HP OpenView
Service Quality Manager

Service Designer User Interface

User's Guide

Edition: 1.4

for the Microsoft Windows Operating System

March 2007

Legal Notices

Warranty

The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be liable for technical or editorial errors or omissions contained herein.

The information contained herein is subject to change without notice.

Restricted Rights Legend

Confidential computer software. Valid license from HP required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Copyright Notices

Trademark Notices

Adobe®, Acrobat®, and PostScript® are trademarks of Adobe Systems Incorporated.

HP-UX Release 10.20 and later and HP-UX Release 11.00 and later (in both 32 and 64-bit configurations) on all HP 9000 computers are Open Group UNIX 95 branded products.

Java™ and all Java based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

Microsoft®, Windows®, Windows NT® and Windows® XP are U.S. registered trademarks of Microsoft Corporation.

Oracle® is a registered US trademark of Oracle Corporation, Redwood City, California.

UNIX® is a registered trademark of The Open Group.

Contents

Preface

HP OpenView Service Quality Manager **Service Designer** has been developed as an Add-In to Rational Rose. This document describes the Graphical User Interface of OpenView Service Quality Manager **Service Designer**. In addition to the facilities provided by the standard version of Rational Rose, OpenView Service Quality Manager Service Designer allows you to graphically design a Service in the Unified Modeling Language (UML) and enables the generation of XML Service definitions, which are then used to import the model definition into Service Center. These generated XML files also allow Reverse Engineering, described in Chapter 5.

Important

This guide is not a guide to Rational Rose, it describes how to design and define a Service using the Service Designer Add-In. Rational Rose is a very rich tool, therefore we describe only the single recommended way to design a Service.

Prerequisites

Rational Rose Modeler or Enterprise edition must be installed.

Intended Audience

This document is intended for personnel who will use HP OpenView Service Quality Manager Service Designer. It is recommended that such personnel have knowledge of the Unified Modeling Language (UML) and of Rational Rose.

Software Versions

The software versions referred to in this document are:

Service Quality Manager	Service Designer	Windows	Rational Rose
1.4	1.4	XP	Rational Rose Modeler or Enterprise edition V7

Typographical Conventions

`Courier` Font:

- Source code and examples of file contents.
- Commands that you enter on the screen.
- Path names
- Keyboard key names

Italic Text:

- Filenames, programs and parameters.
- The names of other documents referenced in this manual.

Bold Text:

- To introduce new terms and to emphasize important words.

Associated Documents

OpenView Service Quality Manager

- *OpenView Service Quality Manager Overview*
- *OpenView Service Quality Manager SLA Monitoring UI User's Guide.*
- *OpenView Service Quality Manager SL Administration UI User's Guide*
- *OpenView Service Quality Manager Service Designer UI User's Guide (this manual).*
- *OpenView Service Quality Manager Planning Guide*
- *OpenView Service Quality Manager Installation Guide*
- *OpenView Service Quality Manager Administration Guide*
- *OpenView Service Quality Manager Sampling Scheduling Guide*

Rational Rose

This guide documents how to use the OpenView Service Quality Manager Service Designer Add-In to Rational Rose. For information about the Rational Rose standard product, please refer to the Rational Rose documentation and the Rational Rose Standard on-line Help.

Support

You can visit the HP OpenView support web site at:

http://support.openview.hp.com/support.jsp

This Web site provides contact information and details about the products, services, and support that HP OpenView offers.

HP OpenView online software support provides customer self-solve capabilities. It provides a fast and efficient way to access interactive technical support tools needed to manage your business. As a valued support customer, you can benefit by using the support site to:

- Search for knowledge documents of interest
- Submit enhancement requests online
- Download software patches

- Submit and track progress on support cases
- Manage a support contract
- Look up HP support contacts
- Review information about available services
- Enter discussions with other software customers
- Research and register for software training

Chapter 1

Introduction

This chapter describes the principles of designing and defining a Service using the HP OpenView Service Quality Manager **Service Designer**.

1.1 What is the HP OpenView Service Quality Manager?

The HP OpenView Service Quality Manager is a service level management product that automates the definition, real-time monitoring, and reporting of operational and customer Service Level Agreements (SLAs) and Service Levels.

The **Service Designer** is the part of the OpenView Service Quality Manager that allows you to design a Service, and all its constituent parts.

1.2 What is a Service?

A service is a set of independent functions. A service can include anything from a single leased-line service, to a complex application, such as video conferencing. You can model three types of services with OpenView Service Quality Manager:

- Single-customer services: these services are associated with a single known customer. For example, connection services (such as SDH, asynchronous transfer mode (ATM), FR, and asymmetric digital subscriber line (ADSL)) are usually single-customer services.

- Multi-customer services: these services are shared by various customers, meaning that OpenView Service Quality Manager must deduce the customer name using information extracted from the external data source.

- No-customer services: these are services that are shared by various customers, but OpenView Service Quality Manager does not know the customer name. For example, the customer identity is 'lost' when an ADSL customer uses an IP service with an IP address assigned by a dynamic host configuration protocol (DHCP) server during the connection phase.

1.3 The Service Designer Graphical User Interface

The Service Designer Graphical User Interface (GUI) allows the user to design and model services in the Unified Modeling Language (UML). The Service Designer GUI is based on the Rational Rose product, widely used software that offers support of the UML standard.

OpenView Service Quality Manager provides predefined meta-definitions for services. You can use the Service Designer GUI to customize the predefined definitions included with OpenView Service Quality Manager, thereby reducing the effort needed to design a new service.

In the Service Designer GUI, you can fully design a service and check its model without any interaction with the service level management components. Once you have designed and validated a service, OpenView Service Quality Manager allows you to generate an XML document representing the service and import it into the OpenView Service Quality Manager data repository. You can enrich the model during the SLA lifecycle by creating new service definitions or by updating existing service definitions.

1.4 The SLA Lifecycle

The SLA lifecycle has five phases defining the sequence of the processes that comprise an SLA. These phases are:

- **Service design**
  Customer needs are identified and the service is modeled.

- **Service level instantiation**
  Individual customer instances are created and the service is enabled.

- **SLA creation**
  SLAs are negotiated and created for individual customers.

- **Service monitoring**
  Services and service level agreements are monitored and real-time violations are handled.

- **Service reporting**
  Reports on the performance of the Service are generated.

This sequence of events and the parts of OpenView Service Quality Manager used for each phase are illustrated in Figure 1.

Figure 1   **The SLA Lifecycle**
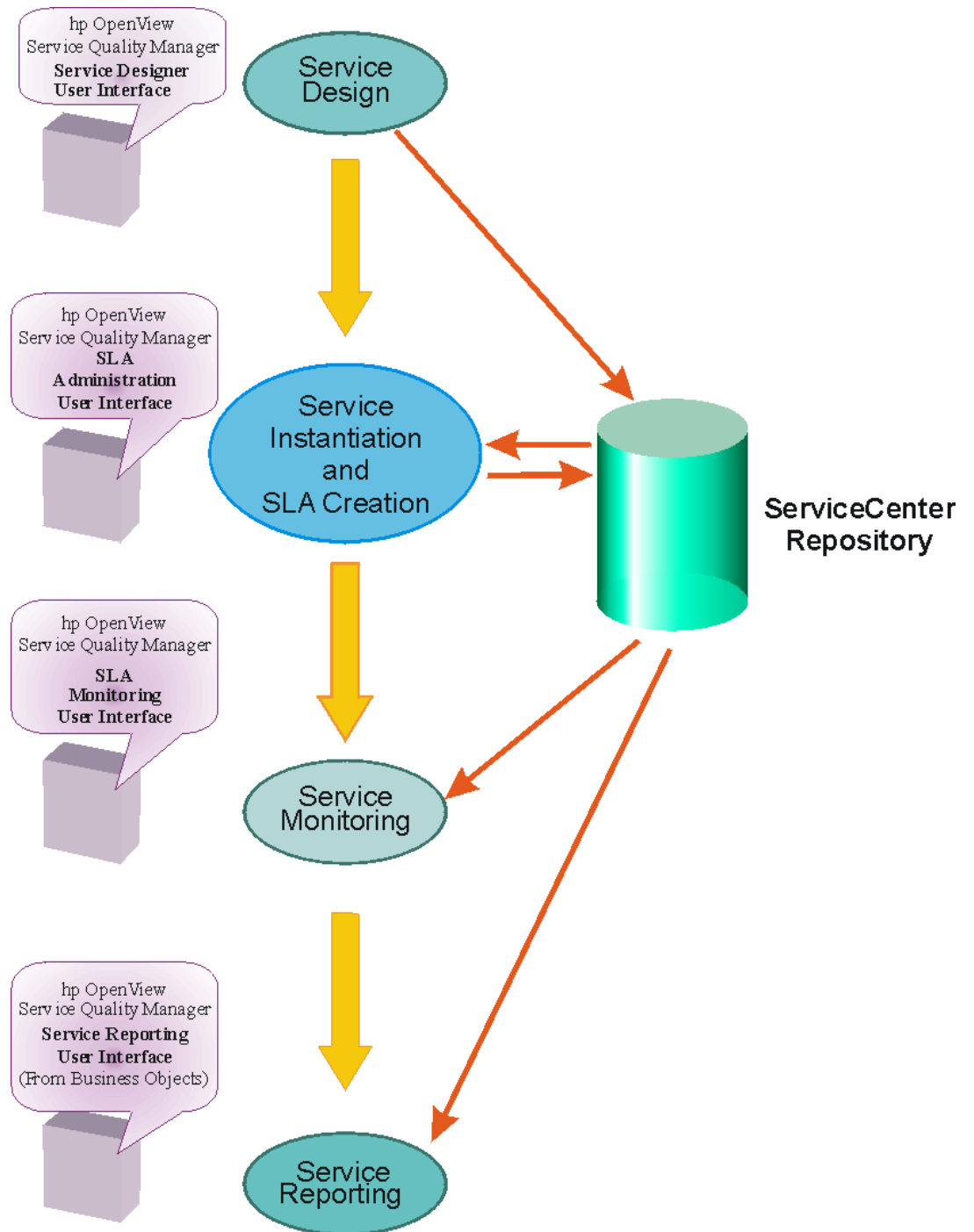
Chapter 2

The Service Model

During the first phase of the SLA lifecycle (see Section 1.4), a new service is designed; this includes modeling and defining the new service. Service models and definitions are built with the OpenView Service Quality Manager object model. The service definition can be created with the OpenView Service Quality Manager Service Designer provided with OpenView. Service Designer has a Graphical User Interface (GUI) that allows the model to be viewed.

The following sections describe the OpenView Service Quality Manager object model and the Service Designer GUI.

2.1 The Object Model for Designing Services

OpenView Service Quality Manager uses a technology-neutral object model for modeling and defining services and the entities that collect data about these services. You can map the object model to any service. The following sections describe the object model used during the design phase to design the service itself and to design data acquisition.

2.1.1 Designing a Service

In the OpenView Service Quality Manager object model, a **service** is a collection of **service components**. A service component corresponds to hardware and software elements, as well as the underlying communications medium used by the service.

A service can include anything from a single Leased Line service, to a complex application, for example: vision conferencing.

A Service and Service components must have parameters. These are values that are periodically updated and that help determine the quality of service, either from a customer or a network operator perspective.

Service components can themselves be composed of service components.

Figure 2 **A Neutral Meta-model of a Service.**
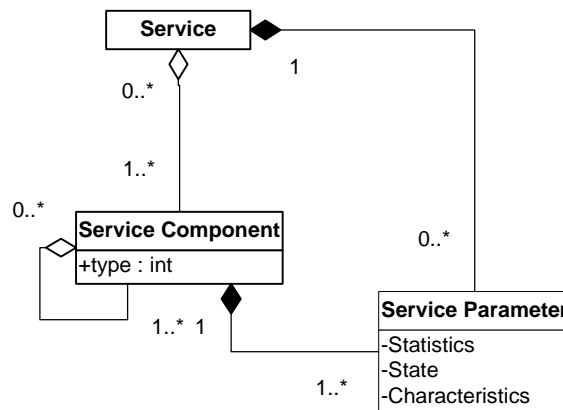
This neutral meta-model can be used to model any service, such as a video service. For example, a video service can be composed of a video server and several web servers (load balancing) hosted by an Internet Service Provider (ISP). This is a specific application of the meta-object model. Such a specific application is called a concrete service model.

Figure 3 **Example of a Concrete Service Model**
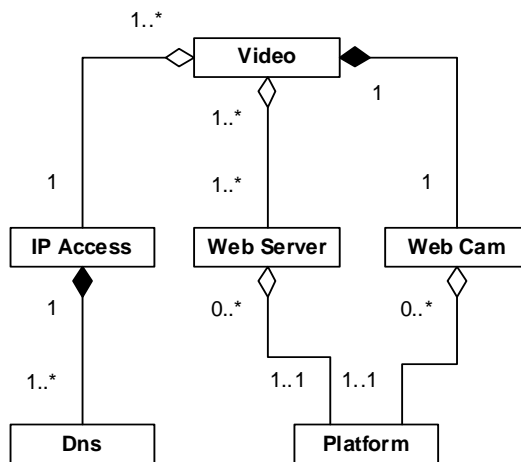
During service design, the service, service components, and service parameters are defined for the service (in this example the Video service) using the Service Designer GUI.

2.1.2 Service

A **Service** is a set of independent functions (service components) that consist of hardware and software elements and an underlying communications medium.

2.1.3   Service Components

Service Components are hardware and software elements of a Service, as well as the underlying communications medium used by the Service. A particular component may or may not be visible by a given Service Customer. This visibility is defined as an attribute of a Service Component. Service Components have both Parameters and Properties. Parameters have values that are calculated according to the performance of the system, whereas Properties have values that do not change.

2.1.4   Data Feeders

Data Feeders are just what their name suggests; they are the source of data used by OpenView Service Quality Manager. They "feed" data, either directly to a Service or to a Service Component, which then "feeds" the data to the Service in turn. What data they feed is defined by means of Parameters and Properties.

The Parameters and Properties of a Data Feeder are set with Service Designer, but the Data Feeder itself can be created by an external application.

2.1.5   Parameters

Services, Service Components and Data Feeders must have Parameters and Properties.

A Parameter is an indicator of the end-to-end Quality of Service (QoS) either from a customer perspective or from a network operator perspective.

The OpenView Service Quality Manager object model supports four kinds of service parameter:

- Quality of service metrics, such as performance, errors, and availability.

- State

- Usage

- Characteristics

All parameters must specify whether their value is specific to a single subscriber (such as the number of transmitted videos) or whether they apply to all customers (such as a host's CPU load).

Service parameters can be one of the following data types:

- Display string: a printable string.

- Integer: a signed, 64-byte integer.

- Enumeration: open enumeration definition.

- Real: a signed float.

- Relative time: number of milliseconds.

- Absolute time: time, given using GMT notation
  (YYYY-MM-DD hh:mm:ss, for example 2002-05-31 13:20:00).

OpenView Service Quality Manager Service Designer allows you to define special **static parameters**, called **properties** that are given a value only when an instance of an OpenView Service Quality Manager object is created. For example, a service component can have a property called "location".

2.1.6   Properties

OpenView Service Quality Manager allows the definition of **static** parameters (that is: Properties), which are only valued when an Instance is created. A good example of

such a parameter is the **Location**. The value for a given instance of the property does not change.

2.1.7 Naming Rules

All the Object Identifiers must respect the following naming rules:

- The first character is one of a-z or A-Z.

- The other characters can be any of a-z, or A-Z, or [0-9] or '-', or '_', or ':'.

- The maximum number of characters for a Parameter is 12.

- The maximum number of characters for all the other objects is 16.

2.1.8 Modeling and Defining Services with the Service Designer Graphical User Interface (GUI)

The Service Designer graphical user interface allows you to model a service.
You can:

- Design services, service components, define relationships with data feeders, and their relationships with one another.

- Design expressions for parameter evaluations.

- Check model consistency.

- Generate XML files used to load the model in SLA Manager.

The GUI uses the Rational Rose product to represent the OpenView Service Quality Manager Object model in Unified Modeling Language (UML). UML is an application modeling language for class and object modeling, component modeling, and distribution and deployment modeling.

OpenView Service Quality Manager provides a generic service information model that can be mapped to any service.

The GUI represents services, service components, and data feeders as Classes with relationships. OpenView Service Quality Manager gives these service elements the following Class types (or "stereotypes") in UML:

- Services have the stereotype **Service**.

- Service components have the stereotype **Service Component**.

- Data feeders have the stereotype **Data Feeder**.

- Expressions have the stereotype **Expression**.

- Enumerations have the stereotype **Enum**.

- Parameters of services, service components, and data feeders are represented in UML by class attributes with the stereotype **Param**.

- Properties of services, service components, and data feeders are represented in UML by class attributes with the stereotype **Prop**.

Once you have designed a new service, it must be instantiated.

During instantiation, the definitions created during the service design step of the Service Level Agreement (SLA) lifecycle are mapped to real-world services, service components, and data feeder objects. Although we are not concerned with the complete SLA process in this manual, it is useful to place Service Designer in its context in this process.

2.2 Rational Rose Standard Features

You can take advantage of Rational Rose features to create your own Data Feeder or Service Component template libraries. Rational Rose can keep a model in one or more files. These files are Petal files or Controlled Unit files. Controlled Unit or Petal files are files into which Rational Rose stores all or part of a model. It is possible in this case to define a set of Service Components and keep them in Petal files or Controlled Unit files and import them and use them whenever it is necessary. This feature is not specific to the Service Designer Add-In, it is a pure Rose feature.

Chapter 3

Getting Started with Service Designer

3.1 Starting Service Designer

Click on the Rational Rose icon [icon]

When you have done this, the window illustrated in Figure 4 will be displayed:

Figure 4   First Rational Rose Window
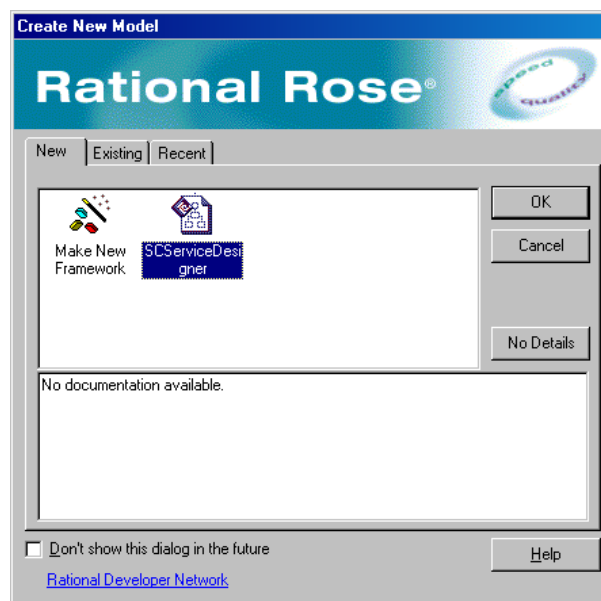
To be able to use all the facilities of the **OpenView Service Quality Manager Service Designer Add-In**, you must ensure that the Service Designer Add-In is activated. You only need to do this when you are using **OpenView Service Quality Manager Service Designer** for the first time, just after you have installed it, or if it has been de-activated at any time.

How to activate the **Service Designer Add-In** is described in Section 3.1.2.

How to activate the **Framework Wizard Add-In** is described in Section 7.1. You only need to do this if you want to be able to use the **OpenView Service Designer Initial Framework** feature.

3.1.1 The Log Window

It is advisable to be able to view the Rose log. The log is not specific to Service Designer. It is a standard Rose facility.

To display the log, select **Log** from the **View** menu.

The Log window will display below the principle Rational Rose window. It displays information which is useful to know.

Figure 5   Example of a Log Window

The Timestamp is not displayed in this window if you do not also select **Timestamp** from the **View** window.

3.1.2 Making Service Designer Available

Before you can use **Service Designer**, you must make it available (you usually only have to do this the first time you use Service Designer, just after installation, but if it is de-activated at any time, then you will need to re-activate it.)

To make Service Designer available:

1. Click on Add-Ins on the menu bar.

2. Select **Add-In** Manager.

3. The **Add-In Manager** dialog box will be displayed.

Figure 6   Add-in Manager box, Service Designer
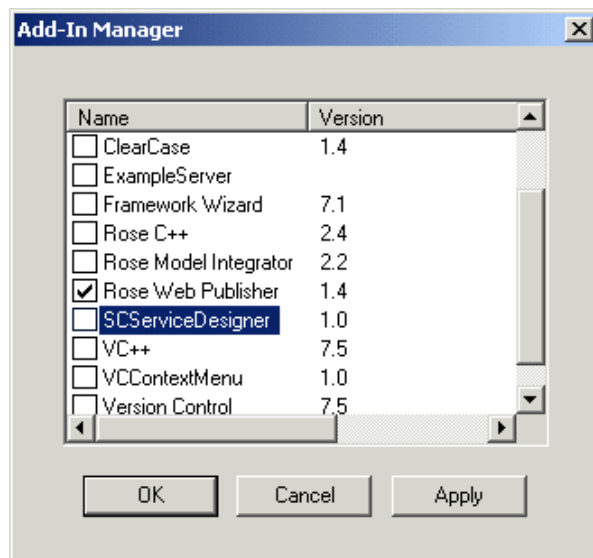
4. Check the box to the left of **SCServiceDesigne**r and click **Apply.**

The message:

```
Info: The SCService Designer add-in has been activated
```

will be displayed in the log window.

5. You must now ensure that the language you are using is SCServiceDesigner. (The current default language is displayed on the bar at the bottom of the Rational Rose window.) If it is not SCServiceDesigner:

6. Choose **Options** from the **Tools** menu to display the **Options** dialog box.

7. Choose the **Notation** tab to display the Notation choices.
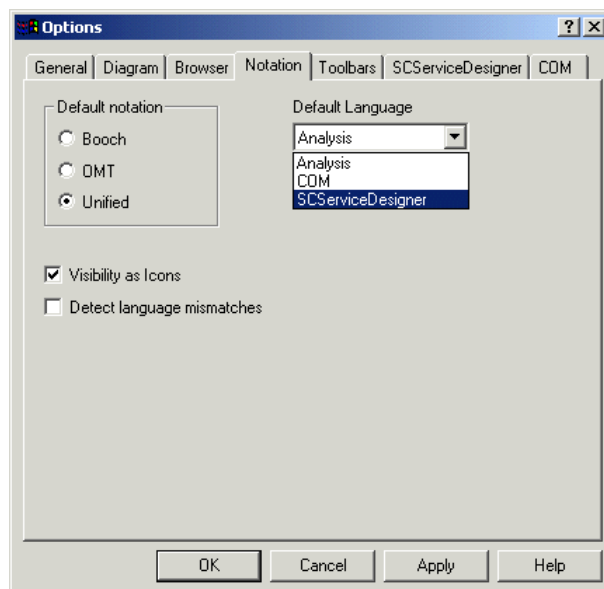
Figure 7   Options Dialog Box, Notation Tab

8. If the default language already displayed in the Default Language box, is not SCServiceDesigner, Choose **SCServiceDesigner** from the drop-down menu of the Default Language box.

9. Choose **Unified** as the Default Notation. (This ensures that the graphical representation of definitions will be displayed in the format illustrated in this document.)

10. Click on **Apply**, and then on **OK**.

Reminder

If, at any time, you de-activate SCServiceDesigner by unchecking it in the Add-In Manager window, a pop-up information box is displayed. This is illustrated in Figure 8.

Figure 8   Pop-up Information Box
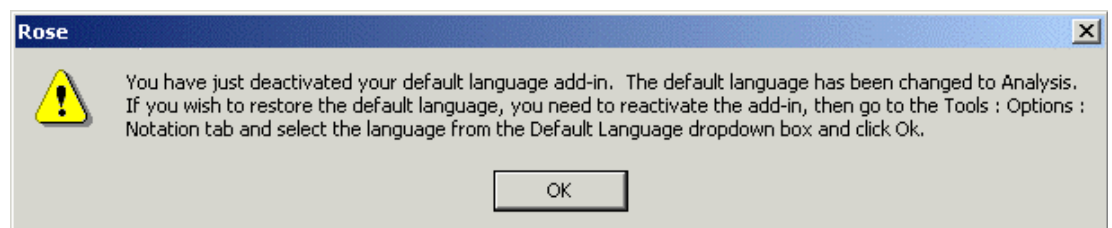
3.1.3 Making the Framework Wizard Available

The **OpenView Service Quality Manager** Service Designer Initial Framework is a feature that allows you to create a new model by using as a basis, a set of UML elements and Expression Classes that have been previously defined. This enables you to create a new model more quickly.

When the **Framework Wizard Add-In** is activated, from the **Create New Model** window, where it will now be displayed, you can choose the **pre-defined Initial Framework** provided with Service Designer.

If you do not see the Initial Framework icon, you need to activate this facility. See Section 7.1.

3.1.4 Placing Icons on the Vertical Toolbar

Customizing the vertical toolbar between the Tree window and the Class diagram window enables you to include icons specific to Service Designer, facilitating the tasks you need to perform to create your model. **You only have to do this once after Installation**.

Icons are provided for creating the classes:

Service ,

Service Component ,

Data Feeder

and Expressions $\sum x^2$

1. From the **Options** menu, click on the **Toolbars** tab.

2. In the Customize toolbars section of the dialog box, select **UML**.

3. This displays a **Customize Toolbar** dialog box.

Figure 9   The Customize Toolbar Dialog Box
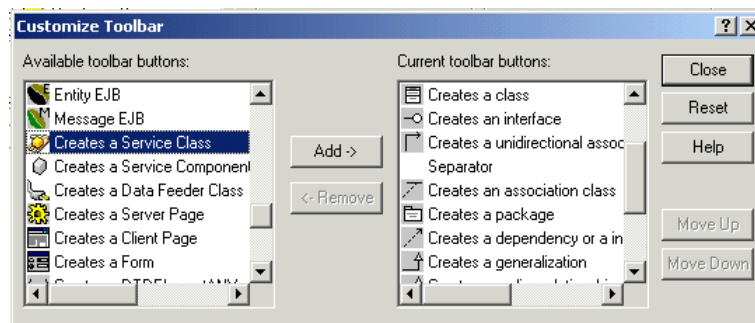
4. Choose the **Creates a Service Class** icon from the **Available toolbar buttons**: list box.

