Table 3: Class Table Editor Symbol Mappings (Continued)

Table Section	PDM Type	Application Type
Attribute	item	UmlAttributeJavaIsTransient
	note	UmlAttributeAda95Definition
	item	UmlAttributeAda95IsAliased
	item	UmlMemberAda95Visibility
Operation	node	UmlOperation
	note	UmlOperationDefinition
	item	UmlOperationReturnType
	item	UmlMemberVisibility
	item	UmlMemberIsClass
	item	UmlOperationIsAbstract
	item	UmlOperationThrows
	note	UmlOperationCxxDefinition
	item	UmlMemberCxxIsConst
	item	UmlOperationCxxIsVirtual
	item	UmlOperationCxxIsInline
	item	UmlOperationCxxCtorInitList
	item	UmlOperationCxxVisibility
	note	UmlOperationJavaDefinition
	item	UmlOperationJavaVisibility
	item	UmlOperationJavaIsNative
	item	UmlOperationJavaIsSynchronized
	item	UmlMemberJavaIsFinal
	note	UmlOperationAda95Definition

Table 3: Class Table Editor Symbol Mappings (Continued)

Table Section	PDM Type	Application Type
Operation	item	UmlMemberAda95Visibility
	item	UmlOperationAda95IsSubunit
	item	UmlOperationAda95IsInline
	note	UmlOperationCxxCode
	note	UmlOperationJavaCode
	note	UmlOperationAda95Code
Signals Received	node	UmlSignal
	link	UmlSignalReceives
Signals Sent	node	UmlSignal
	link	UmlSignalSends

Sequence Diagram Mappings

Table 4 lists Sequence Editor application types.

Table 4: Sequence Editor Symbol Mappings

UML Construct	PDM Type	Application Type
Package	node	UmlPackage
PassiveObject, ActiveObject	node	UmlObjectClassScope
	node	UmlObjectInstance
	node	UmlObjectInState
Actor	node	UmlActor

Table 4: Sequence Editor Symbol Mappings (Continued)

UML Construct	PDM Type	Application Type
Scenario Instance	node	UmlScenarioInstance
	note	UmlScenarioInstanceNote
	item	UmlParentTypeItem
	item	UmlParentNameItem
Extension Point	node	UmlExtensionPoint
	note	UmlMessageDefinition
	item	UmlSequenceExpression
	item	UmlMessageRecurrence
Message (all variations)	note	UmlMessageNote
	item	UmlObjectName
	item	UmlPredecessorGuard
	item	UmlSequenceExpression
	item	UmlMessageRecurrence
	item	UmlMessageArgs
	item	UmlReturnType
Simple Message	link	UmlSimpleMessage
Return Message	link	UmlReturnMessage
Synchronous Message	link	UmlSynchronousMessage
Balking Message	link	UmlBalkingMessage
Timeout Message	link	UmlTimeoutMessage
Asynchronous Message	link	UmlAsynchronousMessage
Transition Name	cntx	UmlTimeSent
	cntx	UmlTimeDelivered

Table 4: Sequence Editor Symbol Mappings (Continued)

UML Construct	PDM Type	Application Type
Constraint	cntx	UmlConstraint

Collaboration Diagram Mappings

Table 5 lists Collaboration Editor application types.

Table 5: Collaboration Editor Symbol Mappings

UML Construct	PDM Type	Application Type
Package	node	UmlPackage
Object, Composite Object, MultiObject	node	UmlObjectClassScope
	node	UmlObjectInstance
	node	UmlObjectInState
Actor	node	UmlActor
Scenario Instance	node	UmlScenarioInstance
	note	UmlScenarioInstanceNote
	item	UmlParentTypeItem
Scenario Instance	item	UmlParentNameItem
Extension Point	node	UmlExtensionPoint
	note	UmlMessageDefinition
	item	UmlSequenceExpression
	item	UmlMessageRecurrence
Collaboration	node	UmlCollaboration

Table 5: Collaboration Editor Symbol Mappings (Continued)

UML Construct	PDM Type	Application Type
Active Object, Active Composite Object	node	UmlActiveObject
	node	UmlObjectInstance
	node	UmlObjectClassScope
	node	UmlObjectInState
Collaboration Context	node	UmlScenarioInstance
Type	node	UmlType
	node	UmlClass
Object Link	link	UmlObjectLink
Collaboration Role	link	UmlCollaborationRole
Message (all variations)	note	UmlMessageNote
	item	UmlObjectName
	item	UmlPredecessorGuard
	item	UmlSequenceExpression
	item	UmlMessageRecurrence
	item	UmlMessageArgs
	item	UmlReturnType
Simple Message	link	UmlSimpleMessage
Synchronous Message	link	UmlSynchronousMessage
Balking Message	link	UmlBalkingMessage
Timeout Message	link	UmlTimeoutMessage
Asynchronous Message	link	UmlAsynchronousMessage

State Diagram Mappings

Table 6 lists State Editor application types.

