Software through Pictures®

Millennium Edition

Features Supplement

UD/UG/ST0000-10119/002

Aonix

Features Supplement

Millennium Edition

March 2002

Trademarks

Aonix

World Headquarters
5040 Shoreham Place
San Diego, CA 92122, Suite 100
Phone: (800) 97-AONIX
Fax: (858) 824-0212
E-mail: info@aonix.com
http://www.aonix.com

Table of Contents

1 What's in StP ME?

Software through Pictures Millennium Edition (StP ME) provides system architects, analysts, designers, and developers with the powerful visual modeling and deployment environment required to address the complex needs of today's e-solution development. StP ME introduces major enhancements and features that provide a complete enterprise-wide modeling solution to visualize, design, implement, and maintain your next-generation systems and applications.

The *Features Supplement* describes features of StP ME that are new to or updated in this release, as well as capabilities introduced in recent releases that are not currently documented elsewhere in the StP document set.

Features Summary

Table 1 broadly summarizes the features that have been introduced in the Millennium Edition (8.0 and beyond) and indicates the chapter in this *Features Supplement* in which they are discussed.

Table 1: StP ME Features

StP ME Feature	For More Information, See
Globally unique identifiers	"Globally Unique IDs" on page 1-7
View-based StP/UML desktop model and repository browser	Chapter 2, "StP/UML View-Based Desktop"

Table 1: StP ME Features

StP ME Feature	For More Information, See
Multi-user model management in StP / UML	Chapter 3, “Model Management”
Model baselining and a version control interface to popular configuration management products	Chapter 4, “Baselining and Version Control”
OMG UML 1.3 compliancy updates for StP / UML	Chapter 5, “Usability and Modeling Enhancements”
Extensive usability enhancements	
Extensive editor enhancements	Chapter 6, “StP / UML Editor Enhancements”
Ability to export and import StP / UML models in XML Metadata Interchange language format	Chapter 7, “Exporting and Importing UML Models and Code”
Additional interfaces to SNiFF and Visual Studio programming environments	
Options for reverse engineering StP models from Visual Studio project files or source files matching specified criteria in an entire directory tree	
Report Maker Editor	Chapter 8, “Report Maker Editor (RME)”

Summary of Desktop Commands

Table 2 lists commands on the StP desktop that have been introduced since the first StP ME release.

Table 2: StP ME Desktop Commands

Menu	Modeling Tool	Submenu or Command	Description
Code > *<language>*	StP/UML	**Generate** *<language>* **by Template**	Uses StP's Architecture Component Development (ACD) domain-specific templates to generate more efficient code for your application. For more information, see "Generating UML Solutions with Architecture Component Development" on page 7-3.
		Open TDL Template	Opens the user-specified ACD/TDL template for editing. For more information, see "Generating UML Solutions with Architecture Component Development" on page 7-3.
Code > XMI		**Export XMI Model** **Import XMI Model**	Facilitates the interchange of metadata between various UML modeling tools and applications. For more information, see "Importing and Exporting UML Models with XMI" on page 7-3.
Tools		**Multiple Global Rename**	Globally renames all packages, classes, operations, attributes, use cases, or actors in the repository whose names match a user-specified search pattern. For more information, see Chapter , "StP/UML Global Rename with Pattern Matching".
Tools > Baseline	StP/SE StP/UML	**Baseline Current System**	Creates or restores baselines of your model. For more information, see "Creating, Restoring, and Comparing Baseline Systems" on page 4-1.
		Restore Selected Baseline System	

Table 2: StP ME Desktop Commands

Menu	Modeling Tool	Submenu or Command	Description
Tools > Model Management	StP / UML	(See referenced discussion for a list of commands)	Displays a submenu of commands for managing subsystem partitioning of models. For more information, see "Managing Subsystems" on page 3-15.

Summary of StP Editor Commands

Table 3 lists the commands in the StP modeling editors that have been introduced since the first StP ME release.

Table 3: StP ME Editor Commands

Menu	Accessible From	Command	Description
File	All diagram editors	**New**	Opens a new blank diagram, after closing the current one, if any, and prompting you to save any modifications. See "Starting a New Diagram" on page 5-3.
Edit (and right-click shortcut menu)	All editors	**Choose** <*apptype*> **Names**	Displays either a property sheet or a list of existing names in the repository for this object's or cell's application type, from which you can select an appropriate name label (formerly, the **Edit > List Labels** command). See "Choose Names Command" on page 5-15.

Table 3: StP ME Editor Commands

Menu	Accessible From	Command	Description
Edit (and right-click shortcut menu)	All StP / UML diagram editors except State and Activity	**Assign Diagram to Subsystem**	Associates the diagram with a selected subsystem. See “Manually Assigning Diagrams to Subsystems” on page 3-12.
	All StP / UML diagram editors	**Object Description**	Displays a text editor window in which you can enter a description of the selected object / cell, which then appears as a note description in the Object Annotation Editor (OAE). For more information. See “Quickly Entering Descriptions” on page 5-8.
	StP / UML Class Table Editor	**Cell Description**	
	StP / UML Class Table Editor	**Fill Type**	Allows you to select class attribute types and operation return types from a list of predefined and user-defined types. See “Choosing Class Attribute and Operation Types” on page 6-11.
		Edit Signature	Allows you to select an operation’s arguments from a pick list. See “Choosing an Operation’s Arguments” on page 6-13.
Go To (and **Go To** shortcut menu)	All editors	**Any Symbol With The Same Name**	Navigates to any node (non-link) object with the same name as the selected symbol. See “Navigation Enhancements” on page 5-19.
		Any Reference Of The Selected Symbol/Link	Navigates to any reference with the same name as the selected symbol. See “Navigation Enhancements” on page 5-19.

Table 3: StP ME Editor Commands

Menu	Accessible From	Command	Description
GoTo (and **GoTo** shortcut menu)	StP/UML Class Table Editor	**Class’s Immediate Super Class(es)**	Navigates to the direct superclass of the selected class, or allows selection of the target superclass from an **Object Selector** dialog box. See “Navigation Enhancements” on page 5-19.
		Class’s Immediate Sub Class(es)	Navigates to the direct subclass of the selected class, or allows selection of the target subclass from an **Object Selector** dialog box. See “Navigation Enhancements” on page 5-19.
	StP/UML Activity Editor	**Diagram with Refined Action State**	Creates/navigates from an action state to a decomposition diagram in which the action state is further refined. See “Refining Action States in Activity Diagrams” on page 6-22.
		Parent Action State	Navigates from the refined activity diagram to the parent action state. See “Sorting Attributes and Operations” on page 6-15.
Sort	StP/UML Class Table Editor	(See referenced discussion for a list of commands)	Changes the order in which rows of attributes or operations display in the class table. See “Sorting Attributes and Operations” on page 6-15.
Tools	All editors	**Start New Editor** **Use Existing Editor**	(Formerly **Reject/Accept Remote Access**) Allows or prevents multiple sessions of the same editor to be started. See “Choosing Multiple or Single Editor Sessions” on page 5-3. **Start New Editor—**Allows StP to open multiple sessions of the same editor. **Use Existing Editor—**Limits StP to opening a single session of each editor. .

Table 3: StP ME Editor Commands

Menu	Accessible From	Command	Description
UML	StP/UML Collaboration and Sequence diagram editors	**Set Message Name**	Displays a property sheet from which you can select an appropriate label for the selected object. See “Sequence and Collaboration Editors” on page 6-16.
UML > Auto Draw	StP/UML Class Diagram Editor	(See referenced discussion for a list of commands)	Automatically draws classes related to a selected class, based on the contents of the repository. See “Auto Drawing Classes” on page 6-6.
UML > Generalization Hierarchy		**Draw Generalization Hierarchy**	Automatically draws a tree-style generalization hierarchy for selected classes. See “Creating a Tree-Style Generalization Hierarchy” on page 6-5.
UML > Construct Class from...	StP/UML Class Table Editor	**Implement Interface Operations**	Adds all the operations of a class’s interfaces to the class table of the supporting class. For more information. See “Adding Methods of an Interface” on page 6-15.

Globally Unique IDs

StP uses globally unique identifiers (GUIDs) for all StP objects. They were introduced in 8.0 in support of StP’s new model management capabilities and the incremental code generation facility from ACD. GUIDs are handled internally within the StP implementation, which means that the majority of users need not be aware of them.

GUIDs are automatically generated when a node, link, or context object is created in the repository. The GUIDs are also stored in the StP diagram and table files, so they are persistent across system repository rebuilds.

In the repository, a GUID is represented as the attribute “guid” (table column) of *node*, *link*, *file*, and *cntx* objects. The values associated with this attribute can be viewed by using the repository browser. You can access the browser from the desktop via **Tools > Browse Repository**. When prompted, enter a valid OMS query (for example, “node[1802]”). The browser will show the node object with all its attributes, including the GUID.

For StP users, it is expected that GUIDs will be used for building customizations or integrations.

Since GUIDs are persistent, it is possible to use them when integrating with third-party tools - for example, where source code merging might be required. GUIDs are also used for ACD-based code generation, where they are put into tags that mark areas for user-defined code.

Factors Relating to GUIDs

Please be aware of the following factors that affect GUIDs:

- Changing the label of a symbol causes regeneration of the GUID, because this action creates a new object in the repository (it clones the existing object).
- Using the global object rename (GOR) functionality ensures that the ID is preserved. Use GOR whenever you want to rename an existing object rather than clone an object by giving it a new label.
- The implementation of GUIDs affects migration from systems created prior to 8.1. Refer to *Migrating to StP 8* for details.

2 StP/UML View-Based Desktop

The StP / UML desktop includes a view-based model browser implemented in StP 8.0 that allows you to examine the structure of your model from different views of its contents.

This chapter describes the organization of the view-based StP / UML desktop, including:

- "Overview of the Desktop Model Browser" on page 2-1
- "Diagram and Table Categories" on page 2-4
- "Repository and Document Views" on page 2-4
- "Tree Views of Your Model" on page 2-4

Overview of the Desktop Model Browser

The desktop displays nine main categories in the model pane:

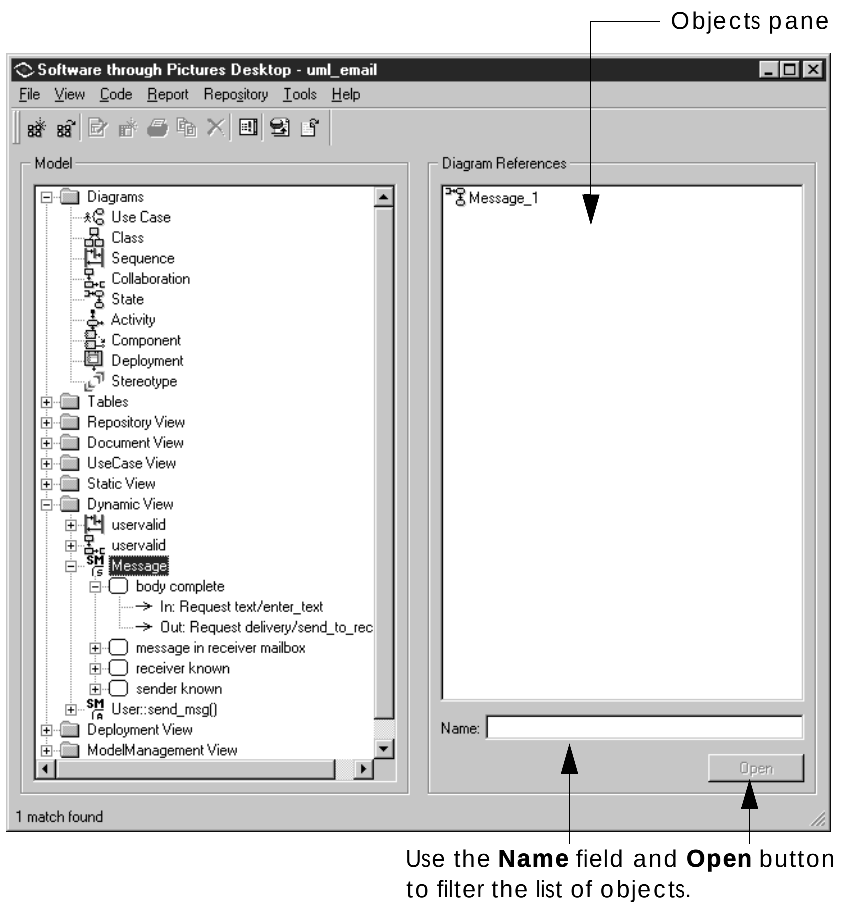
- **Diagrams**—All StP / UML diagrams
- **Tables**—Class, state and requirement tables
- **Repository View**—Use cases, classes, packages, stereotypes
- **Document View**—StP-generated documents
- **UseCase View**—Conceptual view of the modeled system or application
- **Static View**—Descriptive view of the characteristics of individual parts of the modeled system or application
- **Dynamic View**—Behavioral view of the individual parts of the modeled system or application

- **Deployment View**—Implementation view of the components and software / hardware allocations for the modeled system or application
- **ModelManagement View**—Subsystem view of the modeled system or application

Items in the objects pane are listed alphabetically and generally can be filtered using the **Name** field and **Open** button.

Figure 1 shows the current StP / UML desktop organization. In this example, the **Dynamic View** is open and the *Message* state machine is selected. The corresponding diagram, *Message_1*, appears in the objects pane.

Figure 1: StP/UML View-Based Desktop Model Browser



Right-clicking the mouse in the left or right pane displays a context-sensitive menu with commands appropriate to the category or object you have selected.

Each of the categories and views is described in more detail in the following sections. The **ModelManagement View** is described in Chapter 3, “Model Management.”

Diagram and Table Categories

The **Diagrams** and **Tables** categories remain essentially unchanged from the previous StP release. Opening either of these categories displays a list of all the possible types of diagrams or tables that can comprise your StP / UML model. When you select one of the diagram or table subcategories in the model (left) pane, a list of existing diagrams or tables of that type appear in the objects (right) pane.

Repository and Document Views

The **Repository View** replaces the **Model Elements** category that was present in StP 7.x, and includes a new subcategory, **Stereotypes**. The objects (right) pane of the **Repository View** displays repository objects, rather than diagrams or tables.

The **Document View** corresponds to your StP default publishing tool interface. The objects pane displays a list of documents you have generated. The **Document View** contains a new subcategory, **Report Maker Diagrams**. The objects pane in that subcategory will contain diagrams you have created with the Report Maker Editor (RME), a tool introduced in StP 8.2. The RME is discussed in Chapter 8, “Report Maker Editor (RME).”

Tree Views of Your Model

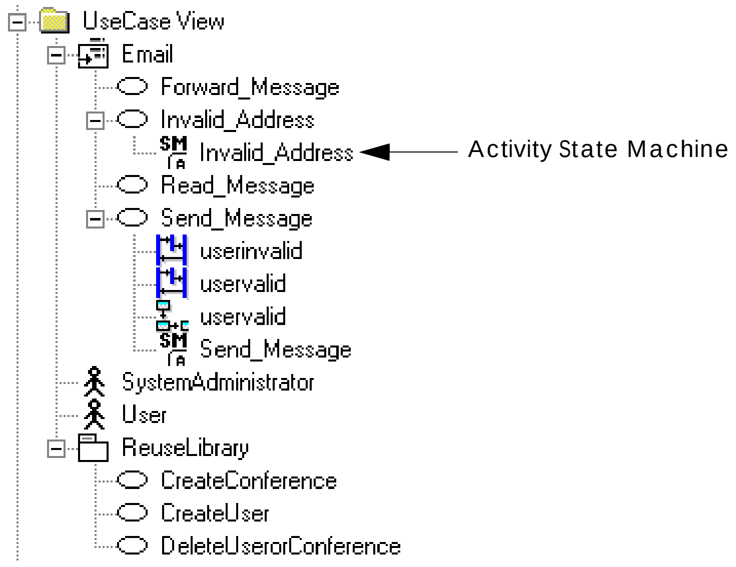
In a tree view of your model, the model pane generally displays model elements, and the objects pane displays references of the selected model elements in diagrams. One exception to this is that scenario and collaboration diagrams without a UmlScenarioInstance (for example, when created without navigating from the use case), appear as top-level file objects in the tree. Both model elements and diagram references have object commands associated with them; you can access them from the shortcut (right-click) menu in each pane. The following sections depict the content and structure of each tree view of your model, and include any special information about the view.

UseCase View

```
UseCase View
    This System
        Use Case
            linked scenarios
            linked activity state machine
        Actor
    Package
        Use Case
    Use Case
        ....
    Actor
```

Note: Activity state machines and scenarios are not further refined in this view, but rather in the dynamic view. An activity state machine is indicated in this view by an “A” on its icon (see Figure 2).

Figure 2: UseCase View Example



Static View

The static view combines both the package hierarchy and the class generalization hierarchy.

```
Static View
    Package
        Package
        ...
        Class
            UmlGeneralization
                SubClass
                    ...
            UmlGeneralization
                SubClass in different package (from package)[1, 2]
            Attribute
            Operation
        Class
            UmlGeneralization
                SuperClass in different package (from package)[1, 3]
    Class
        Attribute
        Operation
```

Notes:

[1] The containing package of a package external class is shown as “Class (from package)” or “Class (no package)”.

[2] A subclass residing in a package other than its parent class is indicated in this view by an "E" in its class icon (see Figure 4).

[3] A superclass residing in another package is indicated in this view by an "E" in its icon, and the icon for the UmlGeneralization appears upside down (see Figure 4).

[4] Package external classes are always leaf objects in a generalization tree, since they also appear in their own package (or at the top level), where the full subclass tree appears.

Figure 3: Static View Class Diagram Example

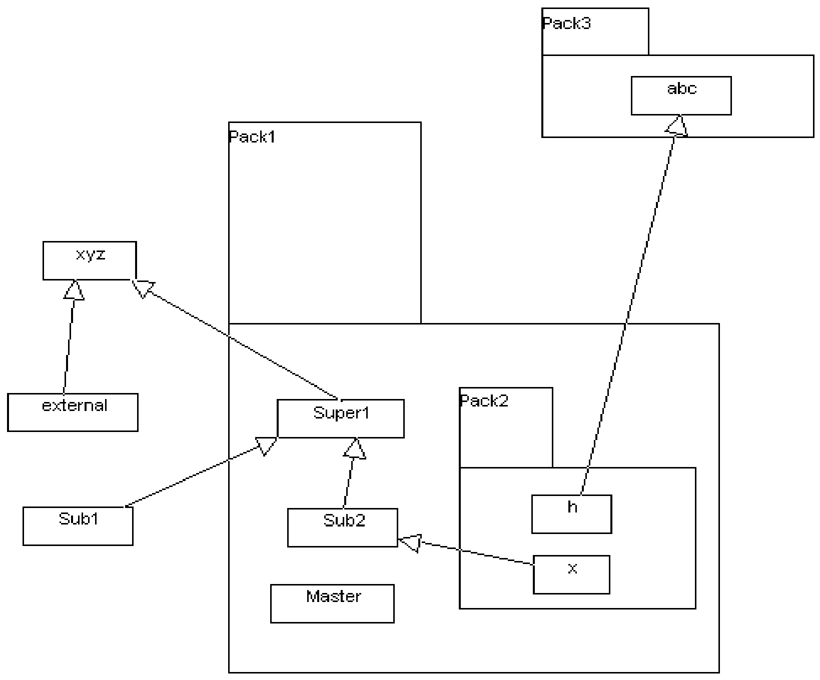
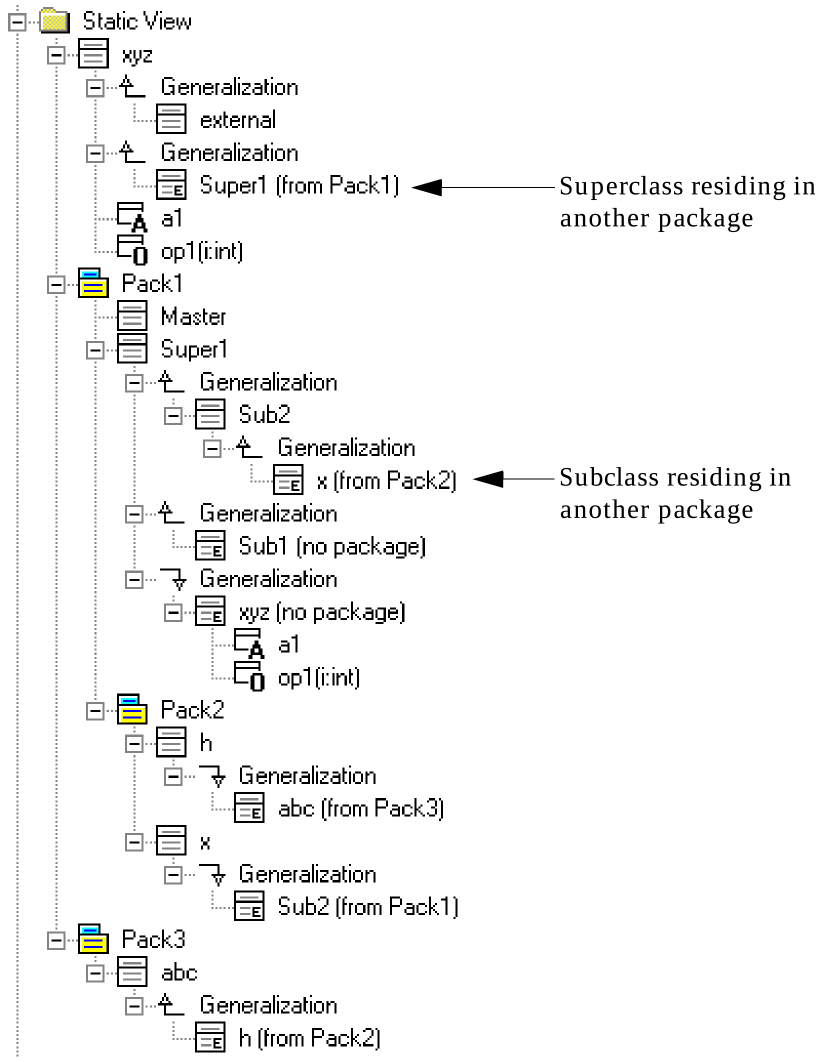


Figure 4: Static View Example



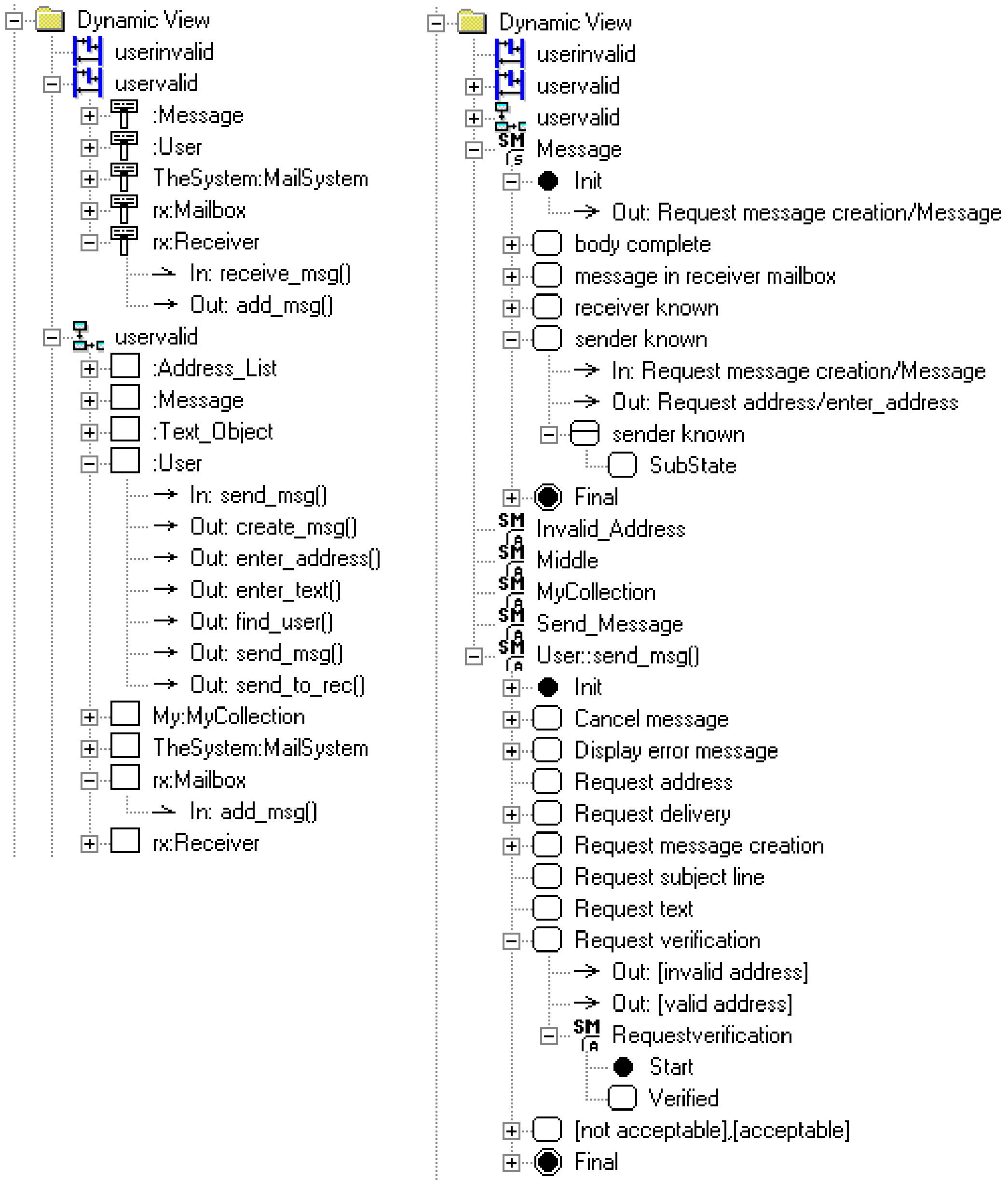
```
Static View
  xyz
    Generalization
      external
    Generalization
      Super1 (from Pack1)
    a1
    op1(i:int)
  Pack1
    Master
    Super1
      Generalization
        Sub2
          Generalization
            x (from Pack2)
      Generalization
        Sub1 (no package)
      Generalization
        xyz (no package)
          a1
          op1(i:int)
    Pack2
      h
        Generalization
          abc (from Pack3)
      x
        Generalization
          Sub2 (from Pack1)
  Pack3
    abc
      Generalization
        h (from Pack2)
```