5. Click on the **Add->** button.

6. The icon and its description will move to the right-hand **Current toolbar buttons:** list box.

Do this for each of the icons that you want to add to the toolbar

- Creates a Service Component Class,

- Creates a Data Feeder Class,

- $\sum x^2$ Creates an Expression.

We recommend that you also add the Unidirectional Association and Unidirectional Aggregation icons to this vertical toolbar. They are:

- ⌐→ Creates an Unidirectional Association

- ↗ Creates an Unidirectional Aggregation

7. When you have added all the icons that you want to the right-hand list box and placed them in the position you want, click on the **Close** button on the dialog box.

8. Click on the **OK** button on the **Options** dialog box.

The icon or icons will have been added to the toolbar, which will now look similar to the illustration in Figure 10.

Figure 10   **Customized Vertical Toolbar**

3.2 Creating a New Model

To create a new model:

1. Click on **New** in the **File** menu, or

2. On the **page** icon

3. If the Framework Wizard is activated, the Create New Model window will be displayed. This is illustrated in Figure 11.

Figure 11   **Create New Model Window**
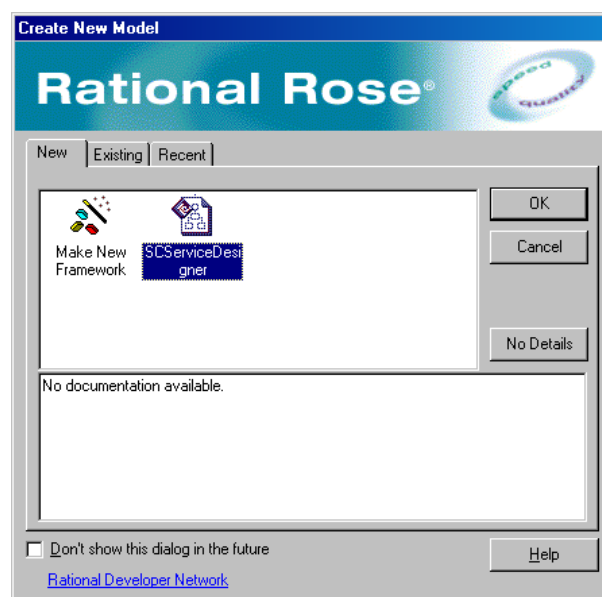
When you use the Initial Framework, the Service Designer window looks like the one illustrated in Figure 12.

Important

It is recommended that you begin to create your model with the Initial Framework provided with Service Designer. If you do begin with an Initial Framework, the Class Diagram Logical View Window will already have the items in it, which you can modify to match your new Service.

Figure 12   First Service Designer Window
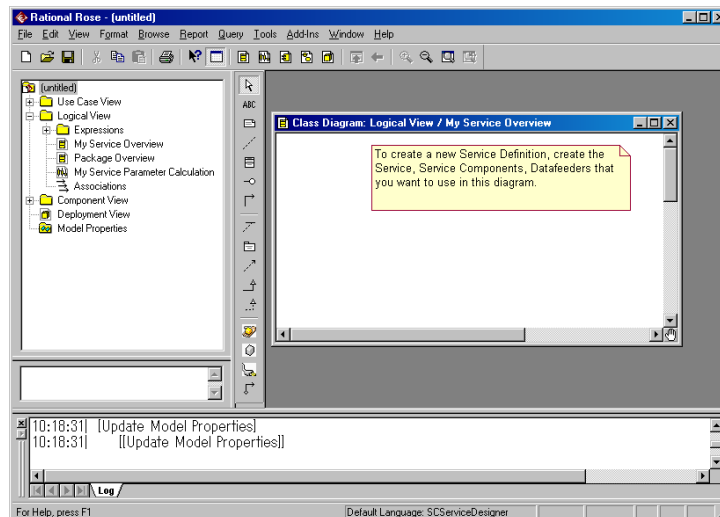
This window displays four "views" from which it is possible to work:

- Use Case View

- Logical View

- Component View

- Deployment View

You can also access Model Properties from this window.

Important

You must only use the **Logical View** to design, define and manage Services. The full menu is not displayed until you have chosen this view and clicked on the Main icon.

3.3 How to Get Help

When you have made the **full** menu bar available, the **Help** drop-down menu becomes available.

When you perform some actions, help in the form of Information, Warning or other messages are displayed in Pop-up boxes.

The **Context Sensitive Help** button on the Icon bar below the Menu bar provides context-sensitive help for the **Standard Rational Rose functions**, but there is no context sensitive help available for this version of **OpenView Service Quality Manager Service Designer**.

To get help on **Service Designer**, go to the **Start** button and choose **Programs.** Choose **HP OpenView\SQM\Service Designer\Documentation** and then **Service**

Designer User Guide. This will display an HTML version of the user manual. You can search for information in this manual using the **Find** facility and by clicking the item you want in the table of contents. Cross-references are also live links to the items referred to.

Chapter 4

Designing a Service

This chapter describes how to define Services.

For an overview of the design process refer to 0.

The Service design process use a very small subset of the UML design.

For a simple Service design, you will only have to use:

- A single class diagram to define the Service, Service Components, DFD and their relationships
- Some sequence diagrams to define the calculation of the parameters

For more complex Services, you can take advantage of some other Rose Features:

- Create other class diagrams to represent other parts of the Service,
- Store part of the model in a 'package' in order to be able to reuse it in other models.

The Creation of new class diagrams and new sequence diagrams must **only** be done in the "Logical View" tree.

The rest of this chapter does not describe standard Rose features, but only features specific to Service Designer.

After starting Service Designer as described in 0, choose the **Logical View** type by double-clicking the icon with the left mouse button.

4.1 Defining Services, Service Components, Data Feeders and Expressions

When the Logical View window is displayed you can make **Main** and **Associations** available by expanding the tree structure (click on the plus sign (+) to the left of the Folder icon). Double click with the left mouse button on the **Main** icon. The menu bar will now display all the available items. From the left these are:

File, Edit, View, Format, Browse, Report, Query, **Tools**, Add-Ins, Window, Help

Important

You **must** double click on **Main** for the full complement of items to be displayed on the menu bar.

It is **ONLY** from the **Tools** menu that **Service Designer** is available. However, if you have configured the Vertical Tool bar, many actions can be performed using the icons available on this toolbar.

Now click on **Tools**, the menu displayed in Figure 13 will be displayed. You will see that **Service Designer** is displayed.

Figure 13   **The Tools Menu**
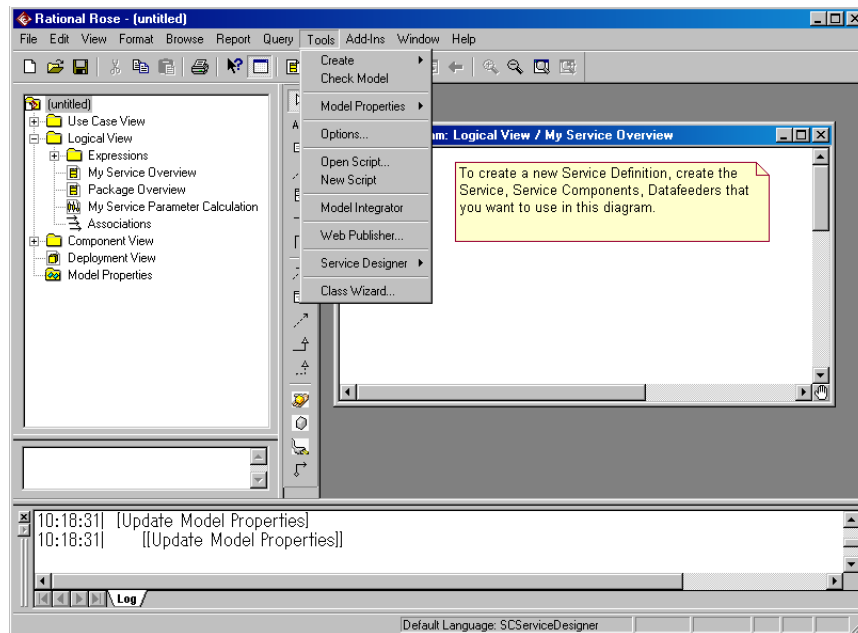
Note

Do not be tempted to choose the **Create** option from this menu, this does not give you access to Service Designer. You cannot create Services, Service Components or Data Feeders from this option.

You can create **Services**, **Service Components, Data Feeders** and **Expressions** in two ways:

- The Icon method, and

- The Menu method.

4.1.1   The Icon Method

- If you have placed the icons on the vertical tool bar to the right of the tree display, you can use the **Icon method**. How to place icons on the vertical toolbar is described in Section 3.1.4.

- The icons are:

    Service

    Service Components

    Data Feeders

    Expressions

The process is the same for each of the four Classes. Left-click on the icon for the Class you want to create and move the cursor to the window to the right and left click again where the **+** cursor stops.

4.1.2   The Menu Method

As with the icon method, the process is the same for each of the Classes:

1. From the **Tools** menu, choose **Service Designer**, another menu will be displayed, which gives you the option **Entity Definition**.

2. Select **Entity Definition**, another menu is displayed giving you the options:

- Define a Service

- Define a Service Component

- Define a Data Feeder

- Define an Expression

The three menus are displayed in Figure 14.

Figure 14   The Definition Menus
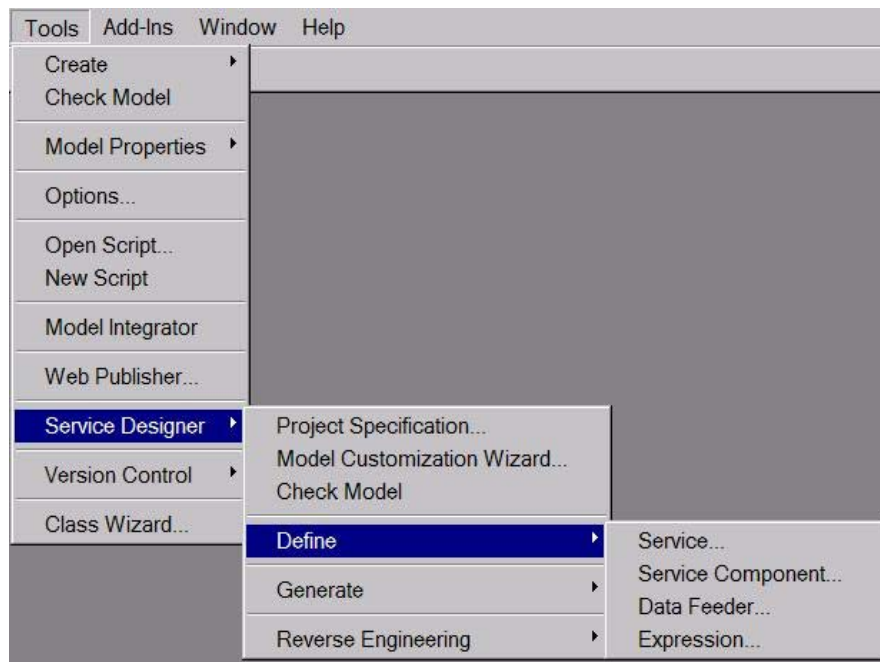
3. Choose the Entity Class you want to create. The graphical representation for the Class will be placed in the **Class Diagram: Logical View\Main** window. You can move it to the position you want, either straight away or later.

When you have created the Service, Service Components and Data Feeder Classes in the Class Diagram View window, the window will look similar to the one illustrated in Figure 15.

Figure 15   Service, Service Component and Data Feeder Classes
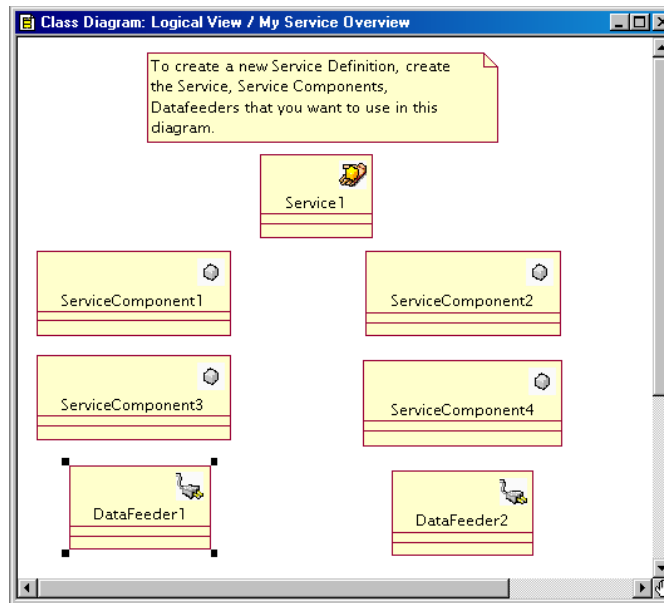
When you have created the Classes, you can move them anywhere you like in this window by selecting a Class with the left mouse button, keeping the button pressed down and moving the object to the position you want. You can do this as many times as you like. There is no restriction to the number of Classes you can create. The number is limited to the needs of your Service.

As the graphical representations, which represent the Classes in the Class Diagram Main window, are created, the model is created and the icons for the new Classes in the model are displayed in the tree window on the left of the screen.

If you have made a mistake and want to delete any item, refer to Section 4.2 for instructions how to do this.

4.2 Deleting Classes From a Model

Although you can remove a **Class representation** from the Class diagram window by clicking on the **Class** and pressing **Delete**, this does **not** delete the Class from the model, it only removes the **graphical representation of the Class** from the display.

You can delete a Class from the model in two ways: the Tree window method and the Icon method.

4.2.1 The Tree Window Method

Select the icon for the Class in the tree window and right click, which will display a menu giving the choice to delete. This is illustrated in Figure 16.

Figure 16 **Deleting Classes From a Model**
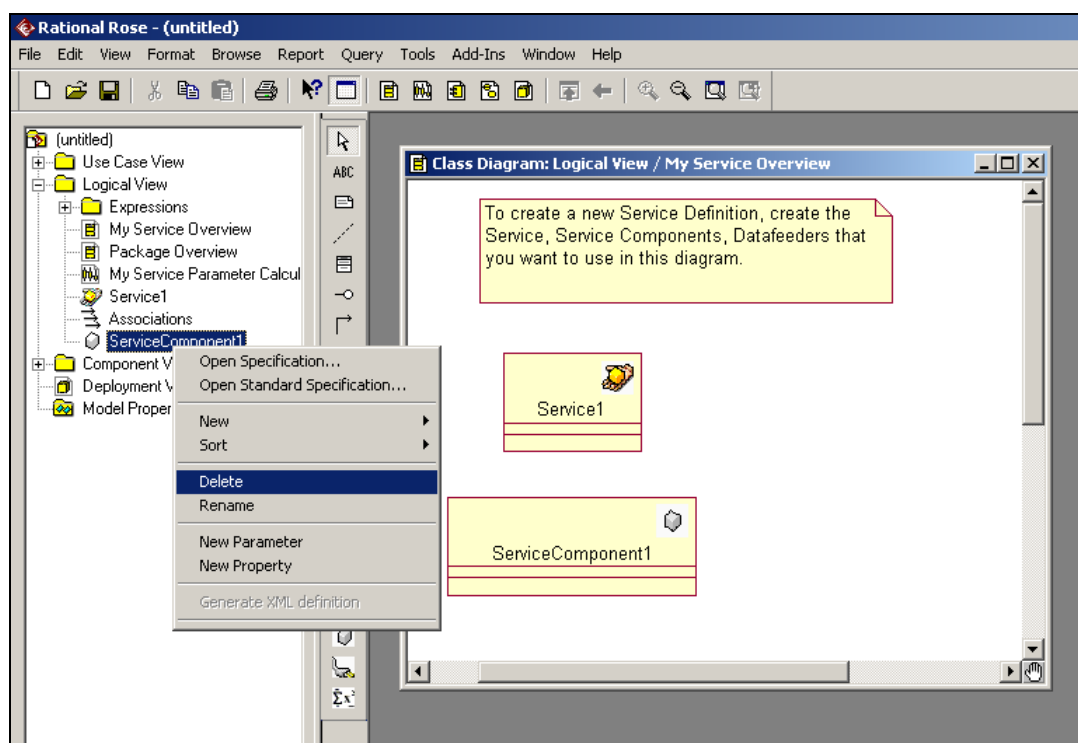
4.2.2 The Class Diagram Window Method

Select the Class in the Class diagram window and press Ctrl/D.

4.3 Copying/Pasting an Object

Rationale Rose does not provide a very intuitive way to duplicate existing object.

If you select an object in class diagram and perform a copy/paste, the selected icon will be duplicated, but in the Rose model, only one object exists.

You can check this in the Logical view tree. Only one object is displayed.

In fact only the representation of the selected object has been duplicated in the Class Diagram.

To duplicate the object you must follow the following steps:

1. Select the object to copy in the class diagram

2. Type ctrl/c to copy it in the clipboard (or Edit then Copy).

3. Change its name (add a character at the end)

4. Type ctrl/v to paste the object (or Edit then Paste).

5. At this stage, you have 2 different objects in the Rose Model. You have to set, now, a name to the newly created object, and restore selected object name.

4.4 Defining and Displaying Specifications

When the Classes are created, they have a default definition. Whichever method you used to create them, you should now define them to your needs.

The procedure is the same for each of the Classes. There are two ways to do this.

You can either click on the Class with the **right** mouse button and choose **Open Specification** from the menu illustrated in Figure 17, or, simply double click on the Class with the left mouse button. The dialog boxes illustrated in sections 4.5, 4.8, and 4.9 will be displayed.

The dialog boxes for Service and Service Component have three tabs: General, Attributes and Parameters. The Data Feeder dialog box has an additional tab: Measure Point.

Figure 17   Definition Options
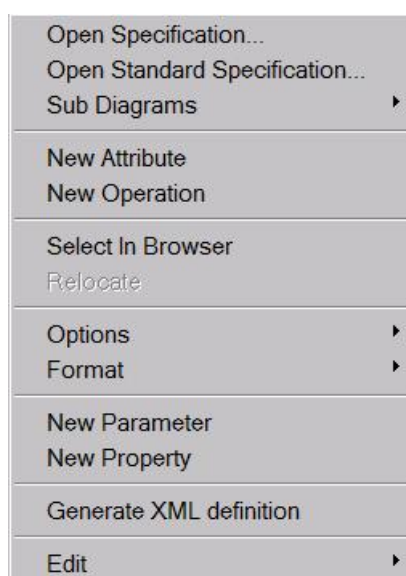
If you choose **Open Specification**, this will display **a dialog box with** the existing specifications for the Class that you had selected**.**

4.5 Service Specifications

Specifying a Service allows you to create a Service, this is the top level of the Service hierarchy.

There are three dialog boxes available to specify a Service. To specify a **Service**, you must fill in **all** the boxes presented.

Here are examples of the three dialog boxes for **Service Specification**.

With each of these three dialog boxes, Click on **OK** when you have finished and are satisfied with what you have entered or on **Cancel** if you want to cancel your entries.

Figure 18   Service Specification Dialog Box, General Tab
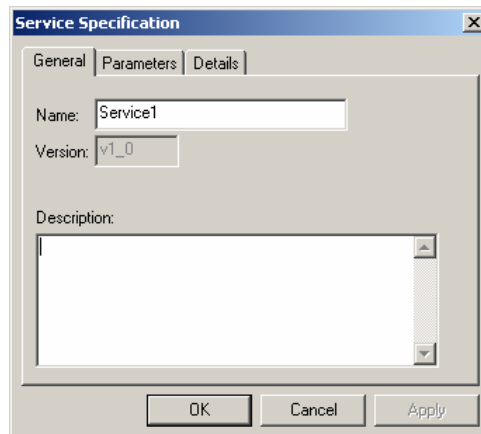
4.5.1 The General Tab

In the **General** Tab, the Version number is displayed. For a Service or Service Component, the Version number has a purely informative role, and is not used in SQM V1.0. For Datafeeders the version can be changed.

You must type a name for the Service.

In the Description box, you can enter free text.

4.5.2 The Parameters Tab

Service Parameters and properties are displayed in this tab with their Name, Type and Stereotype (i.e. Parameter or Property). How to define them is described in Sections 4.6.1 and 4.7.

Services, Service Components and Data Feeders **must** have Parameters and Properties.

Parameters and **Properties** are types of Attributes of a Service, Service Component or Data Feeder. All **Parameters**, with the exception of Data Feeder Parameters must have a **Calculation Expression** associated to them. A **Property** does not have an expression linked to it, as it is a fixed value.

The parameters and properties for a **Service Component** and for a **Data Feeder** are defined in the same way.

Whereas **Inheritanc**e is supported in Service Designer, conflicts are detected, but not resolved. When defining parameters for inherited entities, care should be taken that the parameter name does not conflict with that of the entity from which they were inherited.

Figure 19   The Service Specification Dialog Box, Parameters Tab
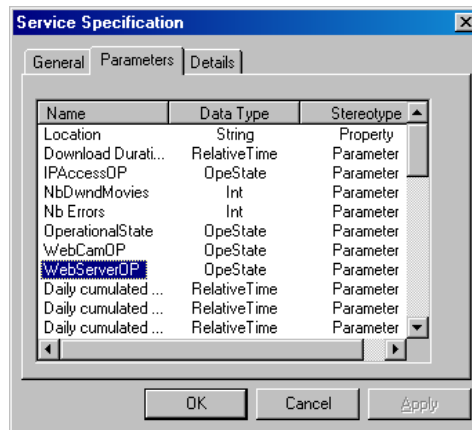
4.5.3   The Details Tab

In the Details Tab of the dialog box, the Customer Dependant option box is disabled. This feature, which allows you to specify whether the service is available to selected customers, is not available in V1.0.

A default Identifier is supplied by Service Designer, but it is recommended that you enter an Identifier, which is significant to you.

Once a Service Definition has been generated and loaded into SQM Repository it is highly undesirable to change the identifiers of all the objects. The Identifiers must remain the same from one generation to another.

For details about sampling related function, refer to the *OpenView Service Quality Manager Sampling Scheduling Guide*.

Note

The Identifiers must not exceed 16 characters and must not contain non-printable characters: tabs, spaces or the characters stop (**.**), asterisk (*****), comma (**,**) or right arrow (**>**).

Figure 20   **The Service Specification Dialog Box, Details Tab**
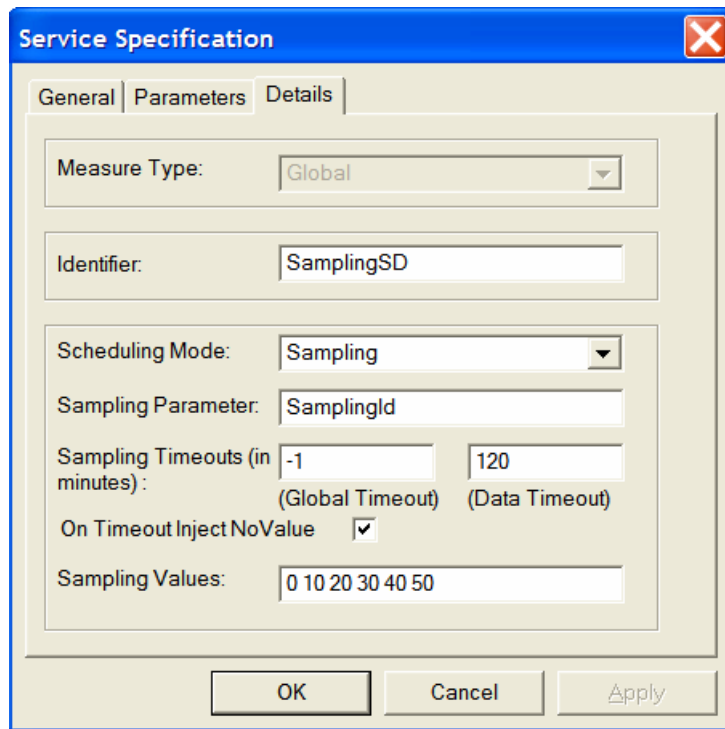
4.6 Defining Parameters

4.6.1 Creating Parameters

Important

There are two methods of defining (or modifying) Parameters.

Whichever method is chosen, Parameters must **always** be changed using the Service Designer dialog boxes. If changing a Parameter without using these dialog boxes, the associated Operation **will not be updated** and the use of the Parameter will be altered. Care should be taken to **never** delete or change a Parameter Name in the Logical View Tree.

To define parameters:

Method One

The Contextual menus provide access to the Parameter creation Dialog Boxes.