Table 6: State Editor Symbol Mappings

UML Construct	PDM Type	Application Type
State, Composite State, Scope Ambassador	node	UmlState
State Machine	node	UmlStateMachine
Initial state	node	UmlInitialState
Final state	node	UmlFinalState
Concurrent Subregion	node	UmlState
	note	UmlStateDefinition
	item	UmlConcurrentState
Split Control	node	UmlSplitControl
	link	UmlTransition
Merge Control	node	UmlMergeControl
	link	UmlTransition
Shallow History, Deep History	node	UmlHistoryState
Transitions	link	UmlTransition
	note	UmlTransitionDefinition
	item	UmlTransitionEvent
	item	UmlTransitionGuard
	item	UmlTransitionAction
	item	UmlTransitionSend

State Table Mappings

Table 7 lists State Table Editor application types.

Table 7: State Table Editor Symbol Mappings

Table Section	PDM Type	Application Type
State	node	UmlState
Entry Action	node	UmlAction
	link	UmlStateEntryAction
Activity	node	UmlActivity
	link	UmlStateActivity
Exit Action	node	UmlAction
	link	UmlStateExitAction
Internal Event	node	UmlAction
	link	UmlStateInternalAction
Internal Action	node	UmlAction
	link	UmlStateInternalAction
State Variable	node	UmlVariable
	link	UmlStateVariable
Deferred Event	node	UmlDeferredEvent
	link	UmlDeferredEventLink

Activity Diagram Mappings

Table 8 lists Activity Editor application types.

Table 8: Activity Editor Symbol Mappings

UML Construct	PDM Type	Application Type
Action State	node	UmlActionState
State Machine	node	UmlStateMachine
Initial State	node	UmlInitialState
Final State	node	UmlFinalState
Decision	node	UmlActionState
	item	UmlStereotype
Object	node	UmlObjectClassScope
	node	UmlObjectInstance
	node	UmlObjectInState
Split Control	node	UmlSplitControl
	link	UmlTransition
Merge Control	node	UmlMergeControl
	link	UmlTransition
Transition	link	UmlTransition
Input Flow	link	UmlInput

Component Diagram Mappings

Table 9 lists Component Editor application types.

Table 9: Component Editor Symbol Mappings

UML Construct	PDM Type	Application Type
Binary	node	UmlComponentBinary
Executable	node	UmlComponentExecutable
Source	node	UmlComponentSource
Objects	node	UmlComponentObjects
Interface	node	UmlComponentInterface
	link	UmlComponentInterface
Dependency	link	UmlComponentDependency
Composition	link	UmlComponentComposit

Deployment Diagram Mappings

Table 10 lists Deployment Editor application types.

Table 10: Deployment Editor Symbol Mappings

UML Construct	PDM Type	Application Type
Binary	node	UmlComponentBinary
Executable	node	UmlComponentExecutable
Objects	node	UmlComponentObjects
Interface	node	UmlComponentInterface
	link	UmlComponentInterface
Node	node	UmlDeploymentNode

Table 10: Deployment Editor Symbol Mappings (Continued)

UML Construct	PDM Type	Application Type
Dependency	link	UmlComponentDependency
Deployment	link	UmlDeployment
Composition	link	UmlComponentComposit

Stereotype Diagram Mappings

Table 11 lists Stereotype Editor application types.

Table 11: Stereotype Editor Symbol Mappings

UML Construct	PDM Type	Application Type
Stereotype	node	UmlStereotype
Inheritance	link	UmlStereotypeInheritance

B Command Line Commands

StP provides a command line interface for running various commands available from the StP Desktop, diagram editors, and table editors.

The table in this section gives command syntax for starting the StP Desktop and each StP/UML editor entered at the command line prompt. Variables appear with angle brackets (<>); optional parts of commands appear within square brackets ([]).

For a list of all StP commands supported from the command line, see [StP Administration](#).

