Rational Rose 2000e
Using Rose
Visual Basic

Contents

List of Figures

List of Tables

Preface

This manual:

- Describes the features of the Visual Basic Language Support add-in in Rational Rose
- Provides information and procedures for round-trip engineering—that is, Visual Basic code generation and reverse engineering
- Provides complete information on mapping UML to Visual Basic and Visual Basic to UML, as implemented by the Visual Basic Language Support add-in

This manual assumes that you are familiar with the Visual Basic programming language, the Windows operating system, object-oriented-design concepts, and how to use Rational Rose.

How this Manual Is Organized

This manual contains the following chapters and appendices:

- **Chapter 1**—Introduction

  Provides an overview of features and the basic round-trip engineering concepts as they apply to Rational Rose Visual Basic.
- **Chapter 2**—Mapping UML to Visual Basic

  Provides a detailed mapping between Visual Basic and UML constructs in both directions (Visual Basic <-> UML) as implemented in Rational Rose Visual Basic.

- **Chapter 3**—Round-Trip Engineering a Visual Basic Application

  Provides detailed procedures for generating or updating a Visual Basic project from a Rational Rose model, and how to update the model from, or reverse engineer, a Visual Basic project.
- **Chapter 4**—Generating Visual Basic Code

  Describes how to generate Visual Basic source code from elements in a Rational Rose model, using the Code Update Tool.
- **Chapter 5**—Reverse Engineering Visual Basic Code

  Describes how to update a Rational Rose model from changes in the Visual Basic source code, using the Model Update Tool.
- **Chapter 6**—Modeling a Visual Basic Project

  Explains the need for the component view in a model. This chapter also describes how to create Visual Basic components in a model, using the Component Assignment Tool, and how to assign classes to components. You can also find information about how to import and use type libraries in Rational Rose.
- **Chapter 7**—Modeling Visual Basic Classes

  Explains how to create and specify Visual Basic classes in a model, using the Model Assistant.
- **Appendix A—**Model Properties Reference

  Describes all model properties provided by the Visual Basic Language Support add-in in Rational Rose. The model properties are used by the Visual Basic code generator to determine what code to generate for each model element.
- **Appendix B**—UML to Visual Basic Mapping Quick Reference

  Provides quick reference tables that show the mapping between Visual Basic and UML constructs for code generation or reverse engineering.

Terminology

The Unified Modeling Language (UML) and the Visual Basic language refer to the same elements by different names. The user interface and documentation of the Visual Basic Language Support add-in tools uses the same terminology as Visual Basic—referred to as Component

Object Model (COM) terminology. In the rest of the Rational Rose user interface, you can choose between the UML or COM terminology. The following table summarizes these terminology differences:

Table 1 Terminology Differences in UML and Visual Basic

UML	**COM**	**Rational Rose**
Operation	Method	Operation or Method
Attribute	Property	Attribute or Property

To customize Rational Rose to use COM terminology in the user interface:

1. Exit Rational Rose.
2. Open rose.ini, which is located in the Rational Rose installation folder.
3. Search for "UseCOMTerminology" and set it to "Yes."

When you restart Rational Rose, the COM terminology is used in all specifications, dialog boxes, and menu items.

Related Documentation

The Visual Basic Language Support add-in provides comprehensive online help with hypertext links and a search index. To display an overview of the online help, click **Help > Rational Rose Help Topics**.

The Visual Basic Language Support add-in online help is found in the *Rose Visual Basic* online book. Information about the basic features of Rational Rose can be found in the *Rational Rose* online book, as well as in the *Using Rose* manual.

Online Help and Manuals

Rational Rose includes comprehensive online help with hypertext links and a two-level search index.

In addition, you can find all the user manuals online. Please refer to the Readme.txt file (found in the Rational Rose installation directory) for more information.

Chapter 1

Introduction to the Visual Basic Language Support Add-In

This chapter provides an introduction to the main features of the Visual Basic Language Support add-in in Rational Rose.

The Visual Basic Language Support add-in enables Rational Rose to model, generate, and reverse engineer code for applications written in Microsoft Visual Basic.

The Visual Basic Language Support add-in tools in Rational Rose are tightly integrated with the Microsoft Visual Basic environment, allowing you to seamlessly progress through the round-trip engineering process. The tools are:

- Class Wizard—helps you create and specify a new Visual Basic class in the model.
- Model Assistant—enables you to specify a Visual Basic class in your model with all the necessary code-specific semantics for complete and robust code generation.

  See the section *The Model Assistant* in chapter 7.
- Component Assignment Tool—provides you with an easy-to-use interface to create new components in the model, associate components with source code projects, and assign classes to components.

  See the section *The Component Assignment Tool* in chapter 6.
- Code Update Tool—generates and updates the Visual Basic source code from the information contained in a model, and preserves existing user-supplied definitions and declarations from the previous iteration's source code.

  See the section *The Code Update Tool* in chapter 4.

- Model Update Tool—extracts design information from the Visual Basic code and updates the application's design model.

  See the section *The Model Update Tool* in chapter 5.

All these tools, except the Model Assistant, are provided by the Visual Studio add-in in Rational Rose, which means that these tools may be shared by other language support add-ins as well.

Chapter 2

Mapping UML to Visual Basic

This chapter explains the mapping between the different elements of a Rational Rose model and the Visual Basic programming language, and vice versa. In *Appendix B* you can also find quick reference tables that show these mappings.

Overview

The code generated from each element in your model is determined by that element's specification and model properties. These properties provide the language-specific information required to transform your model into Visual Basic code.

For each class in a Rational Rose model, the code generator produces a corresponding Visual Basic project item. Class relationships—those representing aggregations and associations—are translated to data members, constants, class properties, or module variables, depending on the desired implementation strategy. Also, Rational Rose Visual Basic produces skeletal function procedures, or property procedures for the class methods (operations) in the model.

Stereotypes and Model Properties

The notations provided by Rational Rose Visual Basic are more abstract than the Visual Basic programming language. Some types of model elements do not have any correspondences in Visual Basic at all, but many of them result in several lines of Visual Basic code when generated.

There is a default mapping for each model element, but you can also control what code to generate by changing the stereotype and model properties for the model element using the Model Assistant (see chapter 7). When a model element is created, Rational Rose assigns each model property a default value, which you can optionally modify. By modifying the model properties, you can control the code that is generated for the model element.

The model properties for a model element are also available on the **Visual Basic** tab of the element's specification (see *Appendix A*). Note, however, when editing the model properties on the **Visual Basic** tab, no consistency checks are being made. Therefore, it is recommended that you use the Model Assistant to modify the model properties.

Component View to Visual Basic Mapping

The component view is used to map the logical view to Visual Basic projects. This section explains the need for components and how they are related to Visual Basic projects.

Please refer to the *Using Rose* manual for general information about the component view.

Components

A Visual Basic project is represented by a component in the component view of a Rational Rose model. The components are needed to map each class in the logical view to the appropriate implementation language and source code project (as illustrated in Figure 1).

You cannot generate Visual Basic code for a class until it has been assigned to one or several Visual Basic components in the model. The implementation language of a component is assigned in its Component Specification.

To reverse engineer or update a model from a Visual Basic project, a component corresponding to that project must exist in the model. For information on how to create components, see the section *Creating Components and Assigning Classes* in chapter 6.

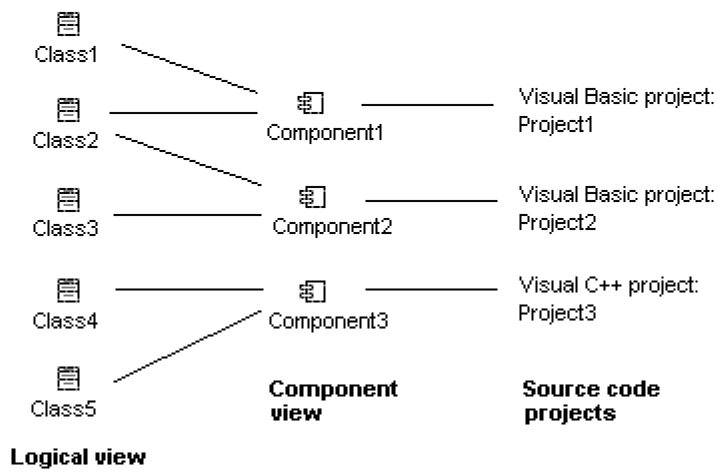


Figure 1   Components Map the Classes in the Model to Source Code Projects

A Component Is Associated with a Project

A component with the Visual Basic language assigned to it corresponds to a Visual Basic project. The physical instantiation of a component is the .exe, .dll, .tlb, or .ocx file that is produced from the associated project. The type of a component—for example, Standard EXE or ActiveX DLL—is specified by its stereotype. The component type corresponds to the type of the associated Visual Basic project.

The name and path to the component's project file is available in the **Visual Basic Component Properties** dialog box for the component. A component can only be related to one project, and the name of a component is the same as the name of the corresponding Visual Basic project. See the section *Associating a Component with a Visual Basic Project File* in chapter 6.

Components May Represent Referenced COM Components

Components are also used to represent COM components that the modeled system is using. (See the section *Using an Imported Type Library* in chapter 6.)

The components referenced by a Visual Basic project are automatically imported into the model when updating the model from the project. You can also import any COM component containing type information into Rational Rose (see section *Importing Type Libraries into the Model* in chapter 6.) A component representing the selected file is then created in the model, and the contents of the component's type library is inserted into a package in the logical view, as illustrated in Figure 2.

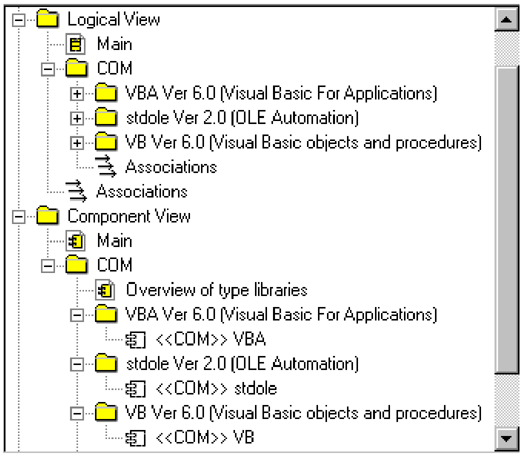

Logical View
Main
COM
VBA Ver 6.0 (Visual Basic For Applications)
stdole Ver 2.0 (OLE Automation)
VB Ver 6.0 (Visual Basic objects and procedures)
Associations
Associations
Component View
Main
COM
Overview of type libraries
VBA Ver 6.0 (Visual Basic For Applications)
<<COM>> VBA
stdole Ver 2.0 (OLE Automation)
<<COM>> stdole
VB Ver 6.0 (Visual Basic objects and procedures)
<<COM>> VB

Figure 2 Example of Imported COM Components

By establishing relationships between classes in the model and the imported type library items, you can show how classes and components depend on the imported COM components (see section *Using an Imported Type Library* in chapter 6.)

An association between a class and a type library item becomes a project reference to the corresponding COM component in the generated Visual Basic project. Also, a dependency relationship between a Visual Basic component in the model and COM component (or its interface) becomes a reference in the Visual Basic component's project.

Note: *No code will be generated for imported type libraries when you generate code from the model.*

A Component May Represent the Compiled Visual Basic Project

The type library of a COM component that has been compiled from a Visual Basic project can also be imported into the model. This allows you to show how other components (projects) in the model use this project. Because Standard EXE projects do not have a public interface, you can only import the type library of compiled ActiveX projects.

You can either import a project's type library automatically, when updating the model from the project, or manually; see *Importing the Binary Component Compiled from a Project in the Model* in chapter 6.

The type library of the compiled COM component is represented and used in the model the same way as any COM component; see *Importing Type Libraries into the Model* and *Using an Imported Type Library* in chapter 6.

Component Stereotypes

The stereotype and language of a component indicates to what kind of software module the component corresponds. To generate a Visual Basic project from a component, the language of the component must be set to "Visual Basic", and the stereotype must be set to one of the values in Table 2.

The following component stereotypes can be used when generating or reverse engineering Visual Basic code:

Table 2 Visual Basic Component Stereotypes

Stereotype	Visual Basic Mapping
none	When generating code for a Visual Basic component without a stereotype, the code generator creates an ActiveX DLL project and sets the stereotype to ActiveX DLL.
ActiveX	When generating code for a Visual Basic component with the stereotype ActiveX, the code generator creates an ActiveX DLL project and changes the stereotype to ActiveX DLL.
ActiveX Control	Represents a Visual Basic project of the type ActiveX Control.
ActiveX DLL	Represents a Visual Basic project of the type ActiveX DLL.

Table 2 Visual Basic Component Stereotypes

Stereotype	Visual Basic Mapping
ActiveX EXE	Represents a Visual Basic project of the type ActiveX EXE.
COM	Represents a COM component (`.exe`, `.ocx`, `.olb`, `.dll`, or `.tlb` file) that has been imported into the model. Thus, such a component corresponds to an external software module, and not to a Visual Basic project. However, for ActiveX projects, the type library of the compiled COM component can be imported into the model.
DLL	When generating code for a Visual Basic component with the stereotype DLL, the code generator creates an ActiveX DLL project and changes the stereotype to ActiveX DLL.
EXE	When generating code for a Visual Basic component with the stereotype EXE, the code generator creates a Standard EXE project and changes the stereotype to Standard EXE.
Standard EXE	Represents a Visual Basic project of the type Standard EXE.
any other value	When generating code for a Visual Basic component with any other stereotype value, the code generator creates an ActiveX DLL project and sets the stereotype to ActiveX DLL.

Note that Rational Rose Visual Basic understands all the component stereotypes above, even though some of them may not be available by default in the **Stereotype** box in the Component Specification.

Component Packages

Component packages have no direct mapping to Visual Basic code.

Dependency Relationships

A dependency relationship between a Visual Basic component in the model and an imported COM component (or its interface) generates a reference in the Visual Basic component's project.

Logical View to Visual Basic Mapping

This section describes the purpose and mapping of the logical view in UML to the Microsoft Visual Basic language for each type of model element. That is, for:

- Classes, interfaces, class utilities, and logical packages
- Properties (attributes)
- Relationships
- Methods (operations)

Please refer to the *Using Rose* manual for general information about the model elements in the logical view.

Classes

A class is a set of objects that share a common structure—properties (attributes) and relationships—as well as a common behavior—methods (operations). A class is by default generated as a Visual Basic class module.

The Stereotype of a Class Defines Its Implementation Type

The stereotype of a class corresponds to a template. The template defines to what kind of Visual Basic project item a class corresponds—for example, a class module or a form. The template initializes the class with the members that are typical for that kind of project item. You assign the stereotype of a class in the Model Assistant or on the General tab in the Class Specification.

Table 3 shows the most important and fundamental stereotypes (templates) for Visual Basic classes in the model. For information about other templates, refer to the description of each template, which is displayed on the Template tab in the Model Assistant.

If the stereotype value is empty or unknown to the Visual Basic code generator (that is, not any of the values in Table 3) Rational Rose generates a class module for the class, and a module definition for a class utility. If you want a class to be generated into some other type of Visual Basic element, you have to change its stereotype. Note that once you have generated a class, Visual Basic will not let you alter its implementation type.

Table 3 Visual Basic Class Stereotypes

Stereotype	Project Item Type	Usage
AddinDesigner	Addin Designer	For generation and reverse engineering of Addin Designers in Visual Basic.
Class Module	Class Module	For generation and reverse engineering of Class Modules in Visual Basic.
Collection	Collection	For generation of user-defined collection classes. Note that when reverse engineering a collection class, Rose creates a class with the stereotype Class Module in the model.
Custom WebItem	Custom Web Item in Web Class	For code generation and reverse engineering of web items in a web class in Visual Basic. A class with this stereotype must be nested within the web class where it belongs. That is, it appears on the Nested tab of the web class's Class Specification.
DataEnvironment	Data Environment	For generation and reverse engineering of Data Environments in Visual Basic.
DataReport	Data Report	For generation and reverse engineering of Data Reports in Visual Basic.
Debug. ClassIdGenerator	Debug. ClassIdGenerator	Represents a class identifier generator module which is used by the debug code that is automatically generated by Rose Visual Basic if the Generate debug code option is selected.
Debug. ErrorHandling	Debug. ErrorHandling	Represents an error-handling module which is used by the error-handling code that is automatically generated by Rose Visual Basic if the Generate error handling code option is selected.

Table 3 Visual Basic Class Stereotypes

Stereotype	Project Item Type	Usage
DTHMLPage	DHTML Page	For generation and reverse engineering of DHTML Pages in Visual Basic.
Enum	Enum	For generation and reverse engineering of enum declarations. A class with the stereotype Enum must be nested within another class. That is, it is defined on the Nested tab of the parent class's Class Specification.
Form	Form	For generation and reverse engineering of Forms in Visual Basic.
Interface	Public Class Module in the modeled Visual Basic project or Interface or dispinterface of an imported COM component	When generating code for a class with the stereotype Interface, which is assigned to a Visual Basic component, Rose creates a class module with the instancing property Public in the generated project (if the project is not a Standard EXE project.) When reverse engineering a project, Rose automatically creates interface classes for the interfaces and dispinterfaces of the project's references.
Module	Module	For generation and reverse engineering of Modules in Visual Basic.
MDI Form	MDI Form	For generation and reverse engineering of Modules in Visual Basic.
Property Page	Property Page	For generation and reverse engineering of Property Pages in Visual Basic.

Table 3 Visual Basic Class Stereotypes

Stereotype	Project Item Type	Usage
Template WebItem	HTML Template Web Item in Web Class	For code generation and reverse engineering of web items in a web class in Visual Basic. A class with this stereotype must be nested within the web class where it belongs. That is, it appears on the Nested tab of the web class's Class Specification.
Type	Data type declaration	For generation and reverse engineering of data type declarations. A class with the stereotype Type must be nested within another class. That is, it appears on the Nested tab of the parent class's Class Specification.
UserConnection	User Connection	For generation and reverse engineering of User Connections in Visual Basic.
User Control	User Control	For generation and reverse engineering of User Controls in Visual Basic.
User Document	User Document	For generation and reverse engineering of User Documents in Visual Basic.
WebClass	Web Class	For generation and reverse engineering of web classes in Visual Basic. The web items are represented in the model as classes nested within the web class.

Classes Must Be Assigned to Components

To generate Visual Basic code for a class, the class must be assigned to a component with Visual Basic as the implementation language of that component. A class can be assigned to, and included in, several Visual Basic components. See the section *Creating Components and Assigning Classes* in chapter 6.

Code Generated for Classes

For each class using the default mapping as a class module, Rational Rose produces the following code constructs:

- A class definition taken from the name and model properties of the class
- A code comment extracted from the Documentation box of the Class Specification
- Module variables generated from the class's properties and relationships
- Method declarations, including skeletal method bodies, for all user-defined methods and property procedures
- Debug code for class modules

Visual Basic supports public and private access control and visibility forms. Therefore, each property, relationship, and method in the model will be mapped appropriately to the corresponding visibility tag in Visual Basic. Protected methods in Rational Rose are mapped as friend methods in Visual Basic.

Caution: *For each generated member, type, type field, and method, Rational Rose adds an identifier (a Model ID) as a code comment—for example "ModelID=3237F8CE0053"—which identifies the corresponding class, property, role, or method in the model. Do not edit or copy those identifiers!*

Naming of Classes

When naming a class, it is recommended that you use only class names that are accepted by Visual Basic. However, the Model Assistant and the Visual Basic code generator check for invalid characters or length of class names and transform them into valid characters.

Abstract Classes

You can define a class (interface) to be abstract on the Class tab in the Model Assistant or the Detail tab of the Class Specification. The Visual Basic code generator creates empty method bodies for abstract classes. Also, the code generator does not generate delegation code for a generalize relationship between a class and an abstract class.

Example of Code Generated for a Class

The following example illustrates the Order class, a typical class from the Business Service layer, and the supporting Visual Basic class module definition:

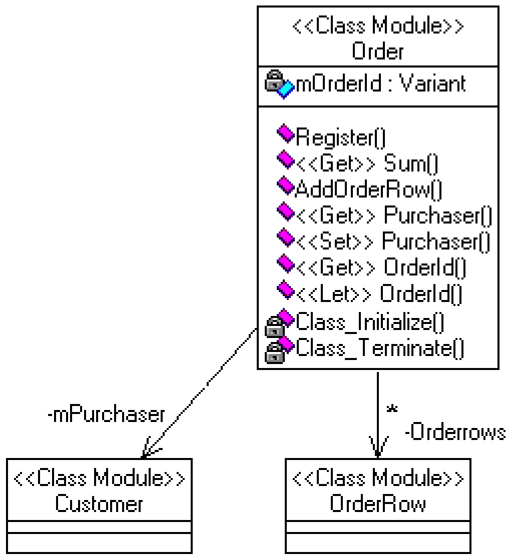


Figure 3 Example of a Class

```
Public Orderrows As Collection 'of OrderRow
Private mPurchaser As Customer
Private mOrder_Id As Variant

' Stores the order, initiates shipping and invoice.
Public Sub Register( )
...
End Sub

' Calculates the total sum of the order.
Public Property Get Sum( ) As Currency
      Dim tempSum As Currency
      Dim thisOrderRow As OrderRow
      'Initilize the sum
      temp_sum = 0
      'Iterate over the order rows and calculate the sum
      For Each thisOrderRow In Orderrows
            tempSum = tempSum + thisOrderRow.Sum
      Next
         'Return the calculated sum
      Sum = tempSum
End Property

' Adds an order row to the order.
Public Sub Add_OrderRow( new_order_row As OrderRow)
      Orderrows.Add new_order_row
End Sub

Public Property Get Purchaser( ) As Customer
      Set Purchaser = mPurchaser
End Property

Public Property Set Purchaser( Value As Customer)
      Set mPurchaser = Value
End Property
```

Interfaces

An interface in Rational Rose is a class with the stereotype Interface. Interfaces in a model of a Visual Basic application are used for two different purposes:

- An interface that is assigned to a Visual Basic component represents a public class module in the corresponding Visual Basic project. This kind of interface is used to model the interface of an

ActiveX project. (It is not meaningful to model interfaces of Standard EXE projects, as those projects cannot have public class modules.)

Note that these are UML interfaces rather than COM Interfaces. In fact, a public class module in Visual Basic corresponds to both a coclass and a default interface in COM.

- An interface that is assigned to an imported COM component represents a COM interface in the type library of that component (see section *Importing Type Libraries into the Model* in chapter 6).

  Note that these kind of interfaces are pure COM Interfaces, which are always abstract.

For detailed information about COM, refer to:

- Ted Pattison, *Programming Distributed Applications with COM and Microsoft Visual Basic 6.0*, Microsoft Press, ISBN 1-57231-961-5
- Don Box, *Essential COM*, Addison-Wesley Pub Co, ISBN 0201634465
- The MSDN Online Library, which can be found at http://msdn.microsoft.com/library—for example, the *Inside OLE* in the *Books* section

Example of a Modeled Public Class Module

The following example shows an interface class that is defined in a model and generated as a public class module in a Visual Basic project. The example is a shared Reporter component that provides a general reporter interface, which can be further specialized by other components and applications. The GeneralReporter interface in Figure 4 is defined in a model called MyComponentModel.

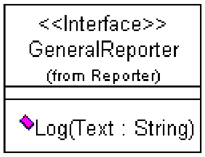


Figure 4 The GeneralReporter Interface

The GeneralReporter interface is assigned to the Reporter component, as illustrated in Figure 5. The Reporter component has the stereotype ActiveX DLL and the implementation language Visual Basic, which means that the component corresponds to an ActiveX DLL project in Visual Basic.

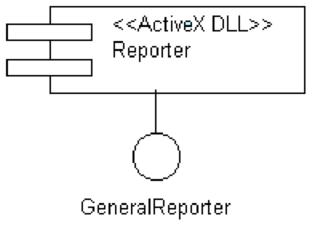


Figure 5 The Reporter ActiveX DLL Component

When generating code for the Reporter component, the code generator creates an ActiveX DLL project with a public class module, GeneralReporter.

When the developer compiles the project, Visual Basic creates a DLL, called Reporter.dll, which can be referenced from other Visual Basic projects and imported as COM components into other models, which is examplified below.

Example of an Imported Type Library

A Standard EXE application, for example MyApplication, that needs a reporter can reference and use the reporter DLL that was created from the Reporter project. The type library of the Reporter.dll component can then be imported and used in the model of MyApplication (which can be the same model as where the Reporter component is modeled.)