Method Two

1. In the Class Diagram window, select the Class for which you want to define Parameters.

2. Click the Class with the right mouse button and choose **New Parameter** from the menu that is displayed.
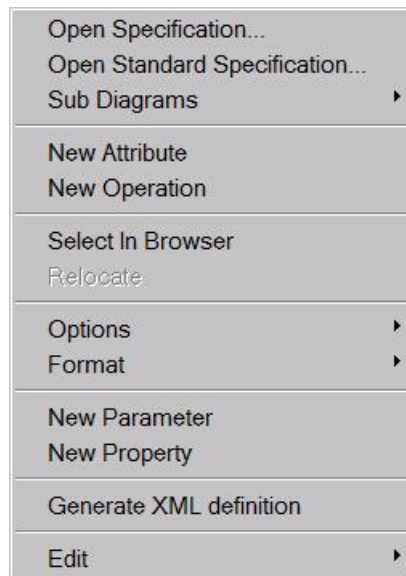
3. This will add a new parameter item to the center section of the Service Class:

Figure 21   New Parameter Item Added to Service
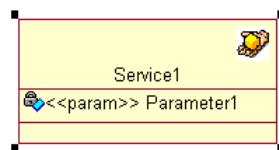
Note

By default, the operations associated to the parameters are displayed in the Class representation. These operations are for internal use only, and it's advised to hide them. To do that, right click on the class representation and select **Options/Suppress Operations**

Figure 22   Operations suppressed from class representation
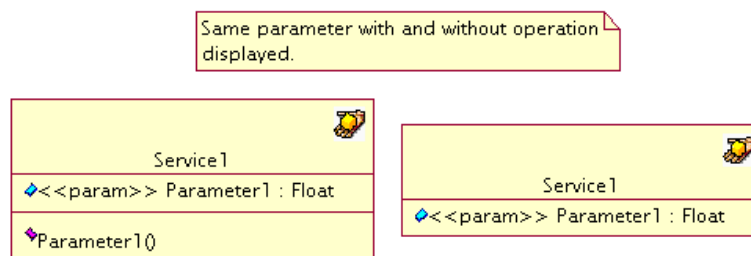
4.6.2   Defining Parameter Specifications

1. Open the Service Specification Dialog box.

2. Click on the **Parameters Tab** of the Service Specification Dialog box.

3. Double click on the Parameter entry.

This will display the Parameter Specification dialog box, which like the Service Specification dialog box has two tabs: the **General Tab** and **Details Tab**.

Figure 23   Parameter Specification Dialog Box, General Tab
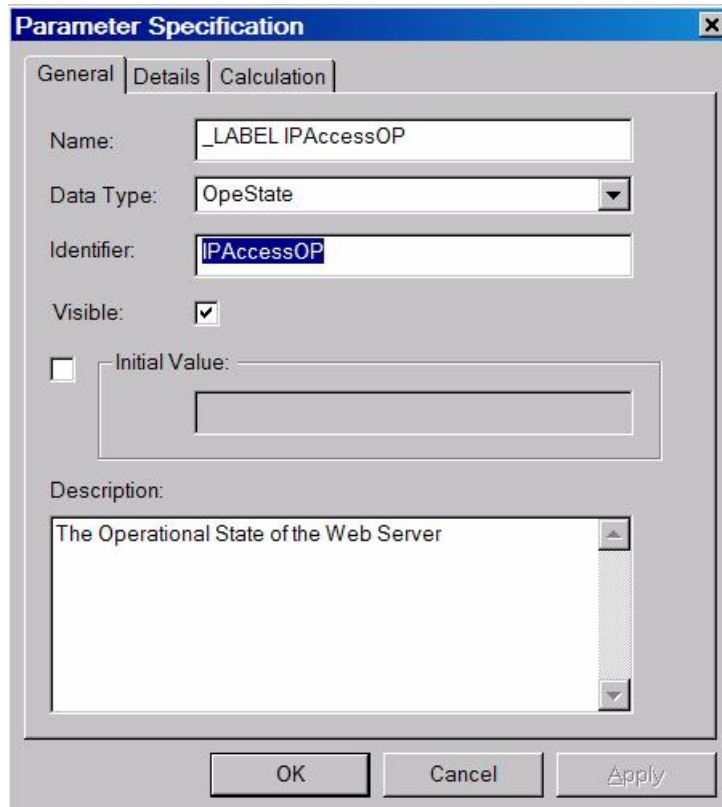
4.6.3 Parameter Specification General Tab

4.6.3.1 Name and Identifier

Text fields allow setting a **Name** and an **Identifier** to the Parameter.

Service Designer supplies a default Name and Identifier, but it is recommended to update them with meaningful values.

Note

The Identifier must not exceed 12 characters for Parameters and 16 characters for Properties. It must not contain spaces, tabs or non-printable characters and must not contain the characters: stop (**.**), comma (**,**), asterisk (*****), or right arrow (**>**).

There is no restriction for the Label, but we recommend that it should not be too long, as this label name will be displayed in all the **OpenView Service Quality Manager** User interfaces.

4.6.3.2 Description

The **Description** field allows entering free text for further describing the parameter.

4.6.3.3 Parameter Type

The parameter **Type** is set using the drop-down list in the Type field. The allowed Types are:

- **AbsoluteTime**: The time, given in GMT notation (YYYY-MM-DD hh:mm:ss for example: 2002-05-16 15:20:12).

- **(Real) Float**: a signed float.

- **Int**: a signed, 64-byte integer.

- **RelativeTime**: number of milliseconds.

- **String**: a printable string.

- Any **Enumeration** defined in the model. (Refer to Section 4.13 for details about defining Enumerations.)

4.6.3.4 Parameter Visibility

Parameters can have one of two visibility levels:

- Visible (Public): ✔ ◆
  The Parameter is visible. It will be displayed in the other GUIs and it is possible to define an objective for this parameter. (This is the default setting and it is always displayed checked until you choose another option.

- Not Visible (Private): 🔑◆, (Protected): 🔒◆
  The Parameter is not visible. It is a parameter used internally to compute other parameters. It will not be displayed in other GUIs and it is not possible to define objectives for this parameter.

Parameter visibility is set by using the **Visible** checkbox in the Specification window or by clicking on the ◆ icon to the left of the Parameter in the Class diagram. This displays a menu with four icons. The default value is 'Visible' ✔ ◆. Only the options **Visible** and **Private/Protected** are used in Service Designer.

Figure 24   Visibility Icons
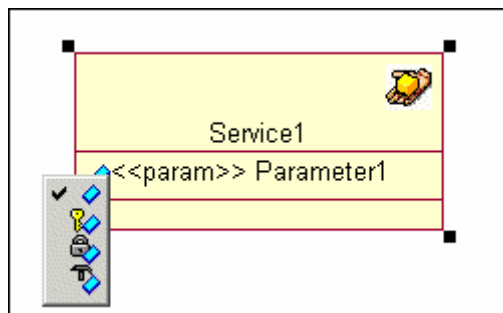
4.6.4 Parameter Specification Details Tab

The information you can define depends on the Entity to which the Parameter belongs. Some fields are read-only.

Figure 25   Parameter Specification Dialog Box, Details Tab
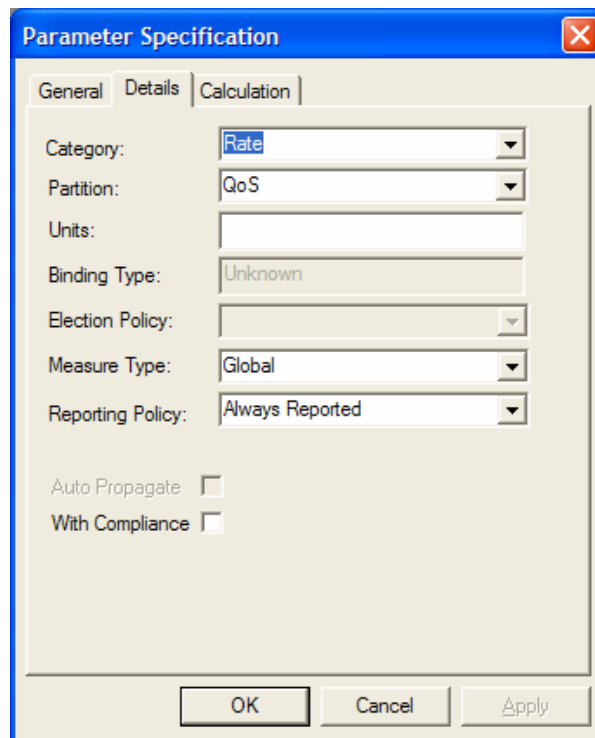
The behaviors of the Parameter inside OV SQM are defined in this tab.

4.6.4.1   Category

The **Category** is the **type** of measure. It determines how the measured parameter is presented graphically in the user interfaces. The choices are:

- Counter:
    Measures in figures between pre-defined levels.

- Gauge:
    Shows the measurement as a gauge.

- Other:
    Any measurement type that is not specified in the options offered.

- Percent:
    Measurement shown as a percentage

- Rate:
    A quantity (such as speed) measured with respect to another measured quantity (such as time).

The **Category** also determines how the parameter values are aggregated in the DataMart. Parameters of type counter are aggregated using the sum function: for each period, the sum of all received values is computed. For all other category, the DataMart computes the min, max and average values.

4.6.4.2   Partition

Parameter Partition is the fundamental type of the measure represented by the parameter:

- QoS:
    Quality of Service

- Char:
  The Parameter describes the characteristics of the Service or Service Component, (for example: the location or the address).

- Other:
  Any category not covered by the options offered.

- State:
  The Parameter describes a state of the Service or Service Component.

- Usage:
  The Parameter describes how the Service or Service Component is used, (for example: the number of hits).

4.6.4.3 Binding Type

The Binding Type cannot be modified. Service Designer automatically computes it as it is deduced from other model elements. The Binding type is displayed for information in a read-only mode. It can take one of the following values:

- Collected:
  The Parameter is associated with values from Data Feeder parameters.

- Computed:
  The Parameter value is computed from other Service Parameter values.

- Unknown:
  This type is only displayed when the parameter has not yet been bound and Service Designer has not yet computed the Binding Type.

The Binding type partly determines whether or not the Election Policy field is available.

See Sections 4.12.1 and 4.12.2 for further details about binding types.

4.6.4.4 Units

This is a free text field for specifying the Parameter Units (e.g.: Packets/s, ^{0}F, etc.). This data is used when displaying the parameter value in the various UIs.

4.6.4.5 Measure Type

- The **Measure Type** field specifies whether the Parameter value is **Customer Dependant**. Two choices are allowed:

- **Customer Related**: the Parameter value is relative to a given Customer. The Parameter value is different for each Customer.

- **Global**: the Parameter value is not tied Customers but shared between them.

4.6.4.6 Auto Propagate

The **Auto Propagate** function can **only** be applied to **primary** parameters. If it is set, the parameter value is immediately and automatically sent to Service Level Monitoring (SLM).

Parameter objectives are calculated on periodic basis (by default every 5 minutes). For some parameters, it can be useful for their values to be updated in real-time.

The Auto Propagate flag allows you to monitor the parameter values in real-time.

To activate Auto Propagation for a Parameter, check the **Auto** Propagate checkbox.

4.6.4.7 With Compliance

This checkbox allows you to activate the computation of the compliance value for the
parameter. Compliance can only be computed, and objectives associated to it if **With
Compliance** is checked.

4.6.4.8 Election Policy

This feature is typically used when the values of measured parameters are collected
more frequently than the calculation period specified in OpenView Service Quality
Manager. When this is the case, several values are available at calculation time, but
only one must be used as input for expression calculations. The Election Policy
allows you to specify how to elect a value from the several values collected.

The **last value collected**, and the **average of the values collected** are the two most
common algorithms chosen for the election policy.

To be able to set the **Election Policy** the Election Policy drop-down list must be
enabled.

For the Election Policy drop-down list to be enabled, the parameter must fill certain
criteria:

- It must be defined in a Sequence Diagram

- Its Binding Type must be Collected.

If these conditions are met, the list will contain all the PL/SQL aggregation
Expressions that match the type of the considered parameter and you can choose the
aggregation Expression you want from the drop-down list. This is illustrated in Figure
26.

Figure 26 Parameter Specification Dialog Box, Details Tab, Showing Binding Type: Primary, Election Policy Enabled
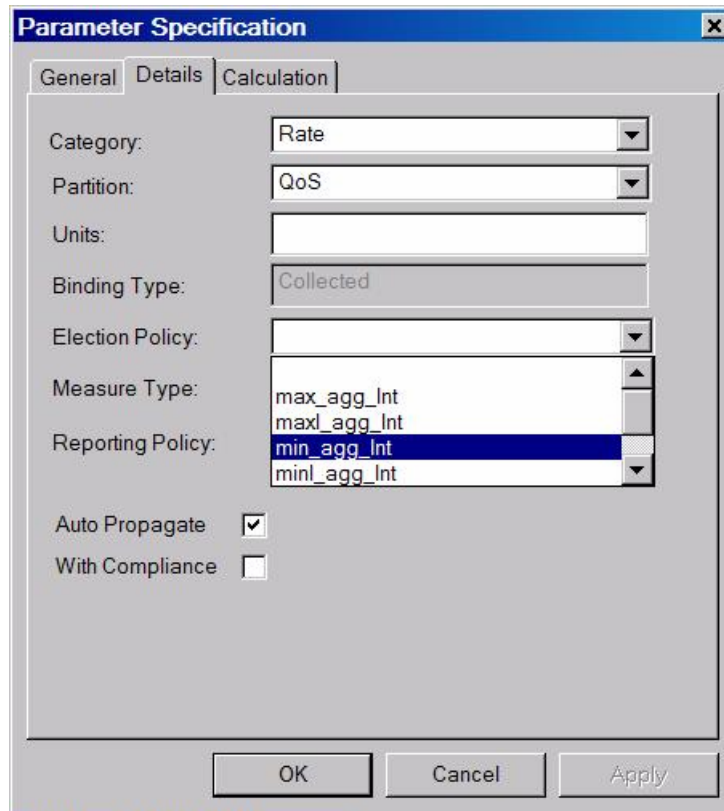
If these criteria are not met, the Election Policy field is not enabled.

4.6.4.9   Reporting Policy

The Reporting Policy is only relevant for Service Component Parameters. This Parameter characteristic defines the behavior of the reporting tools against this parameter.

Three different levels can be assigned:

- Always Reported (Default): this parameter is staged/summarized by reporting tools in all cases.

- Never Reported: reporting tools ignore the parameter. This is useful for intermediary type parameters or parameters for which only the instantaneous value is meaningful.

- Disabled When Late: this identifies components for which measures are only meaningful when published in short delays. Those kinds of components/measures are outdated quickly and are bad candidate for a real reporting. The monitoring should be the preferred tool. The parameter is taken into account by the reporting but dropped when machine load is heavy.

4.6.5   Parameter Specification Calculation Tab

Figure 27   **Parameter Specification Dialog Box, Calculation Tab, Showing Calculation description and Time Aggregation properties**
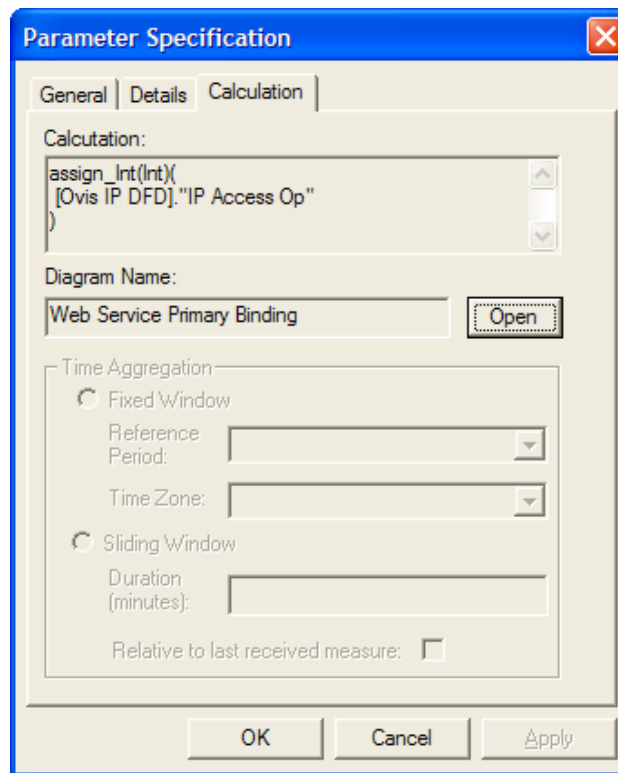
The **Calculation** field displays a textual description of how the parameter is computed. See section 4.12.3 for further details about defining parameter calculation.

The **Diagram Name** field displays the name of the sequence diagram where the parameter calculation is defined. When the user clicks on the **Open** button, the related sequence diagram (if any) is opened.

The **Calculation** and **Diagram Name** fields are read-only.

The **Time Aggregation** section allows setting the Time Aggregation properties for the parameter. This section is available only if it is possible to define a time aggregation for the parameter, that is: if an aggregation expression is used to compute the parameter AND the input parameters come from the same Service Component as the output parameter. In all other cases, the section is disabled.

Figure 28   Enable Time Aggregation for a parameter
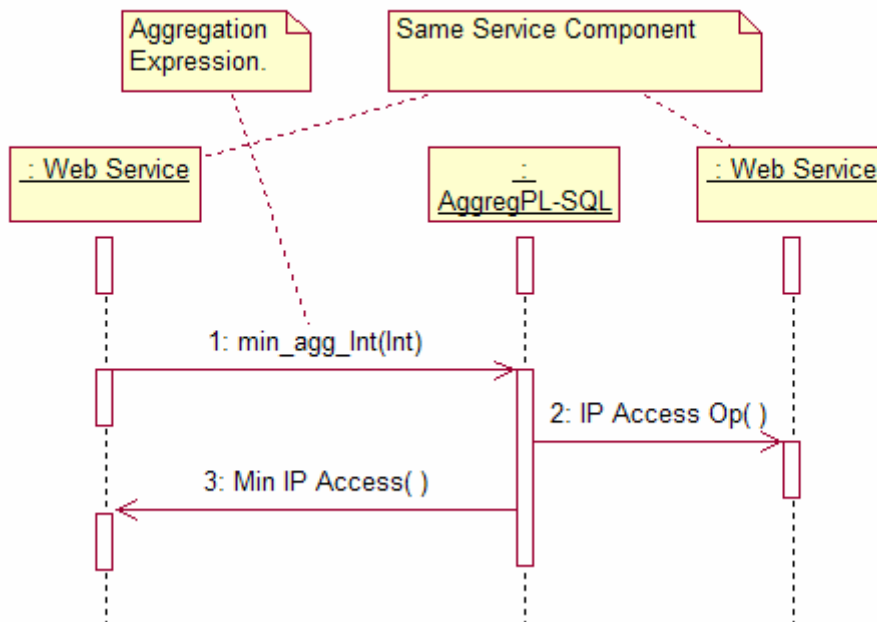
In the example above, SQM will compute the parameter **Min IP Access** as the minimum of **IP access Op** over a specified period of time defined in the window.

SQM manages two kinds of time window:

- Sliding Window Time Aggregation: the window has a constant duration and slides for each calculation.

- Fixed Window Time Aggregation: the window starts at a precise moment (start of a day for example) and the duration increase for each calculation until the window is reset.

Figure 29   Selected input for sliding and fixed window time aggregation
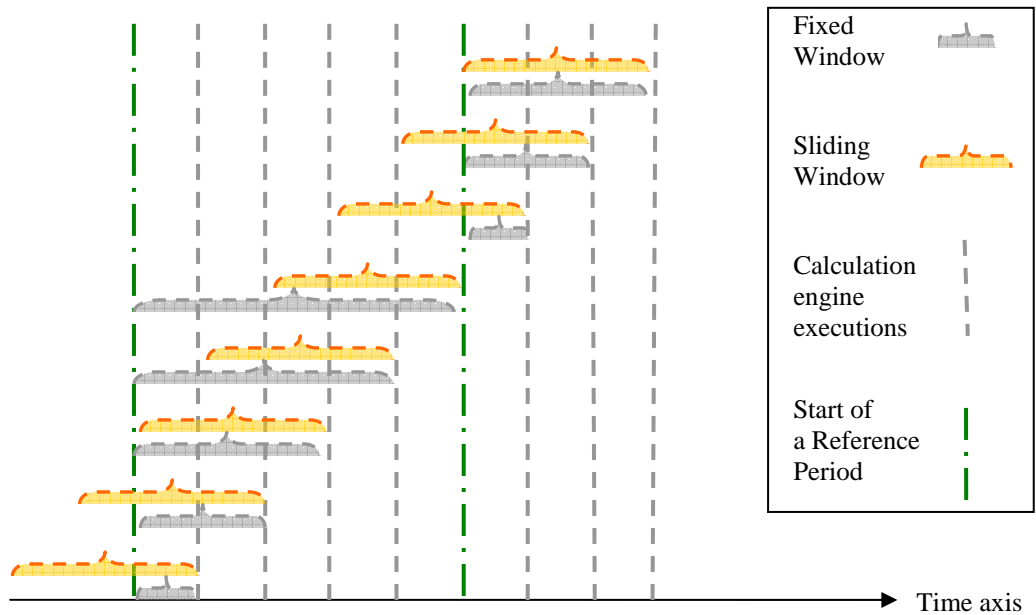
The figure shows for several engine executions which input data is selected to compute the time aggregation.

4.6.5.1   Defining a Sliding Window Time Aggregation

By clicking on the **Sliding** radio button the user can specify the **Duration** of the fixed time window (in minutes). The duration of the sliding window must not be greater than the SPDM retention delay (7 days by default).

The **Relative to last received measure** check box determines if the sliding time window ends at the timestamp of the last received measure or not.

When the Relative to last received measure box is not ticked, then the calculation engine will consider the values received within the last duration period.

Figure 30   Sliding window time aggregation.

Figure 31   Relative to last received measure check box
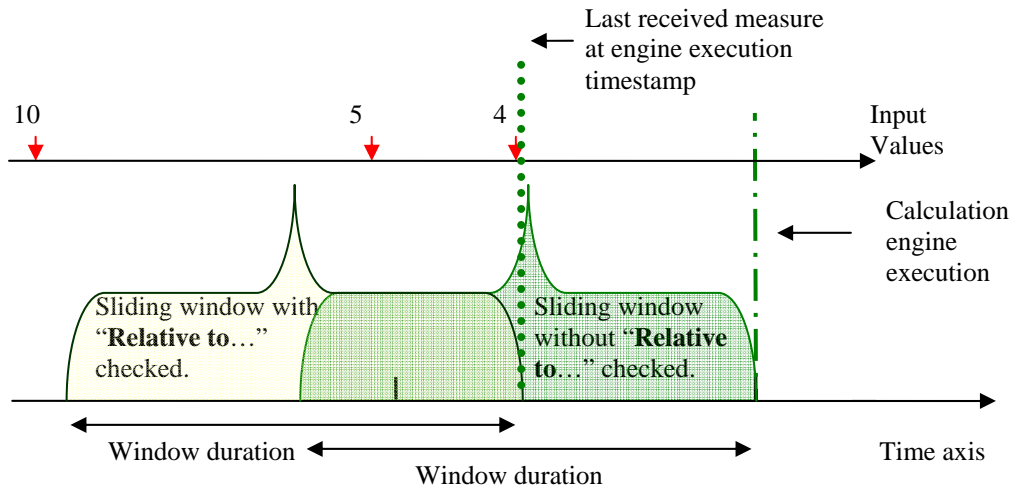
4.6.5.2   Defining a Fixed Window Time Aggregation

To define a Fixed Time Aggregation, the user has to click on the **Fixed Window** radio button. The **Reference period can be set** with the list box.

Possible values are: 1 hour, 2 hours, 3 hours, 4 hours, 6 hours, 8 hours, 12 hours, or 1 day.

The **Time Zone must also be defined** to determine the beginning of the reference period.

Note

In the current version, the possible reference periods are limited to the values listed above.

Figure 32   Fixed Window Time Aggregation
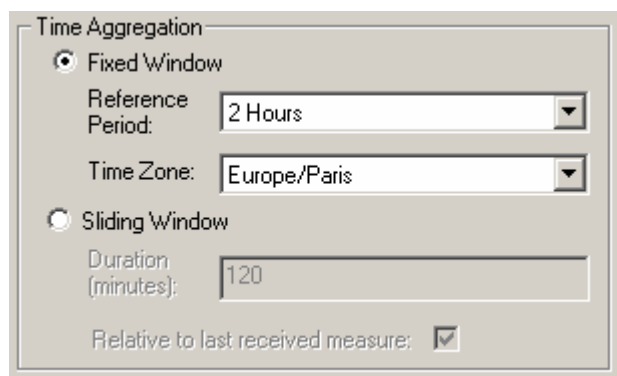
In this example the day is divided in 12 periods of 2 hours. Europe/Paris Time zone is used to determine the beginning of a day.

4.6.5.3   Custom expression for Time Aggregation

Any PL/SQL aggregation expression can be used for Time Aggregation parameters, but these expressions are not specially designed to aggregate over time. For example,

the avg_agg_Float will perform an average of all received values without taking in account the timestamps of the measure and this could lead to inaccurate results.

To handle time aggregation more accurately, a new data type "measure" can be used inside PL/SQL code. Also the usage of a "context" object in PL/SQL code provides some helper to write the suitable expression.

Note

The new type and context object are not visible in the Service Designer. You still have to use standard SQM Type (Float, Int, String…) in your model. Only the PL/SQL code is impacted.

We will explain these new objects based on a simple example: the time_avg_Float function.

We want that the calculated average takes the timestamp of the measure into account.

If no values are collected, the average is equal to the last received value.