Table 1: Commands

To	Type	Comments
Start the StP Desktop	<code>stp [&]</code>	
Start the Class Editor	<code>gde -ed uclassd [-p <proj>] [-s <sys>] [<diagram>] [&]</code>	Optional arguments: proj is the name of the project directory sys is the system name diagram is the diagram name; table is the table name.
Start the Class Table Editor	<code>gte -ed uclasst [-p <proj>] [-s <sys>] [<table>] [&]</code>	
Start the Sequence Editor	<code>gde -ed usequenced [-p <proj>] [-s <sys>] [<diagram>] [&]</code>	
Start the Collaboration Editor	<code>gde -ed ucollaborationd [-p <proj>] [-s <sys>] [<diagram>] [&]</code>	
Start the Use Case Editor	<code>gde -ed uusecased [-p <proj>] [-s <sys>] [<diagram>] [&]</code>	

Table 1: Commands (Continued)

To	Type	Comments
Start the State Editor	<code>gde -ed ustatet [-p <proj>] [-s <sys>] [<diagram>] [&]</code>	<p>Optional arguments: proj is the name of the project directory sys is the system name diagram is the diagram name; table is the table name.</p>
Start the State Table Editor	<code>gte -ed ustatet [-p <proj>] [-s] <sys>] [<table>] [&]</code>	
Start the Activity Editor	<code>gde -ed uactivityd [-p <proj>] [-s <sys>] [<diagram>] [&]</code>	
Start the Component Editor	<code>gde -ed ucomponentd [-p <proj>] [-s <sys>] [<diagram>] [&]</code>	
Start the Deployment Editor	<code>gde -ed udeploymentd [-p <proj>] [-s <sys>] [<diagram>] [&]</code>	
Start the Stereotype Editor	<code>gte -ed ustereod [-p <proj>] [-s <sys>] [<diagram>] [&]</code>	

Index

A

abstract classes 4-26

abstract operations

description 4-18

inherited as concrete operations from
superclass 4-18

action states

description 10-7

final states 10-9

initial states 10-8

actions

definition 8-13

in activity diagrams 10-9

on state transition link label 8-11

active composite objects 7-13

active objects 7-13

activities

in a state table 9-6

Activity Diagram for Class

command 4-5, 5-4

Activity Diagram for Operation

command 4-6, 5-4

Activity Diagram Where Class Is

Referenced command 6-4, 7-4

activity diagrams

action states 10-7

actions 10-9

decisions 10-10

definition 1-4, 10-1

extensibility mechanisms 10-15

guard condition 10-9

objects 10-12

state machine 10-7

state transition links 10-9

state transition syntax 10-9

Activity Diagrams tool commands 2-4

Activity Editor

description 10-2

mappings to the StP repository A-13

navigating to other editors 10-3

symbol palette 10-5

UML menu 10-5

actors

in collaboration diagrams 7-14

in sequence diagrams 6-11

in use case diagrams 3-7

Ada_95 Implementation Items 5-11

Aggregation Indicators display mark 4-9

aggregations

composition 4-45

description 4-44

Align All Objects command 6-7, 6-18

Align Links command 10-10

All Aggregations View command 4-5,
4-59

All Associations View command 4-5,
4-59

All Dependencies View command 4-5,
4-59

All Generalizations View command 4-5,
4-59

All Members View command 4-5, 4-59

AllAggregations View Point 4-58
AllAssociations View Point 4-58
AllDependencies View Point 4-58
AllGeneralizations View Point 4-58
AllMembers View Point 4-58
Allocate Requirements command 3-4,
4-6, 5-5, 6-5, 7-5, 8-5, 9-4, 10-4
Analysis Items 5-10, 5-13
annotations
 adding to model 1-9
 source code notes 5-12
Aonix
 documentation comments xx
 Technical Support xix
 websites xx
application types A-2
association classes 4-36
associations
 constraints 4-48
 creating 4-31
 description 4-33
 drawing in class diagrams 4-33
 labels 4-33
 multiplicity 4-42
 n-ary 4-34
 Or associations 4-37
 ordered multiplicity 4-47
 Properties dialog box 4-39
 reflexive 4-34
 stereotypes 4-47
 tagged values 4-48
 unidirectional 4-46
asynchronous messages 6-13, 7-16
AttributeIsDerived display mark 4-9
attributes
 adding to classes 4-15
 class 4-14
 derived attributes 4-16
 description 4-14
 effects of rename on dependent
 objects 14-18
 in class tables 5-6
 labels 4-14
 link 4-36

 naming rules 4-13
Attributes and Operations command 4-8
AttributeTypeandDefaultValue display
 mark 4-9
automatic alignment 6-17

B

balking messages 6-13, 7-16
binary component 11-6
Binds Arguments display mark 4-10
binds link
 binds arguments 4-52
 definition 4-51