The component diagram in Figure 6 illustrates how the DLL has been imported into the model of MyApplication. The dependency relationship shows that the MyApplication component (and project) depends on the imported Reporter component.

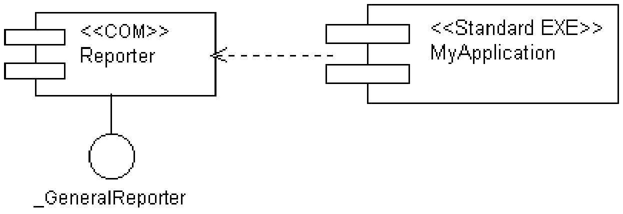


Figure 6 The Reporter DLL Has Been Imported into the Model of MyApplication

The public class module, GeneralReporter, has been imported as both a COM interface, _GeneralReporter, and a coclass, GeneralReporter, in the new model.

MyApplication provides its own implementation of the reporter, which is defined by a class called MyReporter. The realize relationship should be used to specify that a class realizes a certain interface. Therefore, the MyReporter class should have a realize relationship with the imported _GeneralReporter interface. However, because Visual Basic assumes the default interface when implementing a class, and the code generated for a realize relationship with a coclass or its default interface, gives the same compiled result, it is recommended that you create the realize relationship with the coclass. Thus, the MyReporter class has a realize relationship with the imported GeneralReporter coclass, and not the default interface, as illustrated in Figure 7.

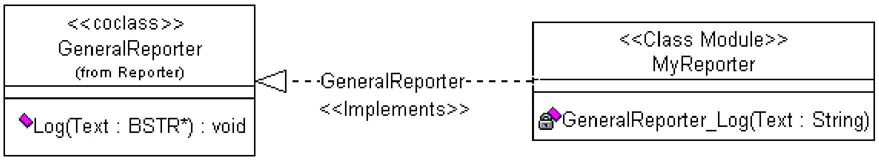


Figure 7 The MyReporter Class Realizes the Imported GeneralReporter Interface

As you can see in Figure 7, the realized method, Log, has automatically been added to the MyReporter class.

Collection Classes

Collections provide methods for adding and removing objects in a collection. They also provide behavior for processing the entire collection, such as behavior for iterating the entire collection using the For Each...Next statement in Visual Basic. A collection object in Visual Basic is created in the same way as an ordinary object. For example:

```
Dim myDatabases As New Databases
```

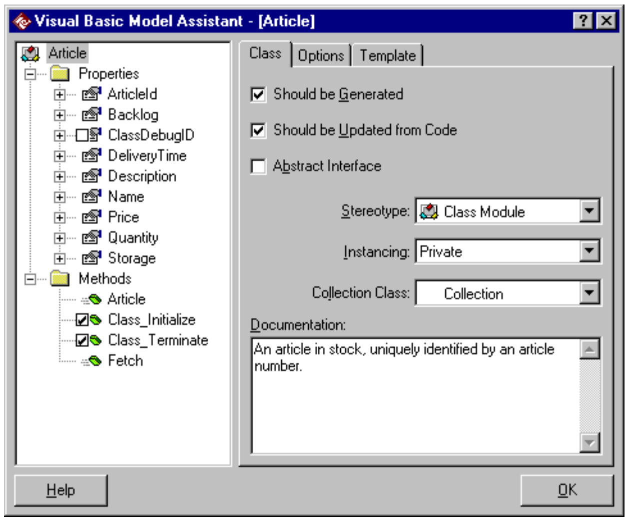
Visual Basic provides a pre-defined standard collection object, Collection, which can be used to create simple collections. However, for collections with more complex behavior, the methods provided by the standard collection are not sufficient. For example, an order administrator may need to know how many orders were received during a specified period. In that case, it is natural to let a collection object, Orders, provide the behavior needed to collect that information.

Collection Classes in Rational Rose

A collection object is modeled as a class with the stereotype Collection in Rational Rose. The class template for a collection class contains the standard collection methods—add, item, remove, count, and enum.

Each Class Has a Collection Class Assigned to It

To each Visual Basic class in the model, there is a collection class assigned—either a user-defined collection class or the standard collection, which is called Collection. The collection class currently assigned to a class is determined by the **Collection Class** option on the **Class** tab in the **Model Assistant** dialog box (Figure 8.)



- [✓] Should be Generated
- [✓] Should be Updated from Code
- [ ] Abstract Interface

Stereotype: Class Module
Instancing: Private
Collection Class: Collection
Documentation: An article in stock, uniquely identified by an article number.

Figure 8 The Article Class Has the Standard Collection Assigned to It

Rational Rose uses the name of the collection class when generating code for an association with unbounded multiplicity. See the *Association Relationships* section in this chapter.

By default, Rational Rose assigns the standard collection class, Collection, to new classes in the model, but you can assign any class with the stereotype Collection to a class.

Note: *The standard collection, Collection, is not explicitly represented in the model. Only user-defined collection classes are represented in the model.*

Collection Classes Can Be Automatically Created

The Model Assistant automatically creates a user-defined collection class when you type a new name in the **Collection Class** option for a class. See the section *Creating User-Defined Collection Classes* in chapter 7.

The created collection class gets the stereotype and class template Collection. A dependency relationship with the stereotype Collection is also created between the two classes.

Code Generated for Collection Classes

The following code is generated by default for the collection class, Orderrows, in Figure 9.

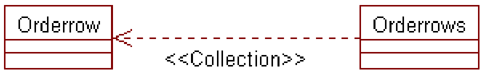


Figure 9   A Collection of Orderrow Objects

```
Private mCol As Collection
Public Property Get Item(vntIndexKey As Variant) As Orderrow
      Set Item = mCol(vntIndexKey)
End Property

Public Sub Remove(vntIndexKey As Variant)
      mCol.Remove vntIndexKey
End Sub

Public Sub Add(Item As Variant, Optional Key As Variant,
Optional Before As Variant, Optional After As Variant)
   If IsMissing(Key) Then
     mCol.Add Item
   Else
     mCol.Add Item, Key
   End If
End Sub

Public Property Get Count() As Long
   Count = mCol.Count
End Property

Public Property Get NewEnum() As IUnknown
```

```
   Set NewEnum = mCol.[_NewEnum]
End Property

Private Sub Class_Initialize()
   Set mCol = New Collection
End Sub

Private Sub Class_Terminate()
   Set mCol = Nothing
End Sub
```

Class Utilities

A class utility denotes a set of services provided by a module in the application under construction. A class utility can therefore be used to collect a set of free methods and data types. For instance, consider a collection of subprograms from Window 3.1 kernel, (for example, GetProfileString and WriteProfileString) that manipulate the **`win.ini`** configuration file. These can be gathered together into a class utility.

A class utility is mapped as a module in Visual Basic. Its properties (attributes) are mapped as public or private module variables, and the methods (operations) are mapped as public or private module methods.

Code Generated for Class Utilities

For each class utility, Rational Rose Visual Basic produces the following code constructs:

- Class annotations extracted from the class specification.
- Module variable declarations that are generated from the class properties and association relationships.
- User-defined function procedure declarations that are defined in the class specification, and skeletal function procedure bodies.

The following example shows the mapping of utilities to Visual Basic code.

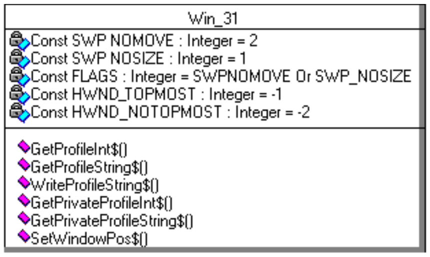

Win_31
Const SWP NOMOVE : Integer = 2
Const SWP NOSIZE : Integer = 1
Const FLAGS : Integer = SWPNOMOVE Or SWP_NOSIZE
Const HWND_TOPMOST : Integer = -1
Const HWND_NOTOPMOST : Integer = -2
GetProfileInt$()
GetProfileString$()
WriteProfileString$()
GetPrivateProfileInt$()
GetPrivateProfileString$()
SetWindowPos$()

Figure 10 Example of a Class Utility

```
Const SWP_NOMOVE = 2
Const SWP_NOSIZE = 1
Const FLAGS = SWP_NOMOVE Or SWP_NOSIZE
Const HWND_TOPMOST = -1
Const HWND_NOTOPMOST = -2

Declare Function GetProfileInt% Lib "Kernel" _
(ByVal lpAppName$, ByVal lpKeyName$, ByVal nDefault%)

Declare Function GetProfileString% Lib "Kernel" _
(ByVal lpAppName$, ByVal lpKeyName$, ByVal lpDefault$, ByVal
lpReturnedString$, ByVal nSize%)

Declare Function WriteProfileString% Lib "Kernel" _
(ByVal lpAppName$, ByVal lpKeyName$, ByVal lpString$)

Declare Function GetPrivateProfileInt% Lib "Kernel" _
(ByVal lpAppName$, ByVal lpKeyName$, ByVal nDefault%, ByVal
lpFileName$)

Declare Function GetPrivateProfileString% Lib "Kernel" _
(ByVal lpAppName$, ByVal lpKeyName$, ByVal lpDefault$, _
ByVal lpReturnedString$, ByVal nSize%, ByVal lpFileName$)

Declare Function WritePrivateProfileString% Lib "Kernel"
_(ByVal lpAppName$, ByVal lpKeyName$, ByVal lpString$, ByVal
lpFileName$)

Declare Function SetWindowPos$ Lib "user"
(ByVal h%, ByVal hb%, ByVal x%, ByVal y%, ByVal cx%, ByVal
cy%, ByVal f%)
```

Enum and Type Declarations

A class with the stereotype Enum or Type, which is nested within another class in the model, corresponds to an Enum or Type declaration in the other class.

The Enums and Types for a class can be displayed in several places in Rose. The Enums and Types are shown:

- On the Nested tab of the parent class's Class Specification
- In the browser under the parent class
- Under the Enums and Types folders in the Model Assistant

For more information about enums and types, see the section *Creating Enums and Types* in chapter 7.

Web Classes

A web class is modeled as class with the stereotype WebClass. The web class's web items are modeled as nested classes with the stereotype Custom WebItem or Template WebItem. You can view these nested classes from several places in Rose. The web items are shown:

- On the Nested tab of the web class's Class Specification
- In the browser under the web class
- Under the Web Items folder in the Model Assistant

Rational Rose generates Visual Basic code for web classes and web items. However, when generating a web class, Rational Rose does not generate any code for any members on the web class and its web items, except for the web items' events. Thus, properties (attributes), relationships, non-event methods, and non-event nested classes are ignored by the Visual Basic code generator.

For information on how to create web classes and web items, see the section *Modeling Web Classes* in chapter 7.

Logical Packages

A logical package is a group of strongly related classes. Each package represents a chunk of the logical architecture of the system, and declares its dependencies to other packages using a dependency diagram as shown below.

The dependency relationships indicate that the classes contained by the BO_ORDER package use the classes exported by the BO_CUSTOMER and BO_ARTICLE packages. The classes exported by the RDO and VB5 packages are visible to all classes in the model, because these packages are declared as global (by clicking **Global** on the Package Specification's **Detail** tab.) Violations to the visibility rules, that is the system architecture, are checked by Rational Rose Visual Basic.

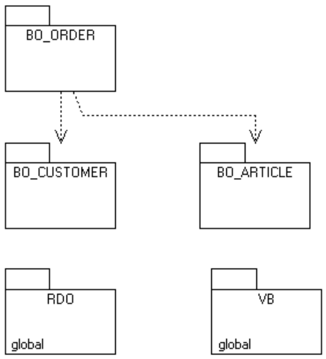


Figure 11 Example of a Diagram Showing Dependencies Between Logical Packages

Logical packages have no direct mapping to Visual Basic code, but they are used to represent imported COM components in the model. See the section *Components* in this chapter.

Properties (Attributes)

A property (attribute) is a data member of a programming-language type that is not a class. If a "property" is an object, it should be modeled as an association to the corresponding object class. For information on how to create properties, see the section *Creating Properties (Attributes)* in chapter 7.

The properties of a class in the model are usually translated into data members in the code. However, a derived property—that is, a property with the Derived option selected—corresponds to a Property Get method instead of a data member. Also, a property in the model may correspond to a constant in Visual Basic. Such a property has the stereotype Const. For information on how to create constants, see the section *Creating Constants* in chapter 7.

Property Procedures may be Associated to Properties

Private properties may have Property Get and Set procedures associated to them. In the Model Assistant you can select the appropriate Property Get or Set procedures for each property. The Model Assistant maintains the relation between a property and its property procedures. Thus, the Model Assistant lets you manage the property and its property procedures together. For example, if you remove a property in the Model Assistant, Rational Rose automatically removes the associated property procedures. For information on how to associate property procedures with properties, see the section *Creating Property Get, Let, and Set Procedures* in chapter 7.

Code Generated for Properties

For a private property, Name, on a class Customer, the following code is generated into the Customer class module:

```
Private Name As Variant
```

If a property Get procedure is associated to the property in the model, the following code is generated:

```
Private mName As Variant
Public Property Get Name() As Variant
   Name = mName
End Property
```

Note that the Model Assistant and code generator automatically changes the name of the property to mName to avoid a name collision with the property procedure.

Also, in each generated data member, Rational Rose adds a unique identifier—a Model ID—to be able to identify the corresponding property in the model.

Property Stereotypes

The stereotype of a property controls the Visual Basic code that Rational Rose produces for the class. You can assign a stereotype to a property in the Model Assistant, by creating the property as a constant, or on the **General** tab of its **Property Specification**.

Table 4 shows which stereotypes of properties are used in Rational Rose when generating or reverse engineering Visual Basic code.

Table 4 Property Stereotypes

Stereotype	Visual Basic Mapping
none	Represents a data member.
Const	Represents a Visual Basic constant. This stereotype is only relevant for properties with an initial value assigned.

Association Relationships

An association is a bi-directional class relationship that denotes a semantic dependency between two classes. Associations map to pairs of data members in Visual Basic. The name for each data member is taken from its association role name.

Associations can be described in detail by assigning a variety of adornments and properties. These adornments include for example: association direction, roles, multiplicity, navigability, aggregate, access, containment, role documentation, and qualifiers. All these adornments are used by the Visual Basic code generator to determine what code to generate.

Property Procedures may be Associated with Roles

Private roles may have Property Get and Set procedures associated with them. In the Model Assistant you can select the appropriate Property Get or Set procedures for each role. The Model Assistant maintains the relationship between a role and its property procedures. Thus, if you remove an association in the Model Assistant, Rational Rose automatically removes any associated property procedures.

For information on how to associate property procedures with roles, see the section *Creating Property Get, Let, and Set Procedures* in chapter 7.

Code Generated for Association Roles

For the private role Purchaser, on an association between an Order and Customer class, the following code is generated into the Order class module:

```
Private Purchaser As Customer
```

If a property Set procedure is associated to the Purchaser role in the model, the following code is generated:

```
Private mPurchaser As Customer
   Public Property Set Purchaser(value As Customer)
End Property
```

Note that Rational Rose automatically changes the name of the role to mPurchaser to avoid a name collision with the property procedure. For more information, see the section *Modifying the Default Data Member Prefix* in chapter 7.

If the name of the referenced class is not unique, you may need to use both the class name and the reference name. The FullName model property for roles is used to instruct the code generator to include the component name in the generated declaration. For example, if the Customer class is assigned to the OrderSys component, Rational Rose Visual Basic generates the following code if FullName is set to TRUE:

```
Private Purchaser As OrderSys.Customer
```

Moreover, in each generated data member, Rational Rose adds an identifier—a Model ID—to be able to identify the corresponding role in the model.

Code Generated for an Association with Unbounded Multiplicity

When generating code for an association with unbounded multiplicity, as the Orderrows role in Figure 12, Rational Rose generates the following code by default for the Orderrows role into the class module of Order:

```
Public Orderrows As Collection
```

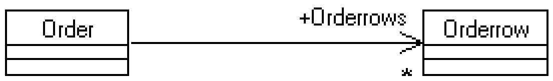


Figure 12 Example of an Association with Unbounded Multiplicity

If there is a user-defined collection class to handle collections of Orderrow objects, the name of that class is used in the declaration instead of the standard collection, for example:

```
Public Orderrows As Orderrows
```

Code Generated for Associations with Imported Interfaces

An association relationship between a class in the model and an imported interface results in a reference to the corresponding ActiveX component in the generated Visual Basic project.

Referenced Component

If the name of the referenced class is not unique, you may need to use both the class name and the component name. This may happen, for example, if two classes with the same name are defined in two different components.

The **Full Name** option on the **Data Member** tab in the **Model Assistant** dialog box (or on the **Visual Basic** tab of the **Association Specification**) is used to instruct the code generator to include the component name in the generated declaration. For example, if the Customer class is assigned to the OrderSys component, Rational Rose generates the following code if **Full Name** is selected:

```
Private Purchaser As OrderSys.Customer
```

Example of Code Generated for an Association

The following example shows the mapping of associations into Visual Basic code. Code has been generated with Get and Set property procedures associated to the mOrders and mPurchaser roles.

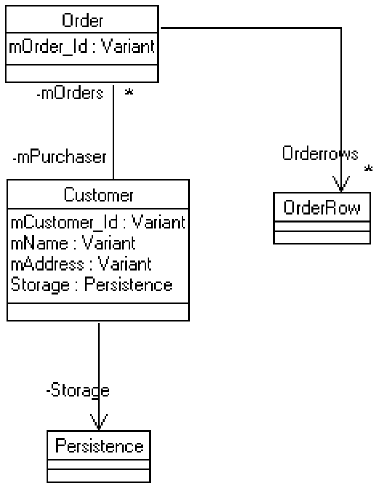


Figure 13 Example of Association Relationships

ORDER.CLS class module.

```
…
   Public Orderrows As Collection 'of OrderRow
   Private mPurchaser As Customer
   Private mOrder_Id As Variant
   …
   Public Property Get Purchaser( ) As Customer …
   Public Property Set Purchaser( Value As Customer) …
…
```

CUSTOMER.CLS class module.

```
…
   Private mOrders As Collection 'of Order
   Private mCustomer_Id As Variant
   Private mName As Variant
   Private mAddress As Variant
   Private Storage As Persistence
   …
   Public Property Get Orders _
      (Index As Variant) As Customer …

   Public Property Set Orders _
      (Index As Variant, Value As Order)
```

Aggregation Relationships

An aggregate relationship is conceptually the same as an association with the aggregate adornment set. The aggregate association adornment denotes a whole or part relationship. Aggregate adornments are purely conceptual and have no effect on code generation, nor reverse engineering. Thus, the code generated for an aggregate relationship is the same as for an association relationship.

The aggregate association may be navigable on both ends, and data members are generated for the associated classes on the navigable side(s).

Note: *Rational Rose Visual Basic does not generate code for aggregation relationships in user interface items, such as forms, but you can subscribe to the events that are defined by an associated controls. See the section Subscribing to Events in chapter 7.*

Dependency Relationships

The dependency relationship in the logical view denotes a client/supplier relationship in which the client object invokes a method on the supplier. Typically this means that the client is dependent on the interface of the supplier, but does not contain an instance of the supplier.

Because Visual Basic does not support the visibility declaration, a dependency relationship between two classes belonging to the same component has no code mapping. However, if a class is dependent upon a module (class utility), the module is automatically added to the same project as the class. Also, if a class is dependent upon a class that belongs to another component, a project reference to that component is added to the project.

The stereotype of a dependency relationship is only relevant if the relationship appears between a class and its collection class. A dependency relationship with the stereotype Collection is automatically created when you assign a new user-defined collection class to a class in the Model Assistant, or when you reverse engineer a collection class into the model. The Visual Basic code generator needs a dependency relationship to automatically generate the appropriate Add, Remove, etc. methods into the collection class. For more information, please refer to the *Collection Classes* section in this chapter.

Generalization Relationships

A generalization relationship between two Visual Basic classes exists when the client class inherits the behavior of the supplier class.

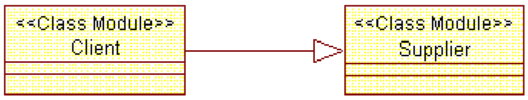


Figure 14 Generalization Relationship

Thus, the generalization relationship means that the client class inherits both the interface of the supplier class and the implementation of the inherited methods. This is also called implementation inheritance.

If the supplier class is abstract—for example, if it is an interface in a COM component— or if the client class is supposed to provide its own variants of the implemented methods, the realize relationship should be used.

Code Generated for Generalization Relationships

In Visual Basic, the closest correspondence to the generalization relationship is the Implements construct and delegation code in the implemented methods. Thus, for a generalization relationship between a class B and a supplier class A, Rational Rose generates the following code into B's class module:

- An Implements A statement
- An object of class A—the delegation object
- Copies of A's public methods, including default dispatching implementations that delegate to class A
- Private Get, Let, and Set procedures for A's public properties and roles

Note: *Generalization relationships between class utilities do not result in any special syntax Visual Basic code. Also, when reverse engineering a class with an Implements statement, a realize relationship is created in the model.*

Referenced Component

If the name of the implemented class is not unique, you may need to use both the class name and the component name. This may happen, for example, if two classes with the same name are defined in two different components.

The **Full Name** option on the **Implements Class** tab in the **Model Assistant** dialog box (or on the **Visual Basic** tab of the **Generalization Specification**) is used to instruct the code generator to include the component name in the generated declaration. For example, if an implemented class Customer is assigned to the OrderSys component, Rational Rose generates the following code if **Full Name** is selected:

```
Implements OrderSys.Customer
```

Abstract Classes

The Visual Basic code generator automatically excludes the delegation object and the dispatching method implementations from the generated code if A is an abstract class. That is, the code generator generates the same code for a generalize relationship between a class and an abstract class as for a realize relationship between two classes.

Example of Code Generated for a Generalization Relationship

The following example shows the mapping of the generalization relationship into Visual Basic code:

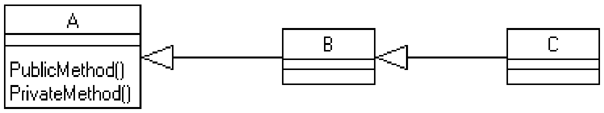


Figure 15 Example of Generalization Relationship

CLASSA.CLS class module.

```
Public Sub PublicMethod()
End Sub

Private Sub PrivateMethod()
End Sub
```

CLASSB.CLS class module.

```
Implements A
Private mAObject As New A

Private Sub A_PublicMethod()
   mAObject.PublicMethod
End Sub
```

CLASSC.CLS class module.

```
Implements B
Private mBObject As New B
```

As you can see, Implements means conformance only to the public interface. For example, B inherits A's public method and gets a private Sub A_PublicMethod(). C implements the public interface of B, but A_PublicMethod() is not public in B. Therefore, according to the Visual Basic programming language, there is no Sub B_A_PublicMethod in C.

Realize Relationships

A realize relationship between a Visual Basic class and another class exists when the client class conforms to the interface defined by the supplier class (usually an interface).

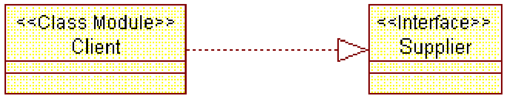


Figure 16 Generalization Relationship

Thus, the realize relationship means that the client class will implement the interface of the supplier class, and provide an implementation of the realized interface's properties and methods. This is also called interface inheritance. Interface inheritance is used if the supplier class or interface is defined in a type library without code—that is, if it is an abstract class or an interface—or if the client class is supposed to provide its own variants of the implemented methods.

To inherit also the implementation of the methods in the supplier class—that is, to delegate the implementation of the inherited methods to the supplier class—the generalization relationship should be used.

Code Generated for Realize Relationships

The realize relationship corresponds to the Implements construct in Visual Basic. Thus, for a realize relationship between a class B and class A, Rational Rose generates the following code into B's class module:

- An Implements A statement
- Copies of A's public methods with empty method bodies
- Private Get, Let, and Set procedures for A's public properties and roles with empty bodies

When reverse engineering a class with an Implements statement, a realize relationship is created in the model.

Referenced Component

If the name of the realized class is not unique, you may need to use both the class name and the component name. This may happen, for example, if two classes with the same name are defined in two different components.