Here is the code of such a function.

```sql
CREATE OR REPLACE FUNCTION Time_Avg_Float(input_v
measure_coll[1]) RETURN NUMBER IS
  -- helper context always named <function name>_ctx
  ctxt            time_avg_Float_ctx[3];

  summ            NUMBER :=0 ;
  duration_v      NUMBER ;
  total_duration NUMBER := 0;
  next_value_m    measure[2];
  -- must be compatible with input parameter
  value_v    NUMBER;
  prev_value_m measure[2];
  i                INTEGER;
BEGIN
  -- Do this call to initialize helper context
  Context.get_context(ctxt) [3];

  -- ---------------------------------------
  -- FIND value at start of period.
  -- ---------------------------------------
  ctxt.last_input_v(prev_value_m) [3];

  -- ---------------------------------------
  -- ACCUMULATION ALGO
  -- ---------------------------------------

  IF input_v IS NOT NULL AND input_v.COUNT > 0 THEN
    FOR i IN input_v.FIRST..input_v.LAST [1]
    LOOP

      -- get next values
      next_value_m := input_v(i) [1];
      prev_value_m.value(value_v) [2];
      duration_v:= Tools.reltime_to_millis(next_value_m.tstamp
- prev_value_m.tstamp) [4];
      summ:= summ + value_v * duration_v;
      total_duration:= total_duration +duration_v;
   prev_value_m := input_v(i);


        END LOOP;
      END IF;

      IF total_duration > 0 THEN
        RETURN summ/ total_duration ;
      ELSE
        prev_value_m.value(value_v);
        RETURN  value_v;
      END IF;
END ;
/
```

Apart from the algorithm use to compute the time average, the important point to notice, specific to time aggregation are:

1. The new datatype is "**measure_coll**". It represents a collection of "measure" object (see 2).
   When the SPDM calls your custom function, the input collections are filled with the values belonging to the current time aggregation window. These values are ordered by ascending timestamp.
   You can access individual measures as any other oracle collection (FOR i IN input_v.FIRST..input_v.LAST   … next_value_m := input_v(i);)

2. A "measure" object is the association of a value and a timestamp. By convention, the "NoValue" are represented by the NULL oracle value. You can access the measure value by calling the "value" method (prev_value_m.**value**(value_v)). When you do this, you must ensure that the parameter passed to the "value" method is compatible with SQM datatype of the related input.
You can also access the timestamp of the measure through the "tstamp" attribute (next_value_m.**tstamp**). The timestamp are represented in the GMT timezone.

3. In some cases, you need additional information to perform your aggregation. In our example, we have to know the parameter value (and timestamp) before the first value of the collection. These information that are not linked to the input collections are stored in a "context" object.
For each custom expression, SQM creates a specific datatype named "**<function_name>_ctx**". Before calling the custom expression, a context is initialized by SQM and you can access several useful information.
For each input parameter of the function, the context gives access to the last measure before the collection through a methode named "**last_<input_parameter_name>**" (eg: ctxt.last_input_v(prev_value_m)). You can also access the last value of the output with the method "current_output_measure". This method is not used in our example but you can use the following statement if needed: **current_output_measure** (last_output_m).
Some other attributes related to the aggregation windows are also available: **offset** the time-zone offset in case of fixed time aggregation (NUMBER):, **start_ref_period** and **end_ref_period**: the window bound (GMT TIMESTAMP)

4. Custom expressions are validated by the SRM and the SPDM, thus you must be careful if you use external functions in your code. In SQM V1.3, you can only access external functions provided in standard oracle package and by the SPDM tool package (eg: TOOLS.reltime_to_millis).

Note

Like any other expression, you can have several input parameters and you can mix measure_coll and other "_coll" type.

4.7 Defining Service Properties

Service Properties are fixed properties of the Service.

To define them:

1. Double click on the Property in the Parameters Tab of the Service Specification dialog box.

2. This will display the **Property Specification** dialog box.

Figure 33   **Property Specification Dialog box**
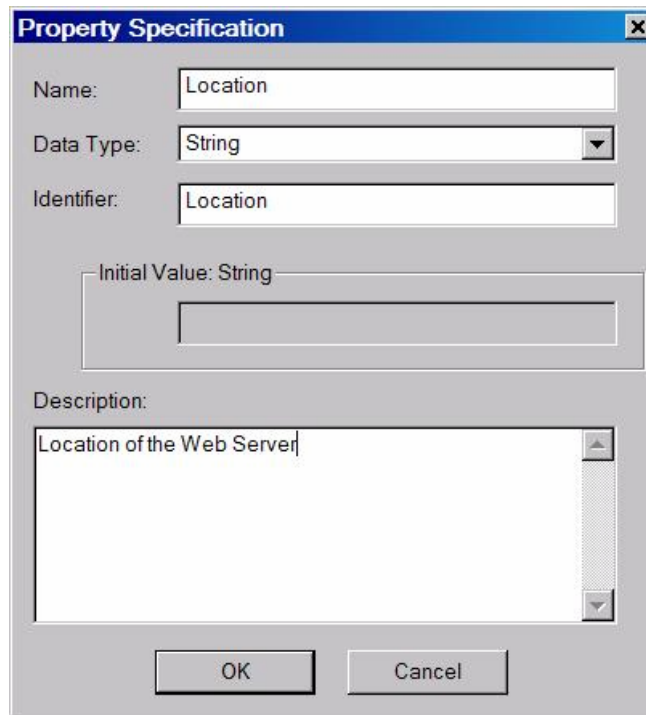
From this box you can define:

- The Property **Type**, which can be:

    - **AbsoluteTime**: The time, given in GMT notation (YYYY-MM-DD hh:mm:ss for example: 2002-05-16 15:20:12).

    - **(Real) Float**: a signed float.

    - **Int**: a signed, 64-byte integer.

    - **RelativeTime**: number of milliseconds.

    - **String**: a printable string.

    - Any **Enumeration** defined in the model. (Refer to Section 4.13 for details about defining Enumerations.)

- The Property **Name**

- The Property **Identifier**

- And you can enter free text to describe the property.

From the illustration you can see a small part of the log displayed, indicating that model properties were updated.

4.8 Service Component Specification

The **Service Component Specification** dialog box has three tabs, General, Attributes and Details.

- In the **General** dialog box, a Name for the Service Component is already displayed and you can change this to the name you want.

- You can enter free text in the Documentation box.

Figure 34   The Service Component Specification Dialog Box, General Tab
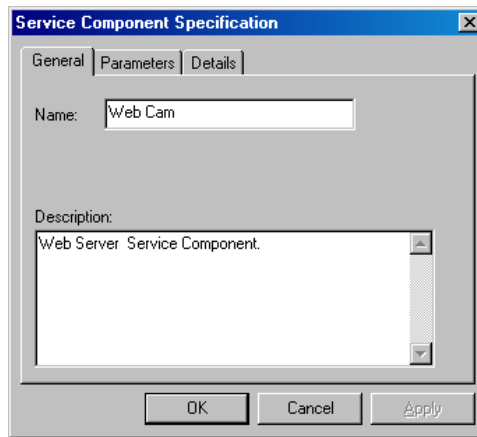
On the **Parameters** dialog box the Stereotype (Property or Parameter), the Type (datatype) and the Name will be displayed if they have already been defined.

Figure 35   Service Component Specification Dialog Box, Parameters Tab
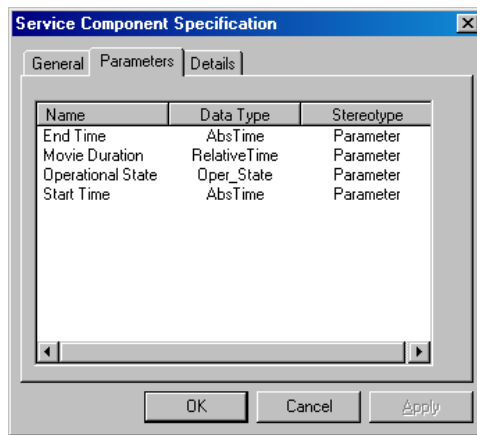
Service Component Parameters and **Properties** are defined in the same way as the Parameters and Properties of a Service.

Figure 36   The Service Component Specification Dialog Box, Details Tab
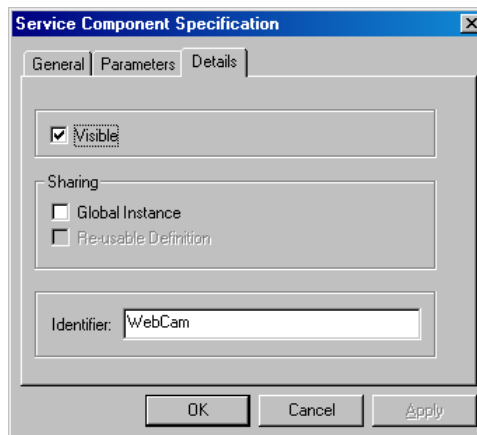
In the Details dialog box, you have the opportunity to make the Service Component visible or not, and to define the Sharing parameters. The Shared Definition flag is automatically set to **True** when the Service Component is imported from a Rose Controlled Unit, or when the Shared Global Instance flag is set to **True**, otherwise it is set to **False**.

The **Identifier** is displayed in the bottom field. It is recommended to change this name but it must remain the same from one generation of an XML definition to another. If it is changed, it can lead to inconsistencies in the model.

4.9 Data Feeder Specification

Data Feeders must be defined to establish how primary parameters are fed.

The Service Designer add in allows to define manually a new Data feeder, but usually, Data Feeders are generated by the Service Adapters tools, and imported into the Rose Model to be used in a Service. This can be done by using the Reverse Engineering feature or by importing control unit.

A Data Feeder Definition (DFD) corresponds to the logical description of parameters that are exposed by a given Data Feeder. It is characterized by:

- A Version,

- A set of DFD Properties,

- A set of DFD Parameters (that is, the parameters used to build primary Service Parameters)

- The Measurement Reference Point (MRP) naming scheme. This is the formal description of how the measurement point name is built, that is, by concatenating the values of Data Feeder properties and fixed strings.

The **Data Feeder Specification** dialog box has four tabs, General, Parameters, Details and Measure Point.

With each of these four dialog boxes, Click on **OK** when you have finished and are satisfied with what you have entered or on **Cancel** if you want to cancel your entries.

In the **General** dialog box, A Version number is displayed, but this can be changed if needed. The role of the Version in a Data Feeder Definition is different from its role for a Service or Service Component definition. For a Data Feeder Definition it is part of the Data Feeder Definition Identifier. There can be more than one Data Feeder Definition with the same Name, but a different Version number in the same model. (For a Service or Service Component, the Version is not used; it has a purely informative role). The Data Feeder default name is displayed and you can edit this to the name that you need. In the **Description** box, you can enter free text.

Figure 37   The Data Feeder Specification Dialog Box, General Tab
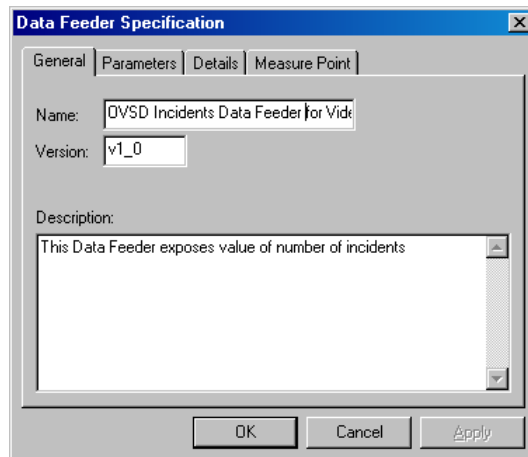
In the **Parameters** dialog box, the Stereotype and the Name are displayed, as well as the Type if it has been specified.

Figure 38   The Data Feeder Specification Dialog Box, Parameters Tab
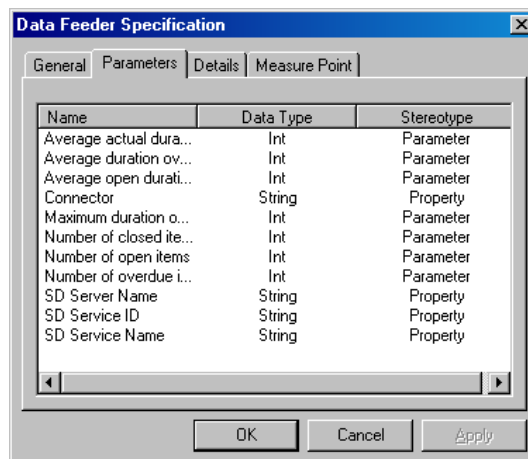
The Details Tab dialog box gives you the possibility of defining the Name of the Service Adapter (SA) that will be in charge of this Data Feeder. The version Number is displayed, but you can change this.

The **Identifier** is displayed in the bottom field. It is recommended to change this name but it must remain the same from one generation of an XML definition to another. If it is changed, it can lead to inconsistencies in the model.

Figure 39   The Data Feeder Specification Dialog Box, Details Tab
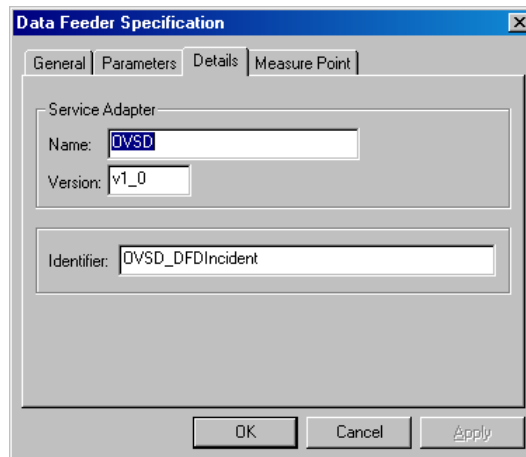
The **Measure Point Tab** dialog box gives you the possibility of setting the Measurement Reference Point (MRP) naming schema that will be used for the Data Feeder instantiation.

MRP Naming Schema describes (or define) a set of common rules for the naming of Data Feeder instances. The MRP naming schema is the formal description of how the measurement point name is built, that is, by concatenating the values of Data Feeder properties and fixed strings. An example is shown in

The MRP indicates from where the measurement is performed and indicates more precisely what is measured.

Example of an MRP Naming Schema

```
Location <Location Property value> Instance <Instance Property Value>
```

Here are some examples of a Data feeder instance name based on this MRP Schema:

- Location_Paris_Instance_1

- Location_Paris_Instance_2

- Location_Lyon_Instance_1

You would create the example above in this way:

1. Type your fixed string in the fixed string box and use the **right double arrow >> button** to move it to the empty, unlabeled box on the right. In the example above this is: `location`.

2. Then choose the Property you want from the drop-down list of the property field and use the **right double arrow >> button** to move it to the box on the right. In the example it is: `property "Location"`

3. Type the fixed string: `instance,` and move it to the right-hand window as before.

4. Then choose the Property you want from the drop-down list of the property field and move it to the right-hand box as before.
(in the example it is: `property.name = "PatrolInstance"`)

5. Type the fixed string: `end` and move it to the right-hand window as before.

6. Click on **OK**.

If you make a mistake, or if you want to change the order, select each item separately in the right-hand box, and use the **left double arrow << button** to remove them. You cannot move the items up or down in the right-hand box. For example, if you select a

property from the property list **before** you have typed your fixed string, you cannot add your 'start' fixed string **above** the property. Each item is added **below** the previously added item.

Figure 40   **Data Feeder Specification Dialog Box, Naming Schema Tab**
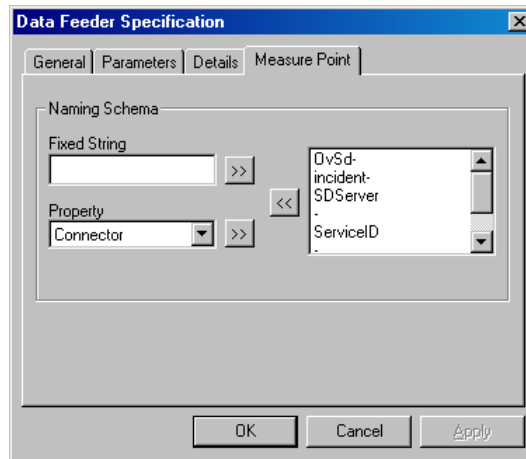
The right-hand fields of the Naming Schema Tab are automatically updated when you delete a Property on the Attributes Tab or modify its name.

Data Feeder Parameters and Properties are defined in the same way as the Parameters and Properties of a Service are defined.

4.10 Defining the Associations or Aggregations of a Service

When you have specified all your Class definitions, you must define the associations or aggregations between them.

An Association represents the ability of one instance to be able to send messages to another instance. An Aggregation is the same as an Association except that instances cannot have cyclic aggregation relationships (a part cannot contain its whole). Therefore, in Service Designer, the association between a Service and a Service Component, or the association between a Service Component and another Service Component, is an Aggregation.

Both **Associations** and **Aggregations** can be created using the Menu Method as well as the Icon Method.

Warning or Information messages are displayed in the log window reminding you of the correct definition of the associations and aggregations if you create the wrong one.

4.10.1 The Menu Method

1. Select **Create** from the **Tools** menu.

This will display the **Create menu**, illustrated in Figure 41.

Figure 41   The Create Menu


```
Text
Note
Note Anchor
Class
Parameterized Class
Interface
Actor
Use Case
Association
Unidirectional Association
Aggregation
Unidirectional Aggregation
Association Class
Generalization
Dependency or Instantiates
Realize
Package

Instantiated Class
Class Utility
Parameterized Class Utility
Instantiated Class Utility
```
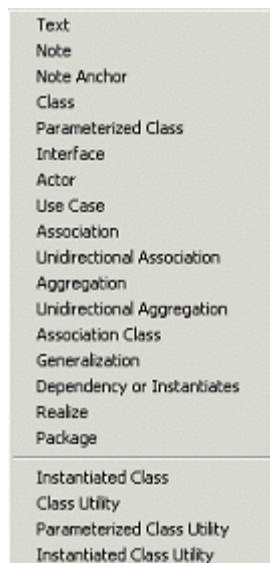
2. Select **Association/Aggregation** or **Unidirectional Association/Unidirectional Aggregation** as appropriate and draw the associations/aggregations directly in the Class Diagram window. The rules that control the type of association or aggregation are detailed in 4.10.3 and 4.10.5.

4.10.2 The Icon Method

The vertical toolbar to the right of the tree window provides icons that you can use to create Unidirectional Associations and Aggregations. You can configure this toolbar to include icons as described in Section 3.1.4.

These Icons are:

- Unidirectional Association

- Unidirectional Aggregation

When the icons are placed on the vertical toolbar, select the icon for the association or aggregation that you want to create, and draw the relationship directly in the Class diagram window. You are not **prevented** from making the wrong type of Association or Aggregation, either of type or direction, but if you do, warning or information messages are displayed in the log window or in a pop-up box. **The error message does not indicate that the Aggregation will be automatically destroyed; you must correct the error yourself.**

The pop-up error message below is the result of drawing an association from a Service Component to a Data Feeder, instead of the other direction.
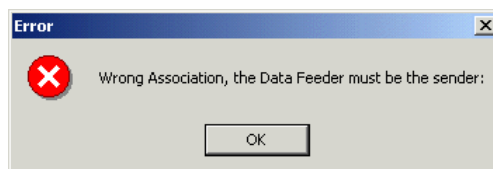
Here is an example of the creation of an incorrect association. A Unidirectional Aggregation was drawn from a Service Component to a Data Feeder.
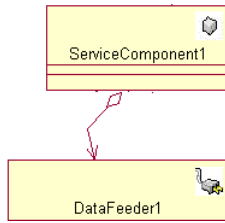
Important: Correcting Association or Aggregation Errors

Whichever method you use to create the associations or aggregations, **if you make an error, you must destroy the line indicating the association or aggregation, and re-create it correctly.**

You can destroy it as described in the same way as removing Classes from the model described in Section 4.2.

4.10.3 Service to Service Component

This association represents the fact that a Service Component is part of a Service. The association from **Service** to **Service Component** and the association from one **Service Components** to another must always be an **Unidirectional Aggregation**. This is illustrated in Figure 42

Figure 42   Part of a Model showing Unidirectional Aggregation
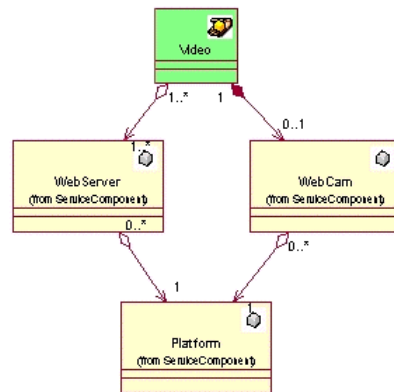
If you make an incorrect association, a Warning is displayed in the log when you create the association indicating which type of association or aggregation you must make. You are not prevented from making the wrong type of association, and the application does not make the correction for you. If you make an incorrect association, you must delete it and re-create it correctly.

4.10.4 Setting the Multiplicity for Associations of a Service

Associations for Components and Data Feeders of a Service must be set, but you do not set Multiplicity for the association of a Data Feeder to a Service Component.

You must set the Multiplicity for both Parent and Child components of the Association. Figure 43 illustrates the function of the Multiplicity setting in both Parent and Child components.

Figure 43   **Parent and Child Multiplicity**
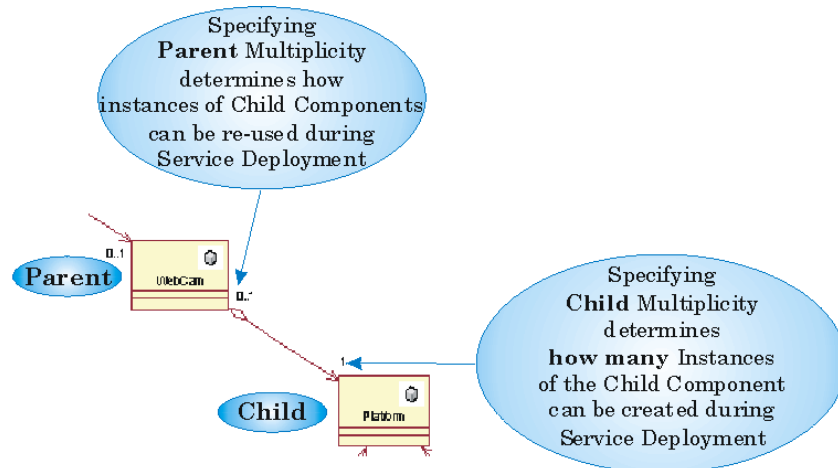
Although the Multiplicity (sometimes called Cardinality) for each Association must be set, you **cannot** set this as part of the creation action.

After you have created the objects, you can set the Multiplicity in two ways:

1. By right-clicking **while the Aggregation line is selected** and selecting the option **Multiplicity** from the menu which is displayed. A further menu is displayed with the Multiplicity options, and with the **Unspecified Multiplicity** item checked. This item is always checked when an association or aggregation has been created as the Multiplicity has not yet been set. This is illustrated in Figure 44.

2. And by using the Standard Specification Dialog.

Figure 44   Menus Showing the Options for Setting Multiplicity with the Service Designer Add-In to Rational Rose
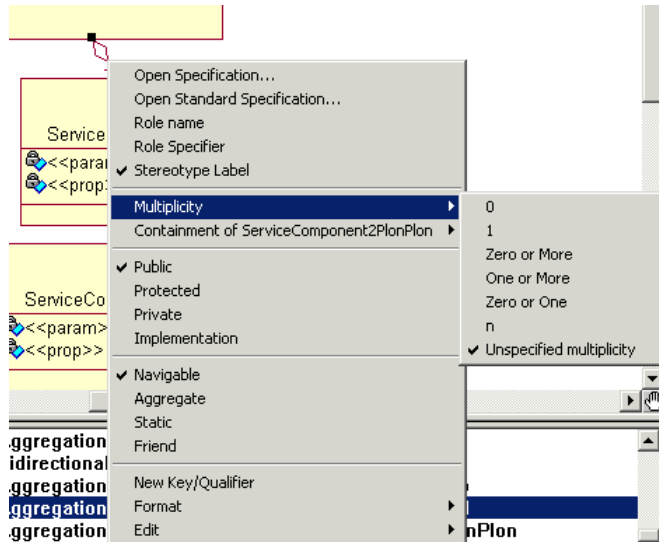
To use the Standard Specification Dialog:
(illustrated in Figure 45).

1. Double-click on the Association/Aggregation in the Class Diagram.

2. This will display the Rose Standard Set of Specification Dialog Boxes.

3. To set the Multiplicity of Role A, you must click on the **Role A Detail Tab**, and to set the Multiplicity of Role B, you must click on the **Role B Detail Tab**.

4. On the Detail Tab, choose the Multiplicity you want from the drop-down list named Multiplicity.

5. Click on OK to validate.

The Standard Set of Specification Dialog Boxes is illustrated in Figure 45.

Figure 45   Rose Standard Set of Dialog Boxes for Setting Multiplicity
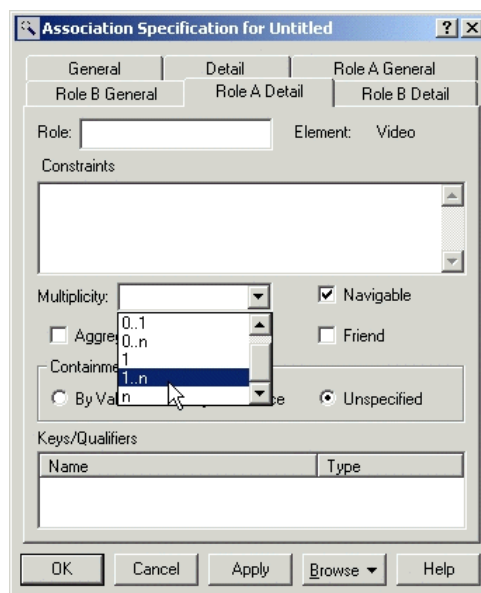
The same type of Association is possible between one Service Component and another Service Component. Aggregate relationships between Service Components are used to model Associations between Service Components.

Figure 46 **Part of a Model Showing a Unidirectional Aggregation Between Two Service Components**
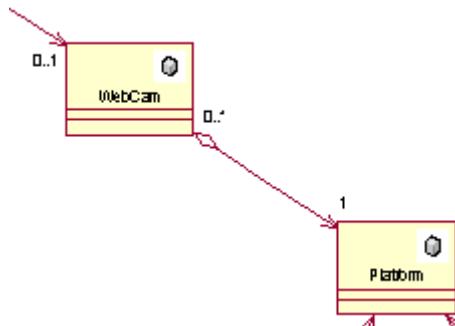
4.10.5 Data Feeder to Service or Service Component

This association represents the fact that a Data Feeder will be used to calculate the Service or Service Component Parameters.