C

C++ Implementation Items 5-11, 5-14
C++ Inheritance is Virtual property 4-54
C++ Inheritance Visibility property 4-54
Check Impact option
 description 14-15 to 14-16
 Rename Object Systemwide 14-13
Check RE Semantic Model Consistency
 command 2-9
Check RE Semantic Model Locks
 command 2-9
Check Semantics command 3-20, 4-61,
5-18, 6-28, 7-21, 8-22, 10-16, 11-11,
12-9, 13-10
Check Syntax command 3-20, 4-60, 6-27,
7-21, 8-22, 10-16, 11-11, 12-9, 13-10
 Class Table Editor 5-18
 State Table Editor 9-9
class attributes 4-14
Class Diagram command 5-4, 7-5
Class Diagram for Class command 8-4,
9-4, 10-4
Class Diagram Where Class Is Referenced
 command 6-4, 7-4
class diagrams
 adding parameters 4-27
 definition 1-3
 extensibility mechanisms 4-28, 4-39,
4-53

-
- interfaces 4-20
 - Properties dialog box 4-22
 - stereotypes 4-28
 - tagged values 4-29
 - updating 4-61
 - Class Diagrams tool commands 2-4
 - Class Editor
 - description 4-2
 - display marks 4-9
 - mappings to the StP repository A-4
 - navigating to other editors 4-4
 - Package Scope display mark 4-57
 - starting 4-4
 - Symbol Palette 4-7
 - Symbols scrolling list 4-7
 - UML menu commands 4-7
 - Class Member Definitions 5-10
 - class names
 - conventions 4-13
 - in objects 6-10, 7-11
 - class operations 4-16
 - class packages
 - containment 4-56
 - description 4-55
 - drawing 4-56
 - identifying 4-55
 - with public classes 4-57
 - Class Table command 4-5
 - Class Table Editor
 - description 5-2
 - hiding and showing sections of 5-8
 - horizontal sections 5-9
 - mappings to the StP repository A-5
 - navigating to other editors 5-3
 - parts of 5-6
 - UML menu 5-5
 - vertical sections 5-10
 - Class Table for Class command 8-4, 9-4, 10-4
 - Class Table Where Class Is Referenced command 6-4, 7-4
 - class tables
 - deleting 5-19
 - inherited operations 5-8
 - signals 5-8
 - table sections 5-8
 - updating 5-19
 - using to define classes 5-12
 - validating 5-18
 - Class Tables tool commands 2-4
 - class visibility 4-25
 - ClassConstraints display mark 4-28
 - classes
 - abstract classes 4-26
 - association or link 4-36
 - changing in class tables 5-14
 - constructing from a class diagram 5-14
 - constructing from the repository 5-14
 - defining in class tables 5-12
 - deleting 5-19
 - description 4-12
 - drawing diagrams 4-12
 - drawing in class diagrams 4-13 to 4-14
 - effects of rename on dependent objects 14-18
 - external classes 4-27
 - in class tables 5-6
 - inheriting abstract operations from superclasses 5-15
 - instantiated classes 4-18
 - labels 4-13
 - naming rules 4-13
 - parameterized 4-18
 - placing constraints on 4-28
 - Properties dialog box 4-22
 - updating 4-61
 - visibility 4-25
 - Classes tool commands 2-5
 - ClassInfo display mark 4-6
 - ClassInfo display
 - mark(UmlClassIsAbstract) 4-10
 - ClassInfo display
 - mark(UmlTaggedValue) 4-10
 - ClassParameters display mark 4-10
 - ClassScoping display mark 4-10
 - Collaboration Context command 7-5, 7-10
 - Collaboration Diagram command 6-4
-