The **Full Name** option on the **Implements Class** tab in the **Model Assistant** dialog box is used to instruct the code generator to include the component name in the generated declaration. For example, if a realized class Customer is assigned to the OrderSys component, Rational Rose generates the following code if **Full Name** is selected:

```
Implements OrderSys.Customer
```

Example of Code Generated for a Realize Relationship

The following example shows the mapping of the realize relationship into Visual Basic code:

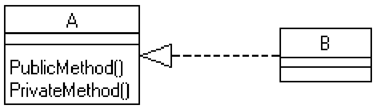


Figure 17 Example of Realize Relationship

CLASSA.CLS class module.

```
Public Sub PublicMethod()
End Sub

Private Sub PrivateMethod()
End Sub
```

CLASSB.CLS class module.

```
Implements A

Private Sub A_PublicMethod()
End Sub
```

Advanced Association Relationship Mappings

This section explains how the following advanced association relationship settings are mapped to Visual Basic code:

- Navigability
- Containment adornment
- Cardinality/Multiplicity
- Link properties
- Qualifiers

Navigability

A navigable association adornment (defined on the Role Detail tabs of the Association Specification) indicates the implementation direction of an association. A data member is generated only on the navigable side of the association. By default, a new association relationship becomes navigable in both directions.

Example of Code Generated for a Navigable Association

The following example shows the mapping of a navigable association into Visual Basic code.

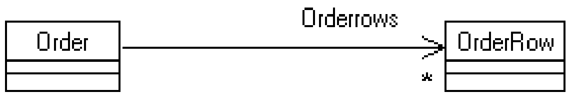


Figure 18 Example of a Navigable Association

ORDER.CLS class module.

```
…
   Public Orderrows As Collection 'of OrderRow
…
```

Containment Adornment

By-Value and By-Reference containment adornments (defined on the Detail tab of Property Specifications and Role Detail tabs of Association Specifications) are not directly applicable to Visual Basic code generation, because all object references are implemented By-Reference.

Conceptually, the By-Value aggregate implies that the lifetime of the containing and contained objects are identical. As such, it is expected that the initialization method for the containing object will create its contained objects, and the termination method of the containing object will destroy the contained objects.

The By-Reference aggregate implies that the lifetime of the containing and contained objects are different. As such, it is expected that the initialization for the containing object will not create its contained By-Reference objects, and the termination method of the containing object will not destroy the objects. Object references are often passed by method or property procedure calls.

The containment adornments may be used by other add-ins in Rational Rose. Therefore, these adornments may need to be specified in the model, and they are preserved by Rational Rose Visual Basic during reverse engineering.

Cardinality/Multiplicity

When code is generated from an aggregation or association relationship, Rational Rose looks at the supplier cardinality (Figure 19) to determine what kind of data member to generate. (The client cardinality of the relationship is ignored.)

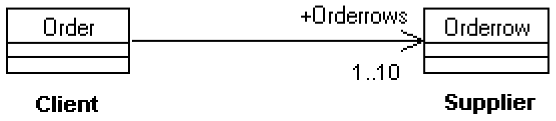


Figure 19 Example of Cardinality

The relationship cardinality indicates the number of links between each instance of the classes. A cardinality of 1 results in the simple data member of one object within another. A cardinality larger than 1 is generated using either an array object definition (for bounded multiplicities) or a collection object (for unbounded multiplicities.)

The cardinality of an array is defined using the Subscript option on the Data Member tab in the Model Assistant dialog box. Note that you should use Visual Basic syntax in the Subscript option box, for example "1 To MaxLen" or "mnuOpen To mnuQuit". If a value has been entered in the Subscript option, the Visual Basic code generator uses that value, and not the cardinality, when generating code.

Table 5 shows what code is generated for a certain cardinality or subscript value.

Table 5 Cardinality Values

Cardinality	Subscript option	Generated code
0, 0..0	none	Nothing. The relationship is ignored by the code generator.
0..1, 1, none	none	A data member of the supplier object.
x..y	*0, 1, x* To *y*, etc.	An array with a subscript. For example, "1..5" is generated as: **`Public Customers(1 To 5) As Customer`** Note, however, that public properties in a Standard EXE component cannot have subscript.
0..n, 0..*, 1..n, 1..*, n, *	none	A data member of a collection class, for example: **`Public Orderrows As Collection`** See the *Collection Classes* and *Association Relationships* section.
any	()	A dynamic array, for example: **`Private My_Prop() As Variant`**

Link Properties (Attributes)

A link property (attribute) represents a value held by an association, rather than one of the classes participating in the association. Link properties are encapsulated by a link class. The association is generated as data members of the link class. Data members for the two associated classes are added to the link class.

The generated code from link property may be optimized by generating the property as a part of the “many” side of the association.

Example of Code Generated for a Link Property

The following example illustrates that the shipment date is added as a link property to the Order—Customer association, that is to the Shipment class, rather than the Order or Customer classes.

After the code has been generated, the association between Order and Customer in the model is replaced by two associations: one between Order and Shipment and one between Customer and Shipment.

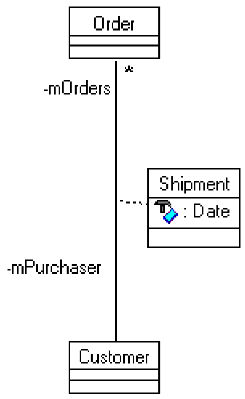


Figure 20 Example of a Link Property

ORDER.CLS class module.

```
…
   Private mPurchaser As Shipment
…
```

CUSTOMER.CLS class module.

```
…
   Private mOrders As Shipment
…
```

SHIPMENT.CLS class module.

```
…
   Private the_Date As Date
   Private mPurchaser As Customer
   Private mOrders As Collection 'of Order
```

Qualifiers

A qualifier is a variant form of a link property that is implemented using a dictionary. Qualifier keys can be used to index the dictionary in order to retrieve a link object.

A qualifier is associated with one of the roles of an association relationship, which means that Rational Rose maps a qualified role into a data member. The multiplicity of the role decides whether a collection object is used in the data member declaration.

You can add property Get, Let, and Set procedures to the generated data member by selecting the appropriate procedures in the Model Assistant. The property procedures automate the maintenance of the dictionary.

Example of Code Generated for a Qualified Association

The example below shows the mapping of the qualified association relationship role mOrder, which has an unbounded multiplicity (*). Note that the property Set and Get procedures were not created by default. They were manually added to the Orders property in the Model Assistant.

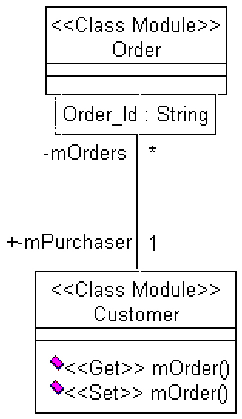


Figure 21   Example of a Qualified Association Relationship

CUSTOMER.CLS class module.

```
…
Private mOrders As Collection
…
Public Property Set Orders(Order_Id As String, ByVal vNewValue
As Order)
   Set mOrders (Order_Id) = vNewValue
Exit Property

Public Property Get Orders(Order_Id As String) As Order
   Set Orders = mOrders(Order_Id)
Exit Property
…
```

Methods (Operations)

This section covers what you need to consider when creating and specifying Visual Basic methods (operations) in Rational Rose, such as:

- User-defined methods
- Method stereotypes
- Method parameter passing
- Property Get/Set/Let procedures
- Declare methods
- Events

User-Defined Methods

Rational Rose allows you to specify several aspects of a method (operation), either in the Method Specification or in the Model Assistant. For example:

- The method stereotype defines the method type, which can be Declare, Event, Get, Set, or Let. See *Method Stereotypes* below. An empty stereotype means that the method is a Sub or Function, depending on whether the method has a return type or not.
- Methods can be declared as:
  - Public - allowing use from other modules. Rational Rose Visual Basic translates the method into a visible function or sub declaration.
  - Private - allowing use only from within the module. Rational Rose Visual Basic translates the method into a private Function or Sub declaration.
  - Protected - allowing use only from within the project. Rational Rose Visual Basic translates the method into a Friend Function or Sub declaration. See *Using Friend Methods in Sequence Diagrams* in chapter 7.
- Method parameters are declared with a name and a parameter type. The type definition can be omitted, implying the use of a Variant data type. The parameter passing mechanism can be specified in the Model Assistant.
- Methods can be static, indicating that the Sub procedure's local variables are preserved between calls. Methods are marked as static by using the **Static** option in the Model Assistant.
- When generating a new method, default code is inserted into the method body from the class's template. See the *The Default Body* section in chapter 7. You can preview the default body on the **Method** tab in the Model Assistant.
- You can instruct Rational Rose to insert debug code, error-handling code, and "Your code goes here..." comments into each class; see chapter 7.

Caution: *Be careful when moving methods from one class to another in the model, because the code generator regards moved methods as new methods. That is, their method bodies get default contents. Also, in each* *generated method, Rational Rose adds an identifier—a model ID—used to identify the corresponding method in the model. Do not edit those identifiers.*

Method Stereotypes

The stereotype of a method controls the Visual Basic code that Rational Rose produces for the method.

The stereotype can be used to declare a property Get, Let, or Set procedure, a reference to an external procedure in a dynamic-link library (DLL), or an event handler. If the stereotype is empty, Rational Rose produces a sub, or a function declaration, depending on whether there is a result type declared for the method or not.

Table 6 summarizes the possible values for the method stereotype when generating or reverse engineering Visual Basic code. In this table, `result` is the return type of the member function, `fname` is the name of the member function, and `params` is the formal parameter list.

Table 6 Visual Basic Method Stereotypes

Stereotype	**Visual Basic Mapping**

No stereotype	(Default) Represents a sub or a function declaration, depending on the availability of a result type. For example:
	`Sub fname (params)`
	`Function fname (params) As result.`
	Sub and function declarations can be both generated and reverse engineered.

Get	Represents a property Get procedure, such as `Public Property Get fname () As result.` Property Get procedures can be both generated and reverse engineered.
Let	Represents a property Let procedure, such as `Public Property Let fname (params).` Property Let procedures can be both generated and reverse engineered.

Table 6 Visual Basic Method Stereotypes

Stereotype	**Visual Basic Mapping**
Set	Represents a property Set procedure, such as `Public Property Set fname (params)`. Property Set procedures can be both generated and reverse engineered.
Declare	Represents a references to external procedures in a dynamic-link library, for example: `Declare Sub fname Lib libname (params)` `Declare Function fname Lib libname (params)As result.` Declare statements can be both generated and reverse engineered.
Event	Rational Rose declares an event handler, such as `Event fname (params)`. Events can be both generated and reverse engineered.

In the Model Assistant, you can automatically create a method of a specific stereotype. See the section *Creating Declare Statements*, *Creating Events*, *Creating Property Get, Let, and Set Procedures*, and *Creating Methods* in chapter 7.

Method Parameter Passing

The Visual Basic programming language allows specification of the parameter passing mechanism as well as dynamic parameter cardinality. The passing mechanism for the parameters of a method can be viewed and modified in the Model Assistant as options on the Parameters tab.

You can also specify the parameter passing mechanism by appending a keyword in front of the parameter name in the Method Specification or in a diagram, as outlined on the following table. When opening the class in the Model Assistant, or when generating code for the class, any keywords that have been entered that way are transformed to option settings in the Model Assistant

You do not have to specify the passing mechanism in Rational Rose. You can do it later, in Visual Basic, after you have generated the code.

You are also able to define default initial values of method parameters. These values are directly mapped to default parameter values in the Visual Basic code.

Table 7 Controlling the Declaration of Method Parameters

If you enter	The parameter is declared as
ByVal argname	Passed by value
ByRef argname	Passed by reference
Optional argname	Optional parameter
ParamArray argname	Indefinite number of parameters

Property Get/Set/Let Procedures

A property procedure in Visual Basic corresponds to a method with the stereotype Get, Set, or Let in the model. A property procedure in the model can be:

- Associated to a role or property in the model. By selecting property procedures for a role or property in the Model Assistant, the created property procedures are associated with the role or property. Thus, if a property is removed, the corresponding property procedures are automatically deleted. See the section *Creating Property Get, Let, and Set Procedures* in chapter 7.
- Independent from the roles and properties in the model. A property procedure can be created by giving a method the stereotype Get, Set, or Let. However, there is no correspondence between the properties or roles in the model and a property procedure that has been created in that way.

Property procedures are also generated for all public roles and properties of a superclass, A, into the class modules of its subclasses. See the *Generalization Relationships* and *Realize Relationships* sections in this chapter.

Declare Methods

A method in a class or class utility can be declared as a reference to a DLL library procedure; see the section *Creating Declare Statements* in chapter 7. Rational Rose transforms the method into a Declare Sub or Declare Function statement. To reference a method to a DLL procedure, the method stereotype must be set to Declare.

The DLL library name and method alias name are set using the **Library** and **Alias** options on the **Declare** tab for the method in the Model Assistant.

The **Library** option specifies the name of the DLL that contains the declared method. The **Alias** option indicates that the procedure being called has another name in the DLL. This is useful when the external procedure name is the same as a Visual Basic reserved word. You can also use **Alias** when a DLL procedure has the same name as a Global variable, constant, or any other procedure in the same scope. **Alias** is also useful if any characters in the DLL procedure name are not allowed in Visual Basic names.

Events

A method in a class or web item can be declared as an event by setting the method stereotype to Event. Rational Rose then transforms the method into an event declaration statement. You can also create event methods in the Model Assistant; see the section *Creating Events* in chapter 7.

A class can subscribe to the events that are defined in other classes. For roles that are declared with the `WithEvents` keyword and for controls (that is, associations with the stereotype Contained Control or aggregation relationships) the Model Assistant lists all events that are defined in the associated class or control. By selecting an event the code necessary to subscribe to that event will be generated into the class; see the section *Subscribing to Events* in chapter 7.

For example, if the form dlg_Order in Figure 22 subscribes to the KeyPress event in the CommandButton interface, the following code will be inserted into dlg_Order:

```
Private Function btn_cancel_KeyPress(ByRef KeyAscii As Integer)
As HRESULT
```

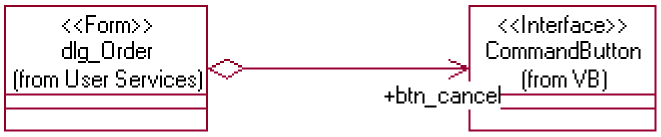


Figure 22 Dlg_Order Subscribes to the btn_cancel_KeyPress Event Defined in CommandButton

Use-Case View to Visual Basic Mapping

Rational Rose Visual Basic does not currently support code generation from actors and use cases. However, you can generate Visual Basic code for any classes that you have created in the use-case view.

Please refer to the *Using Rose* manual or online help for information about the use-case view.

Deployment View to Visual Basic Mapping

Rational Rose Visual Basic does not currently support code generation from the deployment view.

Please refer to the *Using Rose* manual or online help for information about the deployment view.

Visual Basic to UML Mapping

This section describes the mapping of the Microsoft Visual Basic language to the logical and component views in Rational Rose. Rational Rose Visual Basic uses these mapping rules during reverse engineering.

Visual Basic Projects

When reverse engineering a Visual Basic project, a component with the language “Visual Basic” is created. The stereotype of the component is the same as the project type. For example, a Standard EXE project becomes a component with the stereotype Standard EXE in the model.

The model can also contain the type library of the COM component that has been compiled from the Visual Basic project (only relevant for ActiveX projects). The compiled COM component is represented in the same way as the project references; see the section *Project References* below. The Import the compiled VB binary option in the Visual Basic Component Properties dialog box defines whether to automatically import the type library of the compiled project.

The name of the model file that corresponds to a project is stored as a related document to the project. In the Rational Rose tool window in Visual Basic, you can see what models are related to a project. To open the Rational Rose tool window in Visual Basic, click **Add-Ins > Rational Rose**.

Project Items

When reverse engineering Visual Basic project items, the mapping rules in Table 8 are used:

Table 8 Mapping of Visual Basic Project Items to Model Element

In Visual Basic	**Becomes in the model**
project_item_type	Class (stereotype = *project_item_type*)
Implements statement	Realize relationship
Constant declaration	Property with default value
Enum or Type declaration	Nested class with stereotype Enum or Type, which is contained by the reverse engineered class
Web item in web class	Nested class with stereotype WebItem, which is contained by the reverse engineered web class
Data member	Property
	Association (navigable) to object type
Method	Method (stereotype= empty)
Event declaration	Method (stereotype= Event)
Declare declaration	Method (stereotype= Declare)
Property procedure	Method (stereotype= Set, Get, or Let)

Note: *When reverse engineering a project item, A, all project items that A refers to must exist in the model. If the referenced project items were not selected for model update, and if they did not exist in the model before, the Model Update Tool creates empty classes for them in the model.*

Note: *When reverse engineering a new collection class in Visual Basic, the created class in the model gets the stereotype Class Module, and not Collection. However, Rational Rose inserts a dependency relationship between the collection class and the item class in the model.*

Note: *When reverse engineering a class module with an instancing property other than Private, the stereotype of the created class in the model becomes Interface.*

Modules

When reverse engineering Visual Basic modules, the mapping rules in Table 9 are used:

Table 9 Mapping of Visual Basic Modules to Model Elements

In Visual Basic	Becomes in the model
Module	Class utility (stereotype = Module)
Constant declaration	Property with default value
Enum declaration	Nested class with stereotype Enum, which is contained by the reverse engineered class
Data member	Property
	Association (navigable) to object type
Method	Method (stereotype = empty)
Declare declaration	Method (stereotype= Declare)
Property procedure	Method (stereotype= Set, Get, or Let)

Project References

When reverse engineering a Visual Basic project, all COM components that are referenced from the project are automatically imported into the model. Each project reference becomes a package in the logical view and a component, with the stereotype and language COM, in the component view. The type library of the COM component is inserted into its logical package as follows:

Table 10 Mapping of Project References to Model Elements

In type library	Becomes in the model
Interface or dispinterface	Class (stereotype = Interface)
Coclass	Class (stereotype = Coclass)
Module	Class (stereotype = Module)
Enum	Class (stereotype = Enum)
Struct	Class (stereotype = Struct)
Union	Class (stereotype = Union)
Methods and properties	Methods and properties if the component was imported as a full import. Methods and properties are not created when performing a quick import.

Note: *No code will be generated for imported COM components when you generate code from the model.*

Code Comments

Rational Rose Visual Basic takes care of code comments by inserting them as documentation of the corresponding model elements in the model.

For each declaration in the Visual Basic code, Rational Rose Visual Basic copies the directly preceding comment into the Documentation box of the corresponding model element's specification in Rational Rose. Thus, you must be careful where you put the code comments to make them appear in the correct model element.

Example:

```
' This comment will be inserted into the documentation field of
the Purchaser property.
Public Property Get Purchaser( ) As Customer
       Purchaser = mPurchaser
End Property

' This comment will not be reverse engineered because there is
an empty row between the comment and the function declaration
below.

Public Sub Add_OrderRow( new_order_row As OrderRow)
       ' This comment will not be reverse engineered
        because it does not precede code that corresponds
        to a model component of its own.
       Orderrows.Add new_order_row
End Sub
```

Note: *The Model Update Tool does not wrap code comments when inserting them into the model.*

Caution: *The Model Update Tool clears the contents of the* **Documentation** *boxes of Parameter Specifications when updating the model from code. Thus, any parameter documentation that you have entered in the model is removed when you update the corresponding classes from code changes.*

Compiler Directives

If your Visual Basic project contains conditional compiler directives, the reverse engineering applies only to those declarations that are visible under the current conditions. For example, if the project contains the compiler directive "`#if Debug Dim A #else Dim B #endif`," and `Debug` is true, then only the `Dim B` statement will be reverse engineered.

Chapter 3

Round-Trip Engineering a Visual Basic Application

Rational Rose facilitates a controlled iterative application-development process called round-trip engineering. This process enables you to model your application, analyze and refine it as you increase your understanding of its operation, then generate the code elements of a complete Visual Basic application framework based on that model. You then evolve this generated code, using Visual Basic, and reverse engineer your modified code structures into the model—keeping the model and code fully synchronized. The process of alternating between the model and the code is called round-trip engineering.

This chapter provides an overview on how to round-trip engineer a Visual Basic project and the corresponding Rational Rose model. It also explains what synchronization means and how Rational Rose Visual Basic enables synchronization by inserting unique model identifiers into the source code.

The Visual Basic Round-Trip Engineering Tools

The round-trip engineering tools in Rational Rose Visual Basic are tightly integrated with the Microsoft Visual Basic environment, allowing you to seamlessly progress through the round-trip engineering process. The tools are:

- The Model Assistant, which enables you to update and refine a Visual Basic class in your model with all the necessary code-specific semantics for complete and robust code generation.

- The Component Assignment Tool, which provides you with an easy-to-use interface to create new Visual Basic components in the model, associate components with Visual Basic projects, and assign classes to components.
- The Code Update Tool, which generates and updates the Visual Basic source code from the information contained in a model, and preserves existing user-supplied definitions and declarations from the previous iteration's source code.
- The Model Update Tool, which extracts design information from the Visual Basic code and updates the application's design model.

Note: *When further developing the application in Rational Rose or Visual Basic, you should generate or reverse engineer the code before continuing the development in the other tool. Otherwise, if you rename or delete classes, members, and methods in both tools at the same time, you may lose some of the changes.*

Round-Trip Engineering—Starting with a Model

To round-trip engineer a Visual Basic application starting with a model:

1. Create a new model.

   See *Creating a New Model* in chapter 6.
2. Create a component in the component view to represent the Visual Basic project.

   See *Creating New Components in the Model* in chapter 6.
3. Develop a model of the system by creating logical packages, classes, relationships, properties (attributes), and methods (operations), as well as by illustrating the model in diagrams. Specify each class by using the Model Assistant. Also, assign each class to the created Visual Basic component.

   See:
   - The *Using Rose* manual
   - *Creating Visual Basic Classes in the Model* in chapter 7
   - *Assigning Classes to Existing Components* in chapter 6
4. Generate a new Visual Basic project from the model.

   See *Generating a New Visual Basic Project from a Model* in chapter 4.

5. Evolve the generated code.

   See *Evolving the Generated Code* in chapter 4.
6. Update the model with the code changes.

   See *Updating a Component in the Model from Code Changes* in chapter 5.
7. Evolve the updated model.

   See *Evolving the Updated Model* in chapter 5.
8. Update the Visual Basic project with model changes.

   See *Updating an Existing Visual Basic Project with Model Changes* in chapter 4.
9. Continue developing your system in increments by iterating through steps 5-8.

Round-Trip Engineering—Starting with a Visual Basic Project

To round-trip engineer a Visual Basic application starting with a project:

1. Reverse engineer the code to create a new model.

   See *Reverse Engineering a Visual Basic Project Into a New Component* in chapter 5.
2. Evolve the generated model.

   See *Evolving the Updated Model* in chapter 5.
3. Update the Visual Basic project with model changes.

   See *Updating an Existing Visual Basic Project with Model Changes* in chapter 4.
4. Evolve the generated code.

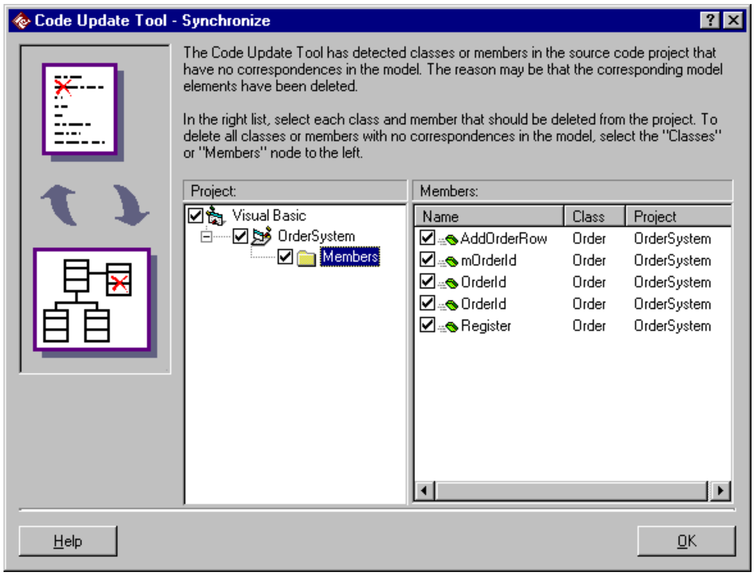
   See *Evolving the Generated Code* in chapter 4.
5. Reverse engineer the code to update the model.

   See *Updating a Component in the Model from Code Changes* in chapter 5.
6. Continue developing your system in increments by iterating through steps 2-5.