You create the associations from a Data Feeder to a Service or a Service Component in the same way.

The Association is a Unidirectional Association, not an Aggregation.

Figure 47 **Part of a Model Showing an Unidirectional Association From a Data Feeder to a Service Component**
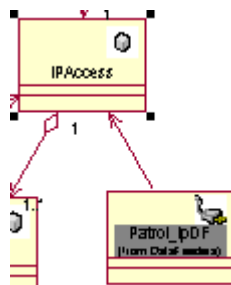
Figure 48   Part of a Model Showing a Unidirectional Association From a Data Feeder to a Service and a Service Component
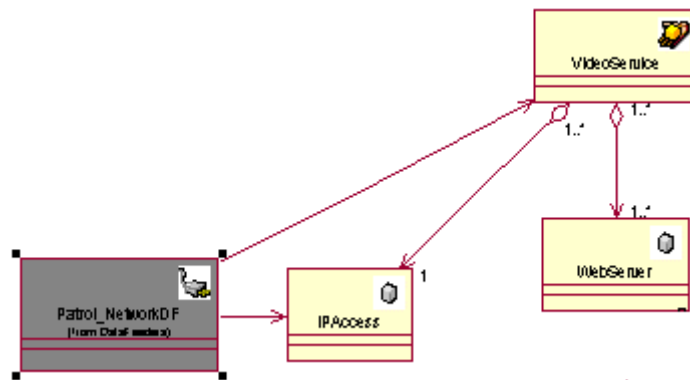
Multiplicity does not have to be set for this type of Association. The Association between Data Feeder and a Service or Service Component allows additional checking on the Service Model. No information related to these associations is generated in the Service and Data Feeder XML definitions.

4.10.6 Inheritance

Generalization of relationships, **Inheritance**, is allowed for Service, Service Component and Data Feeder Classes, with the restriction that Inheritance is allowed **only** between Classes having the same stereotype.

An object can inherit the characteristics of an object of the same type.

Inheritance allows a subclass to share Properties and Parameters with one or more super Classes. Other information specified in the Classes Service, Service component and Data Feeder is **not** inherited.

Whereas **Inheritance** is supported in Service Designer, conflicts are detected, but are not resolved. When defining parameters for inherited entities, you must be careful that the parameter name does not conflict with that of the entity from which they were inherited.

4.11 Calculation Expressions

Calculation Expressions allow the service operator to thoroughly tune the service model population process. With expressions, the service operator can control exactly how OpenView Service Quality Manager handles collected values and how they are propagated in the service model.

In other words, expressions describe how parameters will be calculated to populate the service model with service parameter values.

Therefore in the Service Designer User Interface, expressions are used to describe:

- The election Policy algorithm

- How to perform the data feeder bindings

- How to perform the component bindings.

4.11.1 Expressions Class

The UML Class Utility **Expression** is used to collect the UML operations necessary for the representation of the calculation expressions in Service Designer.

The Expression classes are used to:

- Define the binding in a Sequence Diagram (see section 4.12)

- Add a new custom calculation expression (see section 4.11.2)

- Generate the XML files for the calculation expression in order to load them in the SQM repository (see section 4.15.1.3)

Calculation Expressions can be represented in either of the two languages supported by OpenView Service Quality Manager: Java or PL/SQL.

A Calculation Expression is characterized by:

- The Operation name of the Expression

- The Operation input arguments, which are Service Component or Data Feeder Parameters.

- The Service or Service Component Parameter affected by the value returned by the Operation.

- The Expression Type (aggregation or simple).

- The Expression Language: (Java or PL/SQL).

To make Service Designer easier to use, the initial framework provides a pre-defined set of Expression classes.

Figure 49 illustrates the pre-defined Expression Classes provided with the Service Designer Add-In.

Figure 49   Expression Classes
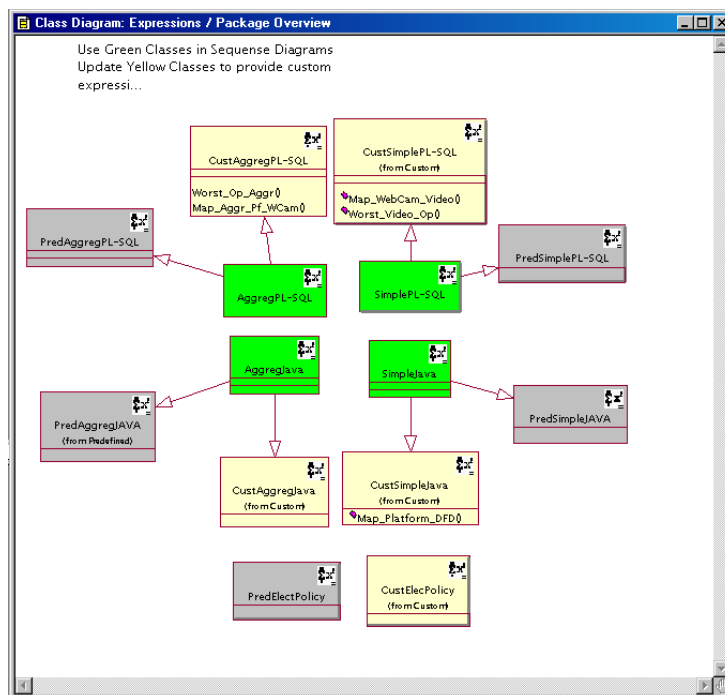
It will not often be necessary to create a new Expression Class, you will only need to add or modify a Calculation Expression contained in one of the Custom Expression Classes provided in the Initial Framework. How to create a new Expression Class is described in Section 6.4.

4.11.2 Creating a Custom Calculation Expression

If you need a specific calculation to compute a parameter, **Service Designer** provides the means of adding a **Custom Calculation Expression**.

You do this by adding a new Operation in the Custom Expression Class whose language (Java or PL/SQL) and type (Simple or Aggregation) match your need.

The process to define and use a custom calculation expression is the following:

- Add the calculation operation in the related expression class, in order to be able to use it in sequence diagrams

- Develop the associated Java or PL/SQL code

- Associate the calculation expression and it's code

- Generate the XML file

- Load this new calculation expression in the SQM Repository

4.11.2.1 Add a Calculation Expression in the Expression class

There is no specific Dialog Box to create a Custom Calculation Expression. You must use the standard Rose Dialog Boxes.

To do this:

1. In the Expression/Custom package, select the Expression Class that you want to contain the new Operation.

2. Right click on the Expression and select **New Operation**. This creates a new Expression under the Expression symbol.

3. Double click on this new Operation and use the Expression Specification Dialog Box to define the details.

4. On the **General Tab** you can define:

- The Expression Name
- The Description
- The Return Type.

5. On the **Details Tab** you can define:

- The Operation Parameters, Name and Type.

When you have done this you will be able to use the newly created Expression in the same way as the other pre-defined Expressions in Service Designer.

Important

To be able to generate an XML file for this calculation to load it into the ServiceCenter Repository, you must ensure that the body of the expression is attached. This is described in Section 4.11.2.2.

4.11.2.2 Creating the Custom Expression custom code

The Body of an Expression is the code that will be executed by OpenView Service Quality Manager when a Parameter is calculated using this Expression.

The 'body' is contained in a file that can be either a Java Class or a PL/SQL code file.

The creation of the code for custom expressions is a development task and no specific tools are provided with service designer. Java and PL/SQL knowledge is a prerequisite to write custom expression. As it is a development task, it is very important to test the produced custom code.

This chapter describes the specificity of the code to produce for Java and PL/SQL expression.

Java Expression

Java expressions are used only for collected (primary) parameters and only simple expression is supported.

The body file for a Java expression is a binary class file. To create this body file, you have to:

- Write the Java code in a text file (<Expression Name>.java)

- Compile this file using JDK 1.4 to produce the associated class file (<Expression Name>.class).
  javac <Expression Name>.java

The Java code for custom expressions must respect the following rules:

- Package must be: com.compaq.temip.servicecenter.expressions

- Class is public and class name must the name of the expression. Example: public class MapPlatformDFD for MapPlatformDFD custom expression.

- It must have an empty public constructor:
  Example: public MapPlatformDFD () {}

- It must have one static method that defines the custom calculation. This function must have the following characteristic:

  - The function must be public and static

  - It must have the same name as the class itself, but the function name must start with a lower case.

  - It must have the same return type as your custom expression

  - It must have the same parameters signature (order and type), plus an additional parameter: String dfdVersion

Example: public static long mapPlatformDFD (long int1, String dfdVersion)

- The following java type must be used for input and return parameters:

  - Double for Float

  - Long for Int, Enum and Relative Time

  - Date for Absolute Time

  - String for String

This is a sample Java class for the long MapPlatformDFD(long a, String dfdVersion) expression:

```
package com.compaq.temip.servicecenter.expressions;
/**
 * Makes the mapping from an Operational State to another
 */
public class MapPlatformDFD {

    public final static long DISABLE = 1;
    public final static long IDLE = 2;
    public final static long ENABLE = 3;

    /**
     * An empty constructor for this class
     */
    public MapPlatformDFD () {}
```

```
    /**
     * The calculation expression
     * @param a the operational state
     * @param dfdVersion the version of the DFD
     */
    public static long mapPlatformDFD(long a,
                                      String dfdVersion) {
        long state;
        if (a == 0) {
            state = DISABLE;
        } else if (a == 1) {
            state = IDLE;
        } else {
            state = ENABLE;
        }
        return state;
    }
}
```

PL/SQL Expression

PL/SQL expressions are used only for computed (secondary) parameters and both aggregation and simple expressions are supported.

The body file for a PL/SQM expression is a text file that contains the PL/SQL code.

The PL/SQL code for a custom expression must respect the following rules:

- It must create or replace a function with the same signature as the custom expression (return type, name, paramters)

- For return type and simple expression input parameter and, allowed PL/SQL types are:

  - NUMBER for Int, Float and Enum,

  - TIMESTAMP for Absolute Time

  - INTERVAL DAY TO SECOND for Relative Time

  - VARCHAR2 for String

- For aggregation expression input parameters are column on which the aggregation will be done. Specific SQM types must be used:

  - float_coll for Float

  - int_coll for Int

  - absTime_coll for AbsoluteTime.

  - relTime_coll for RelativeTime

  - string_coll for String

This is a sample code for a simple PL/SQL expression:

```
CREATE OR REPLACE FUNCTION Map_WebCam_Video( a NUMBER ) RETURN
NUMBER IS
  res NUMBER;
BEGIN
    -- 1 corresponds to the ON value
    IF a = 1 THEN
  -- 1 corresponds to ENABLE value
        res := 1;
    ELSE
  -- 3 corresponds to DISABLE value
        res := 3;
    END IF;

    RETURN res;
END;
```

This is a sample code for an aggregation PL/SQL expression:

```
CREATE OR REPLACE FUNCTION Worst_Op_Aggr( a enum_coll ) RETURN
NUMBER IS
  -- returns the worst of the list
  -- 1 => Enable
  res NUMBER := 1;
BEGIN

    FOR i IN a.FIRST..a.LAST
    LOOP
      IF a( i ) > res THEN
        res := a( i );
      END IF;
    END LOOP;
    RETURN res;
END;
```

4.11.2.3 Assigning a Body File to an Expression

1. Double click on the Expression class that contains the Operation.

2. Click the **Calculation Expressions** Tab. (Figure 50)

Figure 50   **Expression Specification Dialog Box, Calculation Expressions Tab**
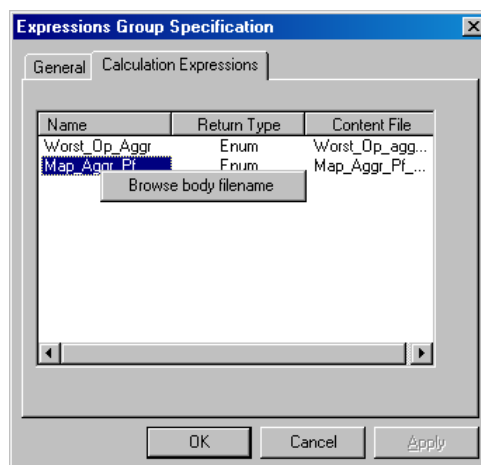
3. Select the Calculation Expression **Name** for which you want to Assign the Body file, and right click to display the **Browse body filename** pop-up menu.

4. Click on this pop-up.

5. From the list of body filenames displayed, select the one that corresponds to the Calculation Expression you have chosen.

6. The filename will be displayed in the **Content file** field.

<div align="center">

Important

</div>

It is extremely important that this should be done for all Calculation Expressions, otherwise, when an XML Definition for an Expression is created, if these files have not been chosen, and you have not verified that they contain the correct information, the XML definition will only generate a skeleton of the XML Definition document.

4.12 Binding Service, Service Component and Data Feeder Parameters

Binding is a way of assigning a value to a parameter. There are two types of binding: Primary and Secondary.

The Binding of **Parameters** is represented by UML Sequence Diagrams.

4.12.1 Primary Binding

A primary binding gets information directly from a data feeder. Therefore,

a **Service Definition** receiving information from a **Data Feeder Definition**,

OR

a **Service Component Definition** receiving information from a **Data Feeder Definition,**

constitutes a **Primary** binding.

When you define a Primary binding, you must only use Simple Java Calculation Expressions in the Sequence Diagram.

4.12.2 Secondary Binding

A secondary binding gets its information from a Service Component Definition, which in turn gets its information from a Data Feeder. Therefore,

a **Service Definition** receiving information from a **Service Component Definition**,

OR

a **Service Component Definition** receiving information from another **Service Component Definition**,

constitutes a **Secondary** binding.

When you define a Secondary binding, you must only use PL/SQL Calculation Expressions (Aggregation or Simple) in the Sequence diagram.

4.12.3 Sequence Diagrams

A sequence diagram is a graphical view of a scenario showing object interaction in a time-based sequence, in other words what happens first and what happens next. Sequence diagrams establish the roles of objects and help in providing essential information for determining Class responsibilities and interfaces. This type of

diagram is best used during the early analysis phase of the design, as they are simple and easy to understand.

UML Sequence Diagrams are generally used to model sequences of operations between Objects.

The principle illustrated here

```
Obj1:   Class1     obj2:   Class2
 |         |
 |       callMethod()        |
 |_____>|
 |         |
 |<--------------------- |
 |         |
```
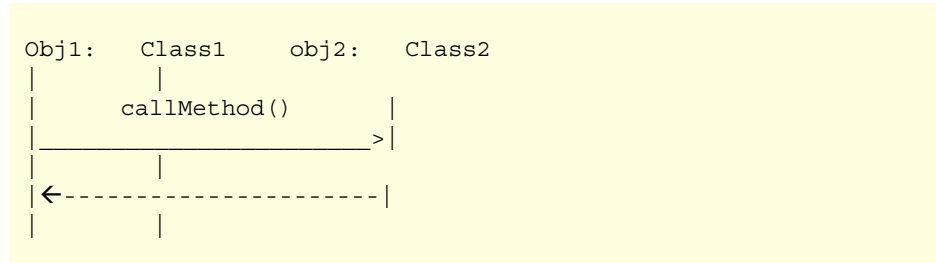
means that the object **obj1** (which is **Class1**) invokes the method **callMethod()** of the object **obj2**, and returns a value.

Each Service and Service Component has **at least** one associated Sequence Diagram representing the Expressions used for Parameter evaluation.

Sequence Diagrams are used in Service Designer to define how Parameters are calculated.

Creating Sequence Diagrams is a Standard Rose feature. How to create the example shown in Figure 51 is described in Section 4.12.3.1.

**Figure 51   Sequence Diagram for a Mail Service
                     (Component of Video2)**
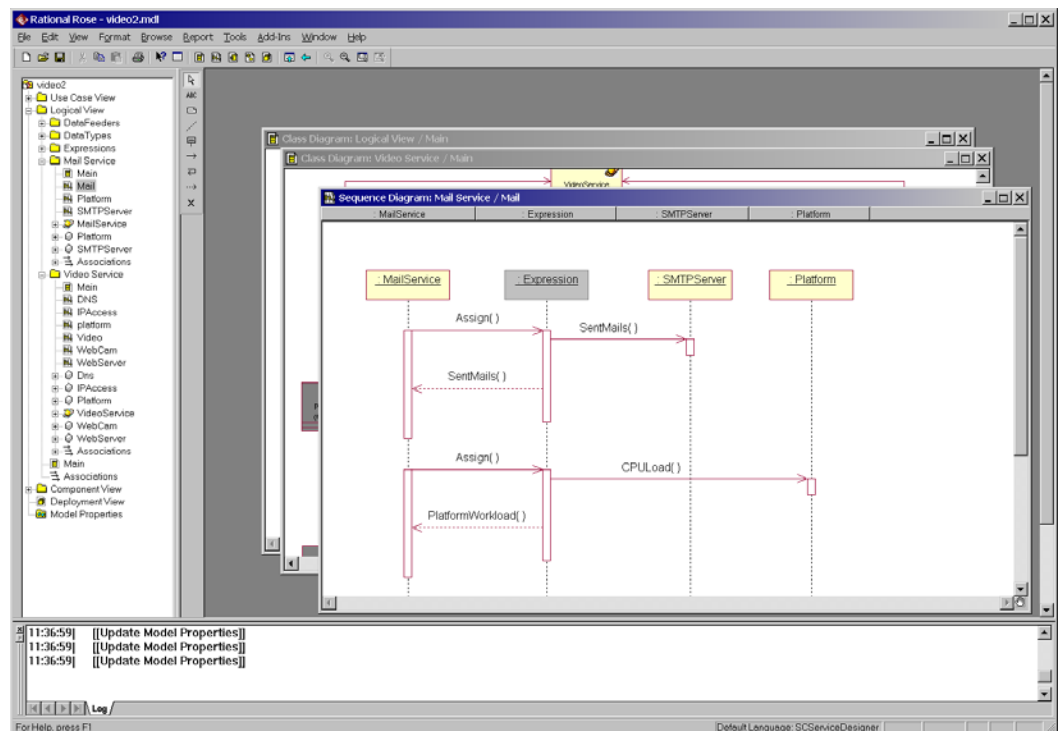
4.12.3.1 Creating a New Sequence Diagram

To create a new Sequence Diagram:

1. Choose: Logical View,

2. New,

3. Sequence Diagram.

Figure 52   Sequence Diagram Menus
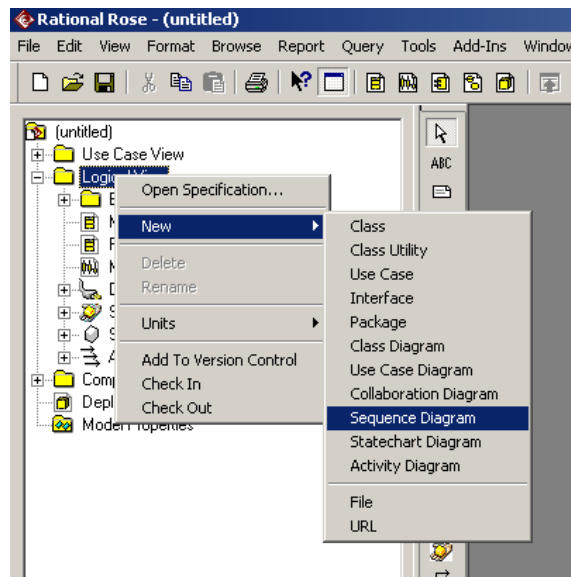
4. This will display a new empty Sequence diagram window.

Figure 53   New Empty Sequence Diagram Window
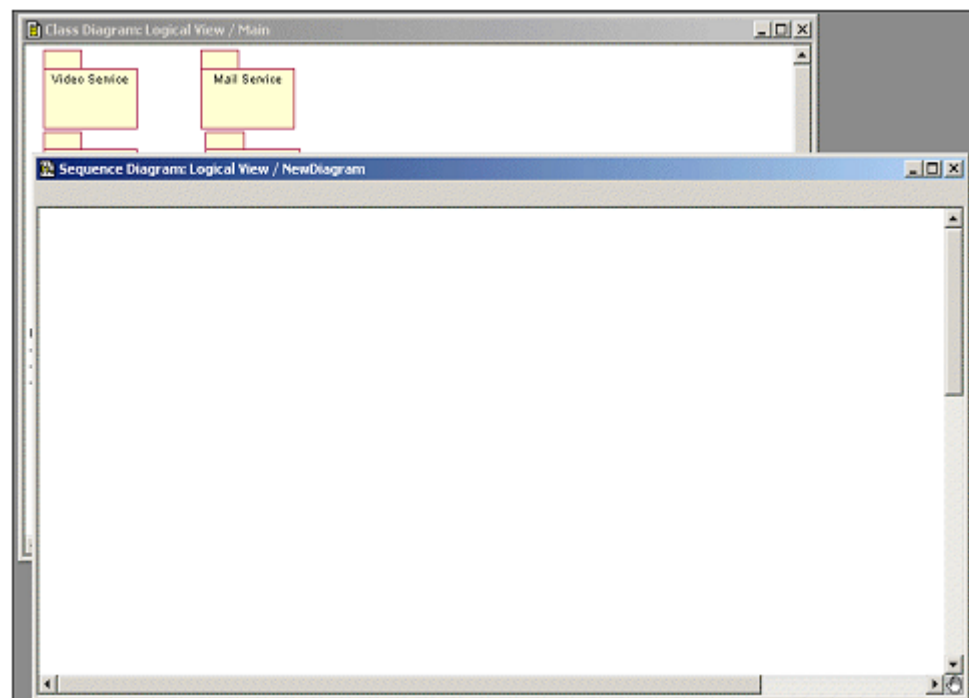
5. You must now insert the elements needed to perform the calculation into the sequence diagram: that is the computed parameter, the operation used for the computation and the input parameter. The order in which you do this is important, they must be inserted into the Sequence Diagram in the order in which they will be used. For the example illustrated you would follow this procedure:

    -   Insert the Mail Service Class into the Sequence diagram by dragging and dropping the Mail Service class from the Logical View **tree window** into the Sequence diagram. (The Mail Service Class contains the calculated parameter.)

The Tree window illustrated in Figure 54 is shown unexpanded. To see all the objects, click on the **plus (+)** sign to expand the Objects.

Figure 54   Logical View Tree Window
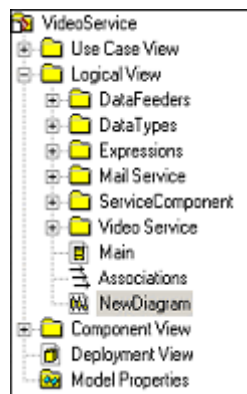
-   Insert the Expression Class into the Sequence diagram in the same way.

-   Insert the SMTP Server in the Sequence diagram in the same way. (The SMTP Server class contains the input parameters.)

Figure 55   New Sequence Diagram with Classes Inserted
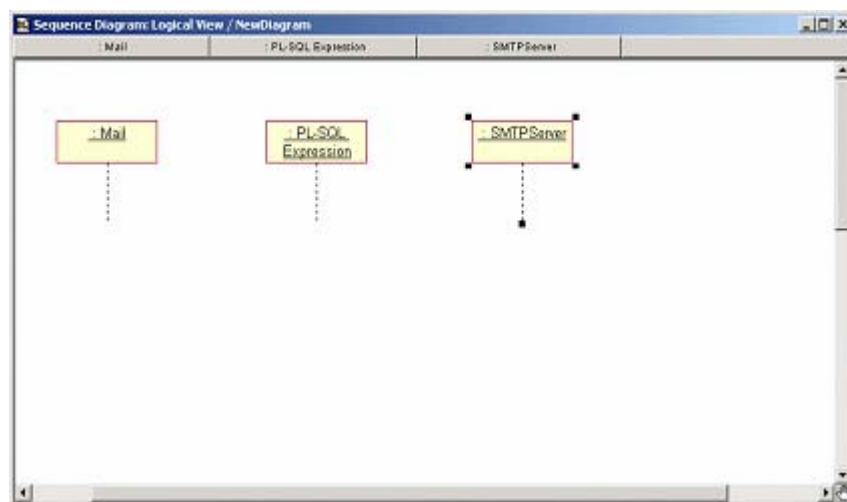
You must now create the sequence of events involved in the computation of the parameter in the diagram. The order in which you do this is important.

1. Create a message from the Source Class to the Expression Class (draw an arrow with a solid line).

2. Select the Calculation Expression (right click on the arrow).
If you don't know at this time the operation you want to use, you can perform next steps and select the operation to use latter.

3. For **each** Input Parameter, create a message (solid line arrow) from the Expression Class to the Class that contains the Parameter.

4. By right clicking on the arrow, select the Parameter Name for each message.

Figure 56   Messages Created and Parameter Name Selected
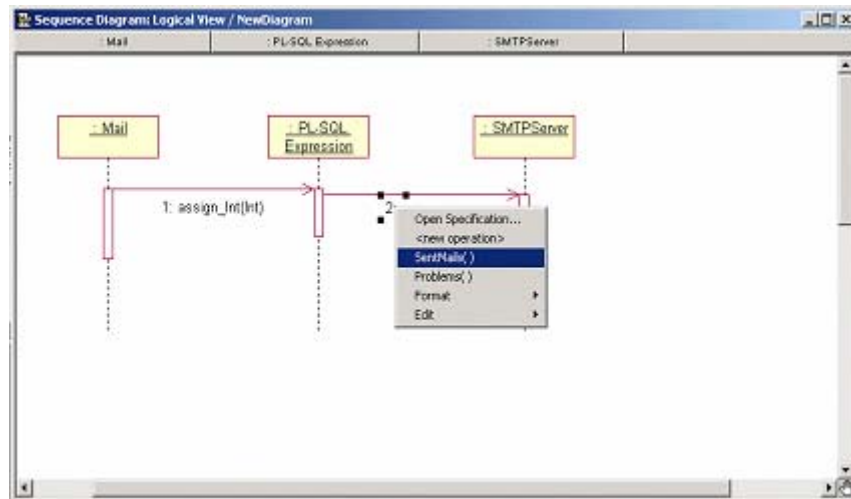
5. Create a Return Message (an arrow with a dotted line) from the Expression Class to the Class that will calculate the Parameter. Then select this parameter by right clicking on the arrow.