Collaboration Diagram Where Class Is
Used command 8-4, 9-4, 10-4

collaboration diagrams

active composite objects 7-13

active objects 7-13

actors 7-14

collaboration symbol 7-9

composite objects 7-12

creating 7-8

creating object diagrams 7-14

definition 1-4

extensibility mechanisms 7-18

extension points 7-19

from a sequence diagram 7-20

messages 7-15

MultiObject 7-13

packages 7-19

Collaboration Diagrams tool

commands 2-4

Collaboration Editor

description 7-2

display marks 7-7

mappings to the StP repository A-9

navigating to other editors 7-3

starting 7-3

Symbol Palette 7-6

UML menu 7-6

CollaborationInfo display mark 7-8

communicates link 3-7

component diagrams

binary component 11-6

creating deployment diagrams 11-10

definition 1-5, 11-1

dependency relationships 11-9

drawing 11-5

drawing interfaces 11-8

executable component 11-5

extensibility mechanisms 11-10

interface component 11-6

labeling components 11-6

object component 11-6

source component 11-5

symbols 11-3

Component Diagrams tool

commands 2-4

Component Editor

display marks 11-4

mappings to the StP repository A-14

starting 11-3

Symbol Palette 11-3

using 11-2

components, labeling 11-6

composite aggregation 4-45

composite objects 7-12

concurrent activities 8-17 to 8-19

constraints 3-12, 4-28, 4-48, 6-21

Constraints display mark 4-11

constraints on associations 4-48

Construct Category by Adding Selectively
(Global) command 4-62

Construct Category from All Components
command 4-62

Construct Class by Adding Selectively
command 4-62

Construct Class from All Definitions
command 4-62, 5-5, 5-15, 5-19

Construct Class from Class Table
Definitions command 4-62

Construct Class from Diagram Definitions
command 5-5, 5-15, 5-19

Construct from Reverse Engineering
command 4-62

constructors

renaming with class 14-14

control flows

merging 8-19, 10-14

splitting 8-18, 10-13

synchronization 8-18 to 8-19,
10-13 to 10-14

Copy Diagram command 2-7

Copy System command 2-7

Copy Table command 2-7

Create Association Class command 4-8

Create Association Roles command 4-8,
4-38

Create N-ary Association command 4-8,
4-35

Create Or Association command 4-8,
4-37

creation
adding 6-22

Creation display mark 6-8

D

decisions 10-10

decomposing
state diagrams 8-5
states 8-14

deep history 8-20

Default Arc Type command 4-32

deferred events, in a state table 9-6

Delete Diagram command 2-7

Delete Table command 2-7

Delete Unreferenced Objects in Current
System Repository
command 14-7

dependency link 4-52

Deploy Component Diagram
command 12-8

deployment diagrams
definition 1-5, 12-1
deploying component diagrams 12-8
extensibility mechanisms 12-7
hardware components 12-5
hardware connections 12-7
nodes 12-7
symbols 12-3

Deployment Diagrams tool
commands 2-4

Deployment Editor
description 12-2
display marks 12-4
mappings to the StP repository A-14
navigating to object references 12-3
starting 12-3
Symbol Palette 12-3
UML menu 12-4

derived attributes 4-16

desktop, *See* StP Desktop

Destroy System command 2-8

destruction

adding 6-22

Destruction display mark 6-8

destructors

renaming with class 14-14

display marks

Aggregation Indicators 4-9

AttributeIsDerived 4-9

AttributeTypeandDefaultValue 4-9

Binds Arguments 4-10

ClassConstraints 4-28

ClassInfo 4-6

ClassInfo, UmlClassIsAbstract 4-10

ClassInfo, UmlTaggedValue 4-10

ClassParameters 4-10

ClassScoping 4-10

CollaborationInfo 7-8

Constraints 4-11

Creation 6-8

Destruction 6-8

Navigability 4-11

OperationReturnType 4-11

Qualifiers 4-11, 4-44

RefinedStateExists 8-5, 8-7

RoleMultiplicity 4-11

SequenceInfo 6-9

StateTableExists 8-5, 8-8

Stereotype 6-9, 7-8, 8-8, 10-6

StereotypeLink 6-9, 7-8, 8-8, 10-6

Stereotypes 3-6, 4-12

UseCaseInfo 3-6

UseCaseScenariosExist 3-6

Visibility 4-12

Distribute All Arcs command 6-7, 6-18

dynamic model 1-2

E

editors

Activity Editor 10-2

Class Table Editor 5-2

Collaboration Editor 7-2

Component Editor 11-2

Deployment Editor 12-2

Sequence Editor 6-2

State Editor 8-2

- State Table Editor 9-2
- Stereotype Editor 13-2
- Use Case Editor 3-2
- entry actions, in a state table 9-6
- executable component 11-5
- exit actions, in a state table 9-6
- Exit command 1-8
- Exit Desktop command 2-8
- Exit StP command 2-8
- Expand All Object to Selected Size command 6-7
- Expand All Objects 50% command 6-7, 6-18
- Expand All Objects to Selected Size command 6-18
- extends link 3-7
- extensibility mechanisms
 - defined 3-12
 - in activity diagrams 10-15
 - in class diagrams 4-28, 4-39, 4-53
 - in collaboration diagrams 7-18
 - in component diagrams 11-10
 - in deployment editor 12-7
 - in sequence diagrams 6-24
 - in state diagrams 8-21
 - in stereotype diagrams 13-6
 - in use case diagrams 3-12
- extension points 6-25, 7-19
- external classes 1-6, 4-27
- external variables, *See* UML reports
- Extract Source Code Comments command 2-9

F

- final states 8-10, 10-9

G

- Generalization Hierarchy command 4-8
- generalizations
 - creating from Reverse Engineering 4-51
 - description 4-50