Superclass residing in another package
Subclass residing in another package

Dynamic View

```
Dynamic View
       sequence scenarios
            object:class
                  In: message
                  Out: message
-     collaboration scenarios
            object:class
                  In: message
                  Out: message
      sequence scenarios without UmlScenarioInstance (file)
            ...
      collaboration scenarios without UmlScenarioInstance (file)
            ...
      activity state machine
            initial state
            action state
                  In: transition
                  Out: transition
                  refined activity state machine
                        ...
            final state
      state diagram state machine
            initial state
            action state
                  In: transition
                  Out: transition
                  refined composite state
                        ...
            final state
```

Figure 5: Dynamic View Example



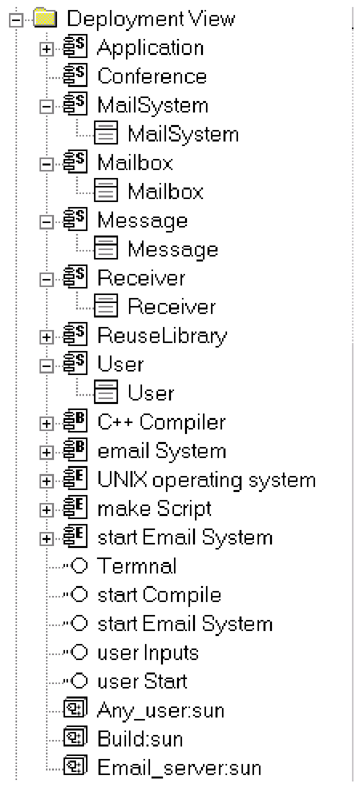
```
Dynamic View
    userinvalid
    uservalid
        :Message
        :User
        TheSystem:MailSystem
        rx:Mailbox
        rx:Receiver
            In: receive_msg()
            Out: add_msg()
    uservalid
        :Address_List
        :Message
        :Text_Object
        :User
            In: send_msg()
            Out: create_msg()
            Out: enter_address()
            Out: enter_text()
            Out: find_user()
            Out: send_msg()
            Out: send_to_rec()
        My:MyCollection
        TheSystem:MailSystem
        rx:Mailbox
            In: add_msg()
        rx:Receiver
```

```
Dynamic View
    userinvalid
    uservalid
    uservalid
    Message
        Init
            Out: Request message creation/Message
        body complete
        message in receiver mailbox
        receiver known
        sender known
            In: Request message creation/Message
            Out: Request address/enter_address
            sender known
                SubState
        Final
    Invalid_Address
    Middle
    MyCollection
    Send_Message
    User::send_msg()
        Init
        Cancel message
        Display error message
        Request address
        Request delivery
        Request message creation
        Request subject line
        Request text
        Request verification
            Out: [invalid address]
            Out: [valid address]
            Requestverification
                Start
                Verified
        [not acceptable],[acceptable]
        Final
```

Deployment View

Deployment View
- ComponentSources
  - Class
- ComponentBinaries
  - Class
- ComponentExecutables
  - Class
- ComponentInterfaces
- ComponentObjects
- DeploymentNodes

Figure 6: Deployment View Example



```
Deployment View
    Application
    Conference
    MailSystem
        MailSystem
    Mailbox
        Mailbox
    Message
        Message
    Receiver
        Receiver
    ReuseLibrary
    User
        User
    C++ Compiler
    email System
    UNIX operating system
    make Script
    start Email System
    Termnal
    start Compile
    start Email System
    user Inputs
    user Start
    Any_user:sun
    Build:sun
    Email_server:sun
```

3 Model Management

StP model management, introduced in StP 8.0, is an StP / UML feature supporting distributed development of large, complex projects through subsystem partitioning and management. StP model management allows independent development of separate subsystems while protecting the integrity of the entire model. Partitioning into subsystems can be done at any time during the development process. Developers are not forced to prematurely partition a system at the beginning of a project.

This chapter discusses:

Using StP Model Management—An Overview

StP model management allows an StP / UML model to be decomposed into subsystems, which can then be developed and managed independently of one another, while maintaining the overall integrity and consistency of the entire model. Users can check out a portion of a model and work "off line" or in a private workspace in a flexible manner, and check their work back in again, with seamless integration of results into a shared repository.

With StP model management, you can:

- Partition an StP / UML model into logical subsystems.
- Create private and shared views of a model.
- Track changes to and dependencies between subsystems.
- Lock and unlock subsystems, as needed.
- Manage subsystems, using version control, with a supported external configuration management (CM) system.

About Subsystems

StP model management uses the subsystem construct introduced in OMG UML 1.3 to provide "containers" for the model partitions. Subsystems are represented graphically by using the package symbol with a *«subsystem»* stereotype. Subsystems are the basis for configuration control and for private workspaces. StP model management provides several commands for managing subsystems. Each command is performed on a single subsystem or a tree of subsystems (if the subsystem contains other subsystems).

Subsystem Contents

A subsystem can contain any UML model elements. It includes those model elements that exist within the subsystem and the interfaces that the subsystem exports. Subsystems can contain any logical grouping of model elements, including other subsystems.

Local and External Elements

Each element contained in a subsystem is called a "local model element" (local class, local use case, and so forth). A model element is considered external to a subsystem ("external model element") if it is referenced in a diagram belonging to that subsystem, but the model element itself is assigned to another subsystem. External model elements are normally used to describe the interface between subsystems.

The Model Management Process

To use model management, you generally perform the following steps:

1. Partition a model into logical subsystems (see “Defining a Subsystem” on page 3-5).
2. Export the subsystem into a private workspace or configuration management system (see “Creating a Private Workspace or Baselining a Subsystem” on page 3-18).

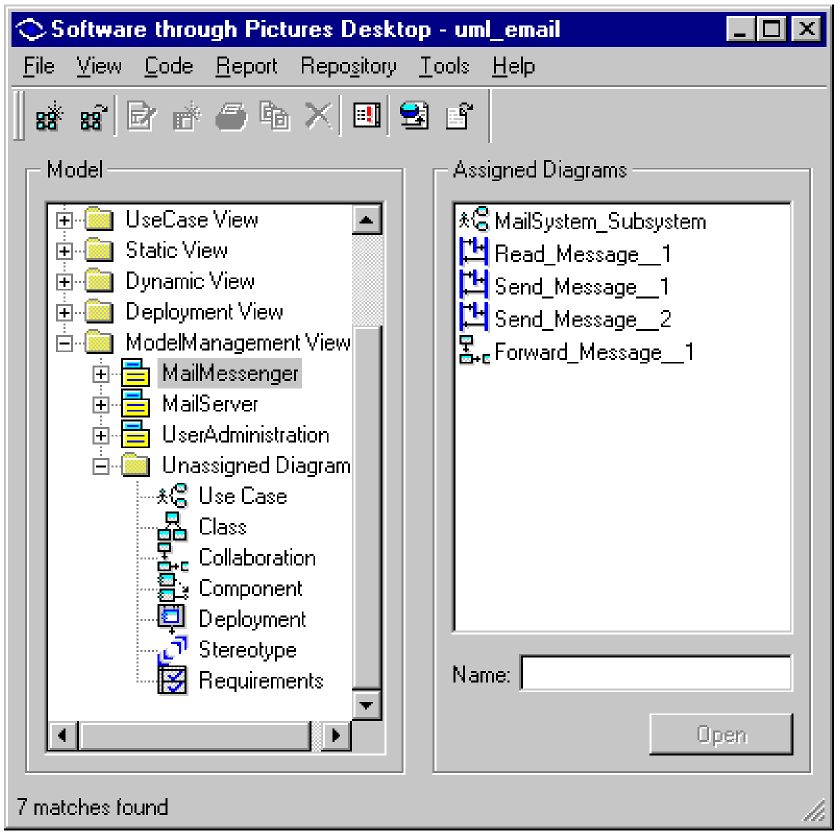
   Exporting a subsystem collects together all of the entities contained within a subsystem and any related elements in the wider model.
3. Work on the individual, locked subsystems (see “Locking and Unlocking a Subsystem” on page 3-21).
4. Reassemble the subsystems by restoring them to the main model (“Restoring a Subsystem” on page 3-21).

   Restoring a subsystem restores all of the model elements of a subsystem into the main model, releasing the locks and resolving all of the changes that have taken place since the subsystem was exported.
5. Update external elements in your current private workspace model with relevant changes made to the main model (see “Reconfiguring External Model Elements” on page 3-23).

Using the Model Management Tree Browser

All subsystems defined in an StP model appear in the **ModelManagement View** category in the StP desktop’s model pane. When a subsystem is selected in the **ModelManagement View**, all of its assigned diagrams appear in the objects (right) pane. Click the plus sign (+) in front of a subsystem to see all the other assigned model elements, such as use cases and classes. An example of the **ModelManagement View** appears in Figure 1.

Figure 1: ModelManagement View Example



The folder *Unassigned Diagrams* in the **ModelManagement View** shows all diagrams that are not yet assigned to any of the defined subsystems.

All actions that can be performed on subsystems are available on the desktop, from the **Tools > Model Management** menu or the shortcut (right-click) menu. To use the shortcut menu, select a subsystem in the model pane and click the right mouse button to display a shortcut menu of available commands. For a description of these commands, see “Managing Subsystems” on page 3-15.

Note: To ensure reliable results and identify potential problems before performing model management operations on subsystems, first ensure that your global model is semantically correct by performing semantic checking on the entire model.

Defining a Subsystem

Subsystems can be used in various ways in different StP / UML diagrams. For example, you can define the architecture of the overall system and show relationships between subsystems in a class diagram. Other diagrams, such as a use case diagram, may define the subsystem containment of model elements.

Three steps are involved in defining a subsystem:

1. Drawing the subsystem as a package with a *«subsystem»* stereotype.
2. Assigning model elements such as use cases and classes to a subsystem by drawing the elements in the desired *«subsystem»* package.
3. Manually assigning diagrams that reference elements of a given subsystem to that subsystem.

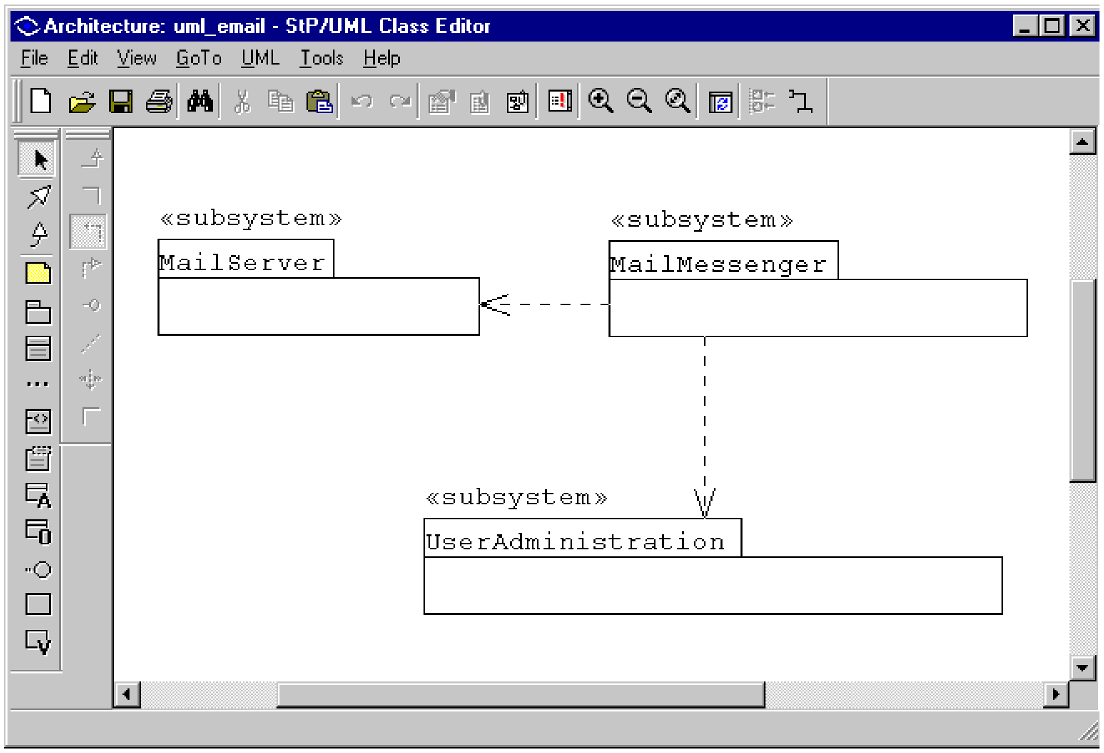
In Step 1, you draw the subsystem package symbols and identify them as subsystems with a *«subsystem»* stereotype, one of the predefined stereotypes built into StP / UML For more information, see “Representing the Subsystem” on page 3-7.

As the model is refined, more detailed diagrams are created and populated with elements that should be in those subsystems. Step 2 uniquely defines which elements belong to which subsystems. Model elements that are refinements or definitions of assigned elements automatically belong to the subsystem of the parent (or “scope”) element. For more information, see “Assigning Model Elements to a Subsystem” on page 3-8 and “Propagating Subsystem Membership” on page 3-9.

Step 3 essentially identifies a diagram as a view that is relevant for the user of that subsystem. The association of a diagram with a subsystem does not redefine any existing association between elements and subsystems; rather, it defines the graphical representation of those elements the user wants to see in a subsystem. For more information, see “Manually Assigning Diagrams to Subsystems” on page 3-12.

Figure 2 shows a simplified class diagram for an example system, in which three subsystems have been drawn.

Figure 2: Subsystems Drawn in a Class Diagram


Architecture: uml_email - StP/UML Class Editor
File Edit View GoTo UML Tools Help
«subsystem»
MailServer
«subsystem»
MailMessenger
«subsystem»
UserAdministration

The title bar displays the name of the diagram or table, followed by the system name and the name of the current editor. If the diagram or table has been modified but not saved, the diagram / table and system names appear within parentheses:

(<*diagram_or_table_name*>: system_name)

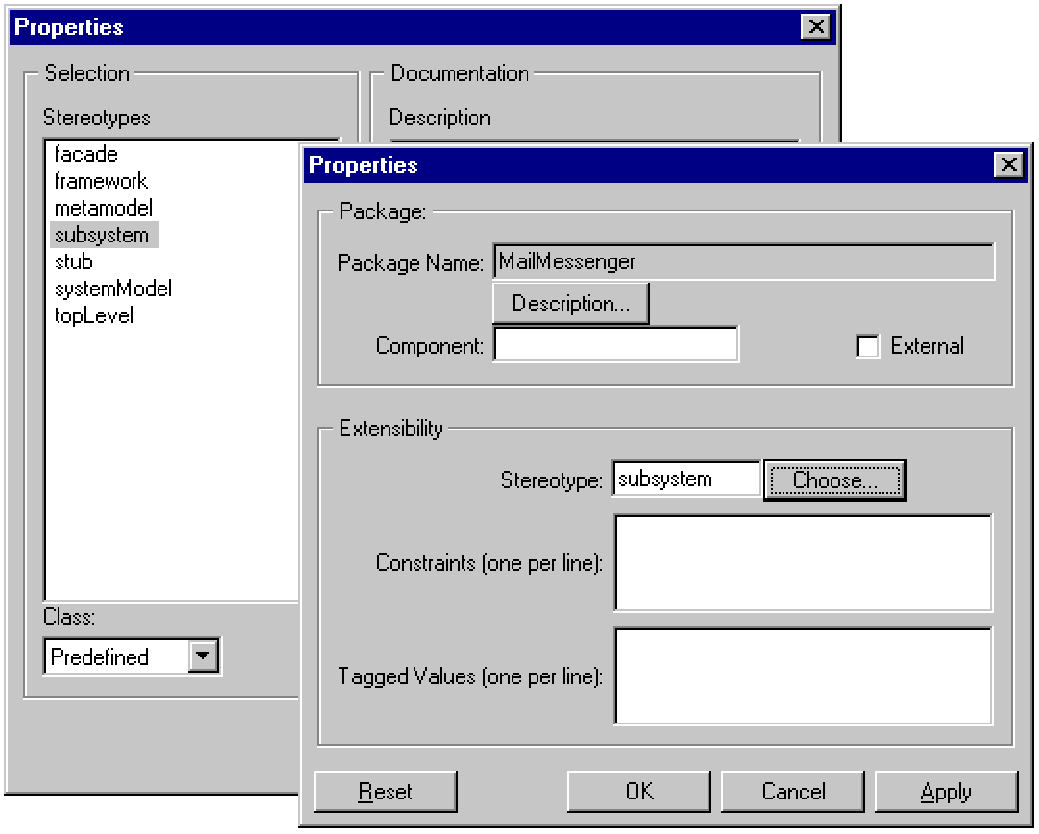
Note: The word <unassigned> may appear in the title bar if the current diagram has not been assigned to a subsystem, but at least one other diagram has been assigned.

Representing the Subsystem

To represent a subsystem in an StP / UML model:

1. On a class or use case diagram, insert and label a package symbol with the subsystem name.
2. Select the package symbol in the drawing area and display its property sheet, using one of the following:
   - **Properties** command on the **Edit** or shortcut (right-click) menu
   - **Properties** button on the editor’s toolbar
3. In the **Properties** sheet, click the **Choose** button adjacent to the **Stereotype** field.
4. From the pick list of predefined stereotypes that appears, select **subsystem**, as shown in Figure 3, and click **OK** on the pick list dialog.

Figure 3: Selecting the Subsystem Stereotype



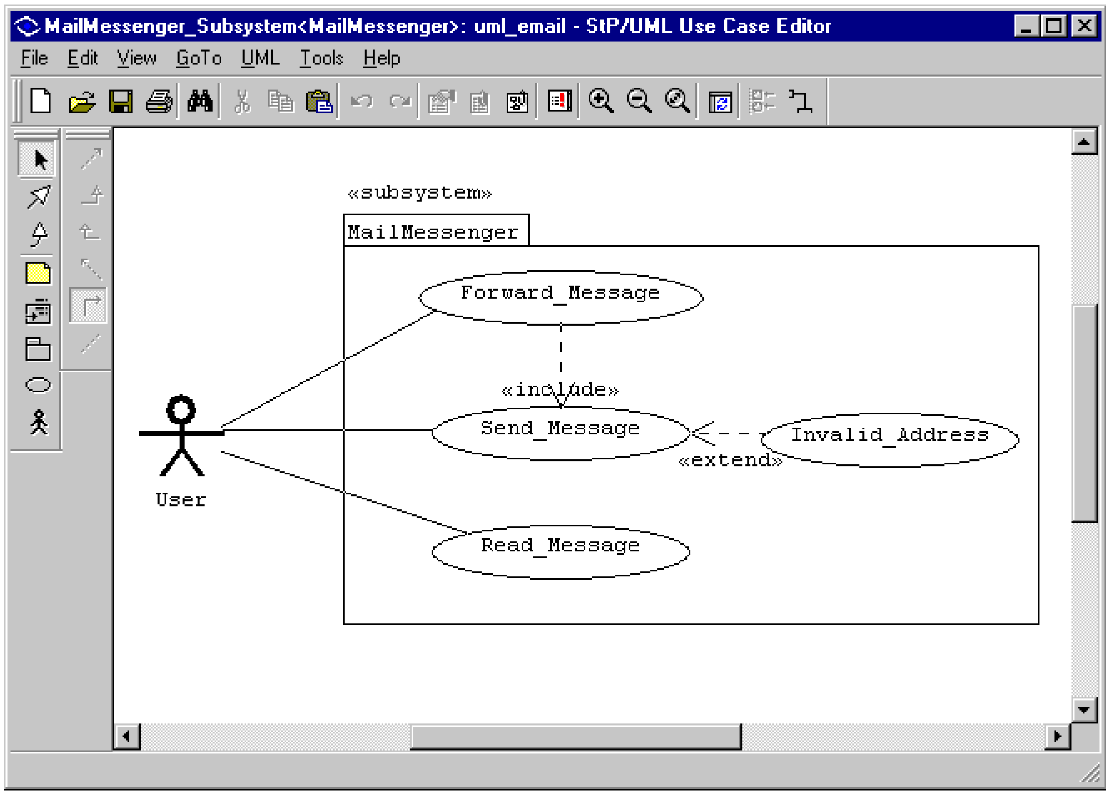
5. Click **OK** on the package symbol's **Properties** sheet.

   The *«subsystem»* stereotype appears above the package symbol, in the diagram, identifying it as a subsystem.

Assigning Model Elements to a Subsystem

Subsystems are containers for other model elements. All model elements that are part of the UML notation are assigned to subsystems when they are drawn within the subsystem package symbol. In Figure 4, the use cases drawn inside the *MailMessenger* subsystem's package symbol are automatically assigned to the subsystem *MailMessenger.*

Figure 4: *MailMessenger* Subsystem with Use Cases


MailMessenger_Subsystem<MailMessenger>: uml_email - StP/UML Use Case Editor
File Edit View GoTo UML Tools Help
«subsystem»
MailMessenger
Forward_Message
«include»
Send_Message
Invalid_Address
«extend»
Read_Message
User

Propagating Subsystem Membership

Logical membership in a subsystem is propagated wherever reasonable. For example, if package *MailMessenger* is part of subsystem *MailLibrary*, all classes in package *MailMessenger* automatically belong to subsystem *MailLibrary*. All directly associated elements, such as a superstate for a class or a scenario for a use case, also belong to the subsystem to which the parent (“scope element”) itself belongs.

Table 1 describes propagation of subsystem membership, where the scoped element (child) belongs to the subsystem of the scope element (parent).

Table 1: Propagation of Subsystem Membership

Scope Element	Scoped Element
Package	Contained elements (Packages, Classes, Use Cases)
Class	Operations, Attributes, and Inner Classes

Implicitly Assigning Diagrams to Subsystems

Wherever possible, diagrams that include elements of a decomposition or navigation are implicitly assigned to the subsystem to which the element's parent belongs.

For example, suppose that after preparing the use case diagram in Figure 4, you further refine the use case *Send_Message* in the sequence diagram *Send_Message_1*. StP implicitly assigns the sequence diagram containing the scenario instance *user valid* (scoped to the use case *Send_Message*) to the subsystem *MailMessenger*, to which the scenario instance's parent use case belongs. Figure 5 illustrates this implicit assignment.