Figure 57   Return Message Created
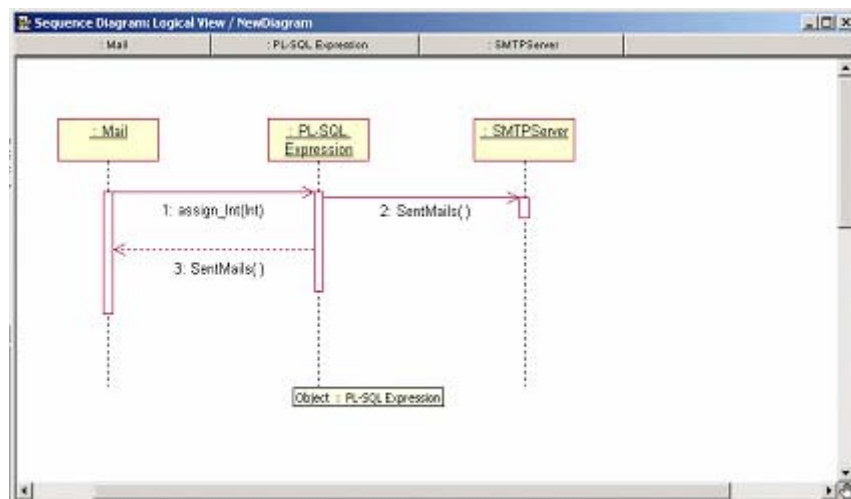
This will complete the Sequence diagram.

4.12.3.2 Actions you should Not Perform in Sequence Diagrams

This section describes some typical mistakes that you should not do in Sequence Diagrams. Most of these actions will raise an error during Model Checking. Refer to Section 0 for further details.

- The Input Parameter comes from Several SCDs. (Illustrated in Figure 58).

Figure 58   Case 1, Example of an Incorrect Sequence Diagram
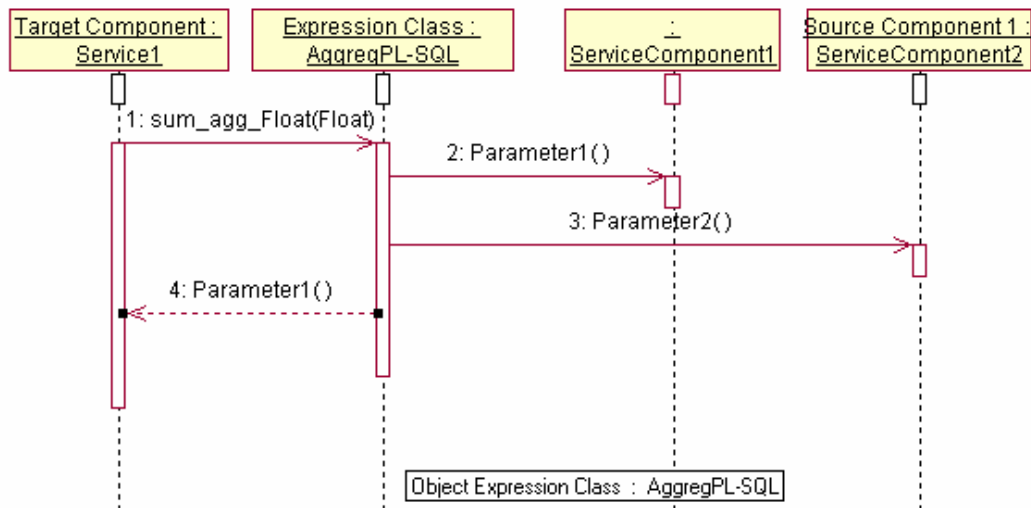
- A PL/SQL expression is used to compute a Primary Parameter. (Illustrated in Figure 59).

Figure 59   Case 2, Example of an Incorrect Sequence Diagram
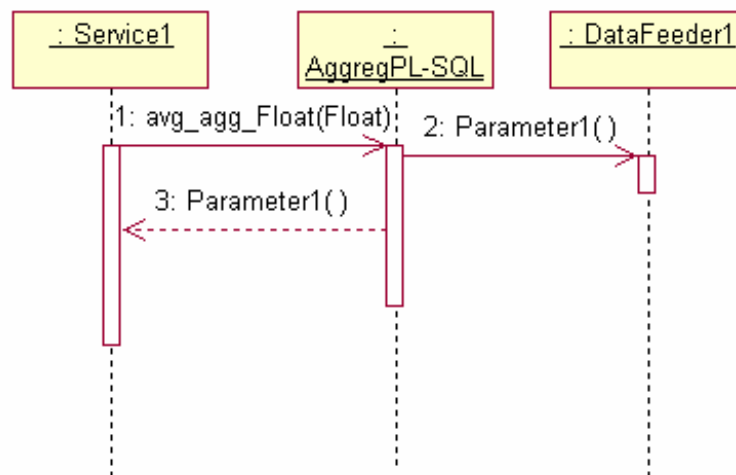
- A Java expression is used with input parameters coming from a Service component. (Illustrated in Figure 60).

Figure 60   **Case 3, Example of an Incorrect Sequence Diagram**
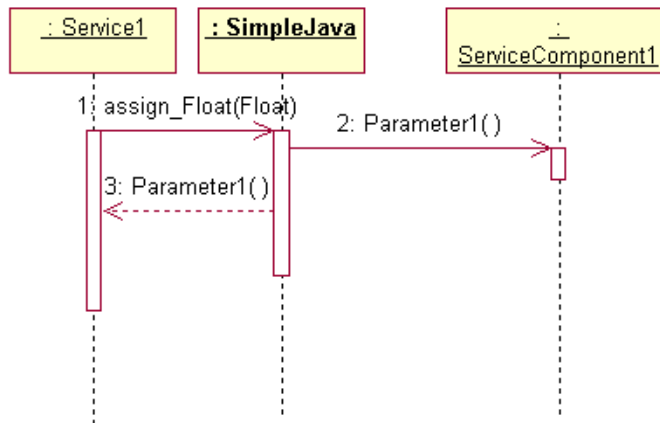
- A simple expression is used on an aggregated component and an aggregate expression is used on a mono component. These two Sequence diagrams will not be correct because of the child multiplicity of the Service Components. (Illustrated in Figure 61).

Figure 61   **Case 4, Example of an Incorrect Sequence Diagram**
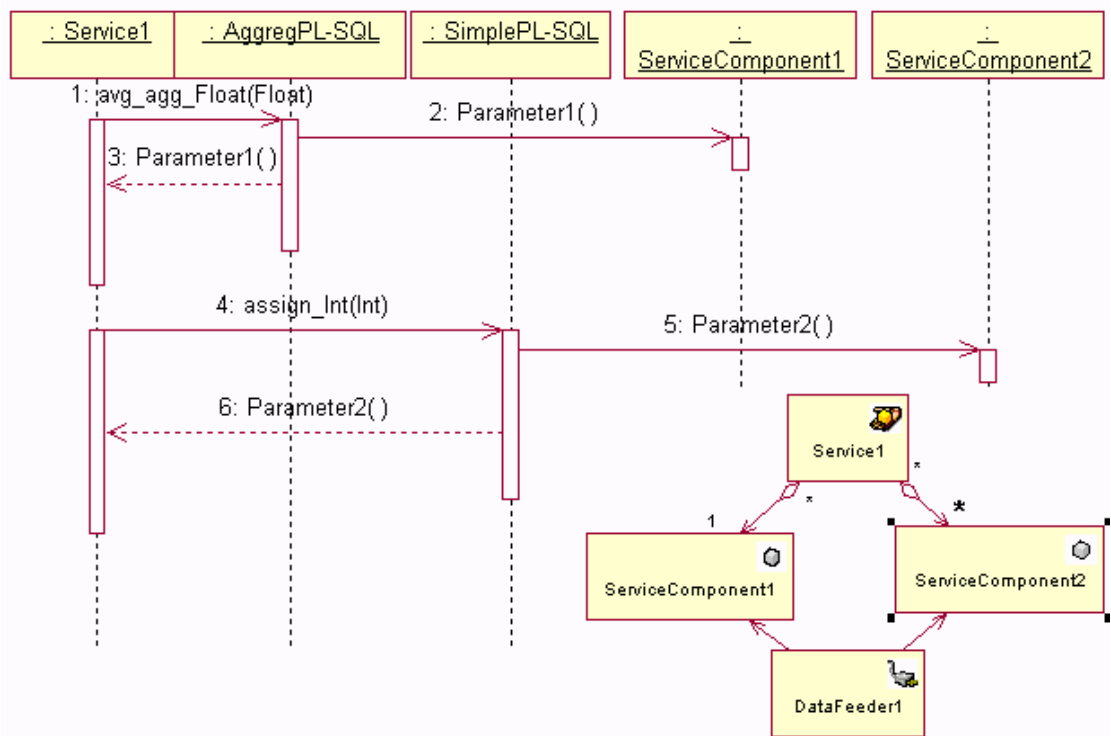
Note

It's possible to use an aggregate expression and to have input and output parameters on the same Service Component. In this case, a Time Aggregation will be applied to compute the output parameter. See section 4.6.5 for further details on Time Aggregation.

4.13 Enumeration

An Enumeration (the stereotype is Enum) is represented as a Rose Class. Enumeration is not yet included in the Add-In interface for this version of Service Designer.

To create this Class, you must:

- Create a Rose Class.
- Set the Stereotype to Enum.
- Create the "necessary Attributes+default values" to represent the Enum values.

For example: if you have the following Enum:

```
enum OperationState {Idle=0, Enable=1, Disable=2};
```

1. Create a Rose Class called **OperationState**,

2. Set the **stereotype** to **Enum** and

3. Create 3 Attributes (Idle, Enable, Disable).

4. For each attribute you must also set the correct default value.

Table 1   Example of Settings for an Enum Class

Enumeration		Mapped to Rose Class as:			
Operational State		Enum Stereotype			
Idle	= 0;	Attribute =	Idle	default	= 0
Enable	= 1;	Attribute =	Enable	default	= 1
Disable	= 2;	Attribute =	Disable	default	= 2

By convention, it is mandatory to have one default value equals to 0 in the Rose class. If your enumeration does not have this value, you must create and use a custom expression to map enumeration values to Rose class values, to follow the convention.

For example: if you have the following Enum:

```
enum OperationState {Idle=2000, Enable=2001, Disable=2002};
```

1. Create a Rose Class called **OperationState**,

2. Set the **stereotype** to **Enum** and

3. Create 3 Attributes (Idle, Enable, Disable).

4. For each attribute you must also set the value following SQM convention.

Table 2   Example of Settings for an Enum Class

Enumeration		Mapped to Rose Class as:			
Operational State		Enum Stereotype			
Idle	= 2000;	Attribute =	Idle	default	= 0
Enable	= 2001;	Attribute =	Enable	default	= 1
Disable	= 2002;	Attribute =	Disable	default	= 2

During the primary binding phase, use the custom expression to map values.

4.14 Validating a Service Definition

There are three types of validation provided with **OpenView Service Quality Manager Service Designer**:

- Some validation is performed in real time in UML as you perform your actions. Refer to Section 4.14.1 for details.

- Service model checking is performed on demand. Refer to Section 4.14.2. You should always perform this action **before** generating XML definitions.

- Model checking is also performed during the generation of an XML definition.

Important

Errors are checked in order of severity, therefore if an error of a higher level is found, Service Designer does not continue checking to find lower level errors. The message displayed will indicate the error that was encountered. This does not, of course, indicate that there are no other errors of a lower level.

4.14.1 Real Time Model Checking

As you create your Service Definition, actions and situations that are not allowed are checked and you are informed, either by a message in the log or by a pop-up message box. This is to help you during the model creation. The creation is not prevented, and if something is incorrect, you must 'undo' the action manually. (You do this while the item is still selected in the Class Diagram window, either by pressing **Ctrl/D,** or **Delete from Model** on the **Edit** menu. If you have unselected it, reselect the item in the Class diagram window and either press **Ctrl/D,** or select **Delete from Model** on the **Edit** menu.)

Real time checking includes:

- Checking the consistency between the Parameter and Operations:

    - The operation is not created at the same time as the Parameter. It is only created when the Parameter becomes 'valid'. Check performed when the Service Designer Parameter Dialog is used.

    - Consistency during the Parameter Life-time. Check performed when the Service Designer Parameter Dialog is used to modify the Parameter. The Operation must be consistent with its Parameter (type, name and so on).

    - Automatic deletion of Operation when corresponding Parameter is deleted.

    - When an Operation linked to an existing Parameter is deleted you are informed of the deletion by a message in the Log. This check is performed using the Service Designer feature where the operation Id is set in the Parameter.

- Checking the Associations between Service Designer Entities:

    - With the exception of Service Component Classes, the creation of an association between two Service Designer Entities of the same type is not allowed. That is, you cannot create an association between two Services, two Data Feeders or two Expressions. Message in Pop-up box.

    - The association between a Data Feeder and a Service or Service Component must be an unidirectional association, with the Data Feeder as the sender. Indicated by a message in a Pop-up box.

- Checking the Inheritance relationships between Service Designer Entities:

- An Inheritance relationship must be created between Classes of the same type (SD, SCD, DFD and Expression). Message in a Pop-up box.

- Checking unsupported relationships between Service Designer Entities:

  - Class Dependency is not allowed between SD Classes. Message in a Pop-up box.

  - RealizeRelation is not allowed between SD Classes. Message in a Pop-up box.

- The overriding of an expression when an expression defined in a Base Class is redefined in a Child Class is not allowed. A warning is issued in the Log window. This conflict is not resolved either by Service Designer, or by Rational Rose. You must take care when re-using definitions to ensure that you have not defined the expression in this way.

4.14.1.1 Checks NOT Performed by Real Time Model Checking

At creation time Rational Rose does not permit distinguishing between an Association and an Aggregation (or a Composition). Multiplicity is never set at Creation time. You must set the Multiplicity after the creation (see section 4.10.4).

- A valid Association between a Service and a Service Component must be a unidirectional Aggregation, the Service must be an Aggregate and the Multiplicity (Cardinality) must be set. Real Time checking does not prevent you from creating a Unidirectional Association between a Service and a Service Component.

- A valid Association between a Service Component and another Service Component must be a unidirectional Aggregation, and the Multiplicity (Cardinality) must be set. Real Time checking does not prevent you from creating a Unidirectional Association between two Service Components.

If you have made a mistake in creating either of these associations, you must correct them yourself by deleting the associations and then re-creating them correctly.

4.14.2 Service Model Checking on Demand

You must always perform this check **before** generating XML definitions.

To initiate a check on the model:

1. From the **Tools** menu

2. Select **Service Designer**

3. Select **Check Model**.

This process checks:

- Classes and Associations. Refer to Section 4.14.2.1

- Expressions. Refer to Section 4.14.2.2.

- **OpenView Service Quality Manager** Rules. Refer to Section 4.14.2.3.

Warning

The Standard Rose Model Checking option, selected from **Tools, Check Model** will discover basic UML errors, but will **not** discover errors related to Service Design.

To ensure that you check the Service Design model, you must be sure to choose **Tools, Service Designer, Check Model**.

4.14.2.1 Classes and Associations

- Checks the use of relationships other than Association and Inheritance in Class Diagrams.

- Checks for Loops in Class associations.

- Checks Service-to-Service Component and Service Component-to-Service Component associations (must be unidirectional aggregations).

- Checks Service-to-Service Component and Service Component-to-Data Feeders associations (must be unidirectional associations).

- Checks consistency between Parameter Class Attributes and operations.

- Checks that there is the same stereotype between child and base Class for Service, Service Component, and Data Feeder Inheritance. (Issues a Warning only).

- Checks whether there are multiple definitions of Parameter or Property in two super Classes.

- Checks whether there are multiple definitions of Parameter or Property in a Class and its super Class (Overriding; issues a Warning only).

- Checks for presence of Classes with an unsupported stereotype (issues a Warning only).

- Checks mandatory tagged values.

4.14.2.2 Expressions

- Checks validity of the Expressions Sequence diagrams

- When Operation signature is defined, checks the input and output operation argument of Expression data type.

- All Service and Service Component Parameters must have an associated Expression defined. Checks completeness of Parameter calculations.

- Only one Binding Expression is allowed for each parameter. Checks uniqueness of Parameter calculations.

- A Warning is issued when there is a risk of an Expression definition being overridden. That is when an Expression already defined in a base Class is redefined in a child Class.

4.14.2.3 OpenView Service Quality Manager Rules

- Checks for Loops in Expressions

- Checks Expressions related to Parameters whose Class is not in direct association with the Class of the target Parameter.

4.14.2.4 Error Logging

- All errors and warnings issued during model checking and generation of XML documents are logged in the Rose Log Console.

4.15 Generating an XML Definition

You can generate an XML definition for an entire Service definition, a Data Feeder definition or for an Expression definition.

The process is the same for whichever Class you want to generate an XML definition.

Model should be saved **before** you generate an XML Definition.

Before you generate an XML definition, you should choose the directory where you want the output file to be saved. A default directory is displayed, but you can change it if you want.

The DTD Prefix is set to DTD/ and should not be changed.

Figure 62   Project Specification Dialog Box, Directories Tab
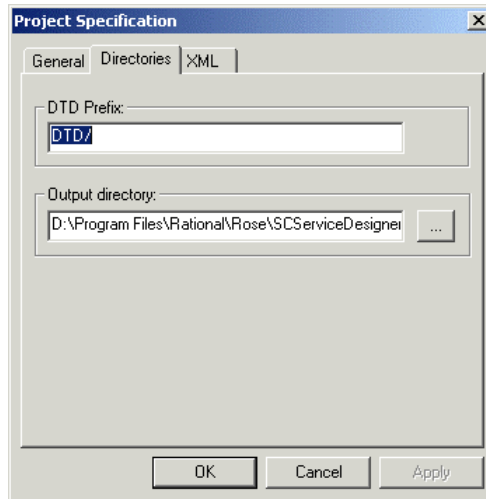
If you do not want XML definitions to be generated when there is an error in your definition, you can prevent XML generation by unchecking the check box **Always generate XML definition** in the Project Specification dialog box, General tab.

Figure 63   Project Specification Dialog Box, General Tab
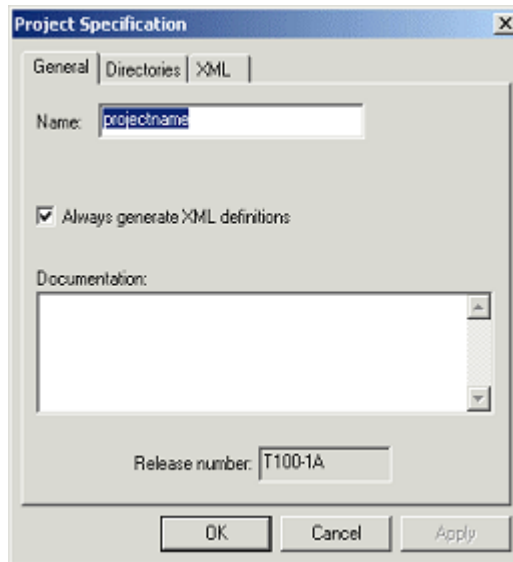
The third Tab of the **Project Specification** set of dialog boxes is the **XML** Tab. In the field XML Header, this Tab displays the default XML header, or the header from your previous definition. It is possible to edit this, but it should be done very carefully.

Figure 64   Project Specification Dialog Box, XML Tab
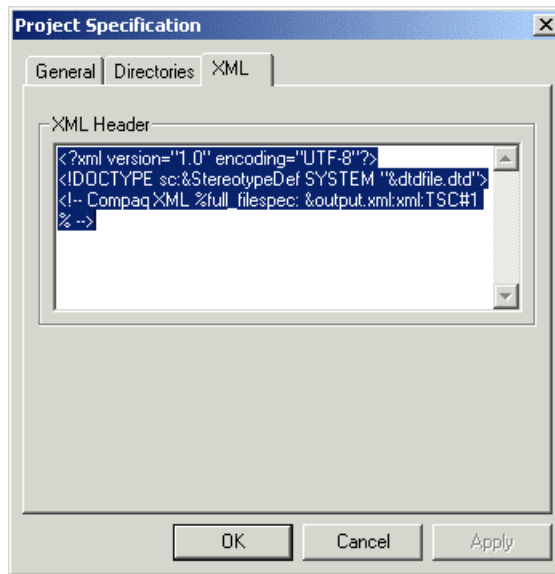
4.15.1 How to Generate an XML Definition

You can do this either by selecting a Class (Service, Data Feeder or Expression) in the Class Diagram window or by selecting the Class icon in the tree diagram.

When you have done this, you also have two ways of accessing the **Generate XML Definition** options, the Menu method or the Right-click method.

The Menu Method

1. From the **Tools** menu, select **Service Designer,** and then **Generate an XML Definition**.

2. This will display the XML generation options menu illustrated in Figure 65.

3. Choose the type of XML definition you want to generate.

Figure 65   XML Generation Menu
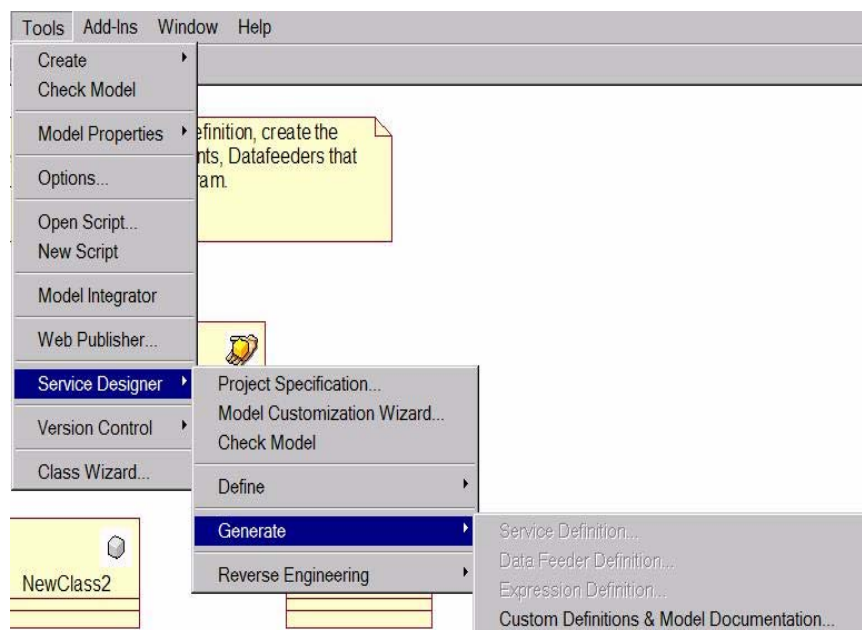
Note

If you haven't selected a Class in the Class Diagram window or a Class icon in the tree diagram, the **Generate an XML Definition** menu is grayed as shown in Figure 65.

If you choose the wrong type of XML Definition to create, such as, the Class selected was a Service Component, and the XML Definition type chosen was a Service, then a Pop-Up error message box is displayed.
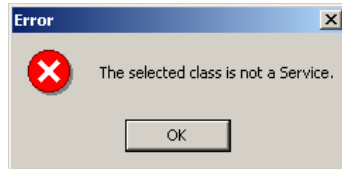
If you choose the correct type of XML Definition to generate and there are no errors in your definitions, a message indicating that it is being generated, and where it will be stored, is displayed in the log.

The Right-click Method

1. While the cursor is over the Class Icon in the tree diagram, or over the Class in the Class diagram window, right-click with the mouse. (If you right-click when the cursor is in another part of the Class diagram window, or is not directly over the icon in the Tree diagram, even though you have selected the Class or the icon, another menu is displayed, **not** the pop-up menu offering the option to Generate an XML Definition.).

2. The XML Definition for that Class is generated according to whether "Always Generate XML" has been defined in the project specification. A message indicating where it is being stored is displayed in the log window. If the model is **not** correct, then a message is displayed in the log window indicating the source of the problem.

If there are errors in your definition, and if you have unchecked the **Always Generate XML definition** check box, the XML definition will **not** be generated.

4.15.1.1 Generating a Service XML Definition

If the Service Model is not complete, or has inconsistency problems, an XML Definition cannot be generated. When component libraries are imported into a model, the component library objects that are not necessary for the generation of the Service XML Definition are not generated.

4.15.1.2 Generating a Data Feeder XML Definition

When component libraries are imported into a model, the component library Data Feeders that are not necessary for the generation of the Service XML Definition are not generated.

4.15.1.3 Generating an Expression XML Definition

When you generate an XML Definition for an Expression, you must make sure that the body of the Expression Operation is included (either the Java or the PL/SQL code. See Section 4.11 for further information.

Note

Operations inherited from Superclasses are NOT generated. Only operations defined locally are generated.

Warning

You must not generate the pre-defined Expressions, or change them in OpenView Service Quality Manager.

4.16 Importing Definitions into OpenView Service Quality Manager

To load the Definitions created with Service Designer into **OpenView Service Quality Manager**, you must use the facility provided with **OpenView Service Quality Manager**:

```
temip_sc_load_definition
```

This is a convenience tool that calls the Command Line User Interface (CLUI) to allow you to load the data into **OpenView Service Quality Manager**.