- Generate Ada 95 for Diagram's Classes command 2-9
- Generate Ada 95 for Whole Model command 2-9
- Generate C++ for Diagram's Classes command 2-8
- Generate C++ for Whole Model command 2-8
- Generate Collaboration from Sequence command 7-7, 7-20
- Generate IDL for Diagram's Classes command 2-8
- Generate IDL for Whole Model command 2-8
- Generate Java for Diagram's Classes command 2-8
- Generate Java for Whole Model command 2-8
- Generate Model from Parsed Source Files command 2-9
- Generate Model from TOOL Code command 2-9
- Generate Sequence from Collaboration command 6-6, 6-27
- Generate TOOL for Diagram's Classes command 2-9
- Generate TOOL for Whole Model command 2-9
- generating code 1-6
- guard condition 10-9

H

- hardware connections 12-7
- Hide/Show command
 - Class Table Editor 5-9
 - State Table Editor 9-6
- history in states 8-20

I

- IDL Implementation Items 5-11
- implementation diagrams
 - definition 1-4
- implements link 4-52

- inherited operations 5-8
- inherits link 3-7
- initial states 8-10, 10-8
- Insert Rows Before command 9-5
- inserting messages 6-16
- instantiated classes 4-18
 - associations 4-20
 - role navigability 4-20
- interaction diagrams
 - definition 1-3
- interface 4-20
- interface component 11-6
- internal events, in a state table 9-6

J

- Java Implementation Items 5-11

L

- Label with Predefined Stereotype
 - command 13-5
- link attributes
 - adding to links in class diagrams 4-36
 - description 4-36
- link classes 4-36
- logical model 1-2

M

- Make Object Active command 7-13
- Make Object Active/Passive
 - command 7-7
- Make Object Passive command 7-13
- Merge Control command 8-7, 8-19, 10-5, 10-14
- merging
 - control flows 8-19, 10-14
- message arguments property 6-21
- message recurrence property 6-21
- message sequence property 6-21
- messages
 - adding details 6-19, 7-17
 - asynchronous 6-13, 7-16
 - balking 6-13, 7-16

- constraints 6-21
- creating 6-12, 7-15
- creation 6-22
- destruction 6-22
- inserting 6-16
- labeling 6-15
- message arguments 6-21
- name component 6-21
- predecessor guard 6-21
- Properties dialog box 6-19
- recurrence 6-21
- renaming 14-10
- reordering 6-17
- Replace command 6-16
- resequencing 6-16
- return 6-13, 7-16
- return type 6-21
- reversing direction 7-17
- sequencing 6-21
- simple 6-13, 7-16
- stereotype 6-21
- supported syntax 6-15
- synchronous 6-13, 7-16
- tagged values 6-21
- timeout 6-13, 7-16
- transition times 6-23

- MultiObject 7-13

- multiplicity
 - description 4-42
 - ordered 4-47

N

- name component property 6-21
- n-ary associations
 - and labels 4-34
 - creating 4-34
 - with classes 4-37
- Navigability display mark 4-11
- New command 2-7
- New System command 2-7
- nodes 12-7
- notes
 - Requirement Allocation 3-17
 - Source Code 5-12

O

- object component 11-6
- object diagrams 4-31, 7-14
- object instances
 - defining 4-30
 - drawing in class diagrams 4-30
- objects
 - in activity diagrams 10-12
 - in class diagrams 4-29
 - in sequence diagrams 6-10
 - renaming 14-10
 - setting class names 6-10, 7-11
- On Selection command 1-9
- Open command 1-8
- Open Diagram command 2-7
- Open System command 2-7
- Open Table command 2-7
- Operation command 8-4, 10-4
- OperationReturn type display mark 4-11
- operations
 - abstract 4-18, 5-15
 - adding to classes 4-17
 - class 4-16
 - description 4-16
 - effects of rename on dependent
 - objects 14-19
 - in class tables 5-7
 - inherited 5-8
 - labels 4-16
 - naming rules 4-13
 - overloaded 14-15
 - renaming constructors and destructors 14-14
 - renaming overloaded operations 14-15
- Or associations 4-37
- ordering 4-47
- overloaded operations
 - renaming 14-15

P

- Package Scope display mark 4-57
- packages
 - in collaboration diagrams 7-19

- in sequence diagrams 6-26
 - in use case diagrams 3-7, 3-10
- See also class packages
- Packages and Classes command 4-8
- parameterized classes
 - adding parameters 4-27
 - associations 4-20
 - drawing 4-18
 - role navigability 4-20
- Parent Collaboration command 7-5
- Parent command 8-5, 8-16
- Parent Use Case command 6-4, 7-4
- Parse Source Code command 2-9
- PDM See Persistent Data Model
- Persistent Data Model 1-10, A-1
- physical model 1-2
- predecessor guard property 6-21
- pre-defined stereotypes 13-6
- Print command 1-8
- Print Diagram As command 2-7
- Print Diagram command 2-7
- Print Table As command 2-7
- Print Table command 2-7
- printing 1-6
- Properties command 3-13, 4-23, 4-40, 4-54, 6-20, 6-24, 7-18, 8-21, 10-15, 11-10, 12-8
- Properties dialog box 3-13, 4-22, 4-39, 4-53, 6-19