Figure 5: Scenario with Scoped Subsystem Membership

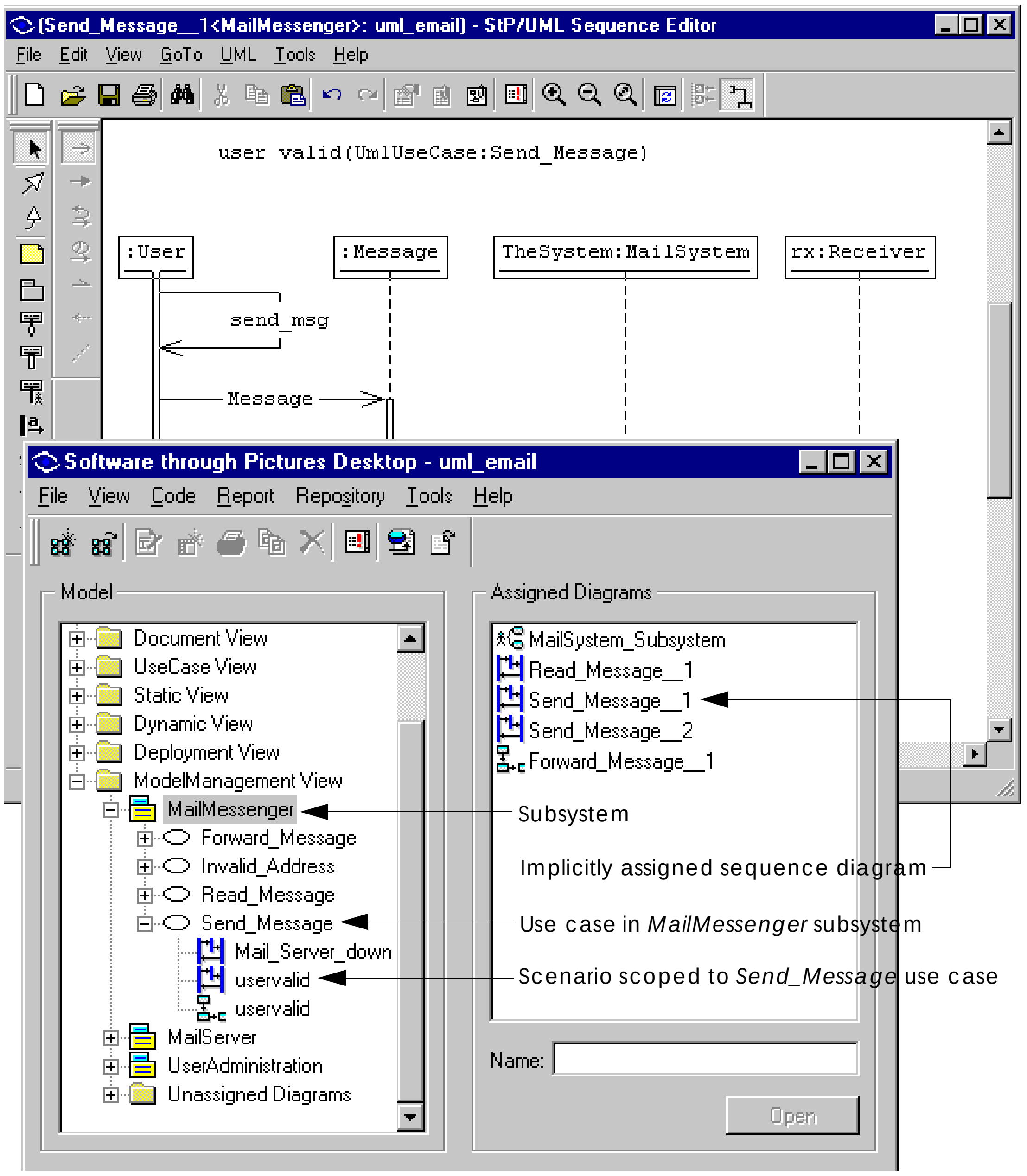


Table 2 shows how diagrams that include elements of a decomposition / navigation (**Scoped Element**) are automatically assigned to the subsystem of the parent (**Scope Element**).

Table 2: Implicit Assignment of Diagrams to Subsystems

Scope Element	Scoped Element	Diagram/Table Implicitly Assigned to Scope Element's Subsystem
Use Case	Scenario Instance	Sequence diagram (*) Collaboration diagram (*)
	State Machine	State diagram
Class	Class Definition	Class table
	State Machine	State diagram Activity diagram (**)
State	State Definition	State table

***** When it is initially created, a scenario defined in a sequence or collaboration diagram is assigned to the same subsystem as its parent use case. However, this initial assignment can be changed by the user. Because sequence and collaboration diagrams can contain objects from classes belonging to different subsystems, it is intentionally left to the user to manually assign these diagrams to the desired subsystem.

****** An activity diagram that refines an activity state from another activity diagram is implicitly assigned to the same subsystem as the parent activity state.

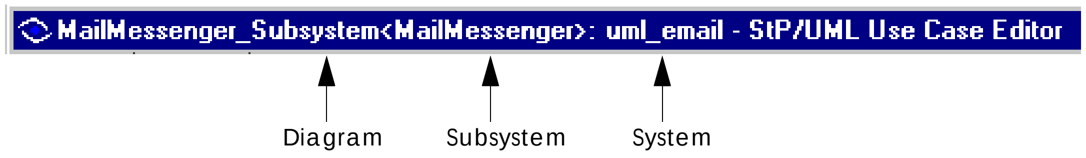
Class and state tables and state and activity diagrams are always implicitly assigned to a subsystem and cannot be manually assigned or re-assigned.

Manually Assigning Diagrams to Subsystems

Diagrams that are not implicitly assigned to subsystems need to be assigned manually. You can assign diagrams to subsystems from an StP editor or from the StP desktop. You cannot assign or reassign diagrams to or from a subsystem that is locked.

To indicate that a diagram has been assigned to a subsystem, the title bar in the diagram displays the subsystem to which it belongs, within angle brackets, between the diagram name and the system name, as in Figure 6.

Figure 6: Subsystem Name in Diagram's Title Bar


MailMessenger_Subsystem<MailMessenger>: uml_email - StP/UML Use Case Editor

After at least one diagram has been assigned to any subsystem in your model, any unassigned diagrams thereafter display the word <unassigned> in the title bar, and also appear in the **Unassigned Diagrams** category in the **ModelManagement View** on the desktop. If no diagrams have been assigned to a subsystem, then the word <unassigned> does not appear in any diagram's title bar.

Note: Class and state tables and state and activity diagrams are always implicitly assigned to a subsystem and cannot be manually assigned or reassigned.

Assigning Diagrams to Subsystems in an StP Editor

To assign the current diagram to a subsystem from within an StP editor:

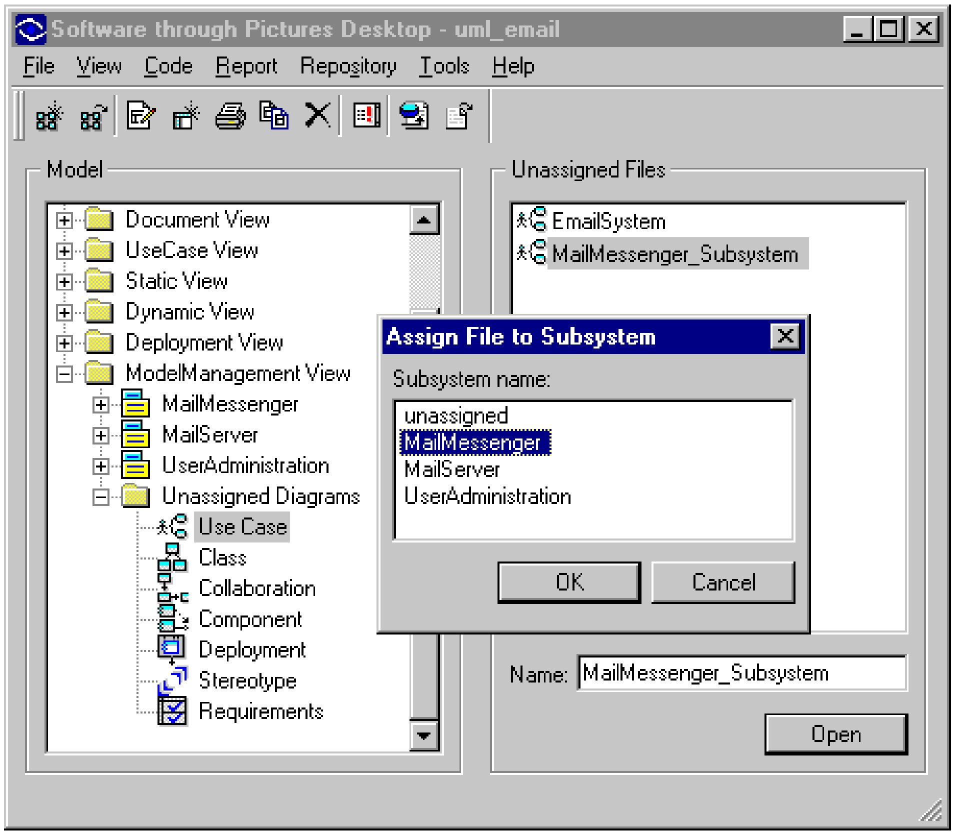
1. From the editor's **Edit** menu, choose **Assign Diagram to Subsystem**.
2. In the **Object Selector** dialog, select the subsystem and click **OK**.

Assigning Diagrams to Subsystems from the StP Desktop

To assign a diagram to a subsystem from the StP desktop:

1. In the **ModelManagement View**, open **Unassigned Diagrams** and select one of the diagram subcategories.
2. In the **Unassigned Files** list (right pane), select a file.
3. From the shortcut menu, choose **Assign File to Subsystem**.
4. In the dialog box (Figure 7), select a subsystem name and click **OK**.

Figure 7: Assigning Files to a Subsystem from the Desktop



Managing Subsystems

Subsystems form the basis for private workspaces and provide the interface to version control systems. When one or more subsystems have been defined, you can perform the tasks described in the following sections:

- "Checking Semantics for Subsystems" on page 3-15
- "Creating a Private Workspace or Baselining a Subsystem" on page 3-18
- "Locking and Unlocking a Subsystem" on page 3-21
- "Restoring a Subsystem" on page 3-21
  (from either a private workspace or a CM tool into a "global" model)
- "Reconfiguring External Model Elements" on page 3-23
  (updating external model elements in a private workspace with the current version of elements from the main model)

Commands for performing these operations are available in the **ModelManagement View** of the desktop tree browser. To access these commands:

1. On the StP desktop, open the **ModelManagement View** and select a subsystem.
2. Choose a command from either of the following:
   - **Tools > Model Management** menu
   - Shortcut (right-click) **Model Management** submenu
3. Make or modify entries in the dialog box, if one appears, as described in the following command-specific sections; then click **OK**.

The following sections describe model management commands.

Checking Semantics for Subsystems

The purpose of checking semantics in subsystems is to guarantee completeness and consistency within the subsystem, even if the subsystem is restored in a private workspace. A link in a diagram belonging to a particular subsystem is semantically correct if it affects local model elements only (those belonging to the same subsystem).

Semantic errors are reported for any links that potentially affect the definition or behavior of model elements from external subsystems. For a definition of local and external model elements, see “About Subsystems” on page 3-2.

Note: model management’s subsystem semantic checking does not take the place of global completeness and consistency checks for the entire model. To perform global semantic checking on your model, use the desktop **Tools > Check > Check Semantics for Whole Model** command.

In addition, when you work with subsystems in private workspaces and then reassemble the subsystems into your global model, it is possible for some elements, such as state machines, to become “orphaned” if the parent class has been deleted from the model. If an element is orphaned in this way, it is not recognized by model management as belonging to the system, and will not appear in the **ModelManagement View**. To detect these potential problems, run the **Tools > Model Management > Check Semantics for Subsystems** command on the desktop each time you reassemble subsystems into the global system. StP issues warnings about apparently orphaned elements, as described in “Orphaned element check,” below.

You can check the semantics of subsystems from the desktop or within the StP editors:

- To check semantics for all subsystems from the desktop, choose **Tools > Model Management > Check Semantics for Subsystems**.
- To check semantics for a selected subsystem only, select the subsystem in the **ModelMangement View** on the desktop, and choose **Check Semantics for Subsystem** from the shortcut (right-click) menu.
- To check semantics in an StP editor for the subsystem to which the current diagram belongs, choose **Tools > Check Diagram’s Subsystem Semantics**.

Checking semantics from the desktop allows you to check the entire model or selected subsystems. Checking semantics within an editor reports only errors caused by the current diagram, and allows you to navigate to the symbol that is the source of the error in that diagram.

Orphaned element check. StP issues a warning message if it detects any of the following:

- A class table contains no UmlClass.
- A class in a class table is not referenced in any class diagram.
- A state diagram, state table or activity diagram contains no state machine.
- The state machine in these diagrams/tables has no corresponding class or use case in a diagram.

General checks. At least one file containing the subsystem package has to be assigned to the subsystem. A model element cannot be assigned to more than one package.

Requirement table. A requirement cannot be referenced in a table assigned to a different subsystem.

Use case diagram checks. The following restrictions apply to diagrams assigned to a subsystem:

- A use case cannot extend a use case belonging to a different subsystem.
- A use case cannot be included by a use case belonging to a different subsystem.
- A use case cannot be the generalization of a use case belonging to a different subsystem.

Class diagram checks. The following constructs are not allowed in diagrams assigned to a subsystem:

- A generalization from external to local class
- A dependency from external to local class
- A dependency with friend stereotype from and to a local class
- An undirected association between an external and local class
- A directed association from an external to a local class

In addition:

- A local class cannot be the aggregation/composition of an external class.
- An association class cannot belong to a different subsystem than the class where the navigation starts.

- For an n-ary association, all partner classes have to be in the same subsystem (according to the UML standard, this navigation does not make sense or would be too complicated).
- An inner class cannot be assigned to a different subsystem than the outer class.

Sequence diagram checks. The following constructs are not allowed in diagrams assigned to a subsystem:

- Any message sent from an external to a local class except for return messages
- Any message to an external class with no corresponding operation

Collaboration diagram checks. These checks are the same as for a sequence diagram.

Activity diagram checks. No checks are needed.

State diagram checks. No checks are needed.

Component diagram checks. Any symbol referenced in a diagram assigned to a subsystem cannot be referenced in another diagram assigned to a different subsystem.

Deployment diagram checks. These checks are the same as for a component diagram.

Stereotype diagram checks. These checks are the same as for a component diagram.

Creating a Private Workspace or Baselining a Subsystem

Use StP's subsystem baselining capabilities to:

- Create a private workspace for a user.
- Export a subsystem (check in a baseline) to a configuration management (CM) version control system.

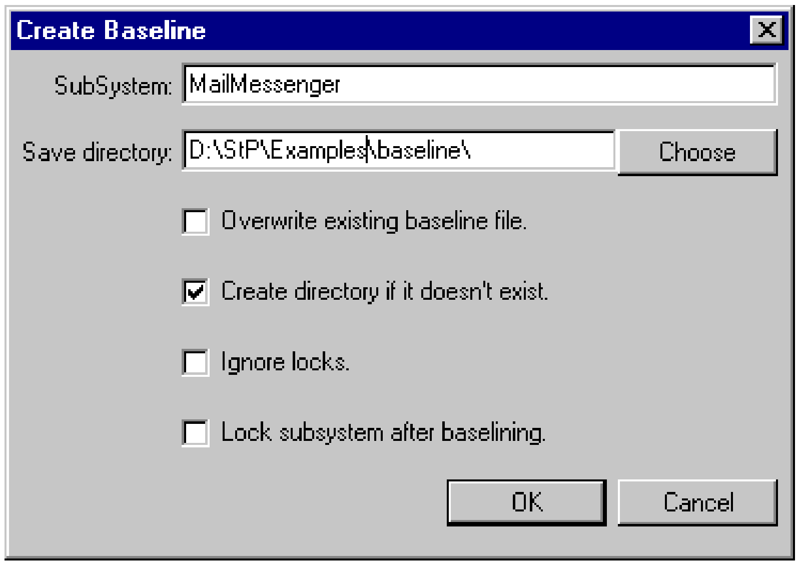
To baseline a subsystem:

1. Select the subsystem on the desktop and choose either of the following:

- **Create Baseline for Subsystem** from the shortcut (right-click) menu
- **Tools > Model Management > Baseline Subsystem**

2. Complete or modify the entries in the **Create Baseline** or **Baseline Subsystem** dialog box, as described in Table 3 on page 3-20.

Figure 8: Create Baseline Dialog Box



Create Baseline

SubSystem: MailMessenger

Save directory: D:\StP\Examples\baseline\

[ ] Overwrite existing baseline file.

[✓] Create directory if it doesn't exist.

[ ] Ignore locks.

[ ] Lock subsystem after baselining.

The **Baseline Subsystem** dialog is similar to the **Create Baseline** dialog shown in Figure 8, except that it allows you to select the subsystem to be baselined from a list of all not currently locked subsystems in the model.

Note: Trying to create a baseline for a currently locked subsystem with the **Create Baseline for Subsystem** command results in an error message, since subsystems need to be unlocked before a baseline can be created.

Table 3: Create Baseline Dialog Box Summary

Field or Option	Description
SubSystem	Name of the selected subsystem
Save directory	Name of the directory in which to create the baseline file
Overwrite existing baseline file	If selected, overwrites an existing baseline file for the selected subsystem
Create directory if it doesn't exist	If selected, creates the output directory if it doesn't exist
Ignore locks	If selected, ignores existing manual or automatic locks on diagrams, tables, and annotations. Locks on subsystems are never ignored.
Lock subsystem after baselining	If selected, locks the subsystem after creating the baseline file

In creating a baseline for a subsystem, StP creates a file named <*subsystem-name.zip*> (in this example, *MailMessenger.zip*) in the specified directory. If the baselined subsystem has nested subsystems, StP generates a separate zip file for each subsystem.

Creating a baseline for a subsystem performs checks that are specific to model management. This ensures the integrity of the subsystem so that it can be safely re-imported at a later time. *Errors reported by those checks need to be corrected before a subsystem can be baselined.* (Note: Error checking can be performed separately via **Tools > Model Management > Check Semantics for Subsystem**.)

If you have previously set up a CM integration (see "Using External CM Systems for Version Control" on page 4-9), it also performs the subsequent check-in sequence with the CM tool.

Locking and Unlocking a Subsystem

A defined subsystem can be manually locked and unlocked, using the **Lock Subsystem** and **Unlock Subsystem** commands on the **Tools > Model Management** menu or shortcut (right-click) menu. You can also lock a subsystem by selecting the **Lock subsystem after baselining** option when creating the baseline.

A locked subsystem cannot be changed and no baseline can be created from it. All the locks can be removed with the **Unlock Subsystem** command.

Note: StP generates an error message if you attempt to lock a subsystem that is already locked, or attempt to unlock a subsystem that is already unlocked.

Restoring a Subsystem

Baselines of subsystems that were created as described in “Creating a Private Workspace or Baselining a Subsystem” on page 3-18 can be restored to create a private workspace or to load a different version. All information from a selected zip file is restored into the current StP system.

To restore a baseline for a subsystem:

1. Select the subsystem you want to restore in the **ModelManagement View** on the desktop.
2. From the **Tools** menu, choose **Model Management > Restore Subsystem**.
3. In the **Restore Subsystem** dialog box, enter or browse and choose the baseline file from which to restore the subsystem.

The behavior and the actions that are performed during the restore depend on the existing information in the system, as described in the following sections.

Restoring to an Empty System

This is the easiest case; no checks are necessary. All information in the zip file is restored and all restored external model elements are locked to prevent them from changing.

Restoring to a System with Other Subsystems

In this case, the StP system contains legal model elements. Therefore, StP has to check for name conflicts before restoring the subsystem. A name conflict occurs if existing model elements in the StP system have the same name as model elements in the baseline, but belong to different subsystems. Name conflicts need to be resolved by the user before the restore operation can succeed.

External model elements that already exist in the current system are not restored, to avoid overwriting the current version.

All external model elements are locked to prevent them from changing.

Restoring to a System Where this Subsystem Exists

This occurs if a subsystem from a previous version or a subsystem from a private workspace is restored.

StP first checks for name conflicts, which must be resolved as described in “Restoring to a System with Other Subsystems” on page 3-22. Next, StP checks to see whether the same model elements in the baseline also exist in the current system, but have different names. If so, the model elements are renamed in the current system to ensure that a consistent model exists after the restore.

All external model elements are locked to prevent them from changing.

Reconfiguring External Model Elements

The **Reconfigure External Model Elements** command allows you to update all external model elements in your current private workspace model with the current version of those elements from the main model. For example, suppose you are working on *MailMessenger* in your private workspace and you learn that *MailServer* (and many others that you do not "own") have changed. You need to re-import the external model elements into your private model.

This command looks for the external subsystem zip files in a path defined by the `baseline_path` ToolInfo variable. The default path is *$(projdir)\baseline*. You can select a different path in the command's dialog box. It then updates all external model elements, such as class tables and annotations of external elements, provided a zip file containing the parent subsystem of that element is present in the selected directory. This means that in order to update all external elements of a system, all subsystem baselines (zip files) need to be put into the selected baseline directory (for example, by checking all those files out of a CM tool).

To invoke this command, select a subsystem in the **ModelManagement View** on the desktop and do either of the following:

- From the **Tools** menu, choose **Model Management > Reconfigure External Model Elements**.
- From the shortcut (right-click) menu, choose **Reconfigure Externals from Baseline(s)**.

Known Issues and Limitations

To see recent changes or additions to subsystems, it may be necessary to refresh the current model management window or dialog box, using the dialog's **Reset** button or the **Refresh Tree** command on the desktop **ModelManagement View** shortcut (right-click) menu. For example, if a new subsystem is added, it is not immediately visible in the **Baseline Subsystem** dialog until you click the dialog's **Reset** button. Likewise, if you move a file into a subsystem as described in "Manually Assigning

Diagrams to Subsystems” on page 3-12, it will not appear in the **ModelManagement View** on the desktop until you refresh the desktop, using the **Refresh Tree** command.

Baselining a subsystem that has nested subsystems generates a separate zip file for each subsystem. When you restore this subsystem, you need to restore each of the zip files separately.

4 Baselining and Version Control

StP ME expands on the baselining capabilities that were introduced in StP 7.2. StP ME includes the ability to create baselines for StP/UML subsystems and an interface to Continuus configuration management software.

This chapter describes:

- "Creating, Restoring, and Comparing Baseline Systems" on page 4-1
- "Using External CM Systems for Version Control" on page 4-9

Creating, Restoring, and Comparing Baseline Systems

Baselining allows you to save the state of your StP system at various points during the development process. StP's baselining capabilities allow you to:

- Create a baseline—Save the state of your StP system, or StP/UML subsystem, at various points during the development process.
- Restore a baseline—Restore the saved baseline state into StP as a separate system in order to manually check what has changed and possibly revert to the previously baselined version.
- Manage the baselines—Use StP's version control interface for supported external configuration management (CM) systems, such as ClearCase or Continuus, to manage the various baselines you have created.
- Compare two baselines.

You can perform an StP baseline operation with or without an integrated version control/configuration management (CM) system. When a version control system is integrated with StP, it is detected with the `use_version_control` ToolInfo variable. The specific version control system is specified by the `vcc_cmds` ToolInfo variable. For more information on setting up a version control integration, see “Using External CM Systems for Version Control” on page 4-9.

Version control for individual files within an StP system is no longer supported. However, baselining and version control for StP/UML subsystems is supported in this release.

Note: StP’s baselining system does not include built-in “differences” or “merge” functions, which are typically associated with CM systems. To detect differences between two systems or subsystems, you can perform a differences operation on the ASCII flat files or open two instances of StP and Alt+Tab between each current diagram and the baseline. The **Restore Subsystem** command effectively merges a baselined subsystem with the entire system.

A Typical Scenario

Several users are working simultaneously on a model. The project administrator wants to make sure that, if undesirable changes are made to the model, it will be easy to revert to a state where the model was still valid. The project administrator uses the StP baseline and version control capabilities to accomplish this goal.

Here is what might be involved:

1. The project administrator creates a baseline of the system on a regular basis.
2. The project administrator uses the StP interface to check the baseline file into a CM system. The CM system keeps track of date and log information about the baseline file(s).
3. A problem occurs: someone made undesired changes to the model.
4. The project administrator checks the most recent version out of the CM system.
5. The project administrator restores the baseline, overwriting the current system. The problem is solved.

Setting Up Baselining Capabilities

Before you can use baselining in StP, you must add the following variables to your ToolInfo file to allow StP's create, restore, and baseline management functions to work properly:

- **`baseline_path`**. Set this variable to any valid directory path. For example:

  ```
  baseline_path=c:\TestSystems\baseline
  ```

  If the variable is set, the variable-specified baseline path will appear as the default path in the:

  - **Save to Directory** field of the **Create Baseline of Current System** dialog
  - **Full Path of Baseline File** field of the **Restore System from Baseline** dialog

  If the `baseline_path` variable is not set, the defaults for both of these fields is *${projdir}/baseline.* If a baseline subdirectory does not exist in your *${projdir}*, StP will create it automatically the first time a baseline is created (see "Creating a Baseline" on page 4-4).

- **`use_version_control`**. Set this variable to either True or False. For example:

  ```
  use_version_control=True
  ```

  If the variable is set to True, the **Version Control Command** dialog appears automatically after a baseline has been created, so you can quickly check the newly created baseline into an external CM system

  Note: You must have configured StP to work with a particular CM system, as described in "Setting Up Your CM Software Integrations" on page 4-9.

  In addition, the `use_version_control` ToolInfo variable controls the default for the **Baseline System/Subsystem name** option on the **Baseline Current System/Subsystem** dialog box (see "Creating a Baseline," which follows). If the variable is set to True, the baseline name is the system or StP / UML subsystem name. If it is set to False, the baseline name is concatenated with the current date (for example, *my_system_082001*). This allows users who are not using a CM system to keep track of their baselines to quickly determine when each baseline file was created.

  If the variable is not set, the default is False.