Synchronization

Changing a model typically involves adding, changing, or deleting model or code elements. Rational Rose automatically adds or changes elements, so synchronization is not required for these operations. When you delete a model or code element for which Rational Rose has generated the code synchronization becomes an issue.

Whenever you perform a code or model update, Rational Rose checks to see that the code and model agree. If the Code Update Tool finds code with no corresponding model element, or if the Model Update Tool finds a model element with no corresponding code, the tool displays a **Synchronize** page as in the following figure:



The Code Update Tool has detected classes or members in the source code project that have no correspondences in the model. The reason may be that the corresponding model elements have been deleted.

In the right list, select each class and member that should be deleted from the project. To delete all classes or members with no correspondences in the model, select the "Classes" or "Members" node to the left.

Name	Class	Project
✓ AddOrderRow	Order	OrderSystem
✓ mOrderId	Order	OrderSystem
✓ OrderId	Order	OrderSystem
✓ OrderId	Order	OrderSystem
✓ Register	Order	OrderSystem

Figure 23 Synchronize Page

The **Synchronize** page allows you to delete or keep the listed model elements or code items; synchronizing the model with the code. If you select the check box next to a model element or code item on the **Synchronize** page, the Model Update Tool or Code Update Tool removes that model element or code item.

Model IDs

To support model-element to code-item matching when renaming or deleting model elements or code items, each generated model element is tagged with a unique identifier in the code—a model ID. This allows the code item to be matched with a model element independent of the actual names. Thus, all model elements that can be renamed and reverse engineered are generated with a special comment tag that contains the model ID.

For example, the following is a model ID comment tag for a property:

```
'##ModelId=351ADC1EF002
Private Name As Variant
```

The Code Update Tool and the Model Update Tool use the model IDs to synchronize the model and code, which means that the model IDs are required to generate and reverse engineer the code properly. Therefore, do not edit or remove the inserted model IDs!

If the ability to synchronize the model and code is not important—for example, if you are going to generate Visual Basic code from this model only once—you can suppress the generation of model IDs by checking the **Suppress model IDs** option in the **Visual Basic Properties** dialog box. Note, however, that Rational Rose will not be able to keep the model and code synchronized if you select that option.

Chapter 4

Generating Visual Basic Code

The Visual Basic components in the component view of a model represent Visual Basic projects. Code is generated from the classes that are assigned to a component into the associated Visual Basic project. You use the Code Update Tool to generate Visual Basic code for one or several components.

For each class in a component, Rational Rose Visual Basic generates a Visual Basic project item according to the class's stereotype. The generated code for a class is defined by the class' properties (attributes), associations, and methods (operations) in the model. You can customize the code to generate for a class or its members by using the Model Assistant in Rational Rose; see chapter 7.

This chapter introduces the Code Update Tool and explains the basic steps for generating Visual Basic code. It also provides information on how to customize the Visual Basic code generation.

The Code Update Tool

Using the Code Update Tool, you can produce Visual Basic source code from the information contained in a model. The Code Update Tool generates code from the components in your model into the corresponding Visual Basic projects. With the Code Update Tool you can:

- Generate and update several projects of different implementation languages at the same time.
- Preview the code to be generated for each class and member.

- Further specify the mapping between the classes in the model and the code, by opening the Model Assistant.
- Keep the model and Visual Basic projects synchronized, as the Code Update Tool detects any project items that may have been renamed or deleted from the model.
- Assign selected classes that are not assigned to any component.

Starting the Code Update Tool

You can start the Code Update Tool in the following ways:

- Click **Tools > Visual Basic > Update Code**.
- Right-click on a component or class in the browser or in a diagram and click **Update Code**.

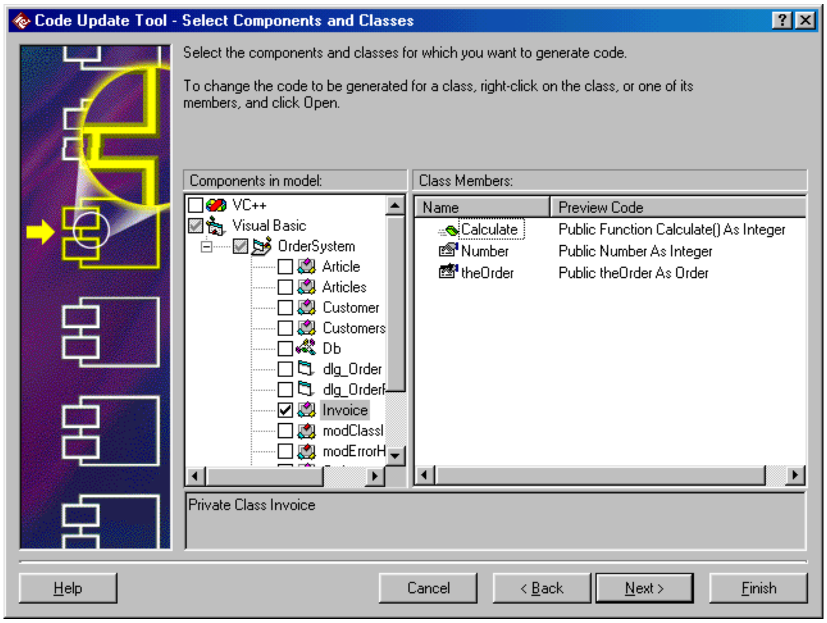
The Code Update Tool Pages

The Code Update Tool leads you through the process of updating a Visual Basic project from model changes on the following pages:

Welcome Page

The first page provides general information about the tool. You can turn this page off by selecting the **Don't show this page in the future** option.

Select Components and Classes Page



Select the components and classes for which you want to generate code.

To change the code to be generated for a class, right-click on the class, or one of its members, and click Open.

Components in model:
☐ VC++
☑ Visual Basic
☑ OrderSystem
☐ Article
☐ Articles
☐ Customer
☐ Customers
☐ Db
☐ dlg_Order
☐ dlg_OrderF
☑ Invoice
☐ modClassI
☐ modErrorH

Name	Preview Code
Calculate	Public Function Calculate() As Integer
Number	Public Number As Integer
theOrder	Public theOrder As Order

Private Class Invoice

Figure 24 Code Update Tool—Select Components and Classes Page

On this page, you select the components or classes from which you want to generate code. Do any of the following:

■ To generate code for all classes in a component, select the check box next to that component. To generate code for only some of the classes that are assigned to a component, select only those classes.

  Note: *You can select several components. Also, the components and classes that are selected in the current active diagram or in the browser are selected by default.*

■ If you have selected classes, before entering the Code Update Tool, which are not assigned to any component, you must assign those classes to a Visual Basic component before you can generate code for them. To assign all selected unassigned classes to one and the same component, select that component and click **Assign new classes to the component** at the top of the page. Otherwise, right-click on any component and click **Assign Classes**, which brings up the Component Assignment Tool.

- To associate a component with a project file, right-click on the component and click **Properties**, which brings up the **Visual Basic Component Properties** dialog box for the component. See the *Associating a Component with a Visual Basic Project File* section in chapter 6.

  Note: *A component marked with ? is associated with a project file that the Code Update Tool cannot find.*

- To preview the code to be generated for a class, select the class in the left-hand list. A list and preview of all its members is displayed in the right-hand list.
- To customize the code to generate for a class or member, open the Model Assistant for that class by right-clicking on the class or one of its members, and then clicking **Open**.

  Note: *Classes marked with ◆ will generate incorrect code. You must change the code mapping of those classes before generating code for them.*

- To customize the Code Update Tool, right-click on the Visual Basic language and click **Properties** on the displayed menu. See *Customizing the Code Generation* in this chapter.

Finish Page

This page displays a summary of the code to be generated. Click **Finish** if you are satisfied, or click **Back** if you want to change something.

Progress Page

This page displays the progress of the code generation.

Synchronize Page

This page is shown if the Code Update Tool detects project items that do not have any correspondences in the model. Here you can confirm the deletion of each such project item. See *Synchronizing Code and Model During Code Update* in this chapter.

Summary Page

This page displays a summary of the result of the code generation. On the shortcut menu, you can customize whether or not to display warnings and errors in color and to use timestamps. The information on the Log tab can also be found in the Rational Rose log window after exiting the Code Update Tool.

Generating Visual Basic Code

Generating a New Visual Basic Project from a Model

In order to generate code for classes into a new Visual Basic project, a component representing that project must exist in the model, and the classes must be assigned to that component.

To generate a new project from a model:

1. First, a component is needed to map classes in the model to the new Visual Basic project. If there is no component for the new project in the model yet, create a component for the new project. (See *Creating New Components in the Model* in chapter 6.) Also, assign the classes that should belong to the new project. (See *Assigning Classes to Existing Components* in chapter 6.)
2. Click **Tools > Visual Basic > Update Code**. The Code Update Tool starts. If the **Welcome** page is shown, click **Next**.
3. On the **Select Components and Classes** page (see Figure 24) select the check box next to the component for which you want to generate a Visual Basic project. Click **Next**.
4. Look at the code to be generated on the **Finish** page, and click **Finish** if you are satisfied.
5. If the **Synchronize** page is displayed, confirm the deletion of each project item that does not have any correspondence in the model. See *Synchronizing Code and Model During Code Update* in this chapter.
6. Examine the results of the code generation on the **Summary** page. Click **Close**.
7. Be sure to review the generated source code before you continue modeling and coding. See *Reviewing the Generated Code* in this chapter.

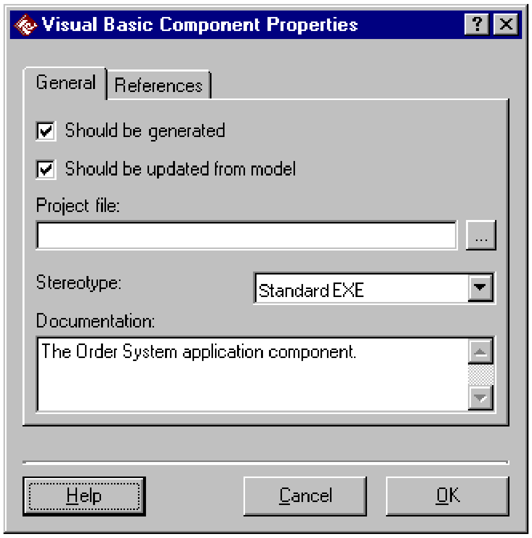
8. In order to associate the generated component with the new project permanently, save the created Visual Basic project and generate code from the component once again. The name of the project file is then stored in the **Project file** option on the component's **Visual Basic Component Properties** dialog box. Rational Rose will then be able to find the associated project file next time you generate code.

Updating an Existing Visual Basic Project with Model Changes

To update a Visual Basic project from changes in the model:

1. First, each new class has to be assigned to the component(s) that correspond to the project(s) that will contain the new class. If there are any unassigned classes, click **Tools > Visual Basic > Component Assignment Tool**, and assign the classes. (See *Assigning Classes to Existing Components* in chapter 6.)
2. Click **Tools > Visual Basic > Update Code**. The Code Update Tool starts. If the **Welcome** page is shown, click **Next**.
3. On the **Select Components and Classes** page, select the check box next to the components or classes that you want to update.

4. If a selected component is not associated with a Visual Basic project yet, associate it with the project that you want to update by right-clicking on the component and clicking **Properties**. The **Visual Basic Component Properties** dialog box opens:



☑ Should be generated

☑ Should be updated from model

Project file:

Stereotype: Standard EXE

Documentation: The Order System application component.

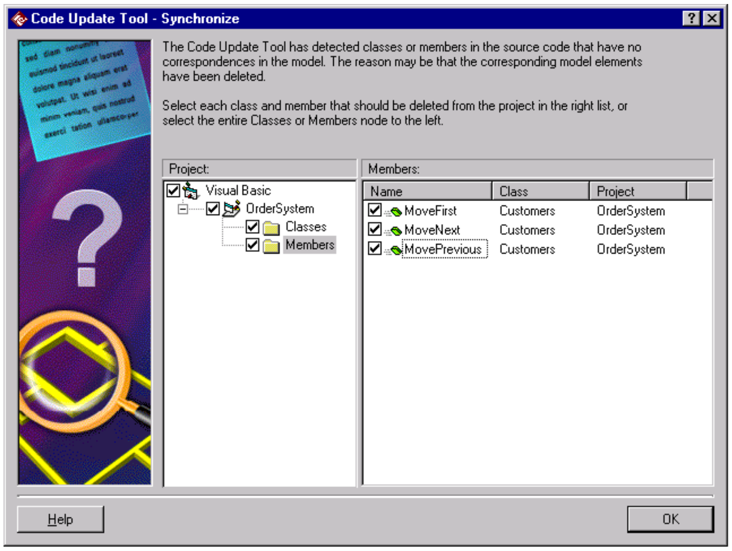
Figure 25   Visual Basic Component Properties Dialog Box—Code Update

   For more information, see *Associating a Component with a Visual Basic Project File* in chapter 6.

5. Click **Next** and examine the code to be generated. Click **Finish** if you are satisfied.
6. If an updated project contains project items or members that do not have any correspondences in the model, a **Synchronize** page is displayed. In the right-hand list, select those project items and members that you want to delete from the project. See *Synchronizing Code and Model During Code Update* in this chapter.
7. Take a look at the results of the code generation on the **Summary** page. Click **Close**.
8. Be sure to review the generated source code before you continue modeling and coding. See *Reviewing the Generated Code* in this chapter.

Synchronizing Code and Model During Code Update

Whenever you perform a code update, Rational Rose checks to see that the code and model agree. If the Code Update Tool finds code with no corresponding model element, the tool displays a **Synchronize** page:



The Code Update Tool has detected classes or members in the source code that have no correspondences in the model. The reason may be that the corresponding model elements have been deleted.

Select each class and member that should be deleted from the project in the right list, or select the entire Classes or Members node to the left.

☑ Visual Basic
☑ OrderSystem
☑ Classes
☑ Members

Name	Class	Project
☑ MoveFirst	Customers	OrderSystem
☑ MoveNext	Customers	OrderSystem
☑ MovePrevious	Customers	OrderSystem

Figure 26 Code Update Tool—Synchronize Page

To delete selected code items:

1. In the right-hand list, select the check box next to each code item that you want to delete.
2. Click **OK** to delete the selected items.

To delete all code items of the same kind:

1. In the left-hand list, select the check box next to the appropriate delete folder (Classes or Members.) The contents of the folder are displayed in the right-hand list and all items that will be deleted are selected.
2. Click **OK** to delete the selected items, or clear those items you do not want deleted.

Reviewing the Generated Code

This section provides you with some tips on how to review the generated code. After code generation:

1. Check the error log on the **Summary** page of the Code Update Tool to track down errors and warnings. Or, exit the Code Update Tool and open the log by clicking **Window > Log** in Rational Rose.
2. In your Microsoft Visual Basic environment, compile the project to make sure that the generated code is free from syntax errors.
3. Inspect the generated source code. To browse between the model and the code, select a component, class, or member in Rational Rose, click **Tools > Visual Basic > Browse Source Code**. The Visual Basic item that corresponds to the selected element in the model is displayed in Visual Basic.
4. Based on your evaluation, make the necessary changes to the model and regenerate the code, or make the changes directly in the code.
5. When you are satisfied, save the code in your Microsoft Visual Basic environment.

Note: *When generating code, Rational Rose uses the procedure separators in Visual Basic as indicators of where declarations start and end. However, if there is an empty line before the `#Endif` statement of a compiler directive in Visual Basic, the location of the procedure separator becomes incorrect. Thus, make sure that the separators are correctly located before generating code again.*

Evolving the Generated Code

Evolve the generated Visual Basic source code by editing the code regions to implement the application's functionality included in the iteration, and changing and adding members, constants, methods (functions, subs, and property procedures), forms, controls, and so on.

Compile and test the edited project in the Visual Basic development environment. Make sure that the project does not contain any syntax errors before generating or reverse engineering the code again.

Caution: *For each generated project item, member, and method, Rational Rose adds an identifier as a code comment—for example "ModelID=3237F8CE0053"—which identifies the corresponding class, property, role, or method in the model. Those identifiers are called Model IDs. Do not edit them!*

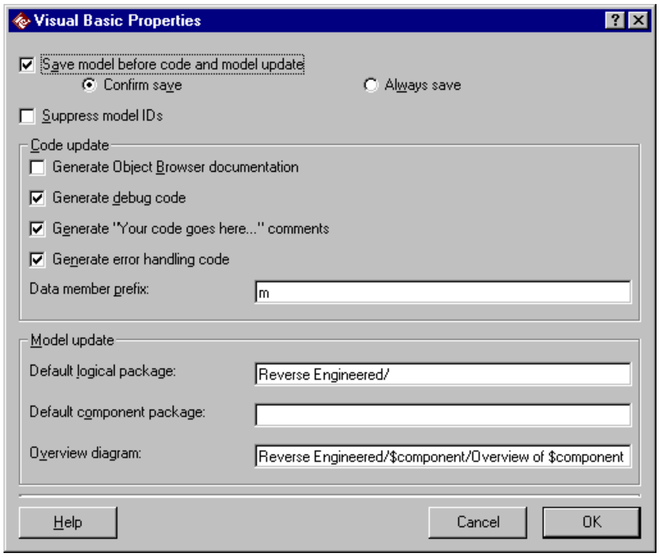
Customizing the Code Generation

You can customize several aspects of the code generation. For example, you can customize the behavior of the Code Update Tool, associate a component with a Visual Basic project file, or preview and specify the code to generate for each class.

Customizing the Code Update Tool

You can customize general aspects of the code generation by modifying the code generation options for the Visual Basic language itself. Those options are available in the **Visual Basic Properties** dialog box.

To open the **Visual Basic Properties** dialog box, right-click on the Visual Basic language node on the **Select Components and Classes** page in the Code Update Tool, and then click **Properties**. (Or click **Tools > Visual Basic > Properties**.)



[✓] Save model before code and model update
[● Confirm save] ○ Always save
[ ] Suppress model IDs

Code update
[ ] Generate Object Browser documentation
[✓] Generate debug code
[✓] Generate "Your code goes here..." comments
[✓] Generate error handling code
Data member prefix: m

Model update
Default logical package: Reverse Engineered/
Default component package:
Overview diagram: Reverse Engineered/$component/Overview of $component

Figure 27 Visual Basic Properties Dialog Box—Code Update Properties

Save model before code and model update

Specifies whether to save the current open model (automatically or after confirmation) before generating anything. If you are not satisfied with the code update, you can revert to the saved model.

Note: *The contents of the model may be affected when updating the model from a Visual Basic project as well as when generating code from the model.*

Supress model IDS

Specifies whether to insert unique model identifiers in the Visual Basic code for new code items when generating or reverse engineering a Visual Basic project. See the section *Model IDs* in chapter 3.

Rational Rose Visual Basic needs the model IDs to synchronize the model with the code when generating or reverse engineering Visual Basic code. If you select this option, Rational Rose will no longer be able to automatically rename or remove classes and members when generating or reverse engineering code.

Generate Object Browser documentation

Specifies whether to make the documentation of the generated classes, properties, and methods available in the Object Browser in Visual Basic.

Other options

For information about the **Generate debug code**, **Generate "Your code goes here..." comments**, **Generate error handling code**, and **Data member prefix** options, please refer to section *Customizing the Default Behavior of the Model Assistant* in chapter 7.

Customizing the Code Generation of a Specific Component

To customize the code generation of a specific component:

1. Open the **Visual Basic Component Properties** dialog box for the component, by right-clicking on the component in the Code Update Tool and clicking **Properties**.
2. Select and specify the options that are relevant when generating code for a component, which are the following:

- **Should be generated**
- **Project file**
- **Stereotype**

  For information about each option, please refer to *Specifying a Component* in chapter 6.

3. Click **OK**.

Previewing and Customizing the Code to Generate for a Specific Class

In the right pane of the **Select Components and Classes** page in the Code Update Tool, you can preview, but not customize, the code to be generated for each member of the selected class.

To customize the code to generate for a class, you must open the Model Assistant as follows:

1. Right-click on the class or one of its members on the **Select Components and Classes** page in the Code Update Tool. Then click **Open**.
2. Define the class' mapping to Visual Basic code on the different tabs. For further information, see chapter 7.
3. Click **OK**.

Chapter 5

Reverse Engineering Visual Basic Code

Visual Basic reverse engineering is the process of examining a Visual Basic project to recover information about its design and to update the model accordingly. Rational Rose extracts design information and uses it to generate or update a model representing the application's logical structure. Rational Rose enables you to view and manipulate this model using the UML notation for object-oriented analysis and design.

A Visual Basic project corresponds to a component in the component view of a model. Reverse engineering takes place between the chosen components in the model and the associated Visual Basic projects.

This chapter provides an overview of the Model Update Tool and the basic steps for generating and updating a model from a Visual Basic project.

The Model Update Tool

Using the Model Update Tool you can reverse engineer a Visual Basic project to create a new model from a project or update an existing model with changes made to the code. With the Model Update Tool you can:

- Update several components of different implementation languages at the same time
- Keep the model and source code projects synchronized as the Model Update Tool detects any model elements that may have been deleted from the code
- Add new components to the model

Starting the Model Update Tool

You can start the Model Update Tool in several ways, for example:

- Click **Tools > Visual Basic > Update Model from Code.**
- Right-click a component and choose **Update Model from Code** from the displayed menu.
- In Microsoft Visual Basic, open a Visual Basic project and bring up the Rational Rose tool window by clicking **Add-Ins > Rational Rose**. Right-click the model file and click **Update Model** on the displayed menu.

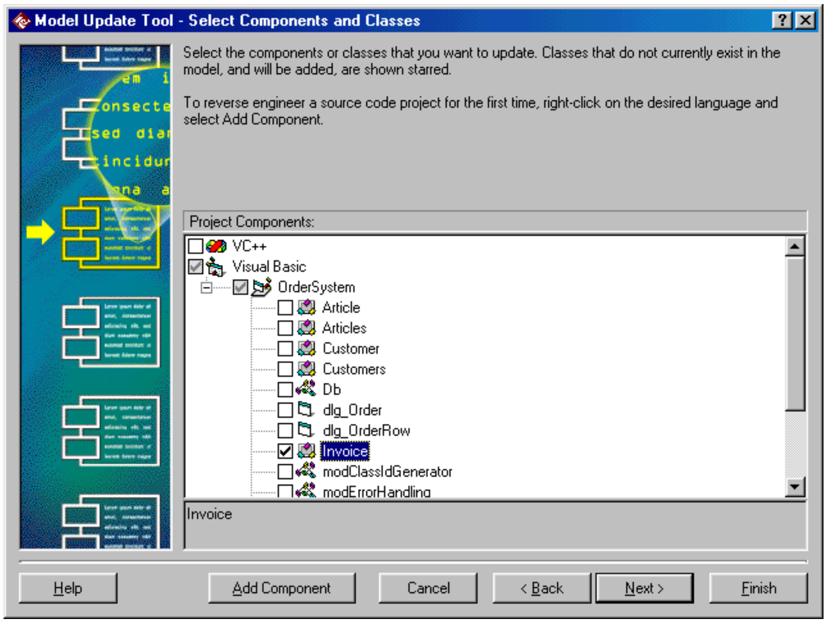
The Model Update Tool Pages

The Model Update Tool leads you through the process of updating a model from code on the following pages:

Welcome Page

The first page provides general information about the tool. You can turn this page off by selecting the **Don't show this page in the future** option.

Select Components and Classes Page



Select the components or classes that you want to update. Classes that do not currently exist in the model, and will be added, are shown starred.

To reverse engineer a source code project for the first time, right-click on the desired language and select Add Component.

Project Components:

- [ ] VC++
- [✓] Visual Basic
  - [✓] OrderSystem
    - [ ] Article
    - [ ] Articles
    - [ ] Customer
    - [ ] Customers
    - [ ] Db
    - [ ] dlg_Order
    - [ ] dlg_OrderRow
    - [✓] Invoice
    - [ ] modClassIdGenerator
    - [ ] modErrorHandling

Invoice

Figure 28 Model Update Tool—Select Components and Classes Page

On this page, you select the components that you want to update. A list of components is displayed. Each component (for example, OrderSystem in Figure 28) represents a source code project and is used to map the classes in the model to items in the project. To reverse engineer a project, the project must be represented in the model by a component.

Do any of the following:

- To update all classes in a project, select the corresponding component. To update or generate only some of the classes that are assigned to a component, expand the component and select those code classes.