Q

- qualified associations
 - description 4-43
- Qualifier display mark 4-44
- Qualifiers display mark 4-11

R

- Refine command 8-4, 8-14, 8-15
- RefinedStateExists display mark 8-5, 8-7
- refines link
 - Replace command 4-51

- reflexive associations
 - and roles 4-34
 - inserting 4-34
- Refresh Inherited Operations Section
 - command 5-6, 5-17
- relationships
 - creating 4-31
 - default 4-32
- rename
 - changing a label 14-7
 - how information is stored 14-2
 - renaming objects in a system 14-8
 - retyping a label 14-4
- Rename Diagram command 2-7
- Rename Object Systemwide
 - alert message 14-20
 - Check Impact option 14-13, 14-15 to 14-16
 - command options 14-11
 - effects on dependent objects 14-17 to 14-19
 - New Label option 14-13
 - rename candidates 14-9
 - Rename Class dialog 14-12
 - Rename Constructors for Class option 14-14
 - Rename Object dialog box 14-11
 - Rename Operation dialog 14-12
 - Rename Overloaded Operations in Class option 14-14, 14-19
 - renaming instances 14-10
 - visual effects 14-20
- Rename Object Systemwide
 - command 14-2, 14-8, 14-10
- Rename Table command 2-7
- reordering messages 6-17
- Replace command 4-32
- repository
 - mapping of objects 1-11
- Requirement Allocation note 3-17
- Requirement note 3-17
- requirement unique identifiers
 - effects of rename 14-19
- requirements model 1-2

- Requirements Tables tool commands 2-4
- Resequencing All Messages command 6-7, 6-16
- resequencing messages 6-16
- return messages 6-13, 7-16
- return type property 6-21
- Reverse Direction command 7-7, 7-17
- reverse engineering 1-6
- role names
 - adding to associations in class diagrams 4-38
- role navigability 4-20
- RoleMultiplicity display mark 4-11
- roles on associations 4-38

S

- Save As command 1-8
- Save command 1-8
- Scenario in Activity Diagram
 - command 3-4, 3-19
- Scenario in Collaboration Diagram
 - command 3-4, 3-19
- Scenario in Sequence Diagram
 - command 3-4, 3-19
- Scenario Involving Actor command 6-4, 7-4
- scope parent
 - decomposed state 8-16
- Sequence Diagram command 7-5
- Sequence Diagram Where Class Is Used
 - command 8-4, 9-4, 10-4
- sequence diagrams
 - actors 6-11
 - definition 1-3
 - drawing 6-9
 - extensibility mechanisms 6-24
 - extension points 6-25
 - from a collaboration diagram 6-27
 - messages 6-12
 - objects, active 6-10
 - objects, passive 6-10
 - use case scenarios 6-5
- Sequence Diagrams tool commands 2-4

Sequence Editor

- automatic alignment 6-17
- description 6-2
- display marks 6-8
- mappings to the StP repository A-7
- messages 6-16
- navigating to other editors 6-3
- starting 6-3
- Symbol Palette 6-5
- UML menu 6-6

SequenceInfo display mark 6-9

Set Label command 5-12

Set Message Name command 6-7, 6-15, 7-7, 7-17

Set Message Name dialog 6-14, 7-16

shallow history 8-20

Shift Messages Down from Selected command 6-7, 6-16

Sibling Collaboration Scenario command 7-5

Sibling Use Case Scenario command 6-4, 7-4

signals 5-8

simple messages 6-13, 7-16

Sort Messages Based on Sequence command 6-7, 6-17

Source Code command 4-6, 5-4

Source Code Declaration command 4-6, 5-4

Source Code Definition command 4-6, 5-5

source component 11-5

specialized relationships 4-53

Split Control command 8-7, 8-18, 10-5, 10-13

splitting

- control flows 8-18, 10-13

Starting the StP Desktop 2-2

State Diagram command 7-5, 9-4

State Diagram for Class command 4-5, 5-4

State Diagram for Operation command 4-6, 5-4

State Diagram Where Class Is Referenced

- command 6-4, 7-4

state diagrams

- decomposing 8-5
- definition 1-4
- extensibility mechanisms 8-21
- state machine 8-8