Here are some examples:

Command	Explanation
`temip_sc_load_definition -e E1.xml`	Loads the Calculation Expression defined in the file **E1.xml** into **OpenView Service Quality Manager**.
`temip_sc_load_definition -d D1.xml`	Loads the Data Feeder Definition defined in the file **D1.xml** into **OpenView Service Quality Manager**.
`temip_sc_load_definition -s S1.xml`	Loads the Service Definition defined in the file **S1.xml** into **OpenView Service Quality Manager**.

Chapter 5

Reverse Engineering

This important feature of OpenView Service Quality Manager **Service Designer** enables you to import definitions that you have previously created. You can import an XML definition for a Service and a Data Feeder. This feature should not be confused with the Reverse Engineering described in the Help of the Rational Rose standard edition.

In Service Designer, you should only use the method described in Section 5.1.

5.1   Importing a Definition

1. Select **Service Designer** from the **Tools** menu.

2. Select **Reverse Engineering** from the second menu displayed.

Select the definition you want to import.

Figure 66   Reverse Engineering Selection
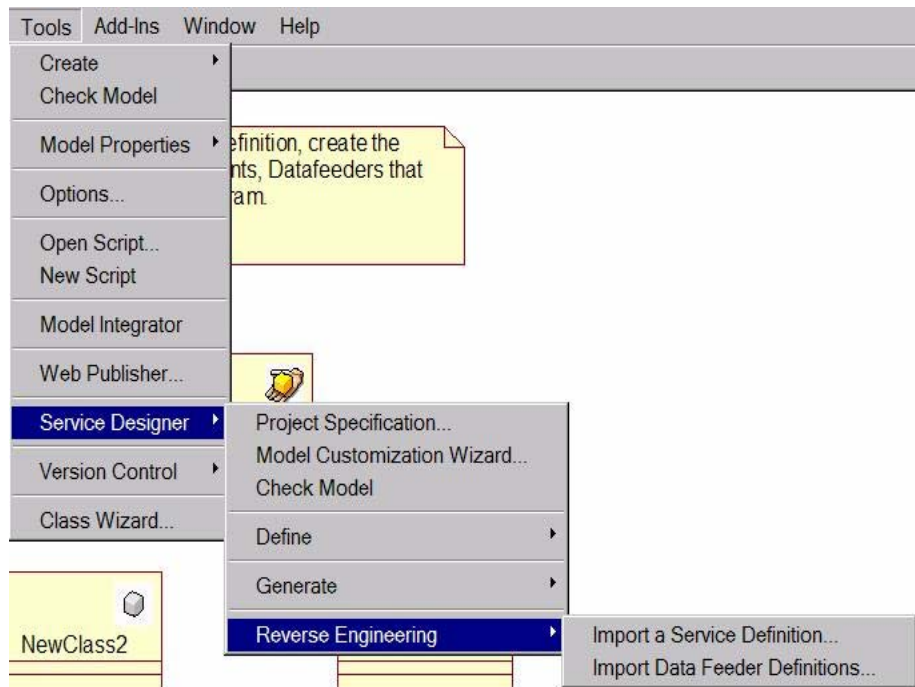
A classical file selector (Figure 67) is displayed upon selection of one of the Import Definition menu entries.

Figure 67   Import an XML Definition Dialog Box
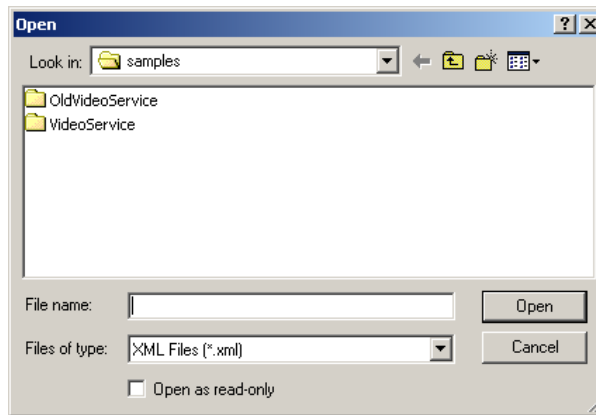
When importing the XML definition for a Service, the Calculation Expressions are not imported. Only the structure is imported. In other words, only the Service and its associated Parameters and Properties are imported, not the associated Sequence Diagram (defining the way the parameter values are computed).

Note

It is also possible to generate an XML Definition using the Command Line User Interface (CLUI). The XML Definition files generated in this way have a "wrapping" around the definition. If you import these files, either using the CLUI or the method described in Section 5.1, then this "wrapping" is ignored.

Chapter 6

Advanced Features

This chapter describes some advanced features connected with **OpenView Service Quality Manager Service Designer**.

6.1 Model Customization Wizard

This tools permits to create sub-model for a given Service Definition. A sub model is built by excluding some Service components and parameters from the model. If the model is correctly designed, it is then possible to exclude the unnecessary part of the model for a given deployment. For example, you can remove every OVO component if the customer is not using OVO.

For a given Service Definition, any number of model customizations can be managed. Example: IPMS without OVO, IPMS without TeMIP…

To start the Model Customization Wizard, click "Tools/Service Designer/Model Customization Wizard…"on toolbar after a service definition is selected. (Figure 68).

Figure 68   **Model Customization Wizard selection**
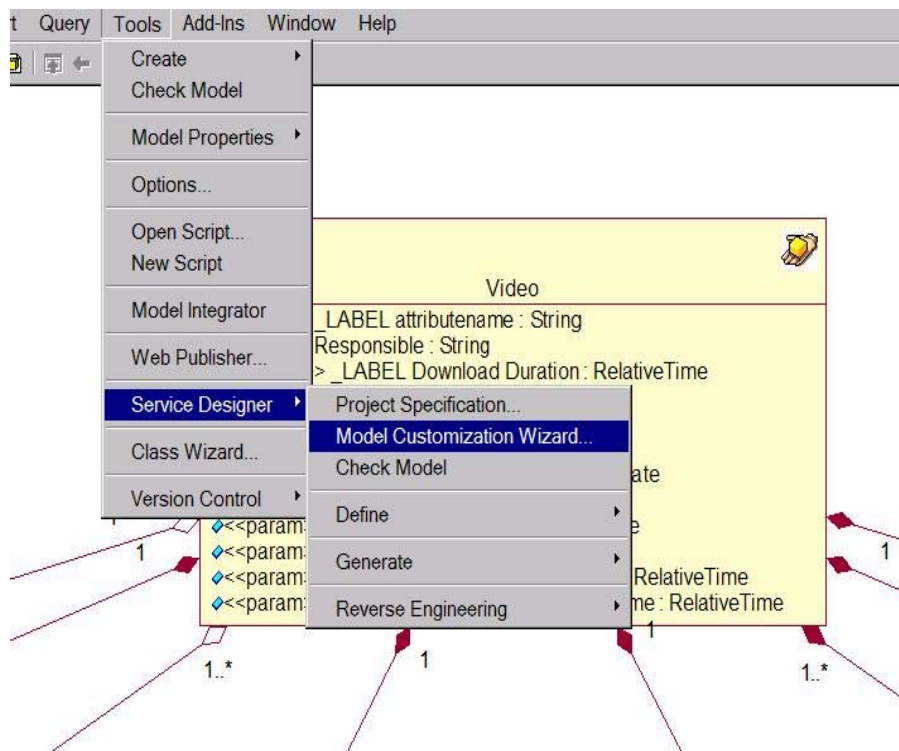
The wizard window (Figure 69   ) is displayed.

In the first panel, you can manage existing model customizations. The List box contains the existing model customizations and you can

- Create a new model customization by giving a new name in the Text field and click "Next",

- Edit the selected model customization by clicking "Next"

- Delete the selected model customization by clicking "Delete"

- Duplicate the selected model customization by clicking "Duplicate" and providing a new model customization name.

Figure 69   Model Customization Wizard
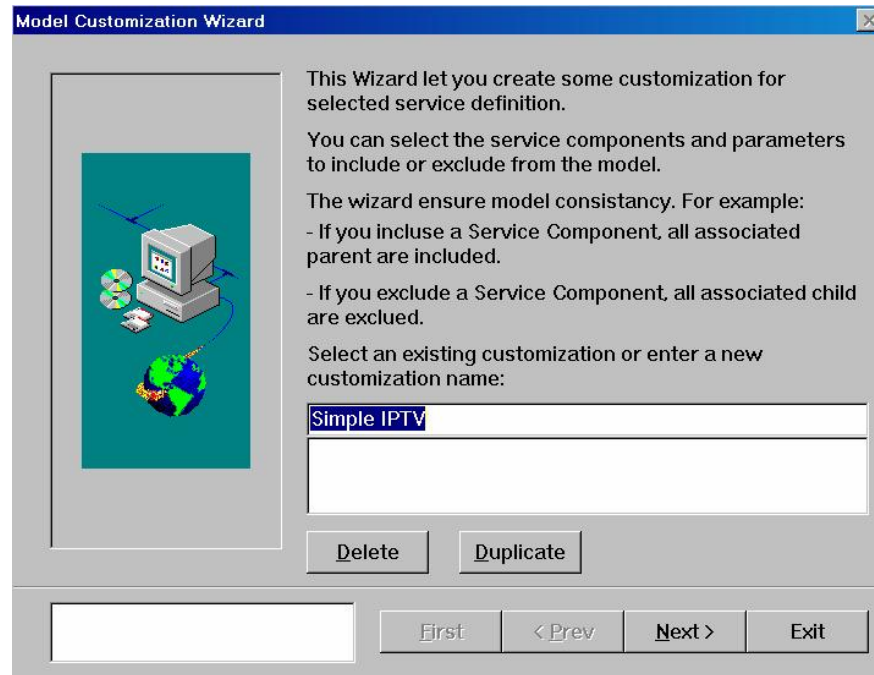
In the next panel (Figure 70   Service Component Selection), you select the component to include or exclude from the model customization.

Figure 70   Service Component Selection
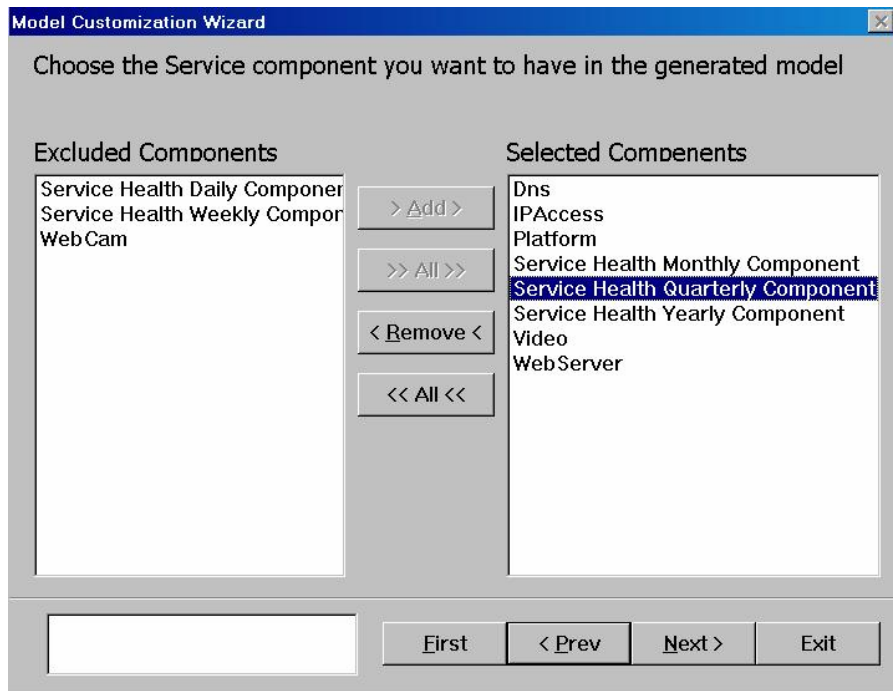
In the next panel (Figure 71   Parameter Selection), you can browse through selected components (Next SCD, Previous SCD) and select the parameter to include or exclude.

Figure 71   Parameter Selection
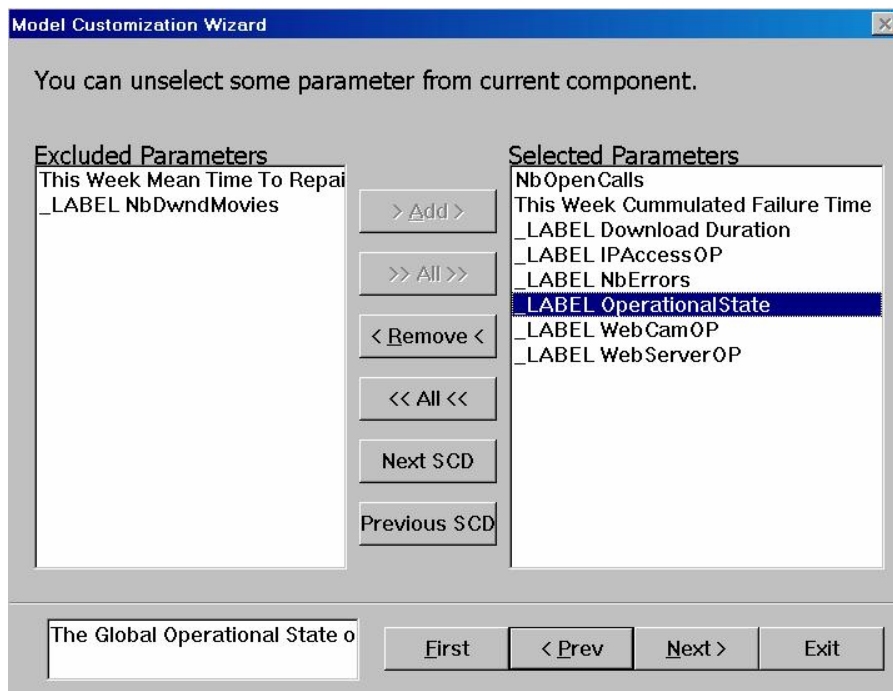
In the next panel (Figure 72   Information for model customization), you can enter the description of the model customization and a prefix/suffix for the model customization.
The prefix and suffix are appended to every service component identifier for the XML generation.

Figure 72   Information for model customization

Note

If you enter a SCD prefix and suffix, you must be sure that the length of the
component identifier (=length(SCD Prefix) + length(SCD Suffix) + length(Id)) is less
that 16 for every component of the model.

In the last panel (Figure 73   Final Screen of Customization), you have a
summary of the created model customization. You can click "Generate" to open the
"Generation Dialog" or quit the wizard ("Cancel").

Figure 73   Final Screen of Customization
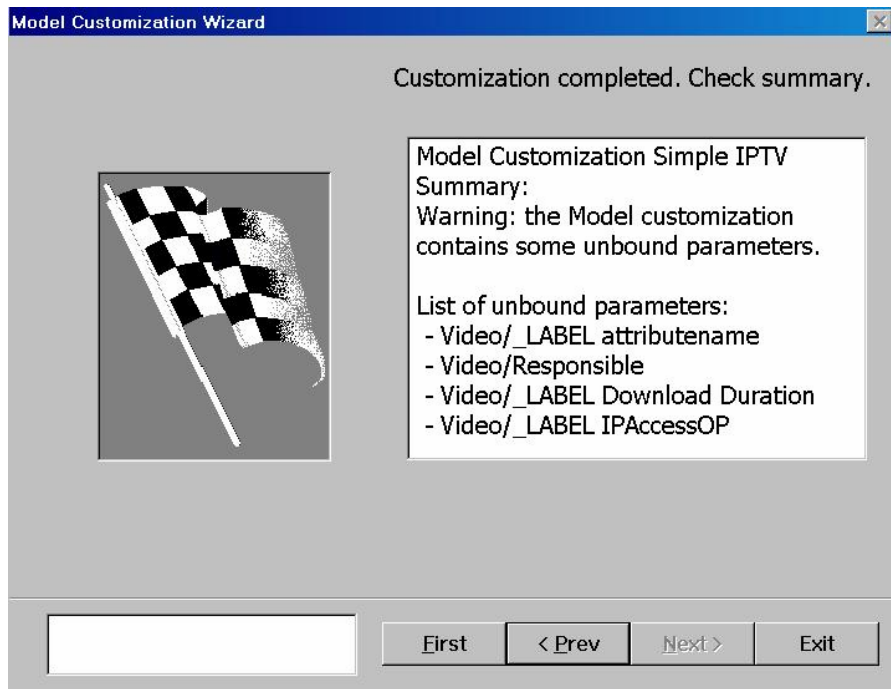
6.2 Custom Definitions & Model Documentation Generation

You can access the generation dialog (Figure 74   XML Generation Dialog) through the "Model Customization Wizard" (see above) or directly from Add-in menu (Tools/ Service Designer /Generate/Custom Generation…").
This tool provides several generation facilities.

- Generate every DFD and expression of the model

- Generate the selected model customization for selected service definition

- Generate documentation in Word or Excel format.

Figure 74   XML Generation Dialog
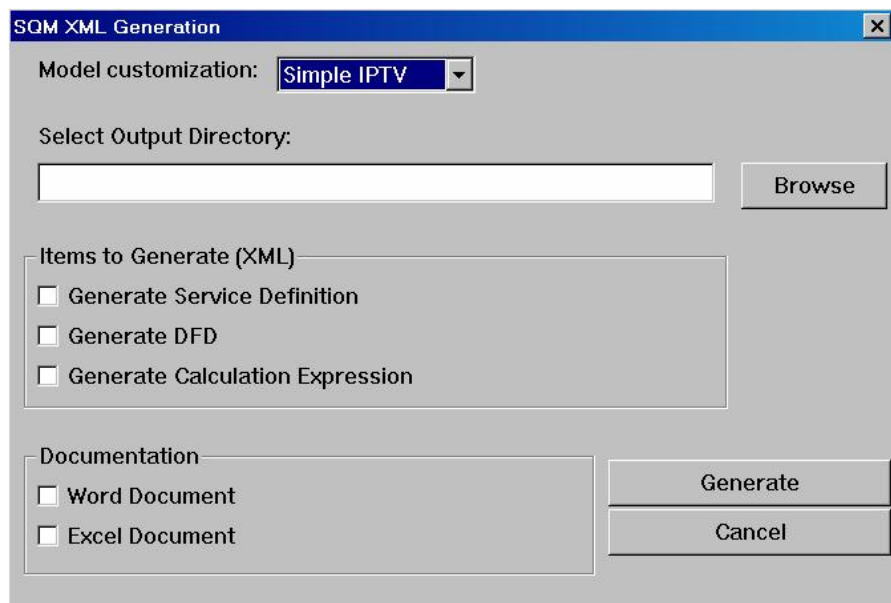
First, select the model customization to generate. If empty, model customization attributes will be ignored and the generated service will be the original one with every SCDs and parameters.
Then select the destination directory (Browse button). Every generated item (Documentation, SD, SCD, and Expression) will be generated in that directory. If empty, the default directory defined for the model is used (See Tools/Service Designer/Project Specification).

Note

In the file browser, you must enter a folder name to select the output directory.

Then select the items to generate.
Then click "Generate".

6.3 Customizing the Initial Framework

Although a pre-defined Initial Framework is provided with **OpenView Service Quality Manager Service Designer**, you might want to create a Custom Initial Framework that contains the specific elements that you always use to create new Services (Custom Expressions, a package with Data feeders or Service Components).

There are two ways to create a Customized Initial Framework:

- You can edit the model used in the Initial Framework provided,

or

- You can create a new model and use the Framework Wizard to create a **new** Initial Framework based on this model.
  (Recommended method)

6.3.1 Editing the Provided Framework

This is the simpler way to customize the initial framework, but if you do, you will no longer be able to retreive the original Initial Framework provided.

To do this:

1. Open the model used by the provided Initial Framework located in:

```
\Program
Files\Rational\Rose\framework\frameworks\SCServiceDesigner\SCSe
rviceDesigner.mdl
```

2. Add the elements required for each new Service in this model (New Custom Expression, Data Feeder and Service Components Packages and so on.)

3. Save the model.

All these new elements will then be available in each new model.

6.3.2 Creating a New Initial Framework

This is more complicated than the previous method, but it has the following advantages:

- You do not change the Initial Framework that is provided and can therefore still use it when you wish.

- You can create several Initial Frameworks for different categories of service.

You only need to do this if the Initial Framework provided with OpenView Service Quality Manager does not correspond to your needs.

6.3.2.1   To create a New Initial Framework

To do this you must first create a new model which will be the basis of each new Service.

To create a new model it is advisable to use the Initial Framework to create it, as you will benefit from the **Expression Classes** it contains.

Add all the elements required for each new Service to this model. (new Custom Expressions, Data Feeder and Service Component Packages and so on).

Save the model in a temporary location, for example: `C:\MyFramework.mdl`

You now use the **Framework Wizard** to create an Initial Framework based on this model.

To do this:

1. Ensure that the Framework Wizard is activated.

2. Click on **New** in the **File** menu, or

3. On the **page** icon.

4. If the Framework Wizard is activated, the Create New Model window will be displayed. This is illustrated in Figure 75 Create New Model Window.

Figure 75   Create New Model Window
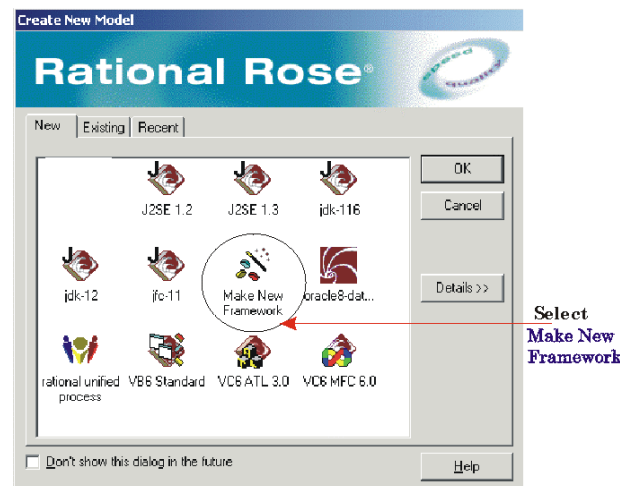
5. Select the **Make New Framework** icon and click on **OK**.

6. The Framework Wizard screen will be displayed.

Figure 76   Framework Wizard Welcome Screen
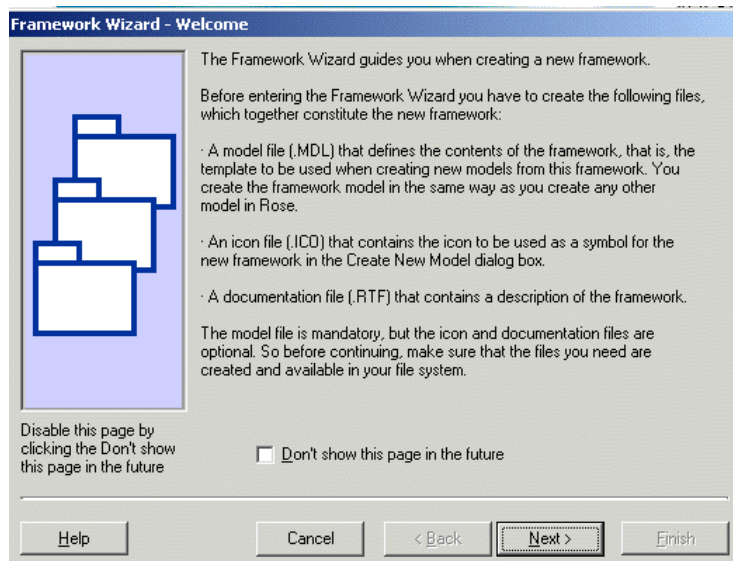
7. Follow the instructions given on screen. Click Next to continue.

8. The screen, **Framework Wizard, Specification of Framework** will be displayed.

Figure 77   Framework Wizard – Specification of Framework Window.
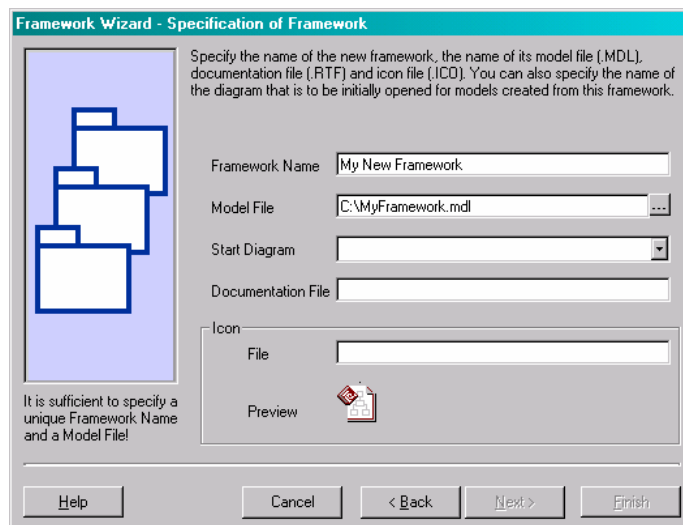
You must provide at least the **Name** of the new Initial Framework (free text) and the **model file location**. (The path to the model created as described at the beginning of Section 6.3.2.1.)

9. Click on Next.

A **Create New Model** window will be displayed containing the icon for your new Framework.