  Note: *Classes marked with a star* *do not exist in the model. When selecting such a class, the class will be created in the model. Also, you can select several components. Each selected component is then updated from the changes of its associated project.*

- In order to update the model from a project, the model must contain a component that is associated with that project. Thus, to generate model elements from a project for the first time, create a corresponding component by right-clicking on the appropriate language node and clicking **Add Component** on the displayed menu.
- To associate an existing component with a project file, right-click on the component and click **Properties**, which brings up the **Visual Basic Component Properties** dialog box for the component.
- To customize general aspects of the model update, right-click on the Visual Basic language and click **Properties** on the displayed menu. See *Customizing the Model Update* in this chapter.

Finish Page

This page displays a summary of what will be updated in the model. Click **Finish** if you are satisfied, or **Back** if you want to change something.

Progress Page

This page displays the progress of the model update process.

Synchronize Page

This page is shown if the Model Update Tool detects any model elements that do not have any correspondences in the code. Here you can confirm the deletion of each such model element. See the section *Synchronizing Model and Code During Model Update* in this chapter.

Summary Page

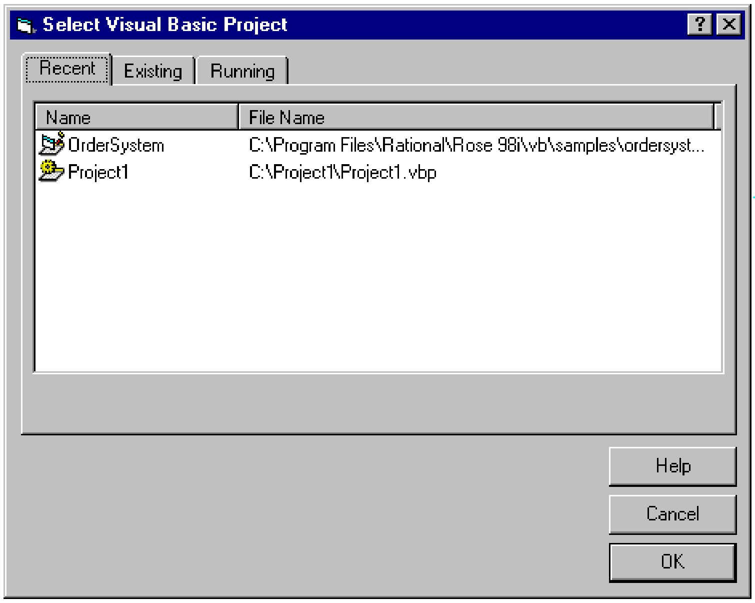
This page displays a summary of the result of the model update. On the shortcut menu, you can choose to display warnings and errors in color and to use timestamps. The information on the Log tab can also be found in the Rational Rose log window after exiting the Model Update Tool.

Updating a Model from Visual Basic Projects

Reverse Engineering a Visual Basic Project Into a New Component

To reverse engineer a source code project into the model:

1. Compile the project that you are going to reverse engineer and make sure that it does not contain any syntax errors.
2. Click **Tools > Visual Basic > Update Model from Code**. The Model Update Tool starts and the **Select Components and Classes** page is displayed. (If the **Welcome** page is shown, click **Next**.)
3. Before you can reverse engineer the project, you must associate the project with an existing or new component in the model:
   - ❑ To associate the project with a new component, right-click on the Visual Basic language and click **Add Component**. In the displayed dialog box (Figure 29), select the project file that you want to reverse engineer and click **OK**.



Name	File Name
OrderSystem	C:\Program Files\Rational\Rose 98i\vb\samples\ordersyst...
Project1	C:\Project1\Project1.vbp

Figure 29   Select Visual Basic Project Dialog Box

   - ❑ To associate the project with an existing component, right-click on that component and click **Properties**.

4. Select the project items that you want to reverse engineer, or select the check box next to the new component in order to reverse engineer the entire project.
5. On the **Finish** page, take a look at the model elements to be generated.
6. When the reverse engineering process has finished, take a look at the results on the **Summary** page. Click **OK**.
7. Evolve the updated model. For example, note that a logical package with the same name as the new component has been added to the logical view. The new logical package contains classes, properties, associations, and methods corresponding to the selected project items. Move the new logical package and its contents to an appropriate place in the logical view of your model. For more information, please refer to the *Evolving the Updated Model* section in this chapter.
8. Save the new model.
9. If the model was untitled before running the Model Update Tool, you must associate the project with the model file. To do that, open the Rational Rose tool window in Microsoft Visual Basic by clicking **Add-Ins > Rational Rose**. Click the refresh button, , in the toolbar. Rational Rose will then be able to find the associated model the next time you update the model from this project.

Updating a Component in the Model from Code Changes

To update a component with source code changes:

1. Compile the project that you are going to reverse engineer and make sure that it does not contain any syntax errors.
2. Click **Tools > Visual Basic > Update Model from Code**. The Model Update Tool starts and the **Select Components and Classes** page is displayed. (If the **Welcome** page is shown, click **Next**.)
3. Select the check box next to the component to update the model with all changes that have been made to the corresponding project, or select specific project items to insert only some of the changes.

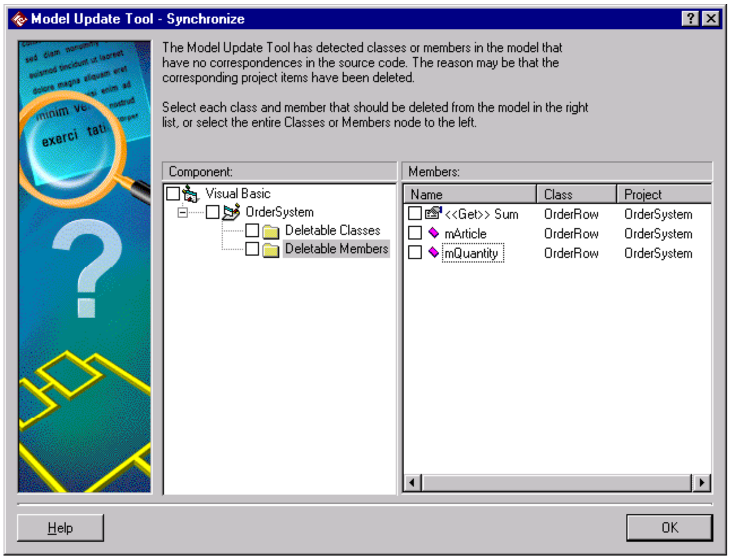
   Note: *For project items marked with a star there are no corresponding elements in the model yet.*

4. Click **Next** and take a look at the summary of what will be updated. Click **Finish** if you are satisfied.

5. If the model contains model elements that do not have any correspondences in the project, a **Synchronize** page is displayed. In the right-hand list, select those elements that you want to delete from the model. See the *Synchronizing Model and Code During Model Update* section in this chapter.
6. Take a look at the results of the code generation on the **Summary** page. Click **Close**.
7. Evolve the updated model. For example, if you updated the model with new classes, the new classes are inserted into a logical package called Reverse Engineered. Move the new classes to an appropriate place in the logical view of your model. For further information, please refer to the *Evolving the Updated Model* section below.
8. Save the model.

Synchronizing Model and Code During Model Update

Whenever you update the model from code, Rational Rose checks to see that the code and model agree. If the Model Update Tool finds a model element with no corresponding code, the tool displays a **Synchronize** page:


Model Update Tool - Synchronize

The Model Update Tool has detected classes or members in the model that have no correspondences in the source code. The reason may be that the corresponding project items have been deleted.

Select each class and member that should be deleted from the model in the right list, or select the entire Classes or Members node to the left.

Component:
- Visual Basic
  - OrderSystem
    - Deletable Classes
    - Deletable Members

Members:

Name	Class	Project
<<Get>> Sum	OrderRow	OrderSystem
mArticle	OrderRow	OrderSystem
mQuantity	OrderRow	OrderSystem

Figure 30 Model Update Tool—Synchronize Page

To delete selected model elements:

1. In the right-hand list, select the check box next to each model element that you want to delete.
2. Click **OK** to delete the selected elements.

To delete all model elements of the same kind:

1. In the left-hand list, select the check box next to the appropriate delete folder (Classes or Members.) The contents of the folder are displayed in the right-hand list and all elements that will be deleted are selected.
2. Click **OK** to delete the selected elements, or clear those elements you do not want deleted.

Evolving the Updated Model

After the Model Update tool has finished, classes, properties (attributes), associations and methods (operations) corresponding to the selected project items are inserted into your new model. Also, any project items that were not selected for model update, but which are referred to from other project items, are created as empty classes in the model.

To evolve the updated model:

1. Save the updated model.
2. After updating the model with Visual Basic project items that do not have any model correspondences yet, a new class is created in the model for each such project item. Each new class is placed in a logical package specified by the **Default logical package** option in the **Visual Basic Properties** dialog box.

   Move any new classes to the logical package where they are supposed to be located in the model. You may need to create new logical packages if some classes do not naturally belong to any of the existing logical packages.
3. After reverse engineering a Visual Basic project for the first time, a corresponding component is created in the component view of the model. The new component is placed in a component package specified by the **Default component package** option in the **Visual Basic Properties** dialog box. Move any new components to the component package where they are supposed to be located in the model.

4. If a reverse engineered project has new ActiveX references, identify the imported ActiveX components as used components, and not classes that need to be implemented.
5. If the reverse engineered Visual Basic project uses another data member prefix (for example "the") than the prefix defined by the **Data member prefix** option in the **Visual Basic Properties** dialog box (which is "m" by default) Rational Rose will not be able to connect the data member with its property procedures in the model. Thus, a data member that has property procedures assigned to it in the code will generate two properties under the Properties folder in the Model Assistant. To combine each such pair of properties into one and the same property, drag one of the properties and drop it on the other as shown in Figure 31.

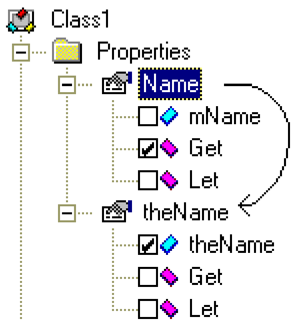


Figure 31 Use Drag and Drop to Combine Properties with Different Data Member Prefix

6. Insert any new classes and components into the appropriate diagrams, by using drag-and-drop from browser to diagram, to illustrate the architecture of the system. Avoid crossed association lines by moving the classes in the diagram, and use the **Edit > Diagram Object Properties** command to control the level of class details in a diagram.

Now you are ready to further develop the model by creating new logical packages, classes, relationships, properties, and methods, as well as illustrating the model in diagrams.

Customizing the Model Update

You can customize several aspects of the model update. For example, you can customize the behavior of the Model Update Tool or choose whether to import project references into the model.

Customizing the Model Update Tool

You can customize general aspects of a model update in the **Visual Basic Properties** dialog box.

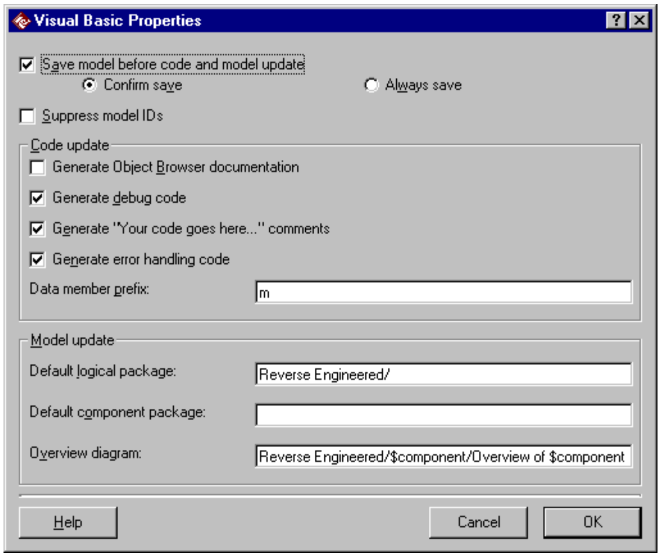

Visual Basic Properties
?
[✓] Save model before code and model update
(●) Confirm save
( ) Always save
[ ] Suppress model IDs
Code update
[ ] Generate Object Browser documentation
[✓] Generate debug code
[✓] Generate "Your code goes here..." comments
[✓] Generate error handling code
Data member prefix: m
Model update
Default logical package: Reverse Engineered/
Default component package:
Overview diagram: Reverse Engineered/$component/Overview of $component
Help
Cancel
OK

Figure 32 Visual Basic Properties Dialog Box—Model Update Tool

To open the **Visual Basic Properties** dialog box, right-click the Visual Basic language on the **Select Components and Classes** page in the Model Update Tool, and click **Properties**. (Or click **Tools > Visual Basic > Properties**.)

The following options are relevant when customizing the model update:

Save model before code and model update

Specifies whether to save the current open model (automatically or after confirmation) before updating the model. When the model update is finished, and if you are not satisfied with the result, you can revert to the saved model.

Suppress model IDS

Specifies whether to insert unique model identifiers in the Visual Basic code for new code items when generating or reverse engineering a Visual Basic project. See the section *Model IDs* in chapter 3.

Note: *Rational Rose Visual Basic needs the Model IDs to synchronize the model with the code when generating or reverse engineering Visual Basic code. Thus, if you select this option, Rational Rose will no longer be able to automatically rename or remove classes and members when generating or reverse engineering code.*

Default logical package

Specifies the logical package into which new classes that are created during model update are inserted. To insert classes into a new or existing logical package, type the name of that package including the path of any enclosing packages. The Model Update Tool will create the packages for you if they do not exist.

For example, if you type “Package1/Package2”, new classes are inserted into the following logical package:

```
Logical View
  Package1
    Package2
```

You can use the following variables in the package name:

- $component—the name of the component where the currently reverse engineered class belongs
- $language—the implementation language of the currently reverse engineered class

For example, “Reverse Engineered/$component” means that a new Visual Basic class called MyClass, which belongs to the component MyComponent, is inserted into the following logical package:

```
Logical View
  Reverse Engineered
    MyComponent
```

Default component package

Specifies the component package into which new components that are created during model update are inserted.

If this box is empty, new components are inserted at the top level of the component view. To insert the component into a new or existing component package, type the name of that package including the path of any enclosing packages. The Model Update Tool will create the packages for you if they do not exist.

For example, if you type “Package1/Package2”, new components are inserted into the following component package:

```
Component View
  Package1
    Package2
```

You can use the following variables in the package name:

- $component—the name of the reverse engineered component
- $language—the implementation language of the reverse engineered component

For example, “Reverse Engineered/$language/$component” means that a new Visual Basic component called MyComponent, is inserted into the following component package:

```
Component View
  Reverse Engineered
    Visual Basic
      MyComponent
```

Overview diagram

Specifies the name and location of the class diagram that Rose creates when you reverse engineer a Visual Basic project. The diagram shows all new classes that are created in the model during the model update.

You can use the following variables in the package path:

- $component—the name of the reverse engineered component
- $language—the implementation language of the component

For example, “Reverse Engineered/$component/Overview of $component” means that a class diagram called “Overview of MyComponent”, is inserted into the following logical package when reverse engineering a Visual Basic project called MyComponent:

```
Logical View
   Reverse Engineered
      MyComponent
```

Customizing the Type Library Importer

In the **COM Properties** dialog box, you can control several aspects of how type libraries are imported into the model. For example, you can control:

- What should happen with existing type libraries when importing new versions
- The name and location of new type libraries in the model
- The name, location, and contents of the overview diagrams that are created when importing type libraries

To open the **COM Properties** dialog box, click **Tools > COM > Properties**. For more information, see the *Customizing the Type Library Importer* section in the *Type Library Importer* chapter in the *Using Rational Rose* manual.

Customizing the Model Update of a Specific Component

To customize the model update rules for a specific component:

1. Right-click on the component in the Model Update Tool and click **Properties.** The **Visual Basic Component Properties** dialog box opens.
2. Select and specify the options that are relevant when updating a component from code changes, which are the following:
   - **Should be updated from code**
   - **Project file**
   - **Import all references**

- **Quick import**
- **Import the compiled VB binary**

  For information about each option, please refer to the section *Specifying a Component* in chapter 6.

3. Click **OK**.

Customizing the Model Update of a Specific Class

To specify that a specific class is not allowed to be updated from code changes:

1. Before entering the Model Update Tool, open the Model Assistant for the class by right-clicking on the class and clicking **Model Assistant**.
2. On the **Class** tab, clear the **Should be updated from code** option.

Chapter 6

Modeling a Visual Basic Project

To model a Visual Basic project means to create a component for the project in the model, create and specify the classes that belong to the project, and to assign those classes to the corresponding component in the model.

The components are needed to map each class in the logical view to the appropriate implementation language and source code project (as illustrated in Figure 1 in chapter 2.) You cannot generate code for a class until it has been assigned to one or several components in the model. Also, to update a model from a source code project a component corresponding to that project must exist in the model.

You can always use the standard procedures that Rational Rose provides to create Visual Basic components in the model and to assign classes to them. See the *Using Rose* manual. However, the Visual Basic Language Support add-in provides a separate tool, the Component Assignment Tool, which is specifically made for the creation and specification of Visual Basic components.

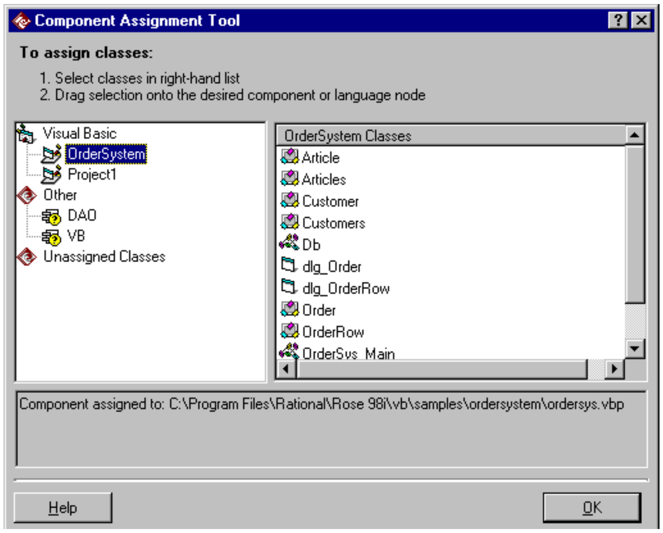
This chapter explains how to create a new model, how to create Visual Basic components in the model using the Component Assignment Tool, and how to assign classes to Visual Basic components. It also explains how to import external software modules into the model and how to browse the model and the code.

The Component Assignment Tool

The Component Assignment Tool provides you with an easy-to-use interface to:

- Create new components (see the section *Creating New Components in the Model* later in this chapter)
- Assign classes to components by using drag-and-drop (see the section *Assigning Classes to Existing Components*)
- Associate a component with a Visual Basic project (see the section *Associating a Component with a Visual Basic Project File*)

The advantage of using the Component Assignment Tool is that the components you create here will contain all the information needed in order to generate Visual Basic code. The tool also provides specific help to assign classes to components, as it finds and displays all currently unassigned classes in a separate folder called Unassigned Classes (see Figure 33.)



To assign classes:

1. Select classes in right-hand list
2. Drag selection onto the desired component or language node

Component assigned to: C:\Program Files\Rational\Rose 98i\vb\samples\ordersystem\ordersys.vbp

Figure 33 The Component Assignment Tool

You can open the Component Assignment Tool in two ways:

- Click **Tools > Visual Basic > Component Assignment Tool**
- Right-click a component in the Code Update Tool and click **Assign Classes**.

For information on how to use the Component Assignment Tool, see the section *Creating Components and Assigning Classes* in this chapter.

Creating a New Model

Rational Rose is delivered with a library of predefined frameworks. By basing new models on one of these frameworks you get immediate access to a lot of reusable components in your model. Note that the code generator does not generate declarations for the predefined data types and class modules in the frameworks. Only references to these types and class modules are generated.

To create a new model, perform the following steps:

1. Click **File > New**.
2. If the Framework add-in is installed and activated, the framework dialog box opens. Open the model framework that corresponds to the system you are going to develop. A new model is created and initialized with the contents of the chosen framework. (If you don't want to use any of the frameworks, click **Cancel**. A new model with nothing but the default contents is created.)
3. If the Framework add-in is not installed or not activated, an empty model is opened.
4. Save the new model and give it a name by clicking **File > Save As**.

Note: *Each package in a framework is stored as a controlled unit in a separate file. To access the contents of a package in a framework, you must load the corresponding controlled unit. To load a unit, double-click on the package in a diagram, or click **File > Units > Load**. For more information about controlled units, please refer to the Team Development chapter in the Using Rose manual.*

The Contents of a New Model

A new model contains by default the following:

Use-Case View—which specifies the behavior and surroundings of the system in terms of use cases and actors. By default it contains a use-case diagram called Main, which is intended to illustrate an overview of the use-case model.

Logical View—which describes the logical structure of the system, that is, the classes and their relationships. If the **3 Tiered Diagram** option on the **Diagram** tab of the **Options** dialog box is selected, the logical view of a new model contains the following:

- Three logical packages representing the three fundamental service layers of a three-tiered model: User Services, Business Services, and Data Services.
- A class diagram, called Package Overview, for each service layer package. Rational Rose Visual Basic automatically inserts each new class diagram, or package, into the Package Overview for the service layer to which it belongs.
- A three-tiered diagram called Three-Tiered Service Model. Rational Rose Visual Basic automatically inserts each new class or logical package into this diagram, in the tier representing the service layer to which it belongs.

  Note: *For more information about three-tiered diagrams, please refer to the online help or the Using Rose manual.*

If the **3 Tiered Diagram** option is cleared, the logical view of a new model contains only a main class diagram.

Component View—which describes the physical structure of the system, that is, how the system is divided into **.exe** files and DLLs. By associating classes to components, and components to Visual Basic projects, the component view defines in which Visual Basic project(s) the classes in the model are contained.

Deployment View—which shows the connections between the system's processors and devices, and the allocation of its processes to processors. It contains by default a diagram called Deployment Diagram.

Creating Components and Assigning Classes

Creating New Components in the Model

To create new components:

1. Click **Tools > Visual Basic > Component Assignment Tool**.
2. In the left list, right-click on the desired language for the new components.
3. On the displayed menu, click **New Component**. The following dialog box is opened:

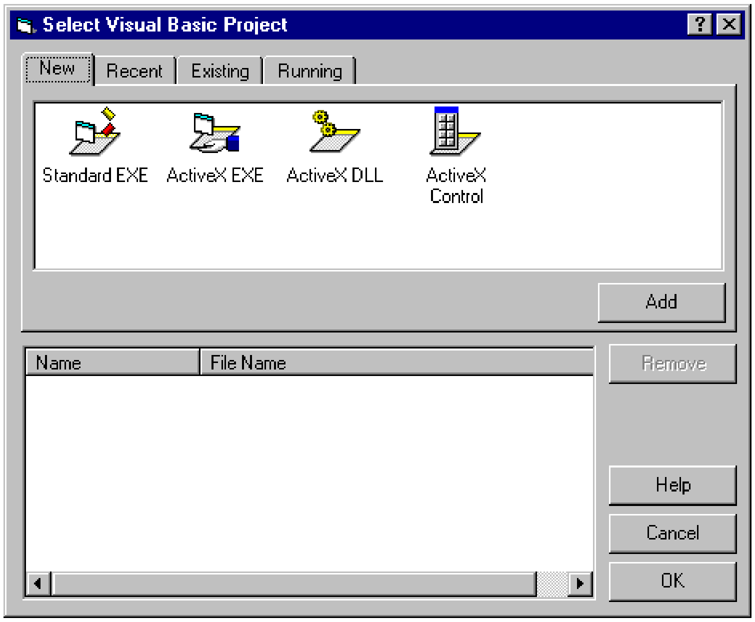


Figure 34 Select Visual Basic Project Dialog Box

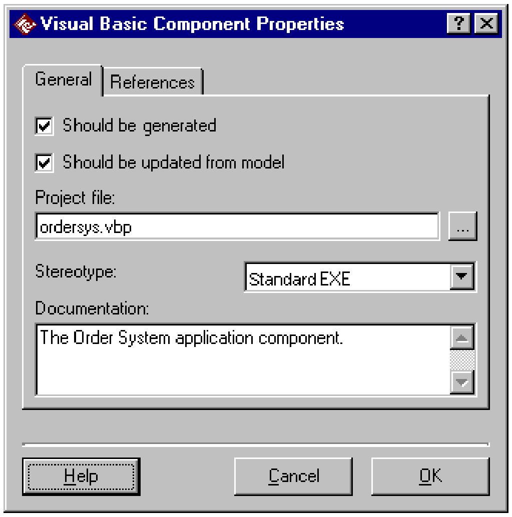
4. For each new component to create: On the **New** tab, select the type of Visual Basic project, to which the component corresponds. Or, on one of the other tabs, select an existing Visual Basic project, which the new component is supposed to represent. Click **Add**.
5. Click **OK**. New Visual Basic components, with the same name as the selected projects, or the default name `Project1,Project2,` etc. for new components, are inserted into the model.