State Diagrams tool commands 2-4

State Editor

- description 8-2
- display marks 8-7, 10-6
- mappings to the StP repository A-11
- navigating from the STE 9-4
- navigating to other editors 8-3
- symbol palette 8-6
- UML menu 8-6

state machine

- in activity diagrams 10-7
- in state diagrams 8-8

State Table command 8-4, 8-13

State Table Editor

- description 9-2
- mappings to the StP repository A-12
- navigating to other editors 9-3

state tables

- creating 9-7
- deleting 9-10
- hiding/showing sections 9-6
- parts of 9-5
- sorting 9-8
- validating 9-9

State Tables tool commands 2-4

state transition links

- discussion 8-11 to 8-13
- in activity diagrams 10-9
- notation syntax 8-11

state variables, in a state table 9-6

states

- creating state tables 9-7
- decomposing states 8-14
- deep history 8-20
- defining in state tables 9-7 to 9-8
- description 8-9
- final states 8-10

- in a state table 9-5
- initial states 8-10
- labels 8-9
- shallow history 8-20
- StateTableExists display mark 8-5, 8-8
- static model 1-2
- stereotype 3-12, 4-28, 4-47, 6-21
- Stereotype Definition command 4-5
- stereotype diagrams
 - definition 1-5
 - drawing 13-5
 - extensibility mechanisms 13-6
- Stereotype Diagrams tool commands 2-4
- Stereotype display mark 6-9, 7-8, 8-8, 10-6
- Stereotype Editor
 - description 13-2
 - display marks 13-4
 - mappings to the StP repository A-15
 - navigating to object references 13-3
 - pre-defined stereotypes 13-6
 - starting 13-3
 - Symbol Palette 13-3
 - UML menu 13-4
- StereotypeLink display mark 6-9, 7-8, 8-8, 10-6
- Stereotypes display mark 3-6, 4-12
- StP Desktop
 - description 2-1
 - list of tool commands 2-7, 2-8
 - list of tools 2-4
 - starting 2-2
- StP Repository
 - object to symbol mapping 1-11
- synchronization of control
 - flows 8-18 to 8-19, 10-13 to 10-14
- synchronous messages 6-13, 7-16

T

- tagged values 3-12, 4-29, 4-48, 6-21
- Technical Support xix
- this system
 - in use case diagrams 3-7

- time constraints 6-23
- timeout messages 6-13, 7-16
- TOOL Implementation Items 5-11
- transition times 6-23
- Turn Auto Alignment Off command 6-17
- Turn Auto Alignment On command 6-7, 6-17

U

- UML menu
 - Activity Editor 10-5
 - Class Editor 4-7
 - Class Table Editor 5-5
 - Collaboration Editor 7-6
 - Deployment Editor 12-4
 - Sequence Editor 6-6
 - State Editor 8-6
 - Stereotype Editor 13-4
- UML reports
 - chapters of 15-6
 - external variables in 15-2, 15-7 to 15-14
 - language-specific 15-2
 - running from Script Manager 15-3
- unidirectional associations 4-46
- UNIX command line commands B-1
- Update from Reverse Engineering
 - command 5-6
- Use Case Diagram Involving Actor
 - command 3-4, 6-4, 7-4
- use case diagrams
 - creating 3-6
 - definition 1-3
 - extensibility mechanisms 3-12
 - packages 3-10
 - this system 3-11
- Use Case Diagrams tool commands 2-4
- Use Case Editor
 - description 3-2
 - display marks 3-5
 - mappings to the StP repository A-3
 - navigating to other editors 3-3
 - symbol list 3-5
 - validating diagrams 3-20

- Use Case Scenario Involving Actor
 - command 3-4
- Use Case Scenario Involving External
 - Event command 3-4
- use case scenarios 3-18, 6-5
- Use Case with Same Name
 - command 8-4, 10-4
- use cases
 - annotating 3-15
 - effects of rename on dependent objects 14-19
 - in use case diagrams 3-7
 - symbols 3-5
- UseCaseInfo display mark 3-6
- UseCaseScenariosExist display mark 3-6
- uses link
 - in use case diagrams 3-7

V

- validations
 - Check Semantics command 3-20, 4-61, 5-18, 6-28, 7-21, 8-22, 10-16, 11-11, 12-9, 13-10
 - Check Syntax command 3-20, 4-60, 6-27, 7-21, 8-22, 10-16, 11-11, 12-9, 13-10
 - class tables 5-18
- value of attributes symbol 4-30
- View Points
 - creating 4-58
 - definition 4-57
- ViewPoints
 - navigating 4-59
- Visibility display mark 4-12

W

- Where Stereotype Is Used
 - command 13-3, 13-6