Creating a Baseline

To create a baseline:

1. Make sure you have opened a system in StP.

   If creating a baseline for an StP / UML subsystem, select the subsystem under the **ModelManagement View** category in the desktop's model pane.
2. Select one of the following commands from the desktop **Tools** menu, as appropriate:
   - **Baseline > Baseline Current System**
   - **Model Management > Baseline Subsystem** (StP / UML only)
3. In the **Baseline Current System/Subsystem** dialog box, enter or make selections as described in "Baseline Current System / Subsystem Dialog Summary" (immediately below) and click **OK**.

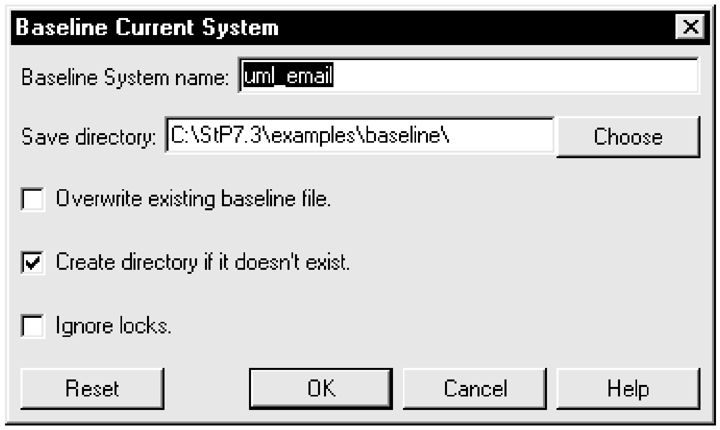
   StP checks for locks, stores the repository information, and archives it all into a zip file, as described in "How StP Creates the Baseline" on page 4-6.
4. In the **Version Control Command** dialog (which appears if StP is configured to use an external configuration management product), do the following:
   - Select one of these commands, as appropriate:
     - Create Initial Version
     - Check In New Version
   - Select appropriate options (see "Version Control Command Dialog Summary" on page 4-11)

   Click **OK** to update your CM system with the newly created baseline archive (see "Using External CM Systems for Version Control" on page 4-9).

Baseline Current System/Subsystem Dialog Summary

Figure 1 shows the **Baseline Current System** dialog. The **Baseline Subsystem** dialog is nearly identical, but includes a pick list of subsystems in your model and an option to lock the subsystem after baselining. Table 1 describes the text fields and options on the dialog.

Figure 1: Baseline Current System Dialog


Baseline Current System

Baseline System name: uml_email

Save directory: C:\StP7.3\examples\baseline\

[ ] Overwrite existing baseline file.

[✓] Create directory if it doesn't exist.

[ ] Ignore locks.

Table 1: Baseline Dialog Summary

Field	Description
Baseline System/ Subsystem name	The name of the file (a zip archive) that will be created. The default is either *${system}* or *${system}_<date>* depending on the value of the `use_version_control` ToolInfo variable (see "Setting Up Baselining Capabilities" on page 4-3). You do not need to add the *.zip* extension; it is appended automatically when the archive is created.
Save directory	The directory in which the zip archive will be created. The default is either `baseline_path`, if this ToolInfo variable is set in the ToolInfo file, or *${projdir}\baseline* otherwise. If the directory specified does not exist, and the **Create directory if it doesn't exist** option is selected, the directory is created automatically.
Overwrite existing baseline file	Overwrites an existing file with the directory and file name specified in the previous two text fields. If this option is not selected, the baseline create routine informs you that a file with this name already exists and exits without creating a baseline.

Table 1: Baseline Dialog Summary (Continued)

Field	Description
Create directory if it doesn't exist	Automatically creates the directory specified in the **Save to directory** text field, if it does not exist already. **Note:** Creates a directory only one level down (a limitation of the **mkdir** function in QRP).
Ignore lock(s)	Checks to see if any locks are set. **Note:** Unexpected results may occur if locks exist at the time the baseline create routine is executed. We recommend that you remove any locks (using the functions in the **Locks** submenu of the **Tools** menu) before attempting to create baselines. If **Ignore locks(s)** is selected, baseline creation continues even if certain locks exist in the system. The list of locks is written to a lock information file called *baseline_lock.log* within the system directory, making it available in the baseline for future reference. Selecting this option will not force baselining of a locked subsystem. If **Ignore locks(s)** is not selected (and locks exist), you will be informed and the baseline will not be created.
Lock subsystem after baselining	Locks the subsystem after creating the baseline (StP/UML only).

How StP Creates the Baseline

StP checks to see if there are any locks on the system and proceeds according to how the **Ignore lock(s)** option was set in the **Baseline Current System/Subsystem** dialog.

If the system uses a Sybase repository, the repository is “dumped” via a Sybase dump function into a temporary directory within the StP system directory structure. This is a time-saving feature, as it is quicker to restore a Sybase repository from a dump than to rebuild it from the StP system files. For Jet systems, the repository information is stored in a file within the system directory, so a “dump” step is unnecessary.

The system directory (including the Sybase repository dump and/or lock information file) is archived into a single file, using *zip.exe***,** a public-domain zip utility provided with StP. The file name contains the name and directory location you've specified and always ends in *.zip*. If you specify a name without the *.zip* extension, it is added automatically.

Restoring a Baseline

To restore a baseline for your entire system or for an StP/UML subsystem:

1. From the desktop **Tools** menu, select either:
   - **Baseline > Restore Selected Baseline System**
   - **Model Management > Restore Subsystem** (StP/UML only)

   The **Restore Selected Baseline System** or **Restore Subsystem** dialog appears.
2. In the **Full path of baseline file** field, enter or browse and choose the name of the baseline file, including the full directory path. For example:

   C:\users\lorib\myBaselines\systemX_072201.zip
3. On the **Restore Selected Baseline System**, enter or browse and choose the following additional information (only if restoring the complete system):
   - **Restore to directory** —Name of the full path to the directory where the baseline file is to be restored, typically your StP project directory.
   - **Restore System name**—Name of the system into which the restored baseline file is to be saved.
4. Click **OK**.

   If the system to be created already exists, the existing system directory is overwritten with the restored files.

   Otherwise, StP creates a new system with the specified name, to which the unzipped files are transferred, overwriting any empty default files. For restored Jet systems, the extracted *.mdb* file is renamed as the new system name. For restored Sybase systems, the extracted Sybase dump files are loaded into the new Sybase database that was created.

   When StP completes the restore operation, the system is ready to use.

Comparing Baselines

Starting with StP 8.3, you can compare two versions of the same StP system or even two different StP systems (although that may have limited utility). The systems to be compared can be in the form of an actual system or a system baseline. The feature is available in both UML and SE. The result of baseline comparison is a report, compare_system, that lists the differences between the two systems. Examples of changes reported are new symbols and diagrams, global renames of symbols and diagrams, annotation / description changes, etc.

Note: The systems to be compared must have the same root. That is, they must have been created with the same version of StP (e.g., 8.3 or greater). You cannot migrate and then compare.

As an example of usage, compare the current system to an external system as follows.

1. Open a system in StP, then choose **Report > Start Script Manager**.
2. In the script manager, open the **Product Scripts** category and then the **report** category. Click on *compare_system* in the set of reports.
3. Specify the location of the second system.

   To do this, select then right click the *system2* category in the **Externals** pane. Then select **Edit External Value** from the shortcut menu. In the edit window that opens, specify the full path of the second system, then close the window after saving your changes.
4. Choose **File > Run Script**. In the dialog that appears, select the output format (e.g., RTF). Click **View After Run**, then click **OK**.

When the process is complete, the report is sent to *compare_systems.<extension>* in the *qrl_files* directory of the current system. You may view the file directly or by choosing **Report > Open Report** from the StP desktop.

You can do other comparisons by changing the *proj*, *system*, or *baseline* external values (step 3) appropriately.

Using External CM Systems for Version Control

You may want to keep track of your StP baselines with a configuration management (CM) system. StP provides integrations between the following configuration management products and StP's baselining facilities:

- Continuus (Continuus Software Corporation)
- ClearCase (Rational Software Corporation)
- GNU RCS (a public domain version control tool)

In addition, since the architecture of the interface is "open," knowledgeable users should be able to customize it for a different CM system, with or without the help of Aonix application engineers (AEs).

Setting Up Your CM Software Integrations

To use StP's baselining utility with an external configuration management system:

- Set the `use_version_control` ToolInfo variable to True in your ToolInfo file, as described in "Setting Up Baselining Capabilities" on page 4-3. This tells StP to display a special dialog after creating the baseline, which enables you to check the baseline directly into the CM system.
- Set other variables as directed for Continuus or ClearCase integrations in the sections that follow.

For more information about the supported and other possible CM integrations, see the *README.vcc* file in *<StP_install_dir> \ templates \ ct \ vcc*.

Setting up the Continuus Integration

StP supports an interface to Continuus 4.5 (Continuus, Inc.). Before using the integration, you must:

1. In your ToolInfo file, set the `vcc_cmds` ToolInfo variable to the *Vcc_continuus_commands* file:

```
vcc_cmds=ct\vcc\continuus\Vcc_continuus_commands
```

For UNIX, use forward slashes.

2. In the *Vcc_continuus_commands* file, set the following appropriately for your particular installation of Continuus:

   ```
   vcc_server
   vcc_datadir
   vcc_dbname
   ```

3. Set the `CCM_PASSWORD` environment variable to your Continuus password.

Setting up the ClearCase Integration

StP ME supports an interface to ClearCase configuration management software.

Before using the integration, you must:

1. In your ToolInfo file, set the `vcc_cmds` ToolInfo variable to the *Vcc_clearcase_commands* file:

   ```
   vcc_cmds=ct\vcc\clearcase\Vcc_clearcase_commands
   ```

   For UNIX, use forward slashes.

2. In the *Vcc_clearcase_commands* file, uncomment the examples of the following variable settings and set them to values appropriate for your installation of ClearCase:

   ```
   vcc_vobtag
   vcc_viewtag
   ```

   Note: To determine the appropriate values, run the ClearCase commands **lsvob** and **lsview**. The `vcc_viewtag` variable may be set to a specific directory within the view where baselines are to be created.

Setting up the RCS Integration

Before using the RCS integration, you must set the `vcc_cmds` ToolInfo variable to the *Vcc_rcs_commands* file:

```
vcc_cmds=ct\vcc\rcs\Vcc_rcs_commands
```

For UNIX, use forward slashes.

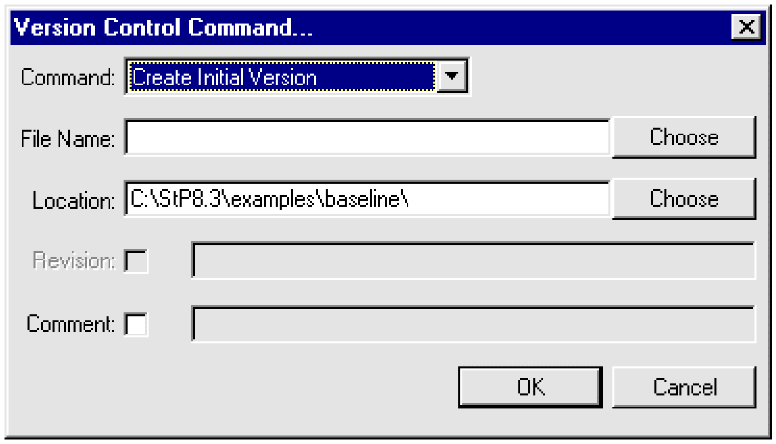
Using Version Control

To use StP's version control tool, you access the **Version Control Command** dialog in either of the following ways:

- Automatically, from StP's baselining utility (if the `use_version_control` ToolInfo variable is set to True)
- From the desktop, by choosing **Tools > Version Control**

Figure 2 shows the **Version Control Command** dialog box.

Figure 2: Version Control Command Dialog



Command: Create Initial Version

File Name:

Location: C:\StP8.3\examples\baseline\

Revision:

Comment:

Version Control Command Dialog Summary

Table 2 describes the text fields and options on the dialog. The availability of each option depends on the command selected from the **Command** option list on the dialog.

Table 2: Version Control Command Dialog Summary

Field	Description
Command	Specifies the command to be performed. Select the desired command from the drop-down options list (see Table 3 on page 4-12 for command descriptions).

Table 2: Version Control Command Dialog Summary (Continued)

Field	Description
File Name	Specifies the name of the file on which the command is to be performed. Required by all commands except **List Files Under Version Control**.
Location	For the **Create Initial Version** and **Check In New Version** commands, specifies the directory path where the file on which the command to be performed is located. For **Get Read-Only Version** and **Get Version & Lock**, specifies the directory path where the file is to be placed.
Revision	Allows access to versions of the file other than the most recent version (available for the **Get Read-Only Version** and **Get Version & Lock** commands only). If it is selected, you can enter appropriate revision information. For ClearCase, an example would be *\main\1*. If not selected, the command defaults to the most recent version of the file.
Comment	Available for the **Create Initial Version**, **Check In New Version**, and **Delete File From Version Control** commands. Allows you to enter comments that will be sent to your selected CM system as a result of executing the command. These comments can be viewed in the history of the file; they provide a mechanism for tracking changes to the file. Quotation marks are not required.

Note: Table 3 describes the commands listed in the **Command** option list on the **Version Control Command** dialog. Some commands may not be available in your particular configuration management system, or may have a slightly different name than that shown in Table 3.

Table 3: Version Control Commands

Command	Description
Create Initial Version	Adds the user-specified baseline file to the CM system. This command allows you to specify a comment to record more information about the file being created.

Table 3: Version Control Commands (Continued)

Command	Description
Get Read-Only Version (check out)	Gets a copy of the file specified in the **File Name** field ("<*file*>") from the CM system and places it in the directory specified in the **Location** field ("<*location*>"). You can specify a particular revision, or get the most recent revision by default.
Get Version & Lock, or **Check Out File**	Gets a "locked" copy of <*file*> from the CM system and places it in <*location*>. You can specify a particular revision, or get the most recent revision by default. The definition of "lock" depends on the implementation of locking in the CM system being used. In ClearCase, "locked" means "reserved."
Unlock File	Performs an unlock operation on <*file*>. The definition of "unlock" is related to the definition of "lock" and depends on how locking is implemented in the CM system being used. In ClearCase, **Unlock File** simply cancels reserved checkouts of <*file*>.
Check In New Version	Checks a new version of <*file*>, located in <*location*>, into the CM system. This command allows you to specify a comment to record more detailed information about the file being checked in.
Show Change History	Provides an interface to the CM system's history function for <*file*>. The history is displayed in the **Version Control Results** scrollable window for easy viewing.
Delete File From Version Control	Provides an interface to the CM system's delete function for <*file*>. This command allows you to specify a comment to record more information about the file being deleted.
List Files Under Version Control	Displays a list of the files in the CM system. Depending on your CM system, this may include all files in the current view. The list is displayed in the **Version Control Results** scrollable window.

5 Usability and Modeling Enhancements

StP ME has introduced extensively enhanced modeling capabilities and usability improvements designed to make StP more powerful, more intuitive, and easier to use. These enhancements began with the StP ME 8.0 release and have continued through to the current release.

These modeling and usability enhancements include:

- "OMG UML Compliancy Updates" (immediately below)
- "Easier Access to Frequently Used Features" on page 5-2
- "Arc Drawing Enhancements" on page 5-10
- "Tree-Structured File Browsing in Dialog Boxes" on page 5-12
- "More Pick Lists for Names and Properties" on page 5-13
- "StP / UML Global Rename with Pattern Matching" on page 5-18
- "Navigation Enhancements" on page 5-19
- "Publishing and Printing Enhancements" on page 5-19

Other usability and modeling enhancements apply to specific diagram and table editors. These are disccused in Chapter 6, "StP / UML Editor Enhancements."

OMG UML Compliancy Updates

Symbols in the StP / UML editors were updated in StP 8.1 to comply with OMG UML 1.3 standards.

In addtion, enhancements to the StP / UML editors also provide users with additional modeling capabilities, in accordance with OMG UML 1.3 standards. These capabilities are discussed in Chapter 6, “StP / UML Editor Enhancements.”

Easier Access to Frequently Used Features

Additional toolbar buttons and related commands provide easier access to more of the frequently-used StP editor features, including:

- “Starting a New Diagram” on page 5-3
- “Choosing Multiple or Single Editor Sessions” on page 5-3
- “Editing Object Properties and Annotations” on page 5-4
- “Editing ToolInfo Variables from the Desktop” on page 5-6
- “Quickly Entering Descriptions” on page 5-8
- “Other Editor Enhancements” on page 5-8

For additional toolbar shortcuts in the StP / UML class table editor, see “Adding New Attributes and Operations” on page 6-10.

StP / UML also provides quick entry and editing of textual descriptions of objects, using a new editor toolbar shortcut, menu item, or the **Description** button on object property sheets. Figure 1 and Figure 2 show the new editor toolbars.

Figure 1: Diagram Editor Toolbar (StP/UML)

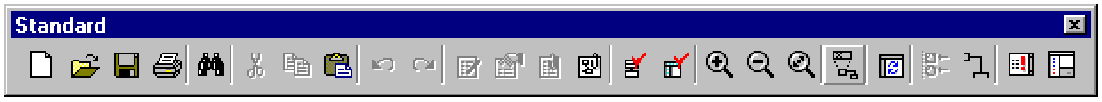
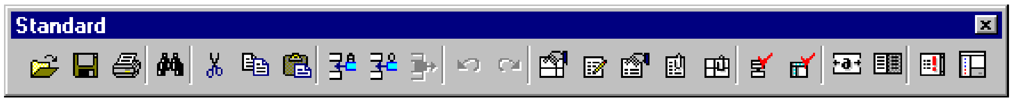


Figure 2: Table Editor Toolbar (StP/UML Class Table)



In addition to the above, StP includes a **Help** menu item to connect to the Aonix Customer Support website. You can use it to submit information for bugs and enhancements.

Starting a New Diagram

StP ME provides a menu command and toolbar button to easily create a new diagram from within an StP editor. See Table 1 below.

Table 1: New Diagram (Command and Toolbar Button)

Toolbar Button or Menu	ToolTip or Command	Description
🗋	**New Diagram**	Opens a new blank diagram, after closing the current one, if any, and prompting you to save any modifications
File >	**New**	

Choosing Multiple or Single Editor Sessions

StP ME improves access to an earlier feature that allows you to choose whether or not StP starts multiple, concurrent sessions of the same editor.

On the StP desktop, you can start multiple sessions of an editor with the Start New Editor toolbar icon. To start a new editing session, select the name of a diagram type (e.g., use case) in the model (right) pane to make the Start New Editor icon available, then click the icon. Each time you click the icon, StP opens a separate session of the editor (e.g., the use case editor).

In an StP editor, you can start multiple sessions of the editor by choosing **File > Open in New Window**. Doing this freezes the current editor window and opens a new editor session. The original editor window will now have "[frozen]" attached to its label. Setting an editor to a frozen state tells StP to open a new editor window of that type every time you open a file from the desktop or navigate to that editor type from another editor.

To unfreeze the original window, choose **Tools > Unfreeze Current View** in that window. To re-freeze, choose **Tools > Freeze Current View**.

Editing Object Properties and Annotations

Toolbar Buttons

The table below describes the new toolbar buttons in StP editors that provide access to the tools and property sheets for entering additional information about an object, diagram, or table.

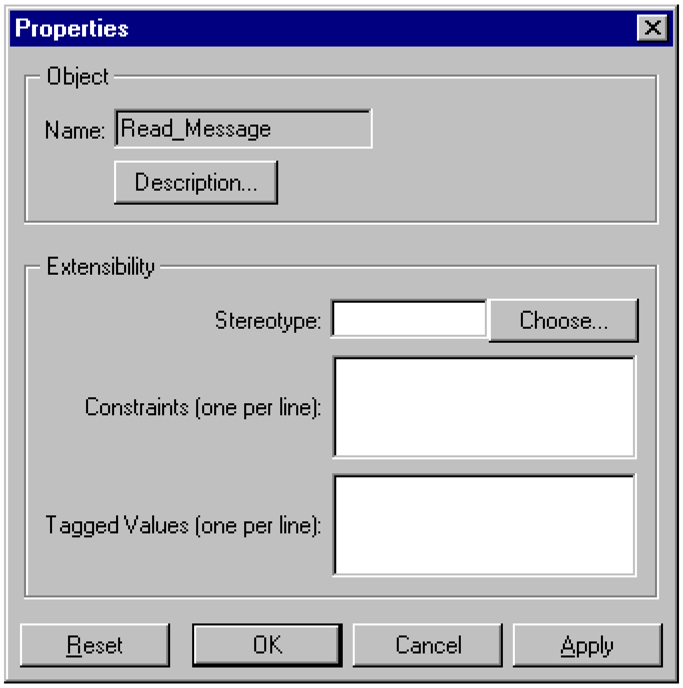
Table 2: Property and Annotation Toolbar Buttons

Button	ToolTip	Description
	Object Description, **Cell Description**	(StP/UML only) Displays a text editor window for entering an object description, which is saved as an annotation for the selected object (shortcut alternative to entering note descriptions in the OAE). For more information, see "Quickly Entering Descriptions" on page 5-8.
	Properties	Displays the property sheet for a selected object (StP/UML only).
	Cell Content	Displays the **Cell Content** dialog box for a selected cell, which allows you to view, enter or edit a long cell label without resizing the column.
	Object Annotation, **Cell Annotation**	Starts the object annotation editor (OAE) and displays any existing annotations for the selected object or cell.
	Diagram Annotation	Starts the OAE and displays any existing annotations for the current diagram or table.
	Table Annotation	

Property Sheets

StP allows you to specify properties of objects by providing object property sheets in all diagram editors and the class table editor. A typical object property sheet (in this case, for the use case Read_Message in the *uml_email* example) is shown in Figure 3.

Figure 3: Typical Object Property Sheet



Name: Read_Message

Stereotype:

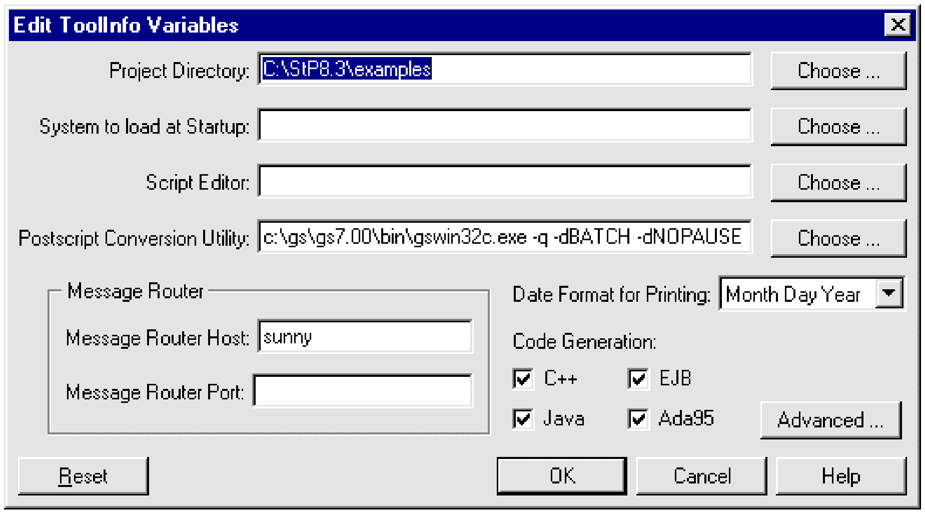
Constraints (one per line):

Tagged Values (one per line):

Starting with StP release 8.3, you can also use property sheets to specify properties of *object instances* in the class diagram, sequence, collaboration, and activity editors. Similar property sheets are available for *attributes* and *operations* in the class diagram editor. These property sheets have a format similar to that shown in Figure 3, except for the label (e.g., “Operation / Operation Name” instead of “Object / Name”).

Editing ToolInfo Variables from the Desktop

Starting with StP release 8.3, you can edit ToolInfo variables from the desktop. To invoke this feature, choose **Tools > Edit ToolInfo Variables**. The **Edit ToolInfo Variable** dialog appears.