Figure 78   Create New Model Window
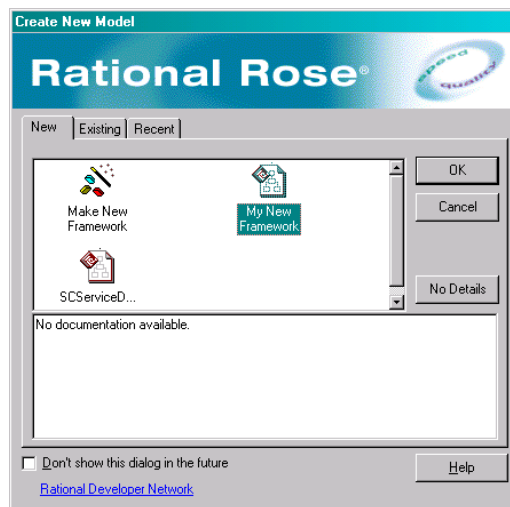
10. Click on the Icon for your New Framework to create a new Service, based on this new Initial Framework.

6.4 Creating an Expression Class

As for Enum, you must follow certain rules to create a valid Expression class.

The new Expression should be created under an existing Expression Hierarchy (provided by the Initial Framework).The goal of this hierarchy is to represent the different types of technology supported in **OpenView Service Quality Manager** (Java or PL/SQL). For each technology type there is a set of pre-defined Expressions and a set of user-defined expressions for the different types of supported Expression (Simple or Aggregate).

Each Expression class has several associated Operations. (Operations used in the Sequence Diagrams).

To Create an Expression Class

1. From the **Tools** menu, choose **Service Designer**

2. The Menu illustrated in Figure 79   Definition Menus will be displayed.

Figure 79   Definition Menus
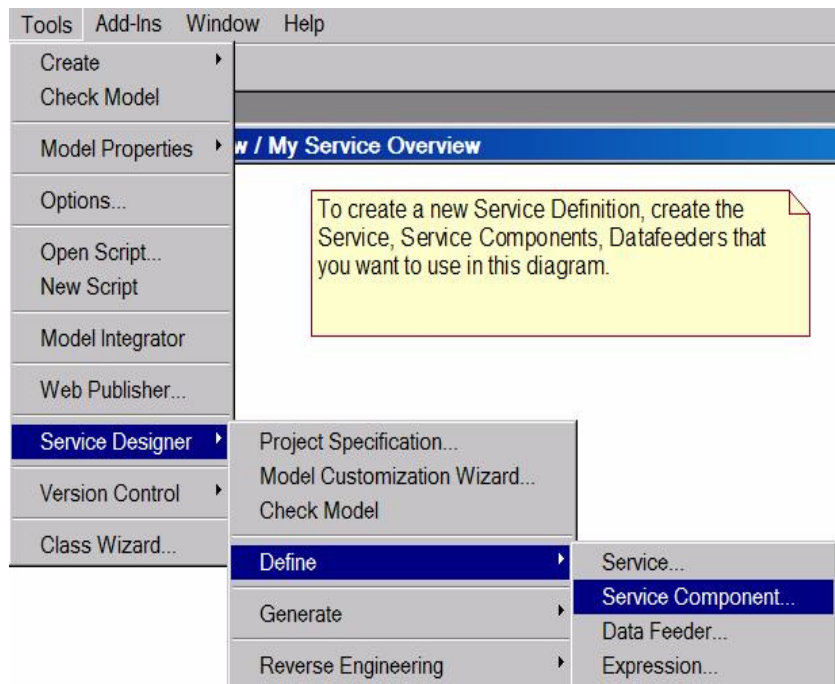
3. Choose **Define**, then **Define an Expression** from the next sub-menu displayed.

4. The **Expression Specification** Dialog box (Figure 80   Expression Specification Dialog Box, General Tab) will be displayed.

This Dialog box has two tabs, **General** and **Calculation Expressions**.

Figure 80   Expression Specification Dialog Box, General Tab
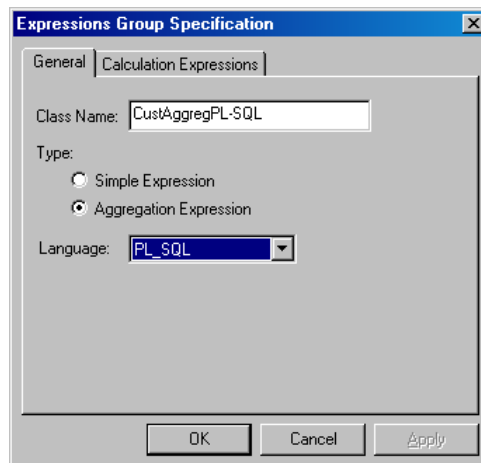
5. The **General Tab** allows you to choose

- The Language (PL/SQL or Java)

- The Name

And

- The Expression Type (Simple Expression or Aggregation Expression).

6. When you have made your choices here, click **OK**.

7. Now you can add the Calculation Expressions in this class. This is described in section 4.11.

6.5 Managing Component Libraries

One of the standard features of Rational Rose enables you to create your own Service Component or Data Feeder libraries. Rational Rose can keep a model in one or more files. These files are called Petal files and Controlled Unit files (Controlled Unit files are sometimes called Control Unit in Rational Rose.). Both Controlled Unit and Petal files are files into which Rational Rose stores all or part of a model. In this way it is possible to define a set of Service Components and keep them in Petal files or Controlled Unit files and import them and use them whenever you need.

Rose provides a rich set of features to manage the libraries of components. The rest of this section describes only the typical use of these Rational Rose features for the needs of Service Design.

6.5.1 Creating Libraries of Components

To do this:

1. Create a new model.

2. In the Logical view, create a new package.

3. Rename this new package.

4. Create a new class diagram in the new package.

5. Open this new class diagram.

6. In this class diagram create the component that will be part of your library (DFD/Service Component) Section 4.10.5 describes how to do this.

Depending on your needs, you can also create associations and sequence diagrams in this package.

Once the components of your library are fully defined, you must save the library in a Petal File or a Controlled Unit file.

6.5.1.1 Storing a Package in a Petal File

If you choose to store your package in a Petal file, the library that you have created will not be shared among different services. It is possible to import this library into a new service, but the changes performed in the package will not affect other services. This type of creation should only be used for Service Components whose definition is not shared (Shared Definition).

To do this:

1. Select the package in the Tree View.

2. From the File menu, select Export <Package Name>.

3. This displays a dialog box where you must enter the file name for your library. The file must have the extension .ptl.

6.5.1.2 Storing a Package in a Controlled Unit

If you choose to store your package in a Controlled Unit, you will be able to use it in a new service, and several services will be able to share these components. If you modify this package while editing a service, all the other services will be impacted. This type of creation should be done for Components that are "Shared Definition" components, and for Data Feeders.

To do this:

1. Right click on the package in the Tree View.

2. Select Units/Control `<Package Name>` from the pop-up menu that is displayed.

3. This displays a dialog box where you must enter the file name for your library. The file must have the extension `.cat`.

The location where you store your library is important both for Petal files and for Controlled Units. You must organize your directories in such a way that you will be able to find your libraries easily.

The contents of two libraries are illustrated in Figure 81 Contents of Two Libraries.

Figure 81   Contents of Two Libraries
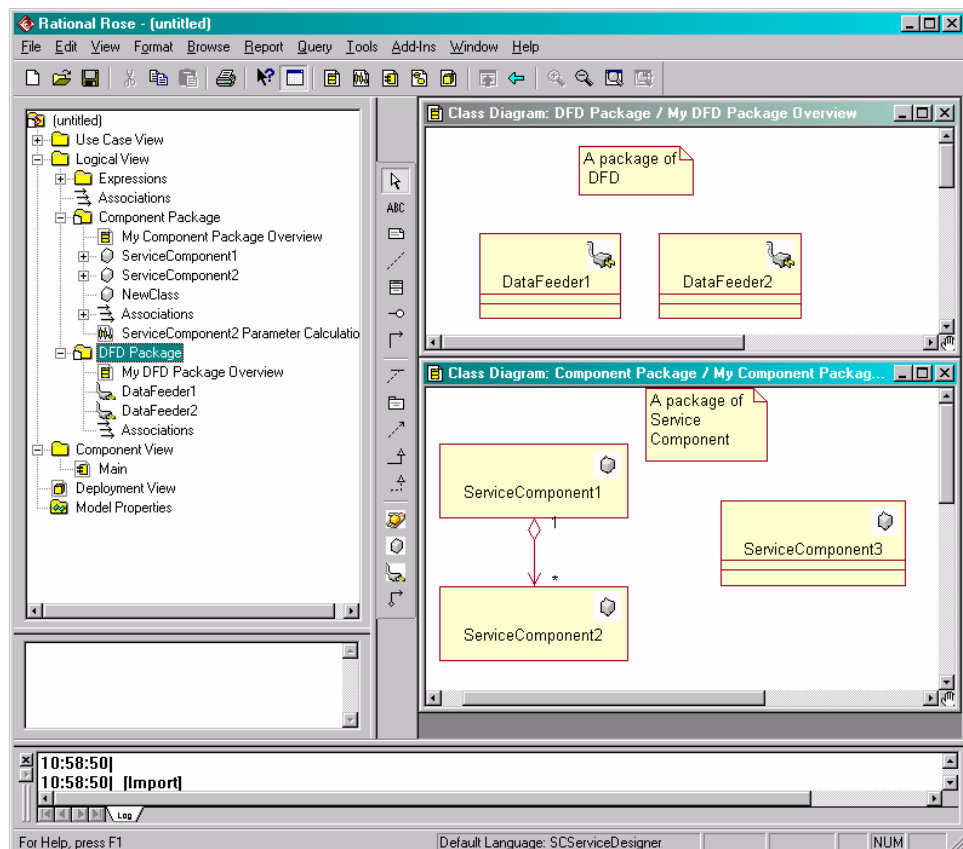
The icon representing the packages in the tree diagram indicates that the packages are Controlled Units.

6.5.2   Using Library Components in a Service

You can use either Petal Files or Controlled Units in a new model. As explained in section xxx, if you import a Petal File, the data stored in this Petal file will be local to the Service. You will be able to modify details without impacting other Services. On the other hand, Controlled Units are shared among all the services that use them; therefore you must be careful if you modify their contents. To use Petal files or Controlled Units you must import them.

Importing Petal Files and Controlled Unit Files

The Import process for both Petal Files and Controlled Units is the same.

You can:

1. Open the **Package Overview** class diagram provided by the Initial Framework. Although this step is not essential, when you do it, a **package icon** will be placed in this diagram.

Or

2. From the File menu, select **Import**.

3. A dialog box is displayed where you must select the type of file to import (`.cat or .ptl`) and select the file you want to import. Click on Open.

4. This will import the selected library and you can now use its Components to define a new Service.

6.5.3   Uncontrolling a Controlled Unit

There may be situations when you need to import a Controlled Unit to get the components defined in it, and edit them without impacting other services that use this Controlled Unit. To be able to do this you must **Uncontrol** the Controlled Units:

1. Import the Controlled Unit you need.

2. Right click on the package icon of this Controlled Unit in the Tree view.

3. Select the **Unit/Uncontrol <Package Name>** entry.

The Controlled Unit now becomes **local** to the Service into which you have imported it (the icon of the package changes) and you can now edit it without impacting the other Services. It remains a Controlled Unit in the other Services sharing its definitions.

6.6   Saving a Model

You can create a Petal file for the complete model by choosing **Export Model** from the Rose **File** menu. (The Export Model option is the option for saving the model.)

You will be prompted for the file name and for the directory where you want to store your Petal file. The **Save as type:** drop-down menu allows you to choose the file type (`.ptl` in Figure 82   Export Model (Save Model) Dialog Box).

Figure 82   **Export Model (Save Model) Dialog Box**
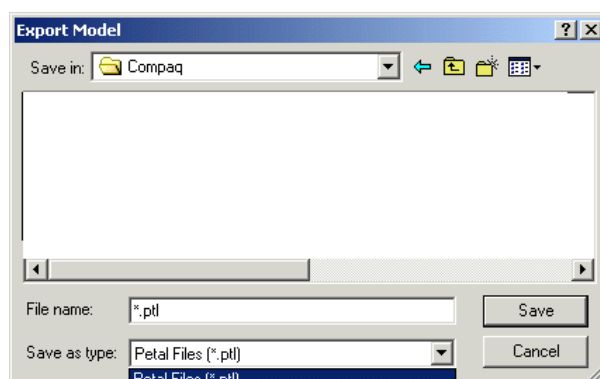
If you want to save only a part of the model in a separate file, select the Class for the part you want to save in the Class diagram window, or the Class icon in the tree diagram. The **File** menu will display **Export** and the name of the item you have selected. (**Export ServiceComponent** in Figure 83   Exporting (Saving) Part of a Model.)

Figure 83   **Exporting (Saving) Part of a Model**
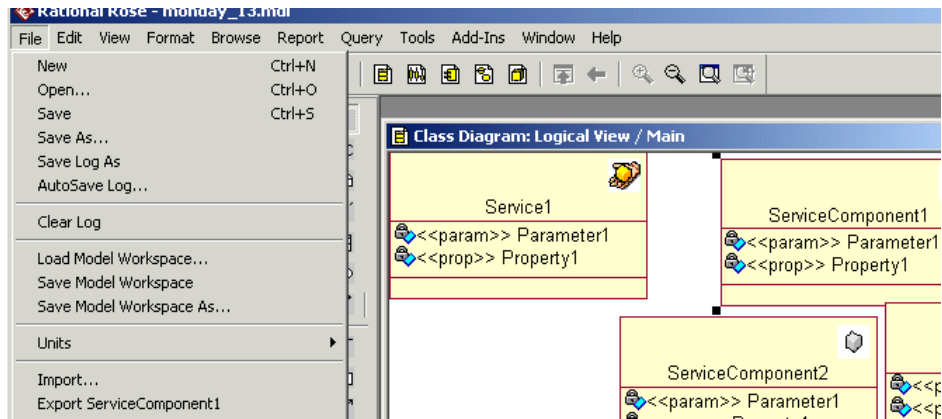
You are prompted for the name of the file in the same way as when you save (export) a Model. You must enter a file name and choose the directory where you want to store your files.

When you have created and stored your Petal files you can decide whether or not to designate them as Controlled Units by using the Version Control Add-In. Both Controlled Units and Petal files can be imported using the **File** menu **Import** option.

For more detailed information about Petal files and Controlled Units, use the Standard Rational Rose online Help. Select the topics: **Controlled Units** and **Petal File**.

Chapter 7

Troubleshooting

This chapter gives some help about how to deal with problems and gives advice about actions it is best to avoid when using Service Designer.

7.1   Activating the Framework Wizard Add-In

If you do not see the Initial Framework icon when you start Service Designer, you need to activate it.

You do this in the same way as activating the Service Designer Add-In.

1. Click on **Add-Ins** on the menu bar.

2. Select **Add-In** Manager.

3. The **Add-In Manager** dialog box will be displayed.

4. Check the box to the left of **Framework Wizard**.

5. Click on **Apply** and then on **OK**.

Figure 84   **Add-In Manager, Framework Wizard**
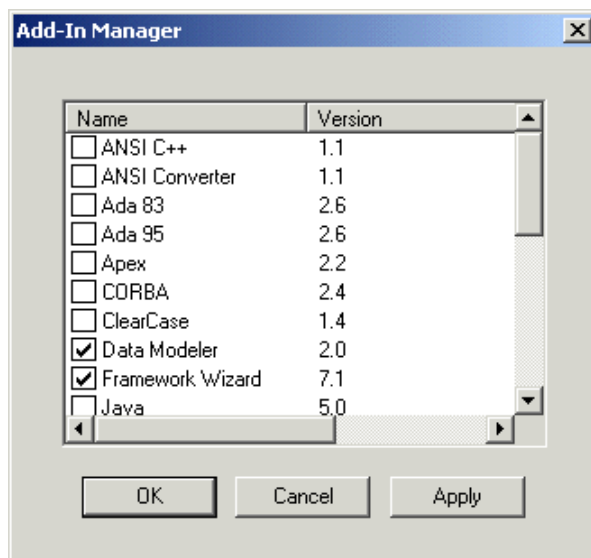
7.2 Service Designer Dialog Boxes are not Displayed

If you cannot display the Service Designer Dialog boxes this may be because the Service Designer Add-in has not been activated, or because there is a Standard Rose Dialog box open.

Therefore first ensure that the Service Designer has been activated. If it has not, follow the directions given in the previous section of this chapter, Section 7.1.

If you still cannot display a Service Designer Dialog box, check that you do not have a Rational Rose Standard dialog box open. If you have, close this dialog box and try to open the Service Designer dialog box again.

7.3 Error and Information Display

Errors and information are displayed in two ways, by Pop-up boxes and through the Log display in the console window.

7.3.1 Pop-up Boxes

Rational Rose provides **Information** and **Error** pop-up boxes whenever your actions require them.

7.3.2 The Log

This is a Rational Rose standard feature. The log records many of the actions performed, providing much useful information. You can choose to display this whenever you wish.

To display the log file, select **Log** from the **View** menu. The Log window will open underneath the principle Rational Rose window. It records the actions it displays with a timestamp. See Section 3.1.1

To close the log display, unselect **Log** from the **View** menu. The log window will close.

Figure 85   Example of the Log Window.
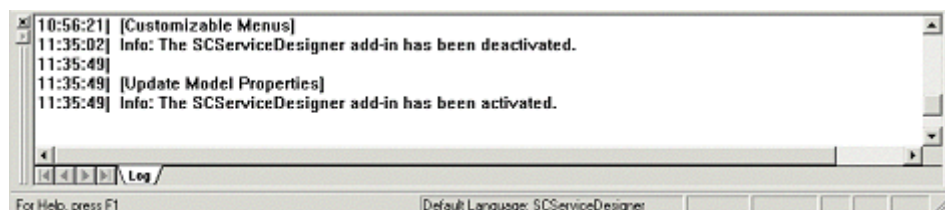
You can clear the log when you wish and you can save it as a file.

Figure 86   Saving a Log Dialog Box
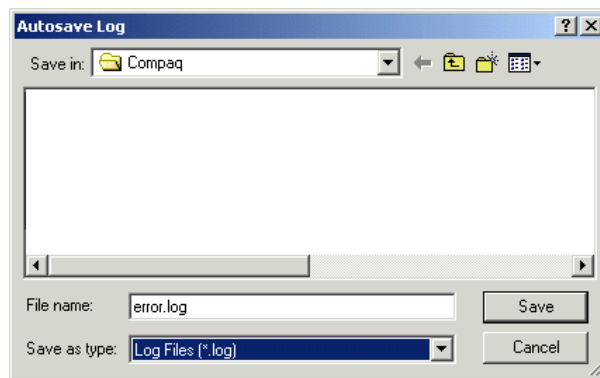
1. From the **File** menu, choose **Save Log As**, or **AutoSave Log**. Whichever you choose, the dialog box illustrated in Figure 86Saving a Log Dialog Box is displayed.

2. Choose **Save.**

If the saved log file already exists, you are asked if you want to replace it.

7.3.3   Correcting Mistakes

If you have made a mistake and created a Class that you do not want in your model, you will want to delete it.

7.3.3.1   Deleting Classes From a Model

Although you can remove a **Class representation** from the **Class** diagram window by clicking on the **Class** representation and pressing **Delete**, this does **not** delete the **Class** from the model, it only removes the **Class representation** from the display.

How to delete Classes from a model is described in Section 4.2

7.3.4   Creating and Defining Enumerations and Expressions

You must use the Rational Rose **Standard** dialog boxes to **create and define** Enumerations and Expressions.

7.3.5   Importing Expression XML Definitions into OpenView Service Quality Manager

When you generate an XML Definition for an Expression to import into Service Center, the document **should** contain the body of the Expression Operation. Refer to Section 4.11 for details of how to do this. If these body files have **not** been selected, only a 'skeleton' of the XML definition document will be generated.

7.3.6   Actions you are Advised Not to Perform

This section gives information about actions that are best avoided when using OpenView Service Quality Manager Service Designer.

7.3.6.1   Closing Rational Rose

- Do not use **Exit** from the file menu to close Rational Rose as this action de-activates Service Designer.

- If you do de-activate Service Designer, by any means, you must re-activate it to be able to use it. Choose the **Add-Ins** menu, choose **Add–In Manager**, and select

the box to the left of Service Designer in the displayed dialog box and click on
OK.

7.3.6.2   Do Not Change the Displayed XML Name

On several Definition screens, the internal XML name is displayed. Although it is
possible to change this name, you are advised **not** to do so, as changing the name can
lead to corrupt XML definitions. The XML name **must** remain the same from one
generation to another.

7.3.6.3   Pre-defined Expressions

You must **not** generate the pre-defined Expressions, or change them in OpenView
Service Quality Manager.

7.3.6.4   Do Not Use the Rational Rose Standard Dialogs to Modify

Rational Rose does not **prevent** you from using the standard dialogs to modify
elements **which have been created using Service Designer**.

However, you should **never** use Rational Rose Standard Dialogs to modify elements
which have been created using Service Designer as it causes consistency violation
in the model, and **the problem is only detected** during a consistency check or during
code generation.

You **must**, however, use the Rational Rose Standard dialog boxes to **create and
define** Enumerations, Expressions and Operations for Expressions.

7.3.6.5   Deleting an Operation

You must **never** delete an Operation linked to a Parameter **without** using the Service
Designer Dialog Boxes.

7.3.6.6   Do not Launch Rational Rose by Double-clicking on a Model
file (.mdl)

It is not advisable to launch Rational Rose by double-clicking on an `.mdl` (Model)
file from a Windows Explorer window, as this can launch Rational Rose **without** the
Service Designer icons in the Vertical toolbar.

Glossary

This glossary defines terminology commonly used in HP OpenView Service Quality Manager.

Auto instantiate (SLA Administration)

This action automatically creates an Instance of the Object selected. When the instance is created, the initial values of its instance variables are assigned.

BI

See business intelligence.

Business intelligence (BI)

A broad category of applications and technologies for gathering, storing, analyzing, and providing access to data that helps users make better business decisions.

CNM

See customer network management

Customer

Companies or organizations that make use of the *services* offered by a *service provider*, based on a contractual relationship.

Customer network management

Customer network management is enabled by means of tools that provide business customers with access to management information originating from the service provider.

Data collection interval

The interval of time over which performance parameters are retrieved from the monitored service resources. This interval does **not** have to be the same as the *measurement interval* because *service adapters* or service resources may buffer statistics.

Data feeder

OpenView Service Quality Manager's source of data. A data feeder models service resources by defining one or more service parameters.

Data feeder definition

The static definition of a data feeder modeling service resources by defining one or more service parameters.

Degraded service

The presence of anomalies or defects causing degradation of the *quality of service*, but do not result in the total failure of the *service*.

Instantiate (SLA Administration)

Instantiate differs from Auto Instantiate in that items are instantiated individually.

Measurement interval

The interval of time over which each service parameter is measured. For example, a parameter may be the number of discarded packets, measured over a 15-minute measurement interval.

Measurement Reference Point (MRP) naming scheme

This is the formal description of how the measurement point name is built, that is, by concatenating the values of Data Feeder properties and fixed strings.

Mobile virtual network operator

A mobile operator that does not own its own spectrum and usually does not have its own network infrastructure. Instead, MVNOs have business arrangements with traditional mobile operators to buy **minutes of use** for sale to their own customers.

MRP

See Measurement Reference Point.

MVNO

See mobile virtual network operator.

parameter

A value or set of values that are periodically updated and that help determine the quality of service.

Parameter instance

Service instances and Service Component instances consist of a set of parameter instances. The value of the parameter instance is provided either by a data feeder or by the parameters of other service components.

Parameter objective

A set of objectives for the parameters belonging to a service.

Property

Special static parameters that are given a value only when an instance of an OpenView Service Quality Manager **Object** is created. For example, a Service Component can have a property called "location".

QoS

See quality of service.

Quality of service (QoS)

The ITU-T has defined quality of service as "the collective effect of service performances that determine the degree of satisfaction of a user of the service".

Service

A Service is a set of independent functions (Service Components) that consist of hardware and software elements and an underlying communications medium. A Service can include anything from a single leased-line service, to a complex application, such as vision conferencing.

Service availability

A measurement made in the context of a *service level agreement* that is expressed as a percentage. This percentage indicates the time during which the *service* is operational at the respective *service access points*.

ServiceCenter Repository

The ServiceCenter Repository is the storage center for all Service Quality Manager data. It receives data from the various Service Quality Manager interfaces and each interface can request information from the Repository.

Service component

An independent function that is part of a *service*, such as a hardware or software element, or the underlying communications medium.

Service component instance

The instance of a Service Component Definition that is active in the network, such as an instance of the IPAccess Service Component definition called "pop".

Service level (SL)

Defines Service Parameters and operational data enforced by the Service Level Agreement (for example, Max Jitter < 10 ms).

Service Level Agreement (SLA)

There are two type of Service Level Agreement, the **Customer** Agreement: a contract between a *service provider* and a *customer*, which specifies in measurable terms what the service provider supplies to its customers, and the Operational Service Level Agreement, which specifies in measurable terms the operational levels of the Service. A *service level agreement* is composed of individual objectives.

Service Level Objective (SLO)

The set of objectives for the parameters belonging to a Service or Service Component.

Service parameter

See *parameter*.

Service provider

A company or organization that provides *services* as a business. Service providers may operate networks or may integrate the services of other providers.

Service instance (SI)

The instantiated service definition that is active in the network, such as an instance of the video service definition called "Paris".

Service instance group (SIG)

A **group** of *service instances* against which the *service availability* must be reported. Each *service instance* belongs to one or more Service Instance Groups and each SIG contains at least one Service Instance. The relationship between the SIG and the Service Instances is defined in their *service level agreement*.

SI

See Service Instance.

SIG

See Service Instance Group.

SL

See Service Level

SLA

See Service Level Agreement.

SLO

See Service Level Objective.

Subscriber

The entity responsible for the payment of charges incurred by one or more users.

User

An entity designated by a customer to use the services of a telecommunication network, such as a person using a UMTS mobile station as a portable telephone.