6. To change the name of a component, right-click on the component, and then click **Rename**.
7. Assign classes to the new components. See the *Assigning Classes to Existing Components* section in this chapter.

Note: *You can also create components in the browser or in a component diagram. The advantage of using the Component Assignment Tool is that it helps you to create Visual Basic components in specific.*

Specifying a Component

You can specify Visual Basic information about a component in the **Visual Basic Component Properties** dialog box. To open that dialog box for a component, right-click the component and click **Properties**.



☑ Should be generated

☑ Should be updated from model

Project file: ordersys.vbp

Stereotype: Standard EXE

Documentation: The Order System application component.

Figure 35 Visual Basic Component Properties Dialog Box

Enter information about the component by selecting or specifying the following options:

Should be generated

Specifies whether it is possible to generate code for this component. If this option is cleared, the component cannot be selected for code generation.

Should be updated from code

Specifies whether it is possible to update the classes assigned to this component from code changes. If this option is cleared, the component cannot be selected for model update.

Project File

Specifies the project file (.vbp) for the Visual Basic project that is associated with this component.

When generating code for a component, Rational Rose Visual Basic generates the code into the project file specified here. If the Project file box contains a name of a project file, but no file path, the code generator assumes that the project file is located in the same folder as the model file.

If the Project file box is empty and the open project has the same name as the component, or if the component has been generated from/into that project before, Rational Rose generates code into the current open Visual Basic project. Otherwise, Rational Rose starts the Visual Basic application and generates code into a new project.

For more information, see *Associating a Component with a Visual Basic Project File* in this chapter.

Note: *If you have defined a virtual path map for your project—for example "$MYPATH=C:\VB_Projects\my_proj"—Rational Rose stores the project reference using the path map symbol—for example "$MYPATH\my_proj.vbp". However, this dialog box will display the reference as the corresponding absolute path.*

Stereotype

Defines the project type of the Visual Basic project that this component represents. For more information, see the section *Components* in chapter 2.

Documentation

A textual description of the component. The text can also be found in the Documentation box of the Component Specification.

Import all references

Specifies whether to import the type library of the COM components that are referred to from the Visual Basic project associated to this component. Use the **Quick import** option to specify how much of the referenced COM components you want to import.

Quick import

If this option is selected, the Model Update Tool imports only the interface classes of the referenced COM components. If this option is cleared, the properties and methods of the imported interfaces are also added to the model. This option is relevant only if the **Import all references** option is selected.

Import the compiled VB binary

If this option is selected, the Model Update Tool quick imports the type library of the binary component that is compiled from this Visual Basic project. Select this option if other components in the model need to use the interface of this component. This option is not available for Standard EXE components.

Associating a Component with a Visual Basic Project File

In order to generate code for a component or update the classes that are assigned to the component from code changes, the component must be associated with a Visual Basic project. The associated project is specified in the **Visual Basic Component Properties** dialog box.

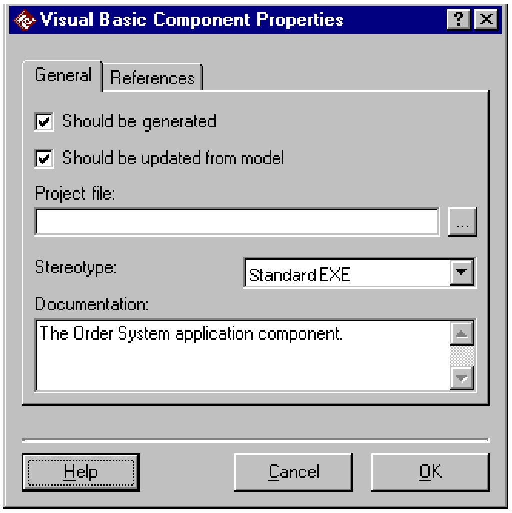
When selecting a component for code generation, not yet associated with a project, the Code Update Tool will automatically associate it with a project using to the following rules:

- Rational Rose generates code into the current open Visual Basic project:
  - if the open project has the same name as the component, or
  - if the component has been generated from/into that project before.
- Otherwise, Rational Rose starts the Visual Basic application and generates code into a new project.

However, you can also associate a component with any Visual Basic project before generating code.

To manually associate a component with a Visual Basic project:

1. In the browser, in a diagram, or in the Component Assignment Tool, right-click the component and click **Properties**. The **Visual Basic Component Properties** dialog box opens:



☑ Should be generated

☑ Should be updated from model

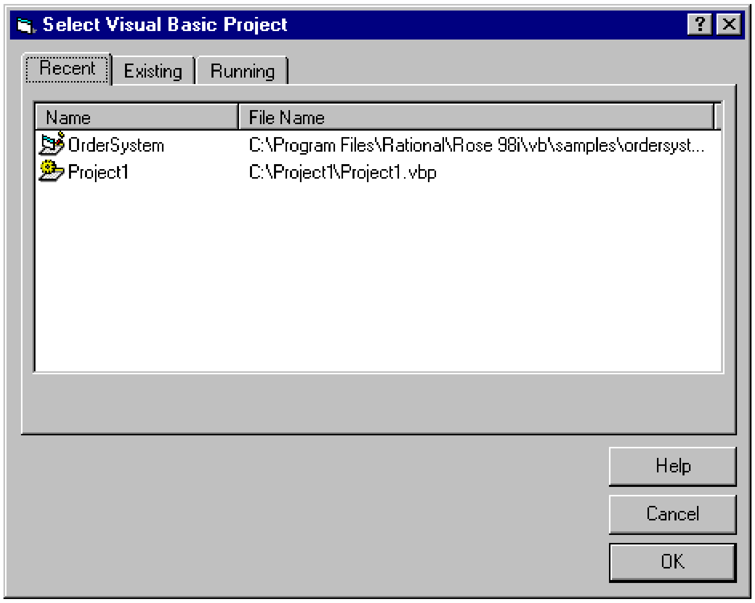
Project file:

Stereotype: Standard EXE

Documentation: The Order System application component.

Figure 36 Visual Basic Component Properties Dialog Box

2. Click the browse button, ... which brings up the **Select Visual Basic Project** dialog box:



Name	File Name
OrderSystem	C:\Program Files\Rational\Rose 98i\vb\samples\ordersyst...
Project1	C:\Project1\Project1.vbp

Figure 37 Select Visual Basic Project Dialog Box

3. Select the Visual Basic project file that should be associated with the component and click **OK** twice. The selected project is then associated with the component. The next time you generate code for this component, the code is generated into the Visual Basic project you selected.

Assigning Classes to Existing Components

To generate Visual Basic code for a class, a component that represents the Visual Basic project where the class belongs must exist in the model, and the class must be assigned to that component. A class or interface can be assigned to a component of a specific implementation language, or to several components with the same language.

Assigning Unassigned Classes to a Component

1. Click **Tools > Visual Basic > Component Assignment Tool**.
2. Select the Unassigned Classes folder.

3. In the right list, select the classes that are to be assigned to the component.
4. Drag the selected classes to the component.

Note: *You can also assign classes on the* **Realizes** *tab of the Component Specification dialog by right-clicking on the class and clicking Assign on the displayed menu.*

Assigning a Class to Several Components of the Same Language

A class can be assigned to one or several components, but all components must have the same implementation language.

1. Click **Tools > Visual Basic > Component Assignment Tool**.
2. In the left list, select a component to which the class is already assigned. Select the class in the right list.
3. Point to the class. Hold down CTRL while pressing the right mouse button. Drag the selected class to the other component. Then release both the CTRL key and the mouse button.

Note: *You can also drag the class from the browser and drop it on the appropriate component in a diagram, in the browser, or on the* **Realizes** *tab of the Component Specification.*

Moving a Class to Another Component

Moving a Class to Another Component with the Same Language

1. Click **Tools > Visual Basic > Component Assignment Tool**.
2. In the left list, select the component to which the class is assigned.
3. In the right list, drag the class and drop it on its new component in the left list.

Note: *You can also drag the class from the browser and drop it on the appropriate component in a diagram, in the browser, or on the* **Realizes** *tab of the Component Specification.*

Moving a Class to a Component with Another Language

A class can be reassigned to a component with a different implementation language. Note, however, that if you have used language specific data types in the specification of the class, you must manually change the data types to conform to the new language. If the reassigned class is also assigned to other components, it will be deleted from those components.

1. Click **Tools > Visual Basic > Component Assignment Tool**.
2. In the left list, select the component to which the class is assigned.
3. In the right list, select the class.
4. Drag the class to the other component.
5. Because the other component is of another implementation language, a warning message is displayed. Click **OK**, and the necessary assignments/reassignments will be made. Or, click **Cancel**, and the class will not be reassigned to the new component.

If you have used language specific data types in the specification of the class, open the Class Specification and change the data types for all methods, formal arguments, and properties to conform to the new language.

Removing a Class from a Component

Removing a Class from One Component

1. Click **Tools > Visual Basic > Component Assignment Tool**.
2. In the left list, select the component from which the class is to be removed.
3. In the right list, right-click on the class and click **Unassign** on the displayed menu.

Note: *You can also unassign the class on the Components tab of its Class Specification dialog by right-clicking on the component and clicking Unassign on the displayed menu.*

Removing a Class from All Assigned Components

1. Click **Tools > Visual Basic > Component Assignment Tool**.
2. In the left list, select a component to which the class is assigned.

3. Drag the class (or several classes) from the right list and drop it on the Unassigned Classes folder in the left list.
4. A warning dialog box is displayed. Click **OK**.

Changing the Implementation Language of a Component

To change the language of a component:

1. Click **Tools > Visual Basic > Component Assignment Tool**.
2. In the left list, select the current language of the component.
3. Drag the component from the right list and drop it on its new language.
4. If any class of the selected component is also assigned to other components, a dialog box is displayed where you must confirm the language change. Those classes are then removed from the component before it is assigned to the new language.

Note: *The data types that are used in the specification of the assigned classes' methods, formal arguments, and properties will remain data types of the original language. You must manually change all data types that do not conform to the new language.*

Importing Type Libraries Into the Model

If you need to use existing software components—.exe, .dll, .ocx, or .tlb files—in the system you are modeling, you should import the type library of those COM components into your model. By importing type libraries into the model, you can show how classes in the model use and depend upon the interface of COM components, regardless of the COM components' implementation languages.

There are several ways to import a COM component's type library into the model. The way you choose depends on which COM component you are importing. You can import the type library for

- Project references
- The binary component compiled from a Visual Basic project in the model
- Any COM component

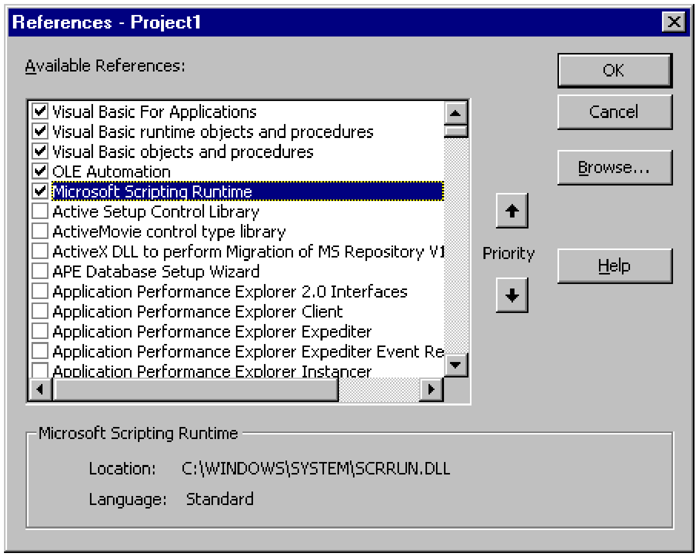
Importing the Type Library of Project References

When you reverse engineer a Visual Basic project, you can automatically import the type libraries of all project references into the model. You can also do this manually for individual components.

To automatically import the type library of project references into the model:

Specify the following settings to make the Model Update Tool automatically import all project references when updating the model from a project:

- The Visual Basic project being reverse engineered must reference the component. To see the components that a project references, click **Project > References** in Visual Basic, which brings up the **References** dialog box:



- ✓ Visual Basic For Applications
- ✓ Visual Basic runtime objects and procedures
- ✓ Visual Basic objects and procedures
- ✓ OLE Automation
- ✓ Microsoft Scripting Runtime
- ☐ Active Setup Control Library
- ☐ ActiveMovie control type library
- ☐ ActiveX DLL to perform Migration of MS Repository V1
- ☐ APE Database Setup Wizard
- ☐ Application Performance Explorer 2.0 Interfaces
- ☐ Application Performance Explorer Client
- ☐ Application Performance Explorer Expediter
- ☐ Application Performance Explorer Expediter Event Re
- ☐ Application Performance Explorer Instancer

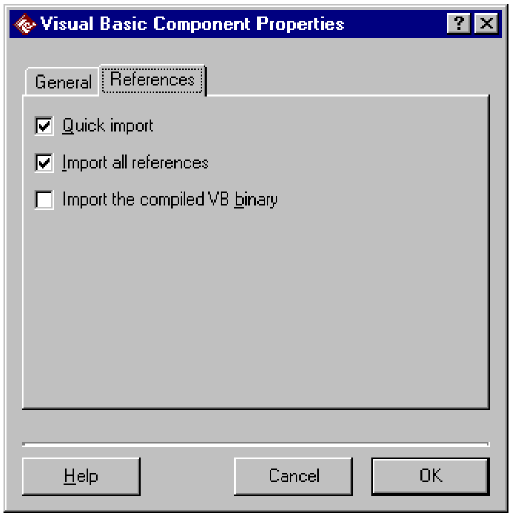
Microsoft Scripting Runtime

Location: C:\WINDOWS\SYSTEM\SCRRUN.DLL

Language: Standard

Figure 38 The References Dialog Box in Visual Basic

- Select the **Import all references** option in the **Visual Basic Component Properties** dialog box (see Figure 39.) By default, the Model Update Tool performs a quick import. That is, it imports the type library items without properties and methods. To also import the properties and methods, clear the **Quick import** option.



Visual Basic Component Properties

General | References

[✓] Quick import

[✓] Import all references

[ ] Import the compiled VB binary

Figure 39 The References Tab of the Component Properties Dialog Box

- If the referenced component is to be used by Visual Basic clients only, it is recommended that the **Show hidden items** option in the **Tools > COM > Properties** dialog box be cleared. See the *Hiding Type Library Items* section in the *Type Library Importer* chapter in the *Using Rational Rose* manual.

The Model Update Tool creates a COM component and a logical package for each of the updated project's references; see the *How Is a Type Library Presented?* section in the *Type Library Importer* chapter in the *Using Rational Rose* manual.

The Model Update Tool also creates a dependency relationship between the updated project's component and the imported COM components.

To manually import the type library of project references into the model:

1. Select the component that corresponds to the modeled project in a diagram.
2. Click **Tools > Visual Basic > Add Reference**.

Rational Rose creates a dependency relationship between the selected Visual Basic component and the imported COM component. The next time you generate code, the COM component is added as a reference to the generated Visual Basic project.

Importing the Binary Component Compiled from a Project in the Model

When reverse engineering a Visual Basic project, the type library of its compiled binary component can be automatically imported into the model. You can also manually import the type library of a project's binary component; see the *Importing a Type Library Into the Model* section in the *Type Library Importer* chapter in the *Using Rational Rose* manual.

After importing the project's binary component, there will be two components with the same name in the model:

- A Visual Basic component representing the source view of the component, which corresponds to the Visual Basic project
- A COM component representing the public view as seen from other COM components, which corresponds to the compiled binary component

By including the type library of a project's binary component in the model, you can show how other components (projects) in the model use this project. It is only possible to import the type library of ActiveX projects, because Standard EXE projects do not have a public interface.

To automatically import the type library of the binary component compiled from a project:

1. If the COM component is to be used by Visual Basic clients only, you may want to clear the **Show hidden items** option in the **Tools > COM > Properties** dialog box.

2. In the browser or in a diagram, locate the component that corresponds to the project for which you want to import the type library. Right-click the component and click **Properties**.
3. On the **References** tab, select the **Import the compiled VB binary** option and click **OK**.
4. Make sure that the Visual Basic project has been compiled and that its binary component has been created.
5. Update the model from the project; see chapter 5.

The Model Update Tool imports the type library and creates a COM component and a logical package for it in the model. The Model Update Tool also creates dependency relationships between the updated project's classes and the corresponding COM coclasses in the type library.

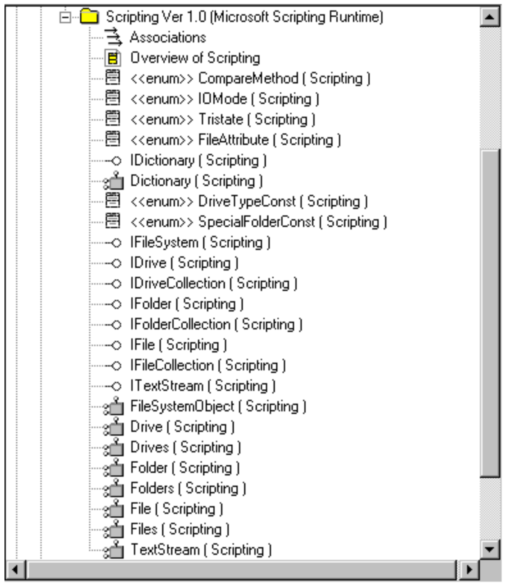
The type library is imported quick. To also import the type library's members, see the *Adding Class Members to a Quick Import Type Library* section in the *Type Library Importer* chapter in the *Using Rational Rose* manual.

Importing any COM Component

See the *Importing a Type Library Into the Model* section in the *Type Library Importer* chapter in the *Using Rational Rose* manual.

Example of an Imported Type Library

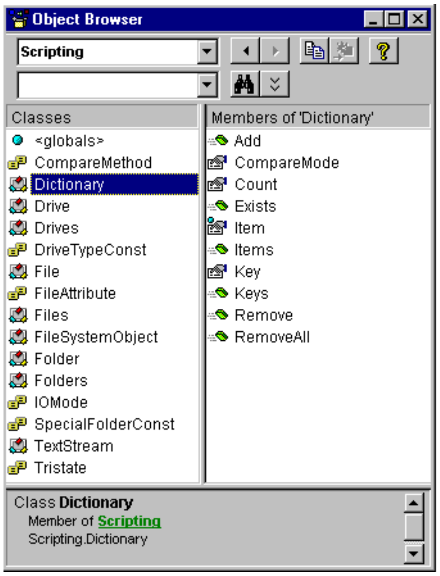
In this example, the type library of the Microsoft Scripting Runtime component has been imported into the model as illustrated in Figure 40.



Scripting Ver 1.0 (Microsoft Scripting Runtime)
Associations
Overview of Scripting
<<enum>> CompareMethod (Scripting)
<<enum>> IOMode (Scripting)
<<enum>> Tristate (Scripting)
<<enum>> FileAttribute (Scripting)
IDictionary (Scripting)
Dictionary (Scripting)
<<enum>> DriveTypeConst (Scripting)
<<enum>> SpecialFolderConst (Scripting)
IFileSystem (Scripting)
IDrive (Scripting)
IDriveCollection (Scripting)
IFolder (Scripting)
IFolderCollection (Scripting)
IFile (Scripting)
IFileCollection (Scripting)
ITextStream (Scripting)
FileSystemObject (Scripting)
Drive (Scripting)
Drives (Scripting)
Folder (Scripting)
Folders (Scripting)
File (Scripting)
Files (Scripting)
TextStream (Scripting)

Figure 40 The Microsoft Scripting Runtime Component Has Been Imported into the Model

In this example, the Microsoft Scripting Runtime type library has been imported with the **Show hidden items** option selected in the **Tools > COM > Properties** dialog box, which means that both coclasses and their default interfaces are shown directly under the type library's package. Because Visual Basic assumes the default interface when referring to a coclass, the Object Browser in Visual Basic hides the default interfaces of coclasses, as you can see in the following illustration:



Class **Dictionary**
Member of **Scripting**
Scripting.Dictionary

Figure 41 The Object Browser in Visual Basic Displays the Microsoft Scripting Runtime Component

Thus, if the COM component is to be used by Visual Basic classes only, it is recommended that you clear the **Show hidden items** option. Then the browser in Rational Rose will give the same view of the imported type library as the Object Browser in Visual Basic.

An imported type library belongs to the logical view, but its interfaces and dispinterfaces can also be displayed on component diagrams as shown in Figure 42.

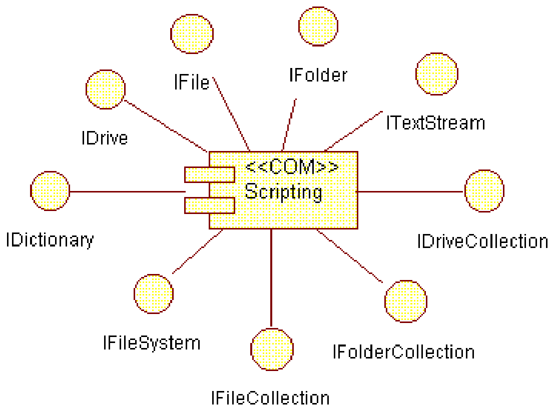


Figure 42 The Appearance of Interfaces on a Component Diagram

The items in the imported Microsoft Scripting Runtime type library can be used by the Visual Basic classes in the model by creating relationships between the classes and the imported coclasses and interfaces. See *Using an Imported Type Library* below.

Using an Imported Type Library

The application you are modeling can use the data types defined by a COM component or realize the interface of a COM component. Before you can use a COM component in the model you must import the type library of that component.

Using the Data Types Defined by a Type Library

You can reuse an imported data type by creating an association relationship between a class and the imported data type. You can associate the class with either a coclass in an imported type library or with the coclass' default interface. The generated code gives the same result when compiled.

For example, you can reuse the Dictionary of the Microsoft Scripting Runtime component by creating an association relationship between the class that has a dictionary and the imported Dictionary coclass, as follows:

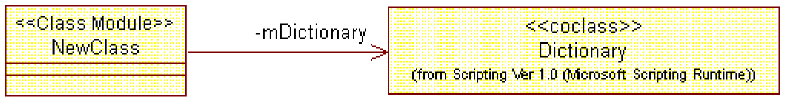


Figure 43 NewClass Has a Relationship with the Coclass Dictionary

Or, you can create the relationship between NewClass and the default interface of Dictionary, which is called IDictionary, as follows:

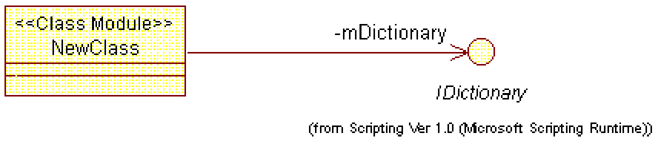


Figure 44 NewClass Has a Relationship with IDictionary—the Default Interface of the coclass Dictionary

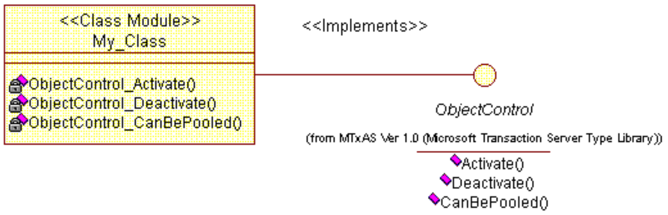
Note that for imported controls, the aggregation relationship (or association relationship with the stereotype Contained Control) is used to specify which controls are used in each form.

Realizing the Interface of a COM Component

Some COM components define interfaces that the classes in other applications may conform to. You can show that a class in the model realizes an interface of an imported type library, by creating a realize relationship between the class and the interface (see *Realize Relationships* in chapter 2).

Because an interface of a COM component is always abstract, and cannot be instantiated, you should use the realize relationship, which does not result in any delegation code, and not the generalization relationship.