Project Directory: C:\StP8.3\examples

System to load at Startup:

Script Editor:

Postscript Conversion Utility: c:\gs\gs7.00\bin\gswin32c.exe -q -dBATCH -dNOPAUSE

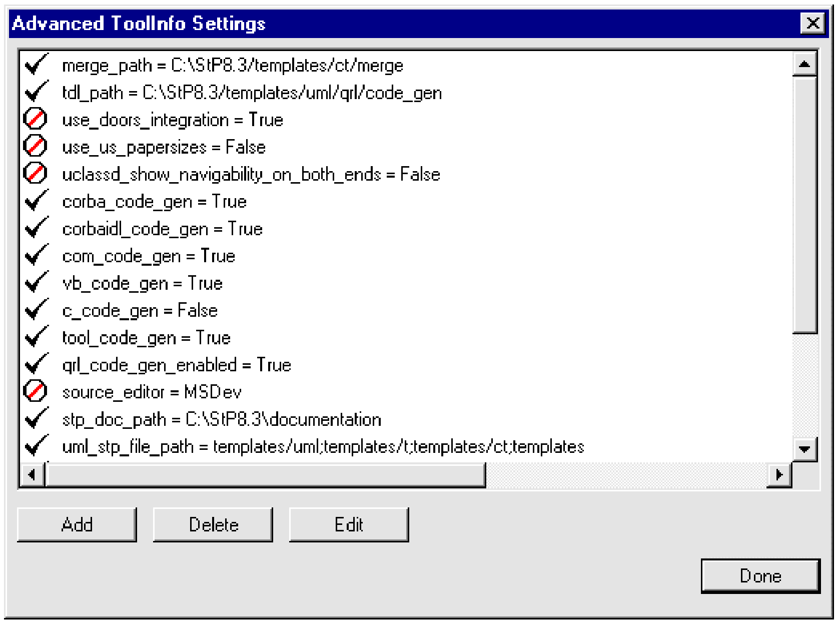
Message Router Host: sunny

Message Router Port:

Date Format for Printing: Month Day Year

Code Generation: ☑ C++ ☑ EJB ☑ Java ☑ Ada95

You can change `postscript_conversion` on the dialog. You can change other variables by clicking the **Advanced** button. An example ToolInfo settings screen is shown below.



```
✓ merge_path = C:\StP8.3/templates/ct/merge
✓ tdl_path = C:\StP8.3/templates/uml/qrl/code_gen
⊘ use_doors_integration = True
⊘ use_us_papersizes = False
⊘ uclassd_show_navigability_on_both_ends = False
✓ corba_code_gen = True
✓ corbaidl_code_gen = True
✓ com_code_gen = True
✓ vb_code_gen = True
✓ c_code_gen = False
✓ tool_code_gen = True
✓ qrl_code_gen_enabled = True
⊘ source_editor = MSDev
✓ stp_doc_path = C:\StP8.3\documentation
✓ uml_stp_file_path = templates/uml;templates/t;templates/ct;templates
```

A check next to the variable indicates the setting in effect. To change the value of a variable, click on its name, click **Edit**, change its value, then click **OK**. When you have finished editing, click **Done** on the settings dialog, then click **OK** on the edit dialog. You will be asked if you want to restart StP now or later. Respond as desired. The new settings will go into effect when StP is restarted.

A circle with a red bar indicates that the variable is defined in the file but has been commented out and has therefore no effect on StP's behavior. To enable a commented-out ToolInfo variable, click on its name and select **Edit**. In the edit window a checkbox is on the left hand side. Click the checkbox, change its value (if desired), and press **OK**. The ToolInfo variable will now be turned on.

Quickly Entering Descriptions

StP/UML diagram editors and the class table editor provide the following new commands and property sheet buttons to enable you to quickly add or edit descriptions for a selected object:

- **Object Description** editor toolbar button
- **Object Description** or **Cell Description** on the **Edit** and shortcut (right-click) menus in diagram or table editors, respectively
- **Description** button on property sheets

These commands and buttons bypass the OAE and open a simple text entry window or external editor in which you can enter a description. The description is saved in the Object annotation note for the selected object.

To use an external editor for the quick note description feature, set the *NoteDescEditor* ToolInfo variable to the name of the edit program you want StP to invoke (e.g., Notepad). Changes made in the external editor are not manifested in StP until you exit the editor.

You can also define a default template for a descriptions to make sure that all objects of a specific type are described the same way. This template is loaded into the description window or the external editor when you enter the object description the first time.

You can easily create such templates by storing files in *<projdir>\stp_env\ndt_files\* with the file name <*type*>_GenericObject (for example, UmlClass_GenericObject). If you are using Microsoft Word as the note description editor, create the templates files in *.rtf* format.

The templates files are not used when you edit a description in the OAE.

Other Editor Enhancements

In addition to the enhancements described previously in this chapter, StP ME provides toolbar access for other frequently used features, such as checking diagram semantics, and an option for specifying whether annotations are to be cloned. These features were introduced in StP release 8.2. See the sections below for descriptions of these enhancements.

Toolbar Enhancements

Table 3 describes other toolbars added to StP editors in StP ME.

Table 3: Other Toolbar Enhancements

Button	ToolTip	Description
	Check Syntax	Checks the diagram syntax to validate that the drawing is correct.
	Check Semantics	Checks the diagram semantics to validate that the underlying model is correct as reflected in the repository.
	Show Desktop	Displays the StP desktop.
	Show Panner	(Diagram editors only) Displays the diagram panner for viewing large diagrams and different areas of a zoom-enlarged diagram.

Cloning Annotations Option

StP diagram editors now have an option to specify whether a symbol's annotations are to be cloned when the symbol is cloned. A symbol is "cloned" when you change the label of the symbol. In this situation, StP creates a *new* symbol with the new label and copies/clones the annotation from the original symbol. The original symbol (and its annotation) remain in the repository, possibly as an unreferenced symbol in case it is not used elsewhere.

If you want to prevent this behavior, choose **Tools > Options > General** and then make sure **Cloning Annotations** is turned off (the default is on). In this case, if you change the label of a symbol, StP also creates a new symbol but *without* copying/cloning the annotations.

Arc Drawing Enhancements

In StP ME, you can draw different types of arcs easily and quickly, replace them with a different arc type, and switch orthogonal arcs on or off with one button. These enhancements are discussed in the following sections:

- "Drawing Different Types of Arcs" on page 5-10
- "Setting the Position of the DefaultArcType Toolbar" on page 5-11
- "Replacing Existing Arcs with a Different Arc Type" on page 5-12
- "Drawing Orthogonal Arcs" on page 5-12

Drawing Different Types of Arcs

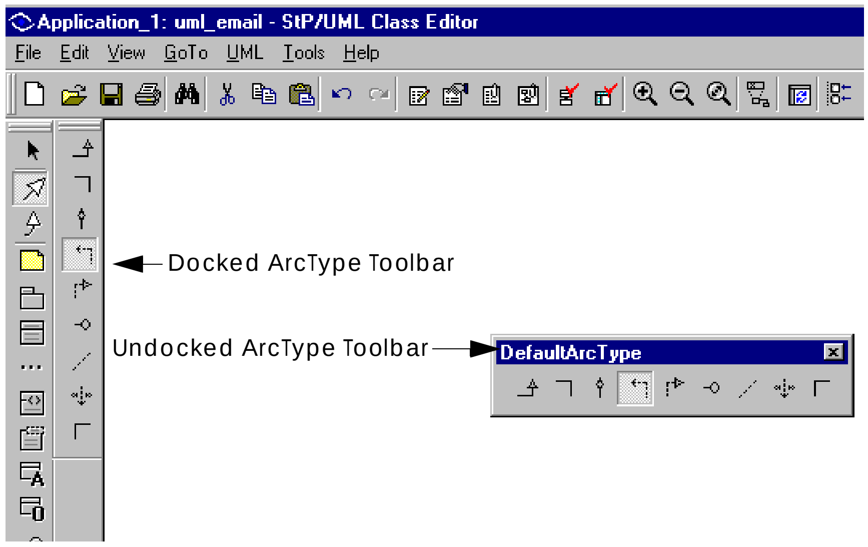
In StP ME you can draw any type of arc—UML generalizations, aggregations, and dependencies, and SE data and control flows, to name a few—easily and quickly, with direct access to each arc type on the new DefaultArcType toolbar (Figure 4).

To use the new DefaultArcType toolbar:

1. Select the Arc or Spline tool on the Symbol toolbar.
2. Select an arc type on the DefaultArcType toolbar.

   Your selection becomes the default arc type until you select another type of arc on the arc toolbar.

Figure 4: DefaultArcType Toolbar


Application_1: uml_email - StP/UML Class Editor
File Edit View GoTo UML Tools Help
Docked ArcType Toolbar
Undocked ArcType Toolbar
DefaultArcType

Setting the Position of the DefaultArcType Toolbar

You can dock or undock the DefaultArcType toolbar the same as you do the Symbol and standard editor toolbars:

- Undock—Drag the toolbar away from its docked position by the double bar “handles” at its top edge.
- Dock—Drag the toolbar into one of the editor’s borders.

In addition, you can hide the DefaultArcType toolbar when it is not being used. Hiding the toolbar causes it to appear only when you select the Arc or Spline tool, and to be hidden when you have finished drawing the arc(s).

To hide the DefaultArcType toolbar:

1. From the **Tools** menu, choose **Options**.
2. On the **Options** dialog, display the **Default Arc** tab.
3. Clear the **Default Arc Type Toolbar Is Permanent** check box.

Replacing Existing Arcs with a Different Arc Type

To replace an existing arc with a different type of arc:

1. In the diagram, select the arc to be replaced.
2. From the **Edit** menu, choose **Replace**.

   If only one other type of arc is possible in this context, StP redraws the arc accordingly.

   Otherwise, StP displays the **Replace Arc Type** dialog.
3. In the **Replace Arc Type** dialog, select the replacement arc type and click **OK**.

Note: StP/SE users can use a **Replace** button on the data flow editor's standard toolbar to quickly redraw a data arc as a control arc, or a control arc as a data arc.

Drawing Orthogonal Arcs

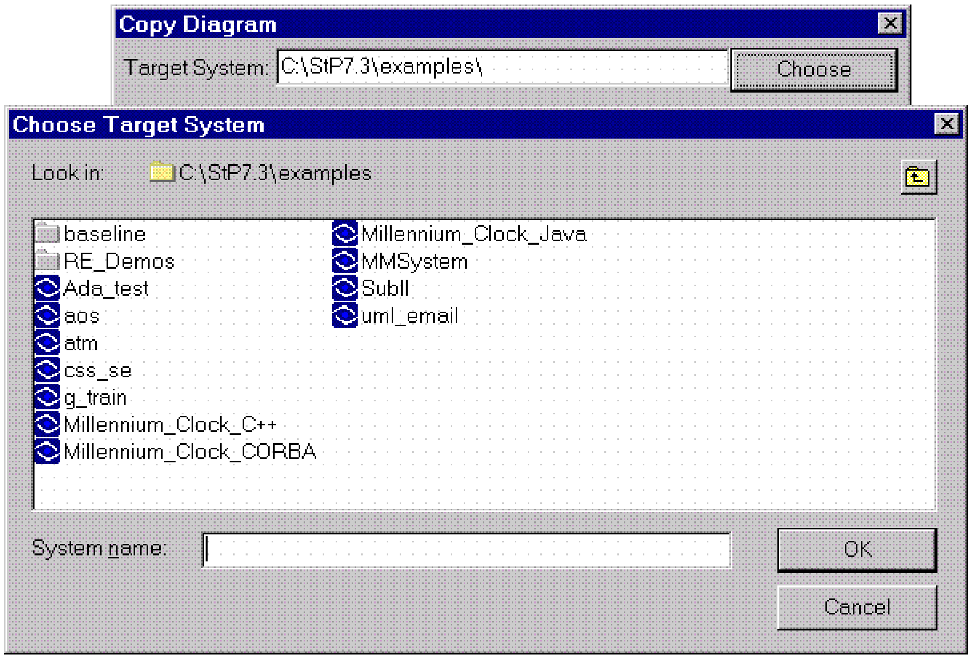
Now you can quickly toggle the orthogonal arc drawing style on or off, using the **Toggle Orthogonal Drawing** toolbar button. This button sets the the default arc-drawing method to orthogonal (right-angled) or non-orthogonal arcs.

Toggle Orthogonal Drawing

Tree-Structured File Browsing in Dialog Boxes

Dialogs that require selection of a directory file name now have a **Choose** button that allows you to use a file browser to select the path name.

Figure 5: Dialog Box File/Folder Browser



More Pick Lists for Names and Properties

StP ME provides several "pick lists" for object names and properties in the StP / UML editors, based on standard and user-defined items in the repository. Reusing existing names and attributes eliminates guesswork and labeling inconsistencies, and improves efficiency. Table 4 summarizes the new pick lists available for each StP / UML editor and property sheet.

The former **List Labels** command, available in StP / SE and StP / UML editors, now has a new context-sensitive command name, **Choose Names** or **Choose** <*Apptype*> **Names**. For more information, see "Choose Names Command" on page 5-15.

Table 4: New StP/UML Properties Pick Lists

For What Object?	Allows Selection Of	To Display Pick List	For More Information, See
Any diagram object or table cell	An existing name from the repository for this object's or cell's label]	**Choose <Apptype> Names** on the **Edit** or shortcut (right-click) menu	"Choose Names Command" on page 5-15
Objects for which you can define a stereotype	Predefined or user-defined stereotype	**Properties** toolbar button, or **Edit** > **Properties**, then **Choose** in the property sheet's **Stereotype** field	"UML Stereotype Pick List" on page 5-16
Class diagram: Association	Predefined multiplicities (or manual entry of any number)	**Properties** toolbar button, or **Edit** > **Properties**, then **Choose** on property sheet	"Editing Role Names and Choosing Multiplicities" on page 6-8
Class table: Attribute type cell or operation return type cell:	Attribute types or return types of operations	**Fill Type** on the **Edit** or shortcut (right-click) menu	"Choosing Class Attribute and Operation Types" on page 6-11
Class table: Operation's argument cell	An operation's arguments	**Edit > Edit Signature**	"Choosing an Operation's Arguments" on page 6-13
Collaboration and sequence diagrams: Messages	A predecessor guard, based on public attributes and operations of the message's target class. Name component, return type, and message arguments, based on the public operations of the target class.	**Properties** toolbar button, **Edit** > **Properties**, or **UML > Set Message Name**; then **Choose** on property sheet	"Sequence and Collaboration Editors" on page 6-16

Table 4: New StP/UML Properties Pick Lists (Continued)

For What Object?	Allows Selection Of	To Display Pick List	For More Information, See
Activity diagram: State transition link	Guard conditions and an action list	**Properties** toolbar button, or **Edit** > **Properties**, then **Choose** on property sheet	"Labeling State Transitions with Events, Actions, and Guards" on page 6-19
State diagram: State transition link	Name component, event (incoming), guard, action list, and event (send)		

Choose Names Command

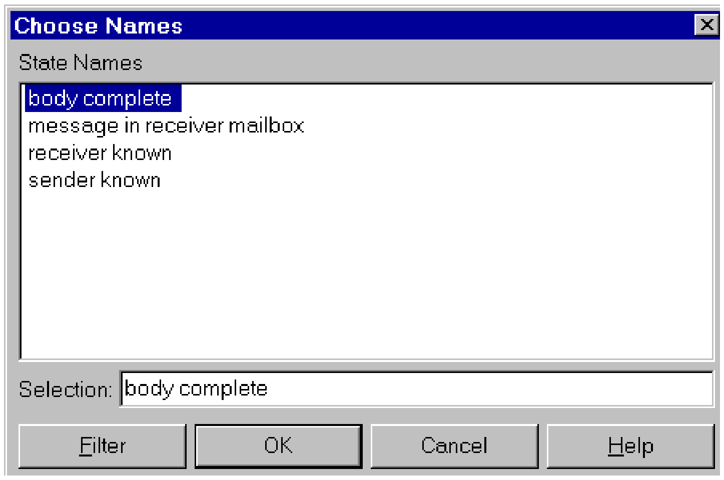
To choose a name for a selected object or cell from a list of existing names in the repository, do one of the following:

- Press Ctrl + Tab.
- From the **Edit** or shortcut (right-click) menu, choose **Choose** <*Apptype*> **Names**, where <*Apptype*> is the application type of the selected object or cell.

You can either invoke this feature from an unlabeled object or cell, or you can type a partial label and then invoke it. The partial label acts as a filter so that only matching names appear in the **Choose Names** dialog.

The **Choose Names** dialog box (Figure 6) displays a list of existing names for the selected application type. Optionally, to filter the list, type a string with an asterisk (*) wild card in the **Selection** field and click **Filter**. If no matching names are found for the selected object, the dialog does not appear and a message to that effect appears in the message log.

Figure 6: Choose Names Dialog



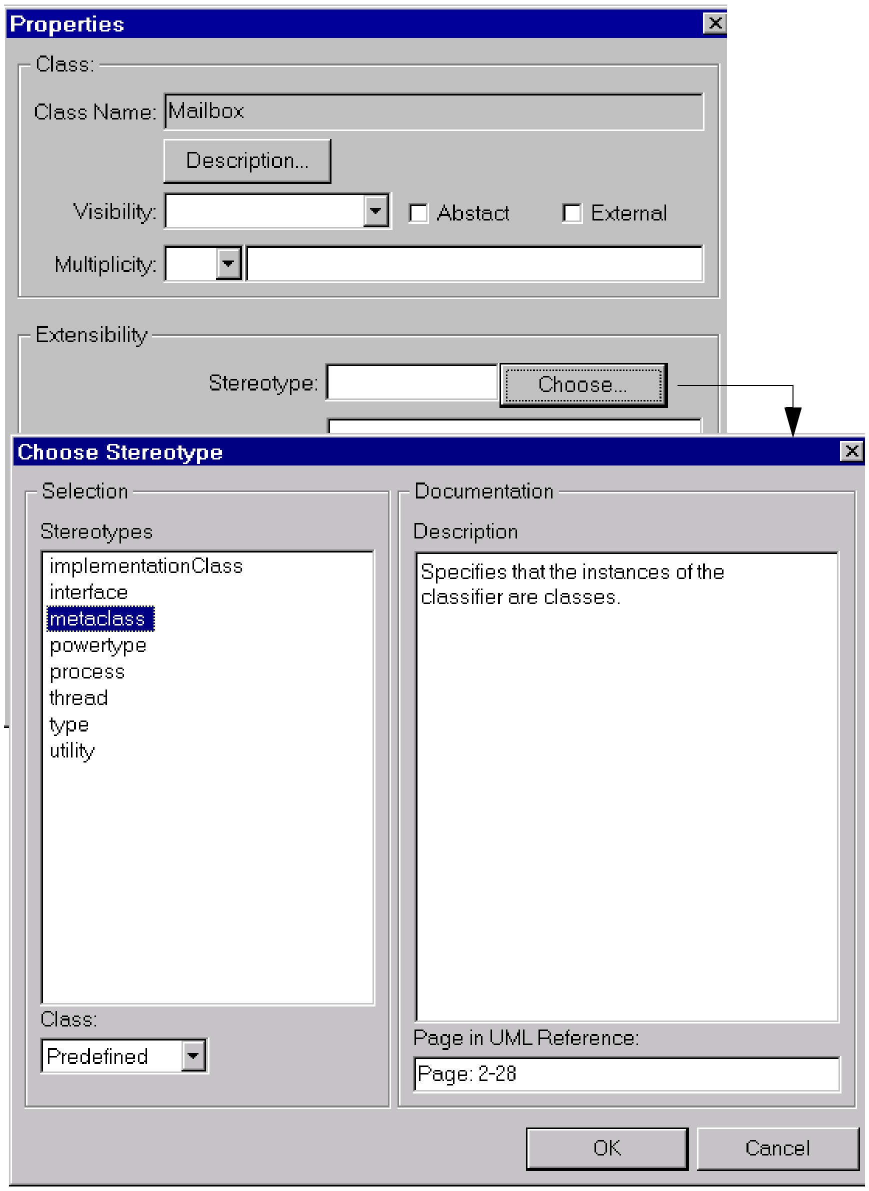
Note: For UML messages on sequence and collaboration diagrams, and for state transition links, this command displays the object's **Properties** sheet instead of the **Choose Names** dialog, to enable selection of additional attributes that are elements of the object's name. For more information, see "Labeling State Transitions with Events, Actions, and Guards" on page 6-19.

UML Stereotype Pick List

Stereotypes may be defined on a property sheet for many types of objects in StP / UML editors. Now you can select predefined or user-defined stereotypes directly from a pick list by clicking the **Choose** button next to the **Stereotype** field on the object's property sheet.

On the **Choose Stereotype** dialog (Figure 7), you can display a pick list of either predefined or user-defined stereotypes. The description for a selected stereotype appears in the **Description** field, and the appropriate page reference in the OMG UML 1.3 PDF file posted on their internet website. For more information on stereotypes, refer to *Creating UML Models*

Figure 7: Choosing StP/UML Stereotypes



Class Name: Mailbox

Visibility:

Abstact

External

Multiplicity:

Stereotype:

Stereotypes

implementationClass
interface
metaclass
powertype
process
thread
type
utility

Class: Predefined

Description

Specifies that the instances of the classifier are classes.

Page in UML Reference: Page: 2-28

StP/UML Global Rename with Pattern Matching

A new StP/UML desktop command, **Multiple Global Rename**, allows you to globally rename all packages, classes, operations, attributes, use cases, or actors in the repository whose names match a user-specified search pattern in the form:

```
[*]String[*]
```

All matching objects are renamed with a user-specified replacement pattern.

To use this command:

1. From the StP desktop **Tools** menu, select **Multiple Global Rename**.
2. In the dialog box that appears, do the following:
   - In the **Type** field, display the options list and select one of the object types to search for.
   - In the **Search Pattern** field, specify a search pattern in the form: `[*]String[*]`
   - In the **Rename Pattern** field, specify the replacement name pattern in the form: `[*]String[*]`
   - To preview the potential name changes without actually changing them, select **Check impact only**. This displays a list of existing names and the proposed name change for each.
3. Click **OK**.

Navigation Enhancements

You can now access the **GoTo** menu from either the menu bar or from the shortcut (right-click) menu for the diagram, table, or selected object or table cell.

In addtion, StP ME provides "global" navigation in various StP / UML editors:

- **Any Symbol With The Same Name** command—Navigates to every node with the same name—for example, an action state with the name of the selected use case, a class with the name of the selected actor, a package or class with the name of the selected component, and so forth.
- **Any Reference Of The Selected Symbol/Link** command—Navigates to any reference of the selected symbol—in other words, to the same node in all diagrams and tables. This in turn allows navigation to every occurrence in the entire model.

  Targets are all references of the same node in all diagrams and tables, which allows navigation to every occurrence in the entire model.
- *<Diagram/Table>* **Where Class is Referenced** command—Navigates from the selected symbol to its corresponding classes. *<Diagram/Table>* is:

  Class Diagram
  Class Table
  State Diagram
  Activity Diagram

  In addition, the StP ME class diagram editor allows navigation from classes to objects of the selected class symbol with the **Objects of the Selected Class** command.

Publishing and Printing Enhancements

Publishing and printing enhancements in StP ME include:

- "New StP Script Manager Toolbar" on page 5-20

- “Printing JPEG Diagrams” on page 5-22
- “Generating HTML Reports (UML)” on page 5-22
- “Using International Paper Size Settings” on page 5-23

New StP Script Manager Toolbar

A standard toolbar in the StP script manager (Figure 8), introduced in StP 8.0, provides faster access to frequently-used scripting tools and commands.

Figure 8: StP Script Manager Toolbar

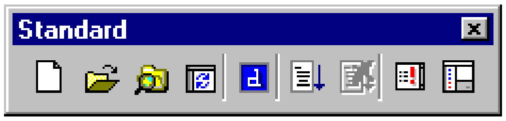


Table 5: Script Manager Toolbar Button Descriptions

Button	ToolTip	Description
🗋	**Create Script**	Creates a copy of a script template in a text editor window.
📂	**Edit Script**	Displays the selected script in a text editor window.
🔍	**Show Path**	Displays the selected script’s path in the message log.
🔄	**Rescan**	Updates the currently selected script in the **Scripts** list. Updates the **Scripts** list if outside scripts are copied into the scripts folder.
d	**Edit External Value**	Allows you to enter a value for the selected external variable in a text editor window.

Table 5: Script Manager Toolbar Button Descriptions (Continued)

Button	ToolTip	Description
	Run Script	Executes the selected script, using your specifications for output file, format file, and optional viewing of the output file when done.
	Terminate Script	Stops a script that is currently executing.
	Show/Hide Message Log	Displays or hides the message log.
	Show Desktop	Displays the StP desktop.

Printing Diagrams

StP has enhanced its print capability to include the printing of JPEG diagrams and HTML report diagrams.

To print JPEG diagrams and HTML report diagrams, you need to have a PostScript conversion tool installed and have the `postscript_conversion` ToolInfo variable defined in your *ToolInfo.<platform>* file. If you do not have a PostScript conversion tool, one possible utility can be obtained at *http://www.cs.wisc.edu/~ghost/doc/AFPL/*.

Once you've installed the tool, uncomment the `postscript_conversion` line in your ToolInfo file. For example, if you've installed GhostScript 7.00, the definition might be something like this:

```
postscript_conversion=<install_path>\gs7.00\bin\gswin32c.exe -q -dBATCH -
dNOPAUSE -dHIDE -sDEVICE=jpeg -r70x70 -Ic:\gs\fonts -g<width>x<height> -
sOutputFile=<OutputFile> <PSFile>
```

Substitute <*install_path*> with the actual path. Do not change other values, because StP generates these values at run time. UNIX users should change the backward slashes to forward slashes.

Note: The definition must be entered as a single line; returns are not allowed.

Note: You can use **Tools > Edit ToolInfo Variables** from the desktop to set `postscript_conversion` (or any other ToolInfo variable) without invoking an external editor. See “Editing ToolInfo Variables from the Desktop” on page 5-6 for details.

Printing JPEG Diagrams

To print a diagram to a file in JPEG format, use **File > Print Diagram As** from the StP desktop or **File > Print As** from an StP diagram editor. (Remember to set `postscript_conversion` in your ToolInfo file first.) Diagrams saved to a file in JPEG format can be opened directly in an email browser, making it easy to share them with other developers.

Note: Currently, JPEG output for StP diagrams is limited to a single page per diagram. Page splitting is not supported for this format at this time. However, you can adjust the overall size of the image by using the scaling options from an StP editor by using **File > Page Setup**.

Generating HTML Reports (UML)

UML report generation was enhanced in StP 8.2 by providing a report that allows you to document a UML system in a set of linked HTML pages. The structure of these pages is similar to those in the JavaDoc standard.

The report consists of a master index that links to subindex pages such as Actor, Use Case, Package, Class Index, as well as to an index that lists all diagrams in the system. Each diagram listed links back to its graphical (JPEG) representation.

High level UML elements are documented textually and are cross-linked with their references in diagrams. The report also provides hyperlinks from symbols in diagrams back to their textual description.

To generate an HTML report for your model, open the script manager from the desktop with **Report > Start Script Manager**, select **HTML_Report** from **Product Scripts > report**, then run the script with **File > Run Script**.

To view the resulting HTML report, open the main index page, *Index.html*, from the StP desktop command **Report > Open HTML Report**. All generated files are placed in *<projdir>/<system>/html_files/*.

Using International Paper Size Settings

To use international (non-U.S.) paper formats and measurements, uncomment the following line in your ToolInfo file:

```
use_us_papersizes=false
```

This changes StP's default paper size print setting from **US Letter** to **A4**, and displays measurements in centimeters (cm) instead of inches (in).

6 StP/UML Editor Enhancements

This chapter discusses the modeling and usability enhancements that relate to specific StP diagram and table editors, namely:

- “Use Case Editor” on page 6-2
- “Class Diagram Editor” on page 6-4
- “Class Table Editor” on page 6-10
- “Sequence and Collaboration Editors” on page 6-16
- “State and Activity Editors” on page 6-18
- “Component and Deployment Editors” on page 6-24
- “Object Annotation Editor” on page 6-26

Usability enhancements that apply to all or most editors rather than specific editors are discussed in Chapter 5, “Usability and Modeling Enhancements.”

Note: One of the “generic” enhancements is that property sheets now exist for most symbols in the UML editors. The standard property sheet for object symbols is shown in Figure 3 on page 5-5.

Use Case Editor

StP ME provides support for the following features in the StP / UML use case editor; they were introduced in StP 8.0.

- "Drawing Generalizations Between Use Cases"
- "Adding Extension Points to Use Cases"

In addition, reflexive links for use cases were disabled in StP 8.2, so that users cannot link a use case to itself.

Drawing Generalizations Between Use Cases

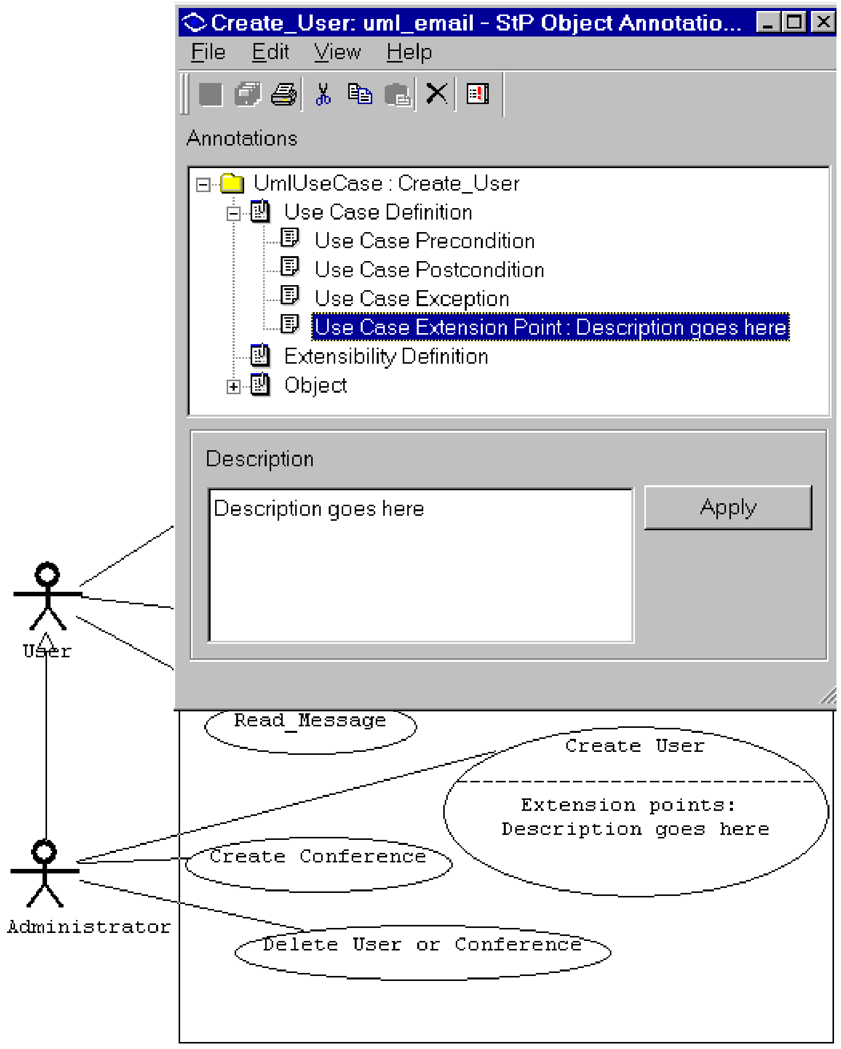
In StP ME, you may draw generalizations between use cases to show a taxonomic relationship between the more general use case and additional use cases representing more specific design details. Select the generalization arc on the arc toolbar to draw generalizations between use cases.

Adding Extension Points to Use Cases

StP ME supports extension points for use cases, in the form of a new annotation item, Use Case Extension Point, for the Use Case Definition note. The item description contains the location of the extension point, entered as ordinary text or in other forms, such as the name of a state in a state machine, or a precondition or postcondition.

Use the object annotation editor to add one or more extension point(s) to the use case, as shown in Figure 1.

Figure 1: Adding Use Case Extension Points



Class Diagram Editor

StP ME provides support for the following new features in the StP / UML class diagram editor:

- “Drawing Aggregations” (immediately below)
- “Enhanced Drawing Generalizations” on page 6-4
- “Auto Drawing Classes” on page 6-6
- “Editing Role Names and Choosing Multiplicity” on page 6-8

The set of new features in the class diagram editor also includes property sheets for object instances, attributes, and operations. These are addressed in “Property Sheets” on page 5-5.

Drawing Aggregations

Now you can draw aggregations easily and quickly by directly selecting the **Aggregation** arc type tool on the default arc type toolbar (see “Arc Drawing Enhancements” on page 5-10).

Choose either **aggregation** or **composition** on the aggregation’s property sheet (**Edit > Properties > Aggregation Type**) or use the **Properties** toolbar button).

Enhanced Drawing Generalizations

StP ME provides the following additional capabilities with respect to drawing generalizations in class diagrams:

- Quick creation of tree-style generalization hierarchies
- Support for ellipses (suppressed classifiers) in generalizations, including navigation to suppressed classes
- Ability to add discriminators to generalization links
- Shared properties for generalizations, through use of a shared **Properties** sheet with multiple generalizations selected

Creating a Tree-Style Generalization Hierarchy

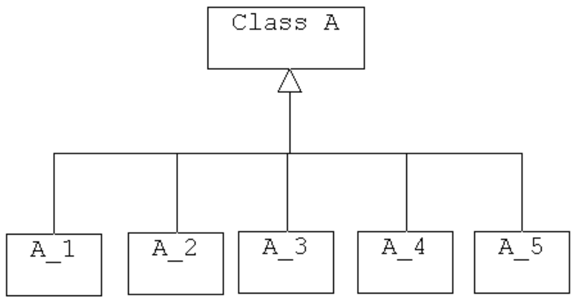
StP ME simplifies the drawing of tree-style generalizations, in which a group of generalization paths for a given superclass are drawn as separate links that combine at a vertex into a single, shared link with one generalization symbol (hollow triangle) connected to the superclass.

To create a tree-style generalization hierarchy:

1. Insert and label a superclass and any number of subclasses.
2. Select the subclasses to participate in the tree-style hierarchy.
3. From the **UML > Generalization Hierarchy** or shortcut (right-click) menu, choose **Draw Generalization Hierarchy**.
4. Move the pointer into the drawing area.

   The tree structure appears, joining the subclasses at a vertex to a single link that has one end attached to the pointer.
5. Drag the end of the link attached to the pointer into the superclass and click the mouse to attach it.

Figure 2: Drawing a Tree-Style Generalization



Adding Classes to a Tree-Style Generalization Hierarchy

To add subclasses to one of the classes in the hierarchy, follow the same procedure you used to create the first hierarchical level of the tree. Use the **Draw Generalization Hierarchy** command to connect the lower-level subclasses to the parent class.

To add sibling classes to a generalization hierarchy, draw individual orthogonal arcs from each new sibling subclass to a vertex of the hierarchy at the same hierarchical level as its siblings.

Showing Suppressed Classes in Generalizations

You can use the ellipses symbol in generalizations to represent the existence of suppressed classes—additional subclasses in the model not shown in the current diagram. The suppressed classes, which must already exist in other parts of the model, are associated by links to a reference of the parent class in the parent generalization structure. To navigate to suppressed classes, select the ellipsis and choose **Suppressed Classes** from the **Go To** or shortcut (right-click) **Go To** menu.

Adding Discriminators to Generalization Links

You may enter a discriminator as a label on a generalization link. In a hierarchy, each link may be labeled individually.

Sharing Properties in a Generalization Hierarchy

To specify properties that are shared by any number of generalizations in a generalization hierarchy:

1. Select multiple generalization links, using the Shift key.
2. Choose **Properties** from the toolbar or from the **Edit** or shortcut (right-click) menu. The property sheet appears.
3. Specify the shared properties and click **OK** when done.

   The properties are applied to all selected generalization links.

Auto Drawing Classes

This enhancement adds auto drawing capabilities and a new **Auto Draw** submenu to the **UML** menu in the class diagram editor. The commands in the **Auto Draw** menu automatically draw the following for any selected class in the diagram, based on the contents of the repository:

- Superclasses (outgoing generalization link)
- Subclasses (incoming generalization link)

- Dependent classes (outgoing dependency link)
- Classes on which it depends (incoming dependency link)
- Associated classes (with roles, labels, aggregations, and so on)

Note: Auto drawing does not check to see whether anything it draws already exists in the diagram.

Auto Drawing a Single Association or Aggregation

To auto draw a single association or aggregation between any two classes:

1. Select the classes.
2. From the **UML > Auto Draw** menu, choose **Draw Associations/Aggregations**.
3. Select an association or aggregation from the **Object Selector** dialog box that appears and click **OK**.

Setting the Generalization Auto Drawing Style

You can specify whether you want StP to auto draw generalizations with direct (default) links or in tree-style hierarchies. To do so, add the *AutoDrawGenTree* ToolInfo variable to your ToolInfo file and set it as follows:

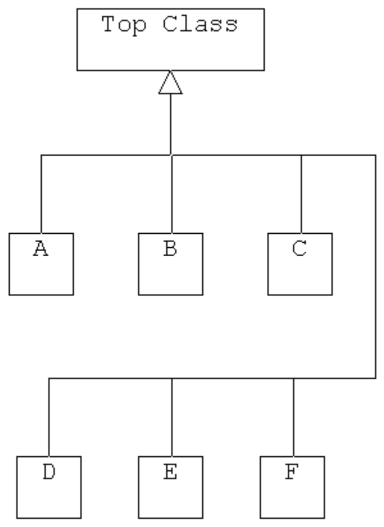
- Not set—Draws direct links (default behavior)
- Set to any value—Draws the generalizations in tree-style hierarchies

Limiting the Number of Classes Drawn Side-by-Side

User-defined limits restrict the number of classes that are automatically drawn side-by-side by the **Auto Draw** commands. Classes exceeding the set limit are drawn below the first "row" of classes, as shown in Figure 3.

To set this limit, set the value of the *AutoDrawPerGroup* ToolInfo variable in your ToolInfo file to an integer representing the number of classes per row.

Figure 3: Auto Drawn Generalization Tree with Group Limit Set to 3



Editing Role Names and Choosing Multiplicity

Modifications and new capabilities on an association **Properties** sheet allow you to:

- Edit role names.
- Choose predefined multiplicities from a pick list.
- Represent bidirectional navigability of an association as the default behavior, with or without arrowheads at each end.

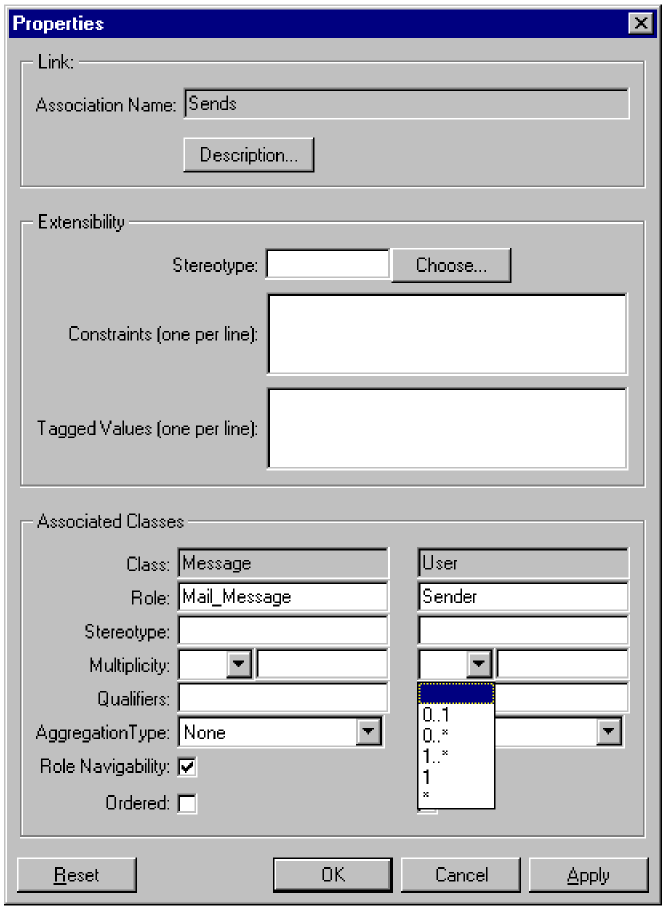
Editing Role Names and Choosing Multiplicities

To edit role names or choose multiplicities:

1. On the class diagram, select an association.
2. Display the association’s property sheet, using the **Properties** toolbar button or the **Properties** command on the **Edit** or shortcut menu.

3. In the **Associated Classes** group, enter or edit role names in the **Role** fields, as desired.
4. Display the options list in a **Multiplicity** field (Figure 4) for an associated class and either:
   - Select a predefined multiplicity from the list.
   - Manually type any allowable entry in the adjacent text entry field on the property sheet.
5. Click **OK**.

Figure 4: Association Property Sheet



Representing Role Navigability

By default, associations are now interpreted as being bidirectional, unless otherwise set by the user. Navigability of an association role is indicated on the association's property sheet by the **Role Navigability** option. The association is bidirectional if the **Role Navigability** option is either set or unset at both ends of the association. Initially, the property sheet shows the **Role Navigability** option set at both ends of the association.

To indicate unidirectional navigability, clear the **Role Navigability** option for one of the association roles on the property sheet. An arrow appears on the set end of the association in the diagram.

To reestablish bidirectional navigability, set or unset role navigability on both ends of the association.

By default, only unidirectional associations have an arrowhead. To show bidirectional arrows heads, add the following to your ToolInfo file:

```
uclassd_show_navigability_on_both_ends=True
```

Class Table Editor

StP ME provides support for the following features in the StP / UML class table editor:

- "Adding New Attributes and Operations" (immediately below)
- "Choosing Class Attribute and Operation Types" on page 6-11
- "Choosing an Operation's Arguments" on page 6-13
- "Adding Methods of an Interface" on page 6-15
- "Sorting Attributes and Operations" on page 6-15

Adding New Attributes and Operations

To quickly create a new blank line in the appropriate table section for adding new attributes or operations to a class table, use the new toolbar buttons shown in the table below.

Table 6: Add New Attribute/Operation Toolbar Buttons

Button	ToolTip	Description
	Add New Attribute	Adds a new empty row below the selected attribute, or if no attribute is selected, adds the row after the last attribute row in the table.
	Add New Operation	Adds a new empty row below the selected operation, or if no operation is selected, adds the row after the last operation row in the table.

The **Table** menu also contains commands for inserting blank rows before or after a selected row.

Deleting Rows in a Class Table

To quickly delete a selected row in a class table, use this new toolbar button:

Table 7: Delete Selected Row Toolbar Button

Button	ToolTip	Description
	Delete Selected Row	Shortcut to **Table > Delete Cells** command, but is only active when a complete row is selected

Alternatively, use the **Delete Cells** command on the **Table** menu.

Choosing Class Attribute and Operation Types

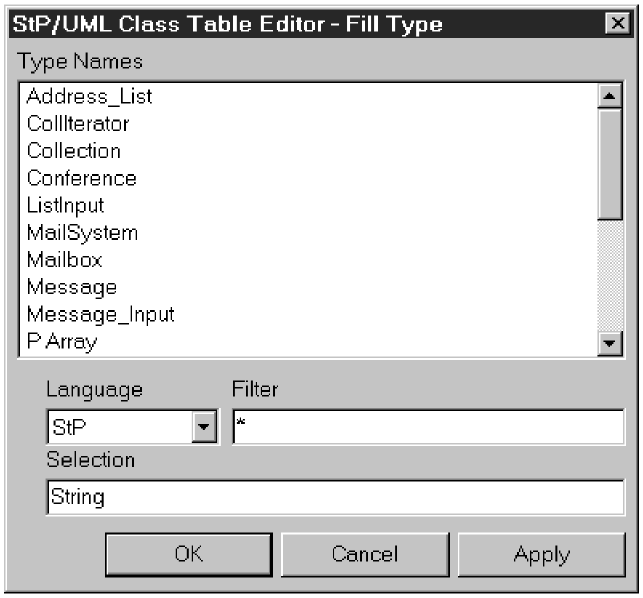
To choose class attribute types and operation return types:

1. In the class table, select an attribute Type cell or operation Return Type cell.

2. From the **Edit** or shortcut (right-click) menu, choose **Fill Type**.

   A dialog box appears, listing types from which you can select.

Figure 5: Fill Type Dialog Box



StP/UML Class Table Editor - Fill Type

Type Names

Address_List
CollIterator
Collection
Conference
ListInput
MailSystem
Mailbox
Message
Message_Input
P Array

Language: StP

Filter: *

Selection: String

3. In the **Language** field, select one of the following:
   - **StP**—to display a pick list of all classes, attributes, and return types of operations used in StP, stripped to their base types
   - One of the standard language types listed

   Use the dialog's **Filter** field for additional filtering of the displayed types.
4. Select a type from the **Type Names** list and click **OK**.

Standard language type files are supplied with StP in files named *<language_ID>_std_types*, which are located in *templates/uml/qrl/code_gen*, where *<language_ID>* is:

cxx	C standard types
java:	Java standard types
idl:	IDL standard types
ada95:	Ada 95 standard types

Note: The set of Ada 95 language type files includes files that define types specified in the three root packages required in each Ada implementation: System, Ada, and Interfaces. For example, *ada95-system_std_type*s includes types defined in package System.

You can customize StP to use a language type file from a different location by defining a ToolInfo variable (having the same name as one of the supplied language type files) with the absolute path to a type file located elsewhere. For example:

```
java_std_types=c:\common\stp_java_std_types
```

Choosing an Operation’s Arguments

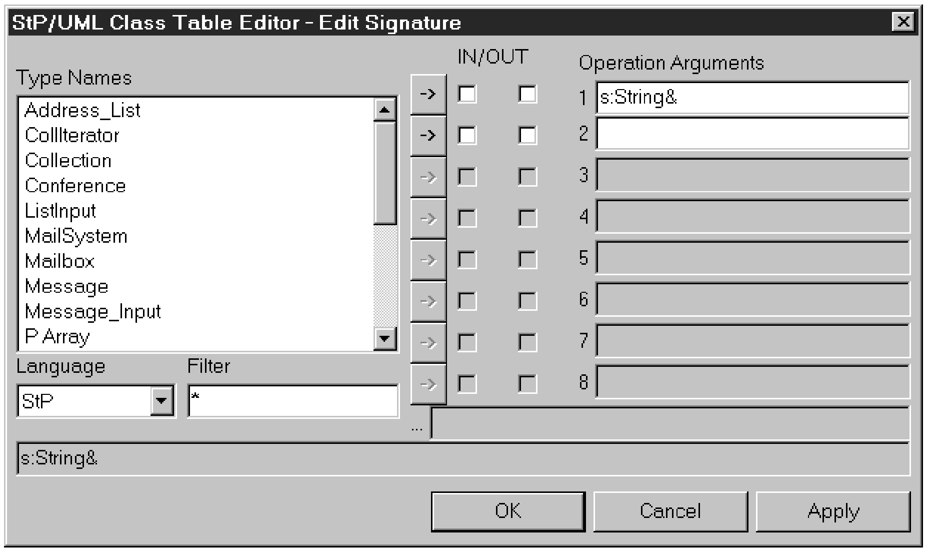
To choose an operation’s arguments:

1. In the class table, select an operation’s argument cell.
2. From the **Edit** or shortcut (right-click) menu, choose **Edit Signature**.

   A dialog box appears, listing the possible types on the left and the first eight arguments for this operation on the right. Any additional arguments are entered in the text field at the bottom of the right column. This field becomes writable after arguments 1 - 8 have been specified.

   The read-only field at the bottom of the dialog displays the complete argument list for the method (see Figure 6).

Figure 6: Edit Signature Dialog Box



3. In the **Language** field, select one of the following:
   - **StP**—to display a pick list of base types used in StP for all classes, attributes, and return types of operations
   - One of the standard language types listed

   Use the dialog's **Filter** field for additional filtering of the displayed types.
4. Select a type from the **Type Names** list and click the arrow button to change the type in the adjacent **Operation Arguments** field.
5. Specify the direction of the data flow for each argument in the non-exclusive **In/Out** check boxes to the left of each argument.
6. Click **OK** when done.

Standard language type files are supplied with StP in files named *<language_ID>_std_types*, which are located in *templates/uml/qrl/code_gen*, where *<language_ID>* is:

cxx	C standard types
java:	Java standard types
idl:	IDL standard types
ada95:	Ada 95 standard types

Note: The set of Ada 95 language type files includes files that define types specified in the three root packages required in each Ada implementation: System, Ada, and Interfaces. For example, *ada95-system_std_type*s includes types defined in package System.

You can customize StP to use a language type file from a different location by defining a ToolInfo variable (having the same name as one of the supplied language type files) with the absolute path to a type file located elsewhere. For example:

```
java_std_types=c:\common\stp_java_std_types
```

Adding Methods of an Interface

A class that supports an interface implements all operations of that interface. StP ME adds a new command, **Implement Interface Operations**, to the following menus in the class table editor:

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

The **Implement Interface Operations** command adds all the operations of the interfaces to the class table of the supporting class.

Sorting Attributes and Operations

You can now change the order in which rows of attributes or operations display, using the following sort criteria:

- Attributes By Name
- Attributes By Type, Name
- Attributes By Current (selected) Column, Name
- Operations By Constructors, Name, Arguments
- Operations By Name, Arguments
- Operations By Current Column, Name, Arguments

To sort the rows:

1. Select row(s) to be sorted by selecting their row numbers in the first column.

   Note: You can extend a selection by holding down the Shift key while selecting the last row in the set to be sorted.

2. Choose a command from the **Sort** menu.

   The table rows reappear in the specified sort order.

Navigating to a Class's Superclass or Subclass

Two new commands on the **Go To** menu in the class table editor allow you to navigate easily between tables for subclasses and superclasses of the current class:

- **Class's Immediate Super Class(es)**
- **Class's Immediate Sub Class(es)**

If multiple potential targets exist, you can select one from an **Object Selector** dialog.

Sequence and Collaboration Editors

StP ME provides support for choosing and setting message attributes.:

New **Choose** buttons on the property sheet for message links in sequence and collaboration diagrams (Figure 7) allow you to:

- Choose a predecessor guard, based on public attributes and operations of the message's target class for non-reflexive messages, and all attributes and operations for reflexive messages.
- Choose a name component, return type, and message arguments, based on the public operations of the target class for non-reflexive messages, and all operations for reflexive messages.