For example, a class, My_Class, that is supposed to run in Microsoft Transaction Server may need to provide the methods defined by the ObjectControl interface of the Microsoft Transaction Server Type Library. The ObjectControl interface provides three methods, Activate, Deactivate, and CanBePooled. In order for My_Class to also provide those methods, you should create a realize relationship between My_Class and the imported ObjectControl interface. When you open My_Class in the Model Assistant or generate code from it, the realized methods are automatically added to My_Class as shown in the following picture:



<<Class Module>>
My_Class
ObjectControl_Activate()
ObjectControl_Deactivate()
ObjectControl_CanBePooled()
<<Implements>>
ObjectControl
(from MTxAS Ver 1.0 (Microsoft Transaction Server Type Library))
Activate()
Deactivate()
CanBePooled()

Figure 45 My_Class Realizes the Interface of ObjectControl

Note: *Instead of creating a realize relationship with a coclass' default interface, you can create a realize relationship with the coclass itself. The result of the generated code is the same when compiled.*

Using Property Procedures of Type Library Items

An imported interface may have methods with the stereotype propget, propput, and propputref. Those method stereotypes correspond to Visual Basic's property procedures as shown in Table 11.

Table 11 Property Procedures on COM Interfaces

Stereotype	**Corresponds to**
propget	Property Get
propput	Property Let
propputref	Property Set

For example, the ObjectContext interface in Microsoft Transaction Server Type Library has four methods with the stereotype propget: Count, Item, _NewEnum, and Security. Thus, if a class realizes the ObjectContext interface, the corresponding Property Get procedures are automatically added to the class.

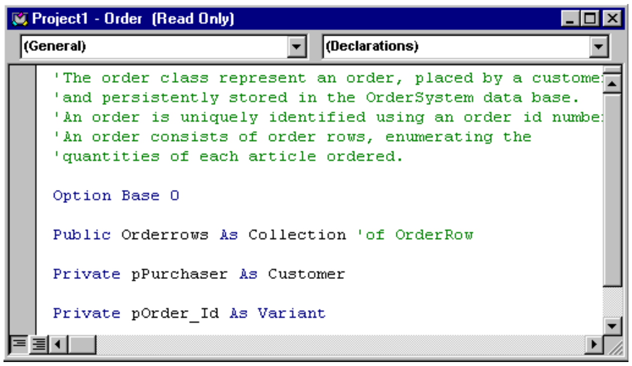
Browsing the Model and Code

The Rational Rose and Microsoft Visual Basic applications are tightly integrated—and not only with respect to code and class generation. You can also use each application to browse corresponding code or models in the other tool. In fact, there is a separate tool in Microsoft Visual Basic, called the Rational Rose tool window, which is used to browse the models that correspond to a project.

Opening the Visual Basic Item that Corresponds to a Model Element

To open the code item for a model element:

1. Select the component, class, or member in Rational Rose.
2. Click **Tools > Visual Basic > Browse Source Code**. (Or, click **Browse Source** on the shortcut menu for the model element.) The Visual Basic item that corresponds to the selected element in the model is displayed in Visual Basic as in the figure below.



```
'The order class represent an order, placed by a custome
'and persistently stored in the OrderSystem data base.
'An order is uniquely identified using an order id numbe
'An order consists of order rows, enumerating the
'quantities of each article ordered.

Option Base 0

Public Orderrows As Collection 'of OrderRow

Private pPurchaser As Customer

Private pOrder_Id As Variant
```

Figure 46 The Browse Source Code Command Displays Code for a Selected Class

Opening a Model that Corresponds to the Open Visual Basic Project

The Visual Basic Language Support Add-In provides a tool window in Microsoft Visual Basic, which displays a list of the currently open Visual Basic projects and the Rational Rose models to which each project is associated. With the Rational Rose tool window you can associate an open project with a model file, open an associated model file, or even create a new model. Also, you can start the Model Update Tool for a selected model.

Open the Rational Rose tool window by clicking **Add-Ins > Rational Rose** in Microsoft Visual Basic, or by clicking the Rational Rose icon, , in the toolbar in Microsoft Visual Basic.

The command buttons on the Rational Rose tool window's toolbar and the list view's shortcut menus allow you to operate on the models that are associated with the currently open Visual Basic projects. Table 12 explains each command.

Table 12 Rational Rose Tool Window Commands

Command	Meaning
Connect Visual Model	Associates a new or existing model with the selected project.
Update Model	Updates the selected model with changes in the corresponding project, by starting the Model Update Tool.
Browse Model	Starts Rational Rose (if needed) and opens the selected model.
Disconnect	Disconnects the selected model from the project.
Refresh	Updates the list view. For example, assigns a model file to untitled models.
Help	Displays help on the Rational Rose tool window.

Note: *These commands are enabled if the project is under version control but not checked-out.*

To open the model that is associated with a Visual Basic project:

1. In Microsoft Visual Basic, bring up the Rational Rose tool window by clicking **Add-Ins > Rational Rose**.
2. Right-click the model file in the tool window and click **Browse Model** on the displayed menu. Rational Rose is started (if needed) with the selected model opened.

Chapter 7

Modeling Visual Basic Classes

When modeling Visual Basic classes in Rational Rose, you can always use the ordinary procedures that Rational Rose provides; see the *Using Rose* manual. However, the Visual Basic Language Support add-in provides a separate tool, called the Model Assistant, for specifying classes that are to be implemented in Visual Basic.

This chapter explains how to create Visual Basic classes in a model and how to specify them using the Model Assistant.

The Model Assistant

The Model Assistant lets you create members on a Visual Basic class in the model, as well as customize the code to generate from the class and its members. With the Model Assistant you can:

- Create constants, Declare statements, Event statements, Enum and Type declarations, properties (attributes), and methods (operations).
- Create Get, Let, and Set procedures for class properties and association roles.
- Define and create a user-defined collection class for the class.
- Preview the code to be generated for the class and each of its members.
- Specify implementation details about the class and its members.

The Model Assistant shows the information about a class as it will be implemented in the Visual Basic code. That is, it maps each UML element found in the Class Specification—generalization relationships, properties, roles, methods, and nested classes—into one of the member folders specified in Table 13.

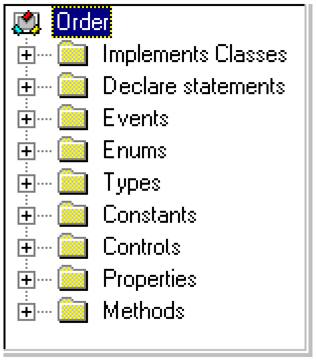


Figure 47 The Model Assistant Shows a Visual Basic View of a Class

The Model Assistant displays only those member folders that currently contain members. The displayed folders contain the following model elements that are related to the current class:

Table 13 Model Assistant—Folder Contents

Folder	**Contains the following model elements**
Implements Classes	Classes with which the current class has generalization or realize relationships; see *Specifying Implements Constructs* later in this chapter.
Declare statements	Methods with stereotype Declare; see *Creating Declare Statements.*
Events	Methods with stereotype Event; see *Creating Event Statements* and *Subscribing to Events.*
Enums	Nested classes with stereotype Enum; see *Creating Enums and Types.*
Types	Nested classes with stereotype Type; see *Creating Enums and Types.*

Table 13 Model Assistant—Folder Contents

Folder	Contains the following model elements
WebItems	Nested classes with stereotype Custom WebItem or Template WebItem; see *Modeling Web Classes*. This folder is only displayed for classes with the stereotype "WebClass".
Constants	Properties with stereotype Const; see *Creating Constants.*
Controls	Associations with stereotype Contained Control and aggregation relationships. Also, any Events of the supplier control are listed; see *Subscribing to Events.* This folder is only relevant for user interface classes, such as forms.
Properties	Properties (attributes) and association roles, as well as any associated property procedures (that is, methods with the same name as a property or role and having the stereotype Get, Set, or Let.) For roles that are defined with the WithEvents option, the Events of the supplier class are listed. See *Creating Properties*.
Methods	Methods (operations) with no stereotype or a stereotype other than any of the above; see *Creating Methods.*

You can open the Model Assistant for a class in several ways:

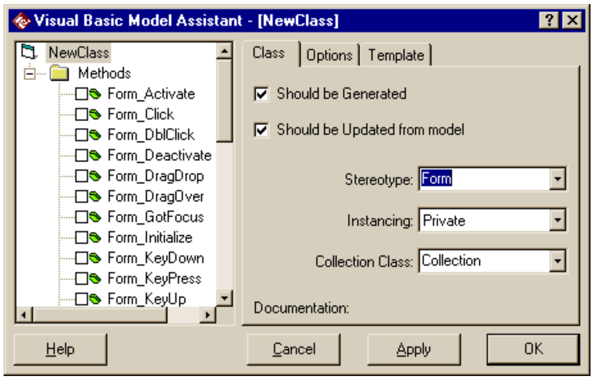
- Select the class in a diagram and click **Tools > Visual Basic > Model Assistant**.
- In a diagram or in the browser, right-click on the class and click **Model Assistant** on the displayed menu.
- On the **Select Components and Classes** page in the Code Update Tool, right-click on a class, or one of its members, and click **Open** on the displayed menu.

Templates

A template defines a number of members that typically appear in a certain type of class—that is, a template describes a certain class stereotype. You can apply a template to a Visual Basic class by selecting the corresponding stereotype in the Class Specification or in the Model Assistant.

A new Visual Basic class gets the stereotype Class Module by default. Thus, when you create a Visual Basic class, it gets a set of predefined members that often appear in class modules. By clicking the check box next to a default member in the Model Assistant you can select those members that this particular class should have.

If you give a class the stereotype Form, as in Figure 48, the Model Assistant automatically adds the standard form event procedures (Click, DragDrop, etc.) to the Methods folder. Those standard procedures are defined in the Form template. For a Form template, none of the template members are selected by default, but for other templates some or all template members may be initially selected.


Visual Basic Model Assistant - [NewClass]

NewClass
- Methods
  - ☐ Form_Activate
  - ☐ Form_Click
  - ☐ Form_DblClick
  - ☐ Form_Deactivate
  - ☐ Form_DragDrop
  - ☐ Form_DragOver
  - ☐ Form_GotFocus
  - ☐ Form_Initialize
  - ☐ Form_KeyDown
  - ☐ Form_KeyPress
  - ☐ Form_KeyUp

Class | Options | Template

☑ Should be Generated

☑ Should be Updated from model

Stereotype: Form

Instancing: Private

Collection Class: Collection

Documentation:

Figure 48 The Form Template

What templates are provided in Rational Rose?

Rational Rose provides a template for each fundamental Visual Basic project item type. There are also templates for project items with specific purposes. For example, there are templates for collection classes and MTS classes.

These additional templates are not fundamental Visual Basic project item types themselves. Instead, they are mapped to one of the fundamental project types. For example, a class that has been created from the Collection template becomes a class module in Visual Basic, and the two Debug templates generate standard modules in the code.

For information about the fundamental project item type templates, see Table 3 in chapter 2. For information about other templates, refer to the description of each template, which is displayed on the **Template** tab in the Model Assistant.

You can create and modify your own templates; see the *Rational Rose Extensibility* / *Rose Visual Studio Extensibility* book on the **Contents** tab of the Rational Rose online help.

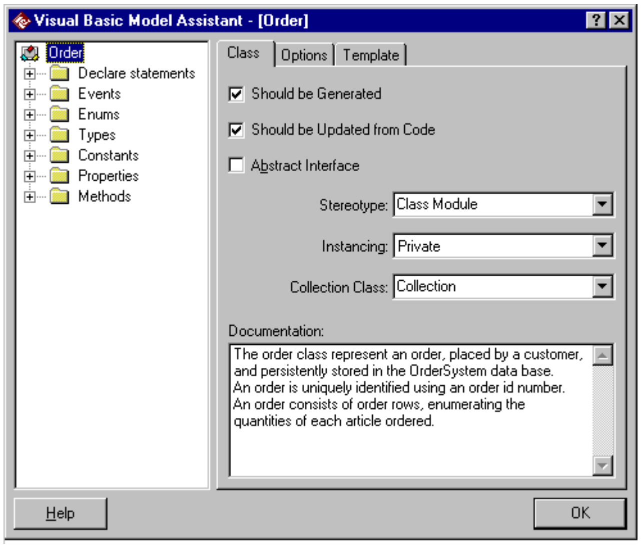
Creating Visual Basic Classes in the Model

You create Visual Basic classes (that is, classes that are to be implemented in Visual Basic code) in the same way as any other class in Rational Rose. (You can also use the Class Wizard, which you open from the **Tools** menu's **Visual Basic** menu.) Then use the Model Assistant to create members on the class and to specify implementation details about the class and its members. In order to use the Model Assistant on a class and to generate Visual Basic code from a class, however, the class must be assigned to the Visual Basic language.

To create a Visual Basic class:

1. Create a class in a diagram or in the browser.
2. If Visual Basic is not the default language in Rational Rose, which you specify on the **Notation** tab of the **Tools > Options** dialog box, the class must be assigned to a component that has Visual Basic as language. See *Assigning Classes to Existing Components* in chapter 6.
3. Open the Model Assistant for the class.
4. Map the class to a Visual Basic project item type by applying a template to the class. See the section *Applying a Template to a Class in the Model* in this chapter.

5. Specify any implementation details on the **Class** tab.


Visual Basic Model Assistant - [Order]

Order
- Declare statements
- Events
- Enums
- Types
- Constants
- Properties
- Methods

Class | Options | Template

☑ Should be Generated
☑ Should be Updated from Code
☐ Abstract Interface

Stereotype: Class Module
Instancing: Private
Collection Class: Collection

Documentation:
The order class represent an order, placed by a customer, and persistently stored in the OrderSystem data base.
An order is uniquely identified using an order id number.
An order consists of order rows, enumerating the quantities of each article ordered.

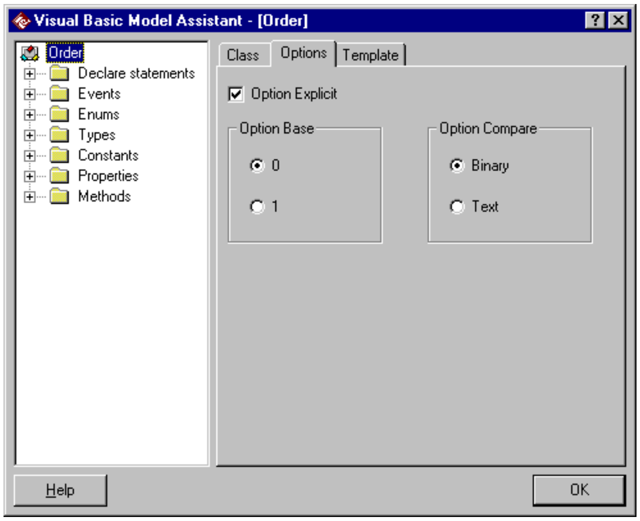
Figure 49 Model Assistant—Class Tab

On the **Class** tab, you can specify the following implementation details about a class:

- **Should be Generated**—Specifies if it is possible to generate code for this class. If this option is cleared, the class cannot be selected for code generation.
- **Should be Updated from Code**—Specifies if it is possible to update this class from code changes. If this option is cleared, the component cannot be selected for model update.
- **Abstract Interface**—Specifies whether this class module is abstract. If a class is abstract, the Visual Basic code generator generates empty method bodies. This option corresponds to the **Abstract** option on the **Detail** tab of the Class Specification.
- **Stereotype**—Defines the type of the class. For example, if the stereotype of a class is set to Class Module, the Visual Basic code generator produces a class module for it. For more information about stereotypes see the section *Classes* in chapter 2. The type of a class also determines what set of pre-defined standard members the class may have.

- **Instancing**—Specifies if the class module is visible outside the project, and if so, specifies rules for how it may be instantiated from other projects. This option is only relevant for classes that correspond to class modules in the code. For more information, see the section *Instancing (Class Property)* in *Appendix A*.
- **Collection Class**—Defines the name of the collection class to be used for data members generated for roles with an unbounded multiplicity (0..n, 0..*, 1..n, 1..*, n, *.) By default, the class "Collection" is used. If you enter a name of a new collection class, the Model Assistant creates a collection class with that name in the model. For more information, see the section *Creating a User-Defined Collection Class for a Class* in this chapter.
- **Documentation**—A textual description of the class. The text can also be found in the **Documentation** box of the Class Specification. When generating code for the class, the text is inserted as a code comment to the corresponding project item.

6. Specify any class option details on the **Options** tab:



[✓] Option Explicit

Option Base: [✓ 0] 1

Option Compare: [✓ Binary] Text

Figure 50 Model Assistant—Class Options Tab

On the **Options** tab, you can specify the following implementation details about a class:

- **Option Explicit**—This statement is used to force explicit declaration of all variables in that module. If you select this option, Rational Rose inserts an Option Explicit statement in the Declarations section of the class.
- **Option Base**—This statement declares the default lower boundary for array subscripts and is used to override the default base array subscript value of 0. As 0 is the default value in Visual Basic, the code generator inserts an Option Base statement in the Declarations section, before any array dimension declaration, only if the value of this option is set to **1**.
- **Option Compare**—This statement is used to declare the default comparison method to use when string data is compared. As **Binary** is the default value in Visual Basic, Rational Rose produces an Option Compare statement for the class only if **Text** is selected.

7. Create and specify the following for the class:
   - Constants
   - Controls
   - Declare statements
   - Event statements
   - Enum and Type declarations
   - Properties
   - Property procedures
   - Methods
   - Implements constructs
   - Web items (if it is a web class)

Applying a Template to a Class in the Model

A template defines a number of members that typically appear in a certain type of class—that is, a template describes a certain class stereotype. The default Visual Basic class stereotype is Class Module which means that Rational Rose by default produces a class module for a new class.

If you want a class to be generated as something other than class module in Visual Basic, you must apply a template to the class before you generate code for it.

Applying Another Template to a Class BEFORE Generating Code

1. If Visual Basic is not the default language in Rational Rose, which you specify on the **Notation** tab of the **Options** dialog box, the class must first be assigned to a component that has Visual Basic as language.
2. Open the Model Assistant for the class.

   Note: *You can also change the stereotype of a class in the Class Specification but the corresponding template is not applied until you open the class in the Model Assistant or generate code for the class.*

3. Select the appropriate template in the **Stereotype** box on the **Class** tab. The Model Assistant updates the class as follows:
   - ❑ Removes all unselected template members.
   - ❑ Adds all members that are defined by the new template.
   - ❑ Adjusts any members that appear both in the class and in the new template. Those members get the required parameters and types from the new template.
   - ❑ Leaves unchanged all members that appear only in the class.

   Also, Rational Rose imports any type libraries that the new template is dependent upon.
4. Depending on the stereotype (template) a pre-defined set of standard members defined by the template may be applied to the class. Select the check box next to the template members that you want the class to have and clear those members that the class should not have.

   For example, for a class module you can specify whether to include a Class_Initialize and a Class_Terminate method by selecting the check box next to those methods.

Note: *You can later remove a template member from the class, for example by clearing the check box next to the member in the Model Assistant. However, the template members must conform to the template where they are defined, so you cannot change the name or the specification of a template member for a class.*

Applying Another Template to a Class AFTER Code Has Been Generated

To apply another template of the same implementation type:

Once you have generated a class you cannot alter its implementation type in Rational Rose. For example, you cannot apply the Module template to a Class Module. However, you can apply another class module template, for example ADO Class, to a class with the Class Module template currently assigned. To do that, see Applying Another Template to a Class BEFORE Generating Code.

To alter the implementation type of a class:

To change a class module into a module, follow these steps:

1. Create a module in the Visual Basic project.
2. Copy the methods and properties from the class module into the new module.
3. Delete the class module in the Visual Basic project.
4. Reverse engineer the changes into the model.

To add members from another template:

You can also add members from another template to a class by temporarily applying another template. To add collection class methods to an "ADO Class", follow these steps:

1. Open the Model Assistant for the class and select **Collection** in the **Stereotype** box.
2. Select the template members that you want the class to have.
3. Switch back to the original stereotype.

Inserting Debug Code for All New Classes

Visual Basic specifies two predefined methods for creating, terminating, and debugging class instances:

- Class_Initialize, which is called when an object is constructed.
- Class_Terminate, which is called before the object is destructed.

You can instruct Rational Rose Visual Basic to include debug code in the default bodies of new Class_Initialize and Class_Terminate methods. The debug code is only applicable to class modules. These two methods, including standard debug code, are added to all new classes that correspond to class modules in the code if the **Generate debug code** option in the **Visual Basic Properties** dialog box is selected.

To insert debug code into all new Class_Initialize and Class_Terminate methods:

1. Open the **Visual Basic Properties** dialog box.
2. Select the **Generate debug code** option and click **OK**.

From now on, the default body of each new Class_Initialize and Class_Terminate method will contain debug code. When you later generate code for the class, Rational Rose generates the following skeleton code for the Class_Initialize and Class_Terminate methods into the class module:

```
Private mlClassDebugID As Long

Private Sub Class_Initialize()
   mlClassDebugID = GetNextClassDebugID()
   Debug.Print "'" & TypeName(Me) & "' instance "
   & mlClassDebugID & " created"
End Sub

Private Sub Class_Terminate()
   Debug.Print "'" & TypeName(Me) & "' instance "
   & CStr(mlClassDebugID) & " is terminating"
End Sub

Public Property Get ClassDebugID()
   ClassDebugID = mlClassDebugID
End Property
```

Rational Rose also creates a logical package called Debug, and adds it to the model. This package contains a class utility, modClassIdGenerator, which is generated as a module in the code. All classes containing debug code will have a dependency relationship with that class utility.

If you have several Visual Basic components in the same model, the same modClassIdGenerator module will automatically be assigned to all those components, and the corresponding Visual Basic projects will include the same .bas file.

To remove the ClassDebugID property for a class:

1. Open the Model Assistant for the class.
2. In the Properties folder, clear the ClassDebugID property.
3. Select the class, and click the **Template** tab.
4. Select **False** in the value field for the **DebugCode** parameter.

To remove generated debug code in the Class_Terminate and Class_Initialize methods for a class:

Remove the code manually in Visual Basic.

Modeling Visual Basic Forms

You can create forms in the model and generate code for them but you cannot use Rational Rose to design the appearance of the form. Rational Rose can generate the form itself, standard members, user-defined members, and event procedures, but not the controls.

To model, generate, and design a Visual Basic form:

1. Create a Visual Basic class in the model and give it the stereotype Form.
2. Open the Model Assistant for the form. Click the "+" sign next to the form's name in the left list, and select the standard members that the form should have.
3. Generate the form into a Visual Basic project by updating the Visual Basic project with model changes or generating a new Visual Basic project from the model. See chapter 4.
4. Design the form in Visual Basic by adding controls to it.
5. Make sure that the **Import all references** option in the **Visual Basic Component Properties** dialog box is selected. Then update the model with the code changes. The Model Update Tool imports the type libraries containing the used controls into the model and adds relationships between the form and its controls.
6. Open the Model Assistant for the form in Rational Rose.

7. Open the Controls folder. For each control, subscribe to the appropriate events. See the section *Subscribing to Events* in this chapter.
8. Update the Visual Basic project with the latest changes. See the section *Updating an Existing Visual Basic Project with Model Changes* in chapter 4.

Modeling Visual Basic Web Classes

You can create Visual Basic web classes and web items in the model and generate code for them.

To create a Visual Basic web class:

1. Create a Visual Basic class in the model.
2. Open the Model Assistant for the class and select **WebClass** in the **Stereotype** box.
3. Select the standard members of the web class.
4. Create web items on the class (see *To add a web item to a web class*).
5. Generate the web class into a Visual Basic project by updating the Visual Basic project with model changes or generating a new Visual Basic project from the model.

To add a web item to a web class:

1. Open the web class in the Model Assistant.
2. Right-click the web class or the WebItems folder in the left list, and click **New Custom WebItem** or **New Template WebItem**. The Model Assistant inserts a new web item under the WebItems statements folder.
3. Edit the default name in the list.
4. Add events to the web item (see *To add an event to a web item*).

After exiting the Model Assistant, you can view the new web items in the model by expanding the web class in the browser. Or, open the **Class Specification** for the web class and click the **Nested** tab.

To add an event to a web item:

1. Open the web class in the Model Assistant.

2. Expand the WebItems folder in the left list.
3. Right-click the web item and click **New Event**.
4. Edit the event's default name in the list and specify any implementation details on the displayed tabs.

Creating User-Defined Collection Classes

By default, Rational Rose Visual Basic assigns the standard collection class, called Collection, to new classes in the model. For collections with more complex behavior, you may want to use a user-defined collection class. This section explains how to create a user-defined collection class in the model and how to transform an existing class into a collection class.