Additional options (**Creation** and **Destruction**) on the property sheet allow you to set message creation and destruction.

To choose message attributes on sequence and collaboration diagrams:

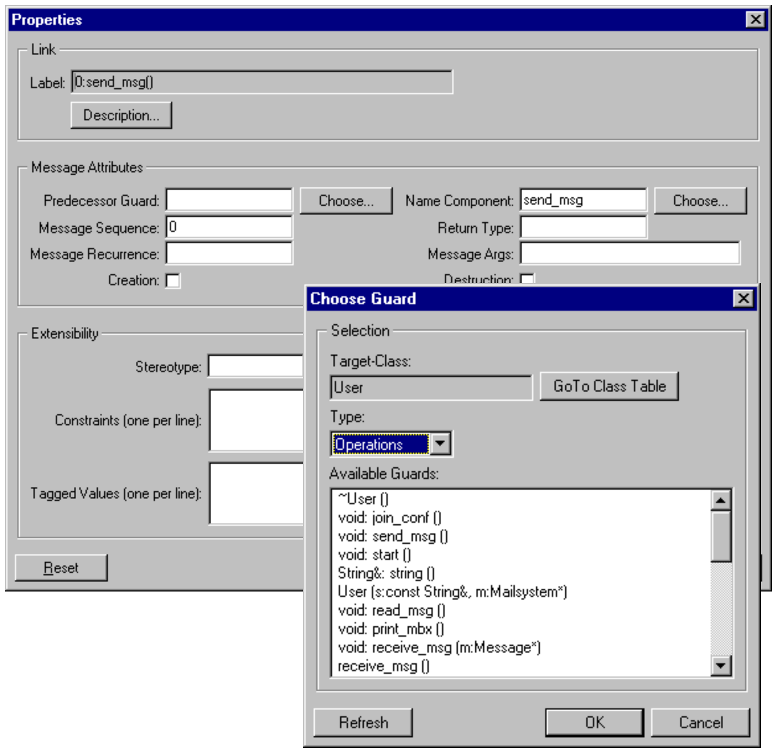
1. On the diagram, select a message link.
2. Use one of these methods to display the link's property sheet:
   - **Properties** toolbar button
   - **Properties** command on the **Edit** or shortcut (right-click) menu
   - **Set Message Name** on the **UML** menu
3. On the property sheet, click the **Choose** button next to the **Predecessor Guard** or **Name Component** field.

   A choose dialog lists appropriate selections defined within StP (see Figure 7).
4. For guard conditions, in the **Type** field on the choose dialog, select **Attributes**, **Operations**, or **States**.
5. If necessary, add potentially missing attributes and operations to the class table: Click **Go To Class Table**; if the class table does not exist, it is created.

   To update the chooser list with changes made to the class table, click **Refresh**.
6. Make your selection from the **Available** list on the choose dialog and click **OK**.

   Your selection appears in the appropriate field(s) in the message's property sheet.
7. Set the **Creation** and **Destruction** options as desired.
8. Click **OK** to save the properties.

Figure 7: Choosing Message Attributes



Label: 0:send_msg()

Predecessor Guard:
Message Sequence: 0
Message Recurrence:
Creation: ☐

Name Component: send_msg
Return Type:
Message Args:
Destruction: ☐

Target-Class: User

Type: Operations

Available Guards:

```
~User ()
void: join_conf ()
void: send_msg ()
void: start ()
String&: string ()
User (s:const String&, m:Mailsystem*)
void: read_msg ()
void: print_mbx ()
void: receive_msg (m:Message*)
receive_msg ()
```

State and Activity Editors

StP release 8.0 introduced support for the following features in the StP / UML state and activity editors.

- “Labeling State Transitions with Events, Actions, and Guards” (immediately below)
- “Refining Action States in Activity Diagrams” on page 6-22
- “Drawing Swim Lanes in Activity Diagrams” on page 6-22

In addition, starting with StP release 8.2, stereotypes, constraints, and tagged values are displayed for action states. Stereotypes are updated continuously, the other two on demand.

Labeling State Transitions with Events, Actions, and Guards

Typically, the event / action label on a state transition link contains:

- An event that maps to a corresponding event sent to that class, as defined in a sequence diagram
- An action that maps to an operation supplied by the class (where the class maps to the state machine for the given state)
- Optionally, a guard condition that maps to an attribute of the calls, which needs to have a defined value in order for the state transition to occur

In StP ME, you can select all of these from names that already exist in the repository, as summarized in Table 8.

Table 8: State Transition Link Event/Action Labels

Diagram	State Transition Label Attributes	Choose From
State	Event (incoming)	Events defined in sequence diagrams for the state machine's class
	Send Event	
Activity, State	Guard conditions	Public attributes or operations of the state machine's class, and superclasses or states used in the scope of the current state machine
	Action List	Operations of the state machine's class and superclasses

To choose events, actions, and guard conditions for state transition link labels in activity or state diagrams:

1. On the diagram, select a state transition link.

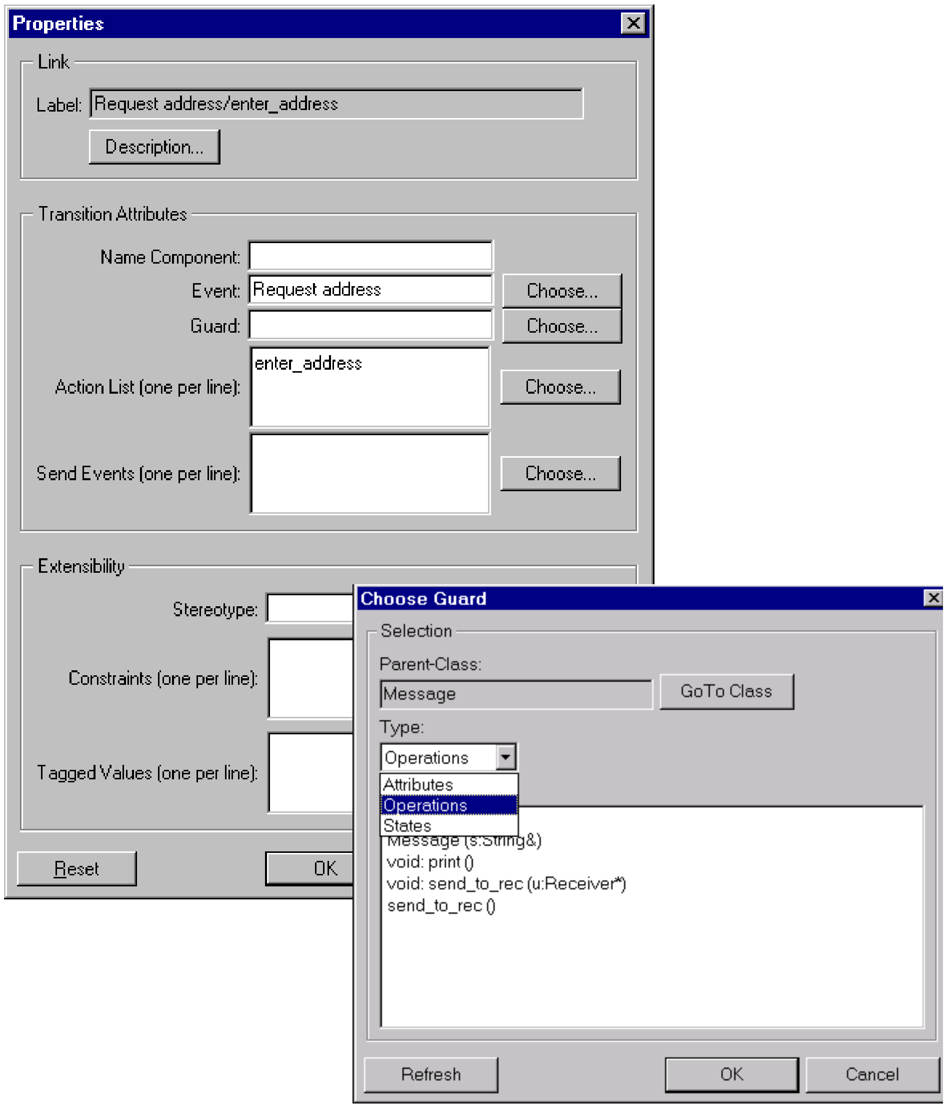
2. Display the link’s property sheet, using the **Properties** toolbar button or **Properties** on the **Edit** or shortcut (right-click) menu.
3. On the property sheet, click the **Choose** button next to a field for which you want to make selections.

   A choose dialog lists appropriate selections defined within StP (see Figure 8).
4. For guard conditions, in the **Type** field on the **Choose** dialog, select **Attributes**, **Operations**, or **States**.
5. To add potentially missing attributes and operations to the class table, click **Go To Class Table**; if the class table does not exist, it is created.

   To update the chooser list with changes made to the class table, click **Refresh**.
6. Select from the **Available** <*item*> list on the **Choose** dialog and click **OK**.

   The selections appear in the state transition link’s property sheet.

Figure 8: Choosing Events, Actions, and Guard Conditions for State Transition Links



Label: Request address/enter_address

Name Component:

Event: Request address

Guard:

Action List (one per line): enter_address

Send Events (one per line):

Stereotype:

Constraints (one per line):

Tagged Values (one per line):

Parent-Class: Message

Type: Operations

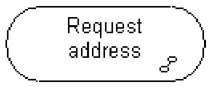
```
Message (s:String&)
void: print ()
void: send_to_rec (u:Receiver*)
send_to_rec ()
```

Refining Action States in Activity Diagrams

This enhancement provides a mechanism for refining action states in activity diagrams. To create the decomposition diagram for an action state, and to navigate between the parent and the decomposition, use these commands on the **Go To** menu:

- **Diagram with Refined Action State**—If the diagram does not exist, creates a decomposition diagram containing a state machine, both of which bear the same name as the parent action state; otherwise, navigates to the existing decomposition diagram.
- **Parent Action State**—Navigates from the refined activity diagram to the parent action state.

A subactivity display mark, ActivityIsRefined, appears on the parent action state symbol (see below) in the parent diagram, indicating that it has been refined in a decomposition diagram.



State machines linked to a parent state (or parent use case, class, and so forth), exhibit a StateMachineContext display mark, for example:

```
Context: refined Action State
```

Note: Globally renaming an action state also renames a linked state machine. However, globally renaming a state machine has no effect on a parent action state.

Drawing Swim Lanes in Activity Diagrams

StP ME provides support for “swim lanes” in activity diagrams. Swim lanes are represented by solid vertical lines between sections of your diagram. They organize responsibility for the actions and subactivities according to class, and often correspond to organizational units in a business model.

Each action state is assigned to one swim lane, while transitions can cross lanes. The relative ordering of the swim lanes may have organizational significance, but does not affect the semantics of the diagram.

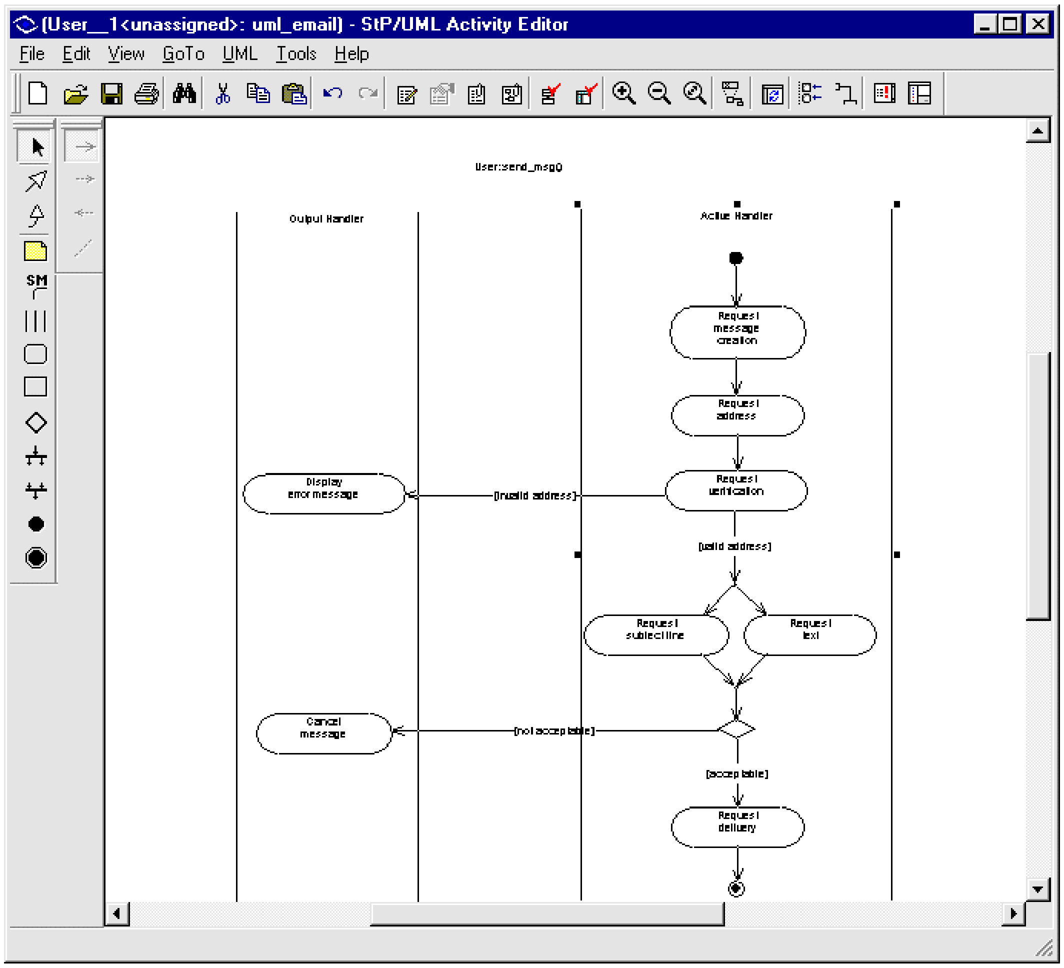
To draw swim lanes in an activity diagram:

1. Insert swim lane symbols into an empty diagram and label them.
2. Drag and drop new or existing action states into a swim lane; label and link the action states as desired.

   The action states are automatically associated with the swim lane that contains them.

   Note: The action state must fit entirely within the boundaries of a swim lane to be associated with it.
3. To lengthen or widen a swim lane, select it and then drag one of its “handles” to the desired position.

Figure 9: Swim Lanes in Activity Diagrams



Component and Deployment Editors

You can now associate a class with a component in component and deployment diagrams to indicate implementation of the class by that component. Classes implemented by a component appear nested within that component. Support for the containment of classes in a component is analogous to the support for the containment of objects within a component; that is, the nesting indicates implementation, not ownership.

Representing Class Implementation by a Component

To indicate that a class is implemented by a component in a component or deployment diagram:

1. Insert a class symbol from the Symbols toolbar into the component.
2. Label the class symbol with the name of the class that is being implemented, by either:
    - Typing the label manually
    - Using the **Edit > Choose Class Name** command

    StP associates the class with the component by generating a UMLContains link between them.

You can edit the class properties on the class's property sheet, which is accessible from the component and deployment diagrams using the **Properties** toolbar button or **Properties** command on the **Edit** or shortcut (right-click) menu.

Navigating to/from Classes in Components

StP provides the same navigations to and from a class in a component or deployment diagram as for a class in a class diagram. In addition, StP provides bi-directional navigation between a class in a component or deployment diagram and the class diagram or table.

Object Annotation Editor

Improvements to the object annotation editor (OAE) extend its capabilities and simplify the process of adding annotations to objects, diagrams, tables, and table cells. These enhancements include:

- "Adding Notes and Items to an Annotation" (immediately below)
- "Linking External Files to StP Objects" on page 6-28
- "Annotating StP / UML Links without Labels" on page 6-29

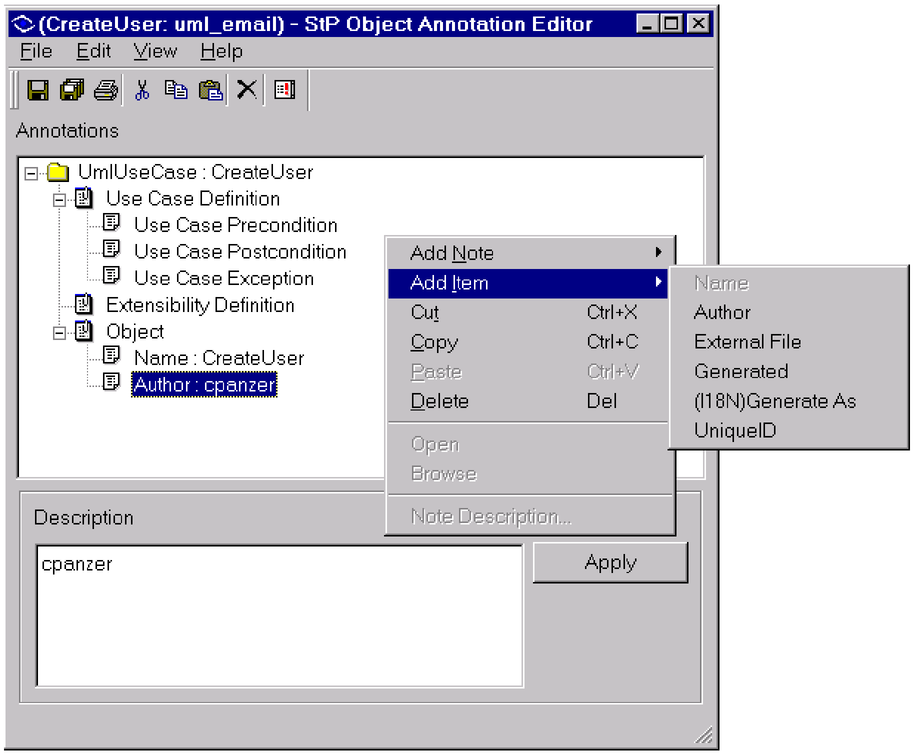
Adding Notes and Items to an Annotation

To add a note and / or item to an annotation in the OAE:

1. Select a note or sibling item in the **Annotations** list.
2. Display a menu of applicable notes or items by right-clicking the mouse or choosing **Add Note** or **Add Item** from the **Edit** menu.
3. Select a note or item from the displayed menu, as shown in Figure 10.

   The note or item appears selected in the **Annotations** list.
4. Enter a description or value for this note or item in the **Description** field, as needed, and click **Apply** to assign the description to the annotation.
5. Save the annotation(s) in one of the following ways:
   - **Save Annotation** or **Save All Annotations** toolbar buttons
   - **File > Save** or **File > Save All** command

Figure 10: Adding Notes and Items in the OAE



Editing Description Text

New commands (**Cut Text**, **Copy Text**, and **Paste Text**) for editing the text in the **Description** field can be found on the **Edit** menu. Note that you can also annotate items from other editors without invoking the AOE. See "Quickly Entering Descriptions" on page 5-8.

Note: **Cut**, **Copy**, and **Paste** toolbar buttons apply only to cutting, copying and pasting entire notes and items, but not text.

Displaying One or Multiple Annotations in the OAE

A new option, **Edit Single Annotation**, on the **General** tab of the OAE **Options** dialog allows you to determine whether the OAE opens one or more annotations at a time:

- Selected—Limits the OAE to displaying only one annotation at a time
- Unselected—Allows you to open additional annotations in the OAE, without saving or clearing the existing annotations

To access the OAE **Options** dialog, choose **Edit > Options**.

Using Microsoft Word as the OAE Description Editor

As described in *Fundamentals of StP*, you can specify an external text editor to be used for editing note descriptions in the OAE **Description** field. If you are using Microsoft Word as the note description editor, do the following:

- Make sure that the path to Microsoft Word is set in your Windows NT environment.
- On the **Annotations** tab of the **Edit > Options** dialog, append *.rtf* as the file extension to be used, as in the following example:

  ```
  winword.exe ${tmpfile}.rtf
  ```

Linking External Files to StP Objects

With StP ME, you can now link external documents, such as MS Word files including object descriptions, screen shots of a RAD prototype GUI, and so forth, to an StP element, such as a use case or class. To link an external file to an object, add an External File item to the Object annotation note in the object annotation editor (OAE).

To define the external file, enter its full path in the External File item's **Description** field. To select the path and file from a browser, choose **Browse** from the shortcut (right-click) menu.

To open the external file, choose **Open** from the shortcut (right-click) menu. Whenever possible, StP automatically opens the file in the correct application, based on the file extension.

Annotating StP/UML Links without Labels

To refer to StP / UML links that have no labels, the OAE now identifies them by the objects they link, as in the following example:

```
UMLRole: from UMLClass A to UMLAssociation A, H
UMLRole: from UMLClass H to UMLAssociation A, H
```

7 Exporting and Importing UML Models and Code

This chapter describes features for exporting, importing, and navigating between StP models and code, including:

- “Generating UML Solutions with Architecture Component Development” on page 7-3
- “Importing and Exporting UML Models with XMI” on page 7-3
- “Integrating SNiFF+ or Visual Studio with StP / UML” on page 7-5
- “Reverse Engineering StP Models” on page 7-8

Table 1 summarizes language support in StP ME for code generation, reverse engineering, and navigation to programming environments.

Table 1: StP/UML Programming Language Support

Language	Code Generation by ACD Template[a]	Code Generation with QRL[a]	Reverse Engineering	Navigation to/from Programming Environment
Ada	X	X	X	
C++	X	X	X	To / from SNiFF+ (Wind River Systems, Inc.), To / from Microsoft Visual Studio
Java	X	X	X	To / from SNiFF+ (Wind River Systems, Inc.), To / from Microsoft Visual Studio
EJB J2EE WebLogic	 X X			

a. Code generation by ACD template is the default (and preferred) method. To enable QRL code generation, you must add `qrl_code_gen_enabled=true` to your ToolInfo file.

Note: Code generation for C, CORBA, Visual Basic, TOOL, and COM is available but not supported. You can turn on code generation for these languages by setting the appropriate "code_gen" variables in your ToolInfo file to True. Please consult your Aonix representative for updated templates, example models, and documentation.

Generating UML Solutions with Architecture Component Development

Architecture Component Development (ACD) is a feature of StP that provides a more powerful and flexible approach for automatically generating solutions from your StP/UML models. Using ACD, a majority of the implementation code for your application can be generated automatically from your model. ACD implements a template-based approach that allows you to specify the architectural aspects of the system under construction.

ACD is designed to let you model your system from a design perspective, rather than focusing on implementation details that can be handled by the templates.

There are two documents that address ACD in detail: *ACD Programming Guide* and *Using ACD Templates*. Refer to those documents for more information.

Importing and Exporting UML Models with XMI

The XML Metadata Interchange (XMI) standard adopted by Object Management Group (OMG) consortium allows the easy interchange of metadata between various UML modeling tools and applications in a heterogeneous environment while utilizing a single repository.

StP uses ACD to generate, export, and import XMI.

The StP/UML implementation of XMI supports: OMG XMI 1.1. document type definitions (DTDs) - packages, classes, attributes, operations, associations, generalizations.

Using XMI Import and Export Features

Using StP's XMI import and export features you can:

- Import XMI (from StP or a third-party tool).

- Create a UML model for it.
- Export XMI based on the StP / UML model.

The XMI import and export features are available on the StP desktop from the **Code** menu or from the shortcut (right-click) menu for packages.

Importing an XMI Model

To import XMI for a UML model:

1. From the desktop **Code** menu, choose **XMI > Import XMI Model**.
2. In the dialog box that appears, enter or browse and choose the **Input File** and **Target System Directory**.
3. Click **OK**.

Note: When you import XMI, the result is a “road map” diagram. This occurs because a standard for layout information is yet to be defined for XMI. From the StP class editor, use **UML > Auto Draw** to restructure the resulting diagram.

Exporting an Entire Model

To export an entire StP / UML model:

1. From the desktop **Code** menu, choose **XMI > Export XMI Model**.
2. In the dialog box that appears, enter or browse and choose the **Output File**.
3. Click **OK**.

Exporting Selected Packages

To export XMI for one or more packages:

1. From the desktop **Repository View**, display and select package(s) in the objects pane.
2. Right-click the mouse on a package, or on a selected group of packages, in the objects pane.
3. From the shortcut menu, choose **Export XMI Model for Package**:
4. In the dialog box that appears, do the following:

- Enter or browse and choose the **Output File**.
- Select **Generate IMF file** if desired.

5. Click **OK**.

Integrating SNiFF+ or Visual Studio with StP/UML

StP ME offers navigational support between StP / UML and these programming environments:

- Bi-directional navigation between StP and SNiFF+ (Wind River Systems, Inc.)
- Navigation from Microsoft Visual Studio (Developer's Studio) to StP

Table 2: Programming Environment Integrations

Programming Environment	StP Supported Languages	Navigation From/To	StP/UML Editor	Object
SNiFF+ (Wind River Systems, Inc.)	C++, Java	StP to SNiFF+	Class Diagram, Class Table	Class, Attribute, Operation
		SNiFF+ to StP	Class Diagram, Class Table, State Diagram	Class
Visual Studio (Microsoft Corporation)	Visual C++, Visual J++	StP to Visual Studio	Class Diagram, Class Table	Class, Attribute, Operation
		Visual Studio to StP	Class Diagram, Class Table, State Diagram	Class, Attribute, Operation, State machine

Setting Up Integrations

In order to navigate successfully between StP / UML and the supported programming environments, you must:

- Set the appropriate environment and ToolInfo variables, as described in the *README* files provided on the CD
- Use source code in which the object names are identical to the object names in the StP / UML model (usually accomplished by either generating or reverse engineering the code in StP)

For directions on setting up the integrations, refer to the following README files on your StP ME installation CD:

- *templates \ ct \ sniff \ README-sniff.txt*
- *templates \ ct \ visual_studio \ README-vs.txt*

Note: StP ME does not support concurrent integrations of both SNiFF+ and Visual Studio. Setting the `ProgrammingEnvironment` ToolInfo variable to SNiFF replaces any existing **Go To** menu **Source Code** commands (for Visual Studio navigation) with the appropriate SNiFF commands.

Navigating to SNiFF+

To navigate from StP / UML to the SNiFF+ programming environment, after setting up the integration, as described in the README file:

1. Start a SNiFF+ session and load the appropriate project into the programming language tool.
2. In StP, select an object that supports this navigation in a class diagram or table (see Table 2 on page 7-5).
3. From the **GoTo** menu or shortcut (right-click) **GoTo** submenu, choose one of the provided context-sensitive commands for navigation to SNiFF+, as follows:
   - For classes:
     SNiFF Source Editor
     SNiFF Class Browser
     SNiFF Class Hierarchy Browser
     SNiFF Class Hierarchy Browser (only relatives)
     SNiFF Retriever

- For operations:
  SNiFF Operation Definition
  SNiFF Operation Implementation
  SNiFF Retriever
- For attributes:
  SNiFF Attribute Definition
  SNiFF Retriever

Navigating to Microsoft Visual Studio

To navigate from StP / UML to the Microsoft Visual Studio programming environment, after setting up the integration, as described in the README file:

1. In StP, select an appropriate object in a diagram or table that supports this navigation (see Table 2 on page 7-5).
2. From the **GoTo** menu or shortcut (right-click) **GoTo** submenu, choose **Source Code**.

   If your model has been reverse engineered, StP starts Visual Studio and loads the appropriate source code file for you.

   Otherwise, StP will ask for the location of the source code file.

Navigating to StP/UML

To ensure correct navigation, you must first:

- Perform the setup steps in the README file, as described in “Setting Up Integrations” on page 7-6.
- Set the `system` ToolInfo variable or `IDE_SYSTEM` environment variable to the name of the system to which you want to navigate
- Se the `projdir` ToolInfo variable or the `IDE_PROJDIR` environment variable to the project directory

Navigating from SNiFF+

To navigate from SNiFF+ to StP / UML, choose one of the following commands from SNiFF+’s **StP/UML** menu:

- **Class Diagram for Class** *<class name>*

- **Class Table for Class** <*class name*>
- **State Diagram for Class** <*class name*>

Navigating from Visual Studio

To navigate from Visual Studio to StP / UML, choose one of the following commands from Visual Studio’s **Tool** menu:

- **NavTo UML Class Diagram**
- **NavTo UML State Diagram**
- **NavTo UML Class Table**

Reverse Engineering StP Models

Reverse engineering capabilities and documentation addressed in this section are:

- “Parsing Files for an Entire Directory Tree or Visual Studio Project” on page 7-8
- “Reverse Engineering ObjectAda Source Code” on page 7-9

Parsing Files for an Entire Directory Tree or Visual Studio Project

StP ME allows you to automatically populate the **Files to Parse** list on the **Parse Source Code** dialog with:

- Visual Studio project files
- Files for an entire directory tree

This shortcut works similarly to the Makefile option. It allows you to specify the file extension(s) for files you want to parse, and then automatically populates the **Files to Parse** list with the matching files from the chosen directory or Visual Studio project.

To automatically populate the **Files to Parse** list:

1. On the **Parse Source Code** dialog, select one of the following:
   - To parse an entire directory tree, select a directory.
   - To parse files for a Visual Studio project, select a Visual Studio project file.
2. Click **Makefile Options**; in the **Source File Pattern** field, specify the file extension(s) for the files you want to parse.

   Note: The **Source File Pattern** field is the only criterion that applies to the parsing options described here. All other fields on the **Makefile Options** dialog box are specific to makefiles only and do not otherwise affect parsing.
3. Click **OK** to return to the **Parse Source Code** dialog.
4. Click the appropriate button:
   - **Parse Directory Tree**—Populates the **Files to Parse** list with all files matching the specified extension(s) in the directory tree.
   - **Read Selected DevStudio Project File**—Reads the selected Visual Studio project file, including all relevant options, and automatically populates the **Files to Parse** list with all project files matching the specified extension(s).

Reverse Engineering ObjectAda Source Code

Reverse Engineering of StP/UML models from ObjectAda source code was included in StP Release 7.2, without documentation, and therefore is documented in this *Features Supplement*. “ObjectAda” refers to any Windows- or Windows NT-hosted version of ObjectAda or ObjectAda Real-Time. The ObjectAda source code is parsed and a model is then generated from the parsed code.

Note: Navigating to ObjectAda source is not currently implemented.

Parsing Ada Code

There are two basic requirements for reverse engineering ObjectAda code:

- In ObjectAda, you must first create a project that includes the Ada source file.
- The project must have been successfully compiled.

Once these two requirements have been met, you can reverse engineer ObjectAda source files, as follows:

1. Open an existing system or create a new one.
2. From the StP desktop **Code** menu, select **Reverse Engineering** > **Parse Source Code**.

   The **Parse Source Code** dialog box appears.
3. In the **Filter** text field, enter or browse and choose the name of the parent directory containing the ObjectAda source file(s) to be parsed; then click **Filter**.
4. From the list of directories or files that appears in **Directory Listing**, do any of the following:
   - Double-click a directory name or **Parent Directory** to display the specified directory's contents.
   - Select individual file(s) to be parsed and use the right-arrow button to move your selections to the **Files to Parse** list.
   - Click **Makefile Options** and specify the types of files to parse in the **Source File Pattern** field of the **Makefile Options** dialog; then use the **Parse Directory Tree** button to move matching files to the **Files to Parse** list. (for details, see "Parsing Files for an Entire Directory Tree or Visual Studio Project" on page 7-8).
5. Click the **ObjectAda Library** button and in the **ObjectAda Library Setting** dialog, specify the directory for the ObjectAda library into which the source code has been compiled; then click **OK**.

   Note: You must specify the correct ObjectAda library for the project; otherwise, the source file(s) will not be parsed. The ObjectAda library is located in the file *Ada.lib* in the specific target directory for the project. The name of the target directory generally includes "Deploy" ("Release") or "Development" ("Debug") at the end of the path. An example target directory might be:

   c:\Ada95\Proj2\Proj2-Raven(Intel)-Development
6. On the **Parse Source Code** dialog, click **OK**.

   A progress bar appears while the source files are being parsed.

When parsing is complete, you can generate diagrams and tables for the parsed source code, as described in "Generating a Model from Parsed Source Files," which follows.

Generating a Model from Parsed Source Files

After you have successfully parsed your ObjectAda source files, you can generate diagrams and tables from them:

1. From the desktop **Code** menu, select **Reverse Engineering > Generate Model from Parsed Source Files**.
2. Click **OK** on the **Generate Model from Parsed Source Files** dialog. This generates the diagrams for the respective source files.

8 Report Maker Editor (RME)

The Report Maker Editor (RME) is a tool that allows you to describe in graphical terms a report consisting of StP diagrams, tables, text, and database objects. Sections of the report are assembled into a diagram, which is then used to generate the set of QRL (Query and Reporting Language) commands that make up the report. This capability allows even novice users to easily define a report structure and to generate finished documents without much knowledge of QRL.

In this chapter, RME usage information is provided in:

- "Overview of RME Usage" (immediately below)
- "Creating RME Diagrams" on page 8-2
- "Generating the Report" on page 8-3

Reference information is provided in:

- "RME Objects and Annotations" on page 8-6

This chapter is summarized in:

- "Summary" on page 8-13

Overview of RME Usage

The RME is accessed from the StP desktop's **Document View** (for UML) or the **Report Maker Diagrams** subcategory in the **Model Elements** view (for SE). To generate reports with the RME, you:

- Create a new diagram: On the StP desktop, right-click on **Report Maker Diagrams** and choose **New**.
- Graphically define the structure for your report. To do this, you insert document icons from the Symbols toolbar and then connect the icons with “Next Section” arcs.
- Generate the QRL for the report (**Reports > Generate QRL Report**).
- Run the generated QRL (**Reports > Run Report > Run Generated QRL** <*output_type*>).
- View or print the report as desired.

The RME puts scripts in the *rme_files* directory and reports in *qrl_files*.

Creating RME Diagrams

To create a report using the RME, you begin by graphically defining the structure of the report.

A report can have the following elements:

- A cover page
- A single diagram or a set of diagrams
- A single table or a set of tables
- A set of text paragraphs
- A set of database objects

Each of these items is represented by an icon. By connecting the icons to each other, you define the order of the report. By annotating the icons, you define the type of diagram, table, or object, as well as information on paragraph styles, orientation, and other attributes.

There are only a few simple rules for connecting the icons:

1. You cannot create circular reports. Semantic checking will flag this as an error.
2. The cover page icon can only have an outlink - no inlinks.
3. All other icons can have a maximum of one outgoing link and one incoming link.

4. Links may be either a straight arc or a spline.
5. All icons must have a label.
6. Icons may be reused on different reports; they will retain their properties.

Once you have constructed a diagram and have labeled the icons, you should then save the diagram. At that point you can start annotating the individual icons.

Generating the Report

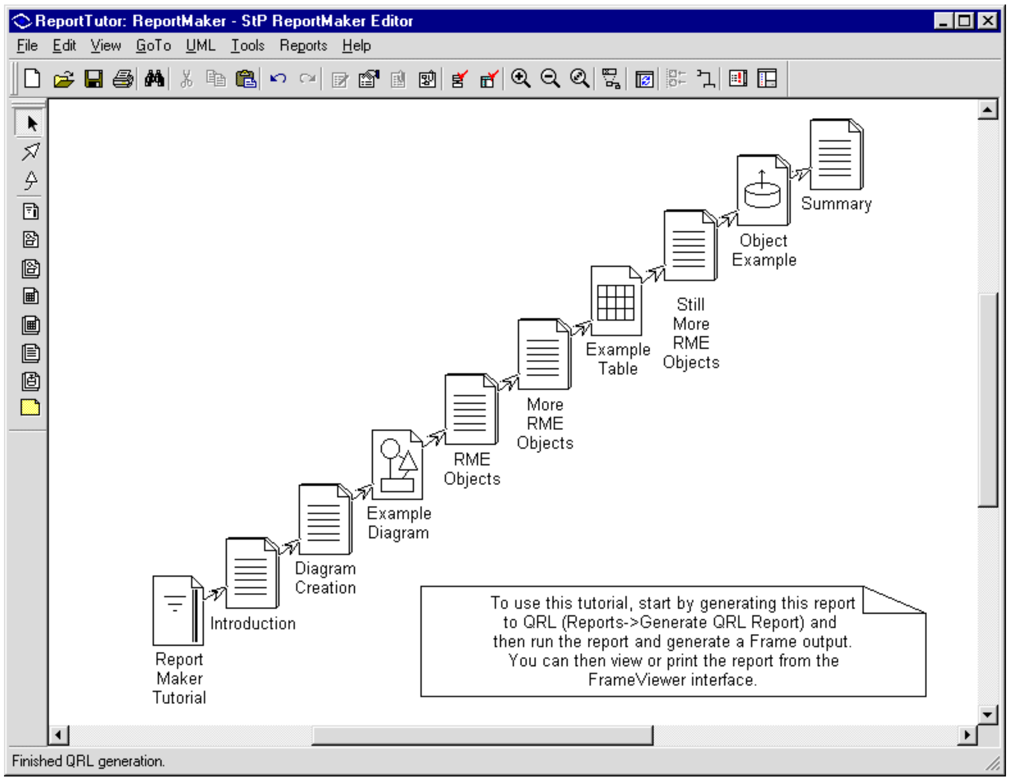
Once you have filled in the attributes for the icons, you can generate the QRL code that will create the report. This function is accessed via the RME's **Reports > Generate QRL Report** command. During this process, you will see status messages that describe where in the process the QRL generation is. When finished, it will produce a QRL file in the *rme_files* directory of your system.

The file that was created can then be viewed and manually edited, or run to produce the final report. To view the file, choose **Edit Generated QRL**. This will allow you to edit the file if you wish.

If you are satisfied with the generated QRL, you can then choose to run the QRL. This is done by selecting **Run Report > Run Generated QRL**. You can choose to run the report to one of three target document systems: standard ASCII, FrameMaker, or RTF. This produces a file of one of the types in the *qrl_files* directory.

Figure 1 shows a sample diagram called ReportTutor. In it, you can see examples of the various icons, each of which represents a document section. The report begins with a cover page followed by two text sets, and then prints the ReportTutor diagram.

Figure 1: Sample Diagram Generated by the RME



To use this tutorial, start by generating this report to QRL (Reports->Generate QRL Report) and then run the report and generate a Frame output. You can then view or print the report from the FrameViewer interface.

Finished QRL generation.

Below is the portion of the QRL code produced that defines the cover page (“Report Maker Tutorial) part of the diagram. You can download a full sample model, including the QRL, from *www.aonix.com/content/downloads/stp/ReportMaker.zip*.

Note: The sample also includes the source text, in RTF and other formats, that was used as a starting point for this chapter.

```
// ReportTutor.qrl
// This file was automatically generated by StP’s Report Maker
// 05/22/01 11:22:57

//  GLOBALS
string query = "";
```

```
void main()
{
    file file_var;
    string inc_file = "";

    target_enum target = target();
    message("Starting report processing...");

    if (target == Ascii) format("rme_report.asc");
    if (target == Framemaker) format("rme_report.mif");
    if (target == RTF) format("rme_report.rtf");

    // Report Maker Tutorial
    // CoverPageSection
    message("-- Processing Report Maker Tutorial...");

    if (target == Ascii) paragraph("center");
    if (target == Framemaker) paragraph("Title");
    if (target == RTF) paragraph("Title");

    print_line("");
    print_line("Report Maker Tutorial");
    print_line("");

    if (target == Ascii) paragraph("center");
    if (target == Framemaker) paragraph("Author");
    if (target == RTF) paragraph("Author");

    print_line("Project Directory : C:/StP-Systems/");
    print_line("System Name : ReportMaker");
    print_line("Prepared by : Aonix");
    print_line("05/22/01 11:22:57");
    print_line("");

    page();

   ...
```

RME Objects and Annotations

This section discusses each RME object, along with the specific annotations that apply to it. To see what the icon for each object looks like, refer to Figure 1 on page 8-4.

Generic annotation items (those associated with all or most icons) are listed below.

Annotation/Item	Description
Generic Report Options ◆ Start On New Page	◆ Specifies that generated output begin on a new page. If it is not present, or is set to False, the icon's information will flow immediately after the last icon's information on the same page (if possible).
Text Report Options ◆ Custom / ASCII / Frame / RTF Para. Style ◆ Include File	◆ Specifies paragraph format. Custom allows a user-defined format to be specified. ◆ Allows you to specify an external file to be included in the output text. If a file is specified, it will appear after any note descriptions and will allow you to use the same formatting. You should specify the full path to any file.
Object ◆ Name	◆ Specifies the name of the object

Annotations other than those described in the sections below can be ignored; they need not be filled in. Examples of such annotations are Requirement, Glossary Definition, and Requirement Allocation.

Cover Page

The Cover Page icon is typically used to start a document. It provides a convenient way to create a simple cover sheet.

Annotation items specific to the Cover Page icon are shown below.

Coverpage Options	
◆ Title	◆ Set to icon's label by default
◆ Author	◆ Set to user log-in by default
◆ Show Report Date	◆ If True, date is placed under Title and Author.
	Note: These items are typically centered on the page.

Text Set

The Text Set icon allows you to insert text paragraphs and headings into your final document. If you look at the document you are reading now, you can see various section headings that are automatically numbered and text paragraphs that contain descriptions for each section.

Each icon can actually contain multiple sections of paragraphs and heading information. The information for each is contained in an annotation called Text Report Options. The text of the annotation note is displayed in the **Description** area of the annotation editor. An easy way to see this text is to select a text report option in the annotation editor and then double-click on the note. This will display the editable note description.

Annotation items specific to the Text Set icon are shown below.

Text Report Options	
◆ Custom / ASCII / Frame / RTF Para. Style	◆ Specifies paragraph format. *Custom* allows a user defined format to be specified.
◆ Include File	◆ Allows you to specify an external file to be included in the output text. If a file is specified, it will appear after any note descriptions and will use the same formatting. You should specify the full path to any file.

When you define paragraph styles, you are supplied with a set of predefined values specific to that style. The only exception is the *custom* paragraph style, which allows a user-defined style to be specified. Make sure that the report format file that you are using defines any custom paragraph styles. If you fail to set a paragraph style, the document will continue using the last paragraph style specified.

The predefined styles are found in a “format file.” This is a file that contains paragraph and character formatting information. The RME uses a supplied set of format files called *rme_report*. If you need to specify a custom format file (versus the default *rme_report*), you can do so by annotating the diagram itself. There is a special annotation for the diagram that allows you to set your own format file. You will probably want to use the custom paragraph style if you have specified a custom format file.

You can have up to ten (numbered 0 - 9) text report options per *Text Set* icon. This allows you to put a number of document paragraphs and headings together in one place. If you need more text you can always link up additional Text Set icons. Each one can hold an additional ten text sections.

Diagrams and Diagram Sets

Diagrams and diagram sets allow you to include your StP diagrams in your documents. The difference between a Diagram icon and a Diagram Set icon is that the Diagram icon produces only a single diagram to output. A diagram set outputs all diagrams of a given type.

Annotation items specific to the Diagram and Diagram Set icons are shown below. Items below the dashed line in the table apply to the Diagram Set only.

Diagram Report Options	
◆ Diagram Type/Name	◆ Specifies the type/name of the diagram
◆ Caption	◆ Specifies that the caption is to appear at the bottom of the diagram
◆ Caption Orientation	◆ Specifies if the caption is landscape or portrait; default is 'best fit.'
◆ Diagram Frame Height/Width	◆ Sets the height/width of the bounding box for the diagram
◆ Use Print Setting	◆ Instructs the program to use a named print setting
------------------------------	---
◆ Start Each On New Page	◆ Starts every diagram on a new page
◆ Omit Diagram Called	◆ Excludes a named diagram
◆ Sort By	◆ Sorts the diagrams alphabetically by field (e.g., 'name')

The annotations used to support a single diagram differ only in that a specific Diagram Name is required for a single diagram. By default, this name comes from the label on the icon, but it can be changed.

When you add a Diagram or Diagram Set icon, you need to specify the Diagram Type (click on **Diagram Type** under Diagram Report Options and select a value from the **Description** list). This identifies the kind of diagram to be printed. If you are printing just a single diagram, you will also need to specify the Diagram Name.

Tables and Table Sets

Tables and table sets allow you to include your StP tables in your documents. The difference between a Table icon and a Table Set icon is that the Table icon produces only a single table to output. A Table Set outputs all tables of a given type.

Annotation items specific to the Table and Table Set icons are shown below. Items below the dashed line in the table apply to the Table Set only.

Table Report Options	
◆ Caption	◆ Sets the caption to appear at the top of the table
◆ Orientation	◆ Sets the orientation to portrait or landscape
◆ Table Width	◆ Sets the width of the bounding box for the table
◆ Row / Column / Header Row Range	◆ Specifies the numeric range of rows, columns, or header rows to be printed
◆ Show Row / Column Indices	◆ Enables the printing of new row or column indices
◆ Use Print Setting	◆ Instructs the program to use a named print setting
---------------------------------	---
◆ Start Each On New Page	◆ Starts every table on a new page
◆ Omit Table Called	◆ Excludes a named table
◆ Sort By	◆ Sorts the tables alphabetically by field (e.g., ‘name’)

The annotations used to support a single table differ only in that a specific table name is required for a single table. By default, this name comes from the label on the icon, but it can be changed.

When you add a Table or a Table Set icon, you need to specify the table type. This identifies the kind of diagram to be printed. If you are printing just a single table, you will also need to specify the table name.

Table 1 illustrates what a requirements table (REQTable) looks like when printed out using the RME. Because tables are currently output in portrait mode only, you may need to split them into multiple pieces in order to show all the information in a pleasing manner.

Table 1: Sample Requirements Table

Requirement			Definition
1.0; Simplify Document Creation	1.1; Build Graphical Report Tool	1.1.1; Construct RME Editor	Build a prototype editor and start refining the prototype for a time until all necessary features are implemented and working.
1.0; Simplify Document Creation	1.1; Build Graphical Report Tool	1.1.2; Test RME Editor	Repeatedly test the prototype and fix any bugs that are found. Update features along the way.
1.0; Simplify Document Creation	1.1; Build Graphical Report Tool	1.1.3; Update RME Editor	Enhance the prototypes to accommodate new editor types and capabilities in the StP tool.
1.0; Simplify Document Creation	1.1; Build Graphical Report Tool	1.1.4; Market RME Editor Internally	Distribute the prototype to the field AE's and get feedback on new features. Visit with product and corporate marketing to introduce the prototype.
1.0; Simplify Document Creation	1.2; Provide Tutorial	1.2.2; Build RME Tutorial Requirements Table	Build a requirements table (like this table) that describes the process of creating the RME editor. Include this table in the tutorial as an example of printing a table.
1.0; Simplify Document Creation	1.2; Provide Tutorial	1.2.3; Test RME Tutorial	Repeatedly test the tutorial output to make sure that it looks correct and produces aesthetically pleasing output.

Object Sets

The Object Set icon is used to query the StP database and report on individual items contained in it. It does not provide clues as to what the queries should be or how to form them. This icon is for more advanced users because you must know a bit about how the StP database is organized, and how to formulate OMS queries, in order to use an object set effectively. The resulting QRL code is a looping structure that collects a set of data and provides it for output to the report.

Annotation items specific to the Object Set icon are shown below.

DB Object Report Options	
◆ DB Object Type	◆ Specifies the OMS data type for the object to be reported
◆ Start Each on New Page	◆ Prints each resulting output on a new page
◆ Specific Object Name	◆ Specifies a single object by name to be reported
◆ List Object Attributes/ Operations	◆ Lists object attributes or operations, if any
◆ List Object In/Out Links	◆ Lists the object's *in* or *out* links
◆ Show Item Value For	◆ Prints the value of the named item
◆ Show Note Description For	◆ Prints the note description of a named annotation note

You may need to examine the QRL code when you use this icon, so you can get a feel for how the Object Set icon generates QRL. You will probably want to experiment with the features of this icon.

In addition, the same paragraph styles as described in "Text Set" on page 8-7 are also available. They allow you to format the output from the generated QRL query.

Object Set Example Output

The following is a sample output from an object set query.

```
--- Database report on type : MultiObjectSection ---
Object Example
  In Link(s) :
    NextSection :

  Out Link(s) :
    NextSection :

  Note Description for GenericObject :
This text is contained in the GenericObject note description
for the 'Example Object' of the ReportTutor RME diagram in
the ReportMaker system.
```

Summary

This chapter has shown how the RME can be used to graphically describe a report and produce output from an StP database. The RME is useful in automating the creation of QRL code for the purpose of report generation. However, more advanced features exist in the QRL language that are not considered here.

For those who need to use the advanced features, the RME can be used to quickly provide a working prototype and fine tune the output until additional customization is needed. At that point, the QRL code can be manually edited and reworked to create the report format desired.

Be aware that some features will look different when output to different applications. For instance, while FrameMaker format supports vertical spanning cells, RTF does not. Instead, the text will be repeated for each of the cells. RTF also does not support a true landscape diagram. Instead it rotates the paper and your footer will appear on the right edge of the page, sideways. For this reason it is best to keep RTF formatted documents simple and to use portrait mode if possible. Similarly, ASCII output is very limited in its capabilities.

The RME also promotes reusability. When you define an RME icon on a given report, you can reuse that icon on a different report; all its formatting and specifications will be carried over. You just need to give the icon the same name and make sure it is the same type.

If you need to use a set of diagrams in a different report, you just need to recreate the icon on the new report. You can even cut and paste between reports.

For more information on creating an RME report, examine all the pieces that make up the sample provided in *www.aonix.com/content/downloads/stp/ReportMaker.zip*. For help on annotation notes and items, you can also look under the help option for that note or item. The report that forms the basis for this chapter was created with the RME. Larger and more complex reports are possible.