For more information about collection classes, please refer to the section *Collection Classes* in chapter 2.

Creating a New User-Defined Collection Class as a Collection of an Existing Class

To create a collection class as a collection of another class:

1. Right-click the class of which you want to create a collection—for example Orderrow—and click **Model Assistant**.
2. On the **Class** tab, give the new collection class a name in the **Collection Class** box. For example, `Orderrows`.
3. Click **OK**.

The following happens:

- A collection class with the specified name and the stereotype Collection is created in the model. The new class is initiated it with the standard collection methods—add, item, remove, count, and enum. The return type of the Property Get Item() procedure is determined by the selected class—that is, Orderrow in the above example.

- A dependency relationship with the stereotype Collection is created between the new collection class and the selected class (Figure 51).

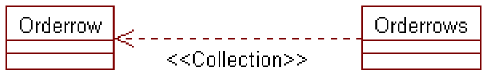


Figure 51 A Dependency Is Created when Creating a Collection Class

Note: *If you want the dependency relationship to appear on a diagram you must add it to the diagram by clicking* ***Query > Filter Relationships*** *and selecting the* ***Dependency*** *option.*

- All associations with unbounded multiplicity—n, *, 0..n, 0..*, 1..n, or 1..*—between the selected class and other classes will be moved to the new collection class. Note, however, that the associations are not moved until you open the Model Assistant or generated code from the associated classes.

  For example, in the following picture, the association between Order and Orderrow is moved to Orderrows when you open the Model Assistant or generate code from Order.

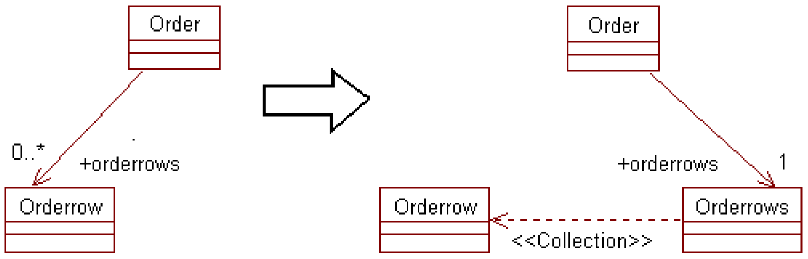


Figure 52 Associations with Unbounded Multiplicity Are Moved to the New Collection Class

The next time you generate code from the model, select the new collection class to create it in the corresponding Visual Basic project as well.

Transforming an Existing Class into a Collection Class

To transform an existing class into a collection class:

1. Right-click the new class and click **Model Assistant**. (You may need to assign the class to a Visual Basic component first.)
2. In the **Stereotype** box on the **Class** tab, select **Collection**.
3. By default, the collection class's standard methods—for example, Property Get Item()—will refer to an item of the type Variant. In order to have the collection class' default methods to refer to a specific class instead, enter the name of that class as the value of the **CollectionOf** parameter on the **Template** tab.
4. Under the Properties and Methods folders, select and clear the standard properties and methods that you want the collection class to have.
5. Click **OK**.

Deleting a User-Defined Collection Class

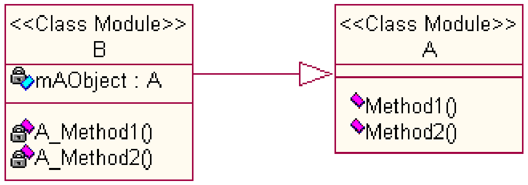
To delete a user-defined collection class from the model:

1. Select the collection class.
2. Click **Edit > Delete from Model**.

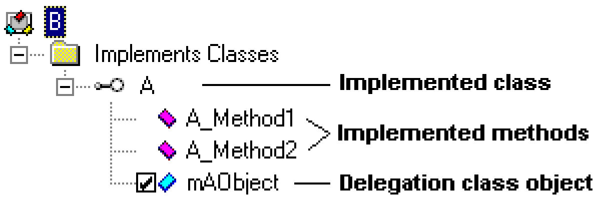
The next time you generate code from the model, the Code Update Tool removes the collection class from the corresponding Visual Basic project as well.

Specifying Implements Constructs

Under the Implements Classes folder, you can find all classes with which the current class has generalize or realize relationships. For the following generalization relationship:



the Model Assistant displays:



As you can see, Rational Rose Visual Basic generates an implements construct for each one of the implemented classes into the class module of the current class.

You cannot add or remove implemented classes in the Model Assistant. However, you can exclude the delegation class object from the class and the dispatching code from the implemented methods for generalization relationships. This is preferable if the delegation class is defined in a type library without code, or if the class is supposed to provide its own variants of the implemented methods. You can also specify if the full name of the implemented class should be used in the generated code.

Note: *If you change the name of the implemented class, for example if you change A to X in the above example, the Model Assistant or code generator creates new delegation methods with the new name in the client class (X_Method1 and X_Method2). Note, however, that the existing implemented methods are not automatically deleted.*

To exclude the delegation class object from a class:

1. Open the Model Assistant for the class.
2. Open the Implements Classes folder and the implemented class—for example, A above.
3. Clear the check box next to the delegation object—for example, mAObject.
4. Click **Yes** in the displayed dialog box.
5. Click **OK**. Rational Rose will not generate a delegation object declaration and the dispatching method implementations into the class module of the current class.

Note: *A generalization relationship generates a delegation object and delegation code in the implemented methods by default. Thus, if you do not want the delegation code, it is recommended that you use the realize relationship instead, which does not generate any delegation code.*

To use the full name, including the component name, of an implemented class:

1. Open the Model Assistant for the class that implements the other class.
2. Select the implemented class under the Implements Classes folder.
3. Select **Full Name** on the displayed **Implements Class** tab.
4. Click **OK**. Rational Rose Visual Basic will use the full name of the implemented class in the generated code—for example, Implements MyComponent.MyClass.

For more information about the generalize and realize relationships, see the *Generalization Relationships* and *Realize Relationships* sections in chapter 2.

Creating Declare Statements

A method in a class or class utility can be declared as a reference to a DLL library procedure; that is, it can be declared as a Declare Sub or Declare Function statement. For more information, see the section *Declare Methods* in chapter 2.

To create a Declare statement on a class:

1. Open the Model Assistant for the class.
2. Right-click the class or the Declare statements folder in the left list, and click **New Declare**.
3. The Model Assistant inserts a Declare statement under the Declare statements folder. Edit the default name in the list.

4. Specify any implementation details on the displayed Declare tab:

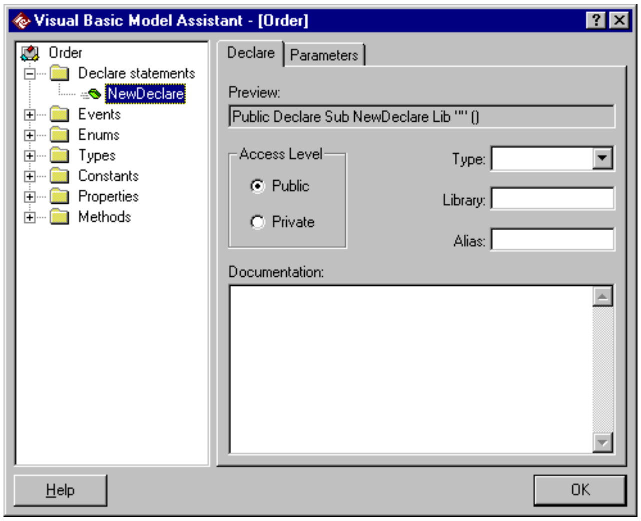


Figure 53 Model Assistant—Declare Tab

On the Declare tab, you can specify the following implementation details about a Declare statement:

- **Access Level**—Defines whether to generate a Public or Private Declare statement.
- **Type**—Specifies the return type of a Declare Function statement. If the **Type** box is empty, a Declare Sub declaration is generated.
- **Library**—Specifies the name of the DLL that contains the declared method. The value must be a string literal.
- **Alias**—Specifies the real name of the procedure, if you are using another name than the DLL. The value must be a string literal.
- **Documentation**—A textual description of the Declare statement. The text can also be found in the **Documentation** box of the Method Specification. When generating code for the class, the text is inserted as a code comment to the Declare statement.

5. Create any parameters on the **Parameters** tab. See the section *Creating Method Parameters* in this chapter.

Creating Event Statements

A method with the stereotype Event corresponds to an Event statement in Visual Basic.

To create an Event statement on a class or class utility:

1. Open the Model Assistant for the class.
2. Right-click the class or the Events folder in the left list, and click **New Event**.
3. The Model Assistant inserts an Event statement under the Events folder. Edit the default name in the list.
4. Optionally, enter a textual description of the Event in the **Documentation** box on the **Event** tab.
5. Create any parameters on the **Parameters** tab for the Event. See the section *Creating Method Parameters* in this chapter. Note, however, that the **Optional**, **ParamArray**, and **Initial Value** options are unavailable for parameters on Events.

To create an Event statement on a web item:

See *Modeling Visual Basic Web Classes* in this chapter.

Subscribing to Events

For roles that are declared with the **WithEvents** option and for controls (that is, associations with the stereotype Contained Control or aggregation relationships) the Model Assistant lists all events that are defined in the associated class or control.

For example, the dlg_Order form in Figure 54 has an aggregation relationship with a CommandButton control in the model. Among the events that are defined in CommandButton, dlg_Order subscribes to the KeyPress event.

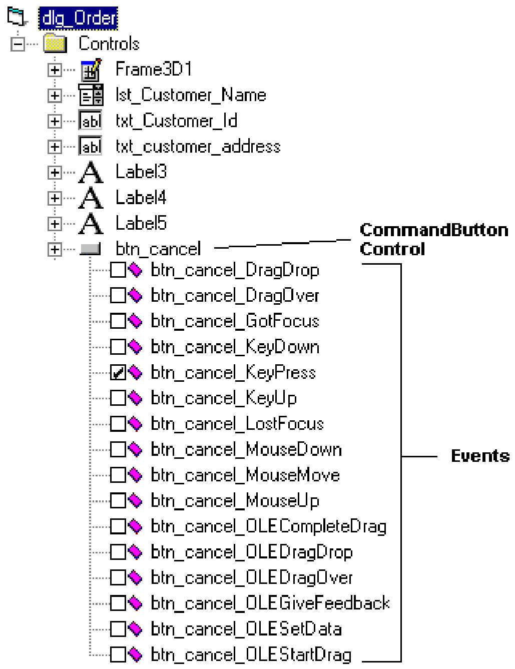


Figure 54 The dlg_Order Form Subscribes to the KeyPress Event Defined in the CommandButton Control

To specify that a user interface item, such as a form called A, should subscribe to certain events in a control, B:

1. Make sure there is an association relationships with the stereotype Contained Control or an aggregation relationship between class A and B.
2. Open the Model Assistant for class A.
3. Under the Controls folder, open control B. All events that are defined in B are displayed.
4. Select the check box next to the events that class A should subscribe to.

To specify that class module, called A, should subscribe to certain events in another class, B:

1. Make sure there is an association or aggregation relationship between class A and B. The role on class B's end of the association must also be navigable.
2. Open the Model Assistant for class A.
3. Under the Properties folder, select the data member that corresponds to role B.
4. On the **Data Member** tab, select **WithEvents**. The Model Assistant then displays all events that are defined in class B in the left list.
5. Select the check box next to the events that class A should subscribe to.

When generating code for the class, Rational Rose Visual Basic inserts a Sub or Function procedure for each event to which the class subscribes.

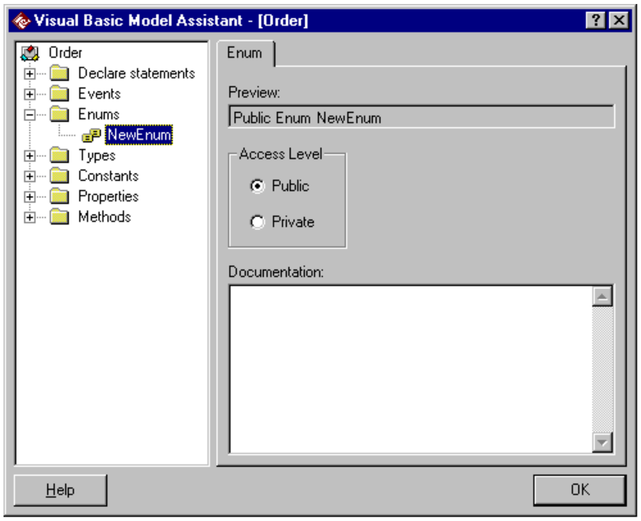
Creating Enums and Types

An Enum or Type declaration in a Visual Basic class corresponds to a class with the stereotype Enum or Type nested within the containing class in the model. An Enum or Type declaration can have data members which correspond to properties on the Enum or Type class in the model.

To create an Enum or Type declaration on a class:

1. Open the Model Assistant for the class.
2. Right-click the class or the Enums or Types folder in the left list, and click **New Enum** or **New Type**.
3. The Model Assistant inserts an Enum or Type declaration under the Enums or Types folder. Edit the default name in the list.

4. Specify any implementation details on the displayed **Enum** or **Type** tab:



Preview: Public Enum NewEnum

Access Level: (● Public) (○ Private)

Documentation:

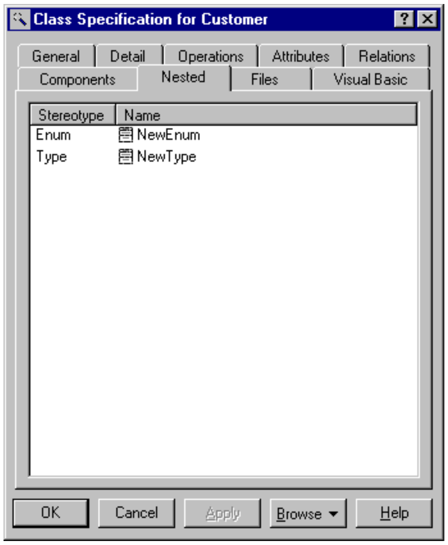
Figure 55   Model Assistant—Enum and Type Tab

On the **Enum** and **Type** tabs you can specify the following implementation details about an Enum or Type declaration:

- **Access Level**—Defines whether to generate a Public or Private Enum or Type declaration.
- **Documentation**—A textual description of the Enum or Type declaration. The text can also be found in the **Documentation** box of the Class Specification for the corresponding nested class. When generating code for the enclosing class, the text is inserted as a code comment to the Enum or Type declaration.

5. To create data members on the Enum or Type declaration, right-click on the Enum or Type declaration and click **New Member**. The Model Assistant inserts a member under the selected Enum or Type declaration. Edit the default name in the list. You can specify implementation details for a selected member (see the section *Creating Properties* in this chapter.) Note, however, that some data member options are not available for Enum and Type members.

6. After exiting the Model Assistant, you can view the new Enum or Type class in the model by expanding the enclosing class in the browser. Or, open the Class Specification for the enclosing class and click on the **Nested** tab:



Stereotype	Name
Enum	NewEnum
Type	NewType

Figure 56 Enum and Type Classes Are Shown on the Nested Tab of the Enclosing Class's Specification

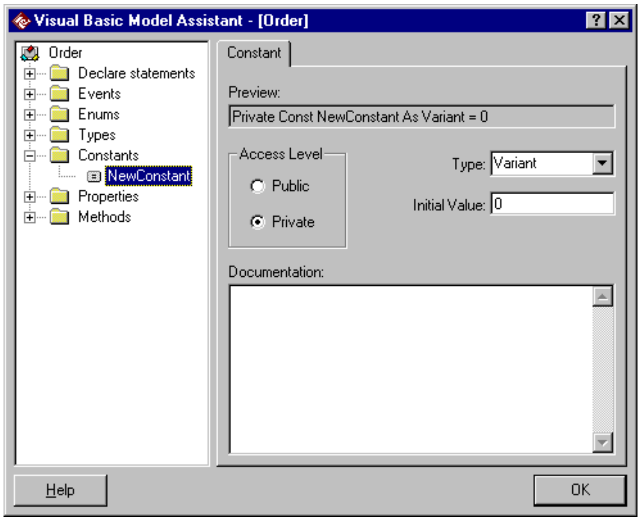
Creating Constants

A constant corresponds to a property in the model, which has an initial value assigned.

To create a constant on a class:

1. Open the Model Assistant for the class.
2. Right-click the class or the Constants folder in the left list, and click **New Constant**.
3. The Model Assistant inserts a new constant under the Constants folder. Edit the default name in the list.

4. Specify any implementation details on the displayed **Constant** tab:



Preview: Private Const NewConstant As Variant = 0

Access Level: Public [✓ Private]

Type: Variant

Initial Value: 0

Documentation:

Figure 57 Model Assistant—Constant Tab

On the **Constant** tab, you can specify the following implementation details about a constant:

- **Access Level**—Defines whether to generate a Public or Private Constant declaration.
- **Type**—Specifies the type of the generated constant.
- **Initial Value**—Specifies the initial value of the generated constant. You must specify the initial value in order to generate a constant.
- **Documentation**—A textual description of the constant. The text can also be found in the **Documentation** box of the Property Specification. When generating code for the class, the text is inserted as a code comment to the constant.

Creating Properties (Attributes)

A property in Visual Basic corresponds to a property or a navigable association role in the model. Except for Rational Rose's standard ways to create properties, you can also use the Model Assistant to view and create them.

The Model Assistant displays each property as a data member together with any associated property procedures. For example, in Figure 58, the property OrderId in the model corresponds to a data member, mOrderId, with which a property Get and a property Let procedure are associated.

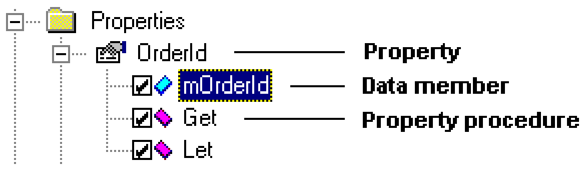
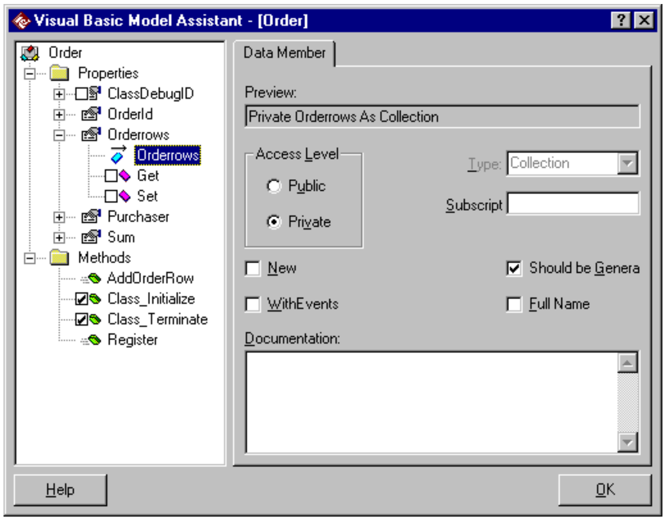


Figure 58 OrderId Corresponds to a Data Member, mOrderId, and Two Property Procedures

To create a property on a class:

1. Open the Model Assistant for the class.
2. Right-click the class or the Properties folder in the left list, and click **New Property**.
3. The Model Assistant inserts a new property under the Properties folder. Edit the default name in the list.
4. Select the corresponding data member and specify any implementation details about it on the displayed **Data Member** tab:



Preview: Private Orderrows As Collection

Access Level: Public [✓ Private]

Type: Collection

Subscript:

New: ☐

WithEvents: ☐

Should be Genera: ☑

Full Name: ☐

Documentation:

Figure 59 Model Assistant—Data Member Tab

On the **Data Member** tab, you can specify the following implementation details about a data member:

- **Access Level**—Defines whether to generate a Public or Private data member declaration.
- **Subscript**—For a property of the type array, the size of the generated array is defined here. To generate correct code you must use Visual Basic syntax for array bound values, for example **`1 To 10`** or **`1 To MaxLen`**. A dynamic array is declared by entering **`()`**.
- **Type**—Specifies the type of the generated data member.
- **Should be Generated**—Specifies whether to generate Visual Basic code for this role or property.
- **Full Name**—Specifies if the full name of the referenced class, including the name of the component to which the class is assigned, should be used in the data member declaration for the role. The full name—for example OrderSys.Customer—is needed to generate correct code for class names that are defined in several components.
- **New**—Specifies if Rational Rose Visual Basic should add the **`New`** keyword to the data member declaration.
- **WithEvents**—Specifies if Rational Rose Visual Basic should add the **`WithEvents`** keyword to the data member declaration. If this option is selected for a role, any events that are defined in the associated class are listed under the data member. To specify that the current class should subscribe to a certain event, select the check box next to that event. See the section *Subscribing to Events*.
- **Documentation**—A textual description of the property. The text can also be found in the Documentation box of the Property Specification. When generating code for the class, the text is inserted as a code comment to the property.

Note: *The* **New** *and* **WithEvents** *options are only available for data members that correspond to roles in the model and the* **Type** *box is only available for data members corresponding to properties in the model. Also, the* **Subscript**, **New**, *or* **WithEvents** *option may not be valid with the current settings. For more information please refer to the Visual Basic documentation.*

5. For private properties, create any necessary Property Get, Set, or Let procedures. See the *Creating Property Get, Let, and Set Procedures* section below.

For more information about properties and associations roles, see the sections *Properties* and *Association Relationships* in chapter 2.

Creating Property Get, Let, and Set Procedures

A property procedure corresponds to a method in the model with the stereotype Get, Set, or Let. Thus, you can create a property procedure for a class by creating a method of one of those stereotypes in the Class Specification. However, by using the Model Assistant you can create property procedures for a specific property or role of a class.

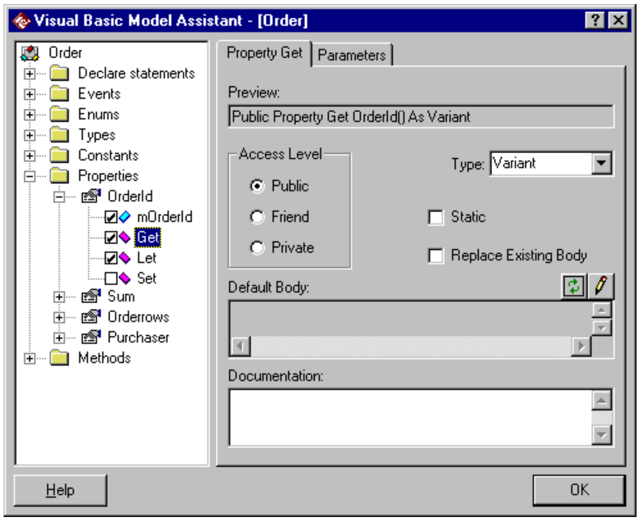
The Model Assistant uses the name of a property procedure determine to which property it belongs to. For example, in Figure 58, the property Get procedure for the mOrderId data member is called OrderId(). If you rename a property, the Model Assistant renames the corresponding data member and property procedures accordingly.

To avoid name clashes between the data member and the property procedure names, the Model Assistant adds a prefix "m" to the data member when creating a property procedure. For example, the Model Assistant renames the data member OrderId in Figure 59 to mOrderId if you associate a property procedure with that property. To use a prefix other than the default prefix "m," please refer to the section *Modifying the Default Data Member Prefix* in this chapter.

Creating a Get, Set, or Let Procedure for a Property

1. Open the Model Assistant for the class that has the property or role for which you want to create property procedures.
2. Under the Properties folder, open the property in question.

3. Select the check box next to the property procedure(s) that you want to associate with the property, such as Get and Set in the picture below. Note that only those property procedures that are relevant for a data member of the current type are available.



Preview: Public Property Get OrderId() As Variant

Access Level: [✓ Public] Friend Private

Type: Variant

☐ Static

☐ Replace Existing Body

Default Body:

Documentation:

Figure 60 Creating a Property Procedure Adds an "m" to the Data Member Name

4. Select the new property procedure, and take a look at the code to be generated for it in the **Default Body** box. As you can see, the generated property procedure simply copies the passed parameter to the data member or returns the value of the data member.
5. Specify any implementation details about the property procedure. See the **Method** tab and **Method Parameters** tab in the *Creating Methods* section in this chapter.

For more information about property procedures, please refer to the section *Property Get/Set/Let Procedures* in chapter 2.

Modifying the Default Data Member Prefix

If you associate a property procedure to a property or role, the Model Assistant adds an "m" in front of the name of the property or role to avoid a name collision with the property procedure.

To instruct the Model Assistant to add another prefix than “m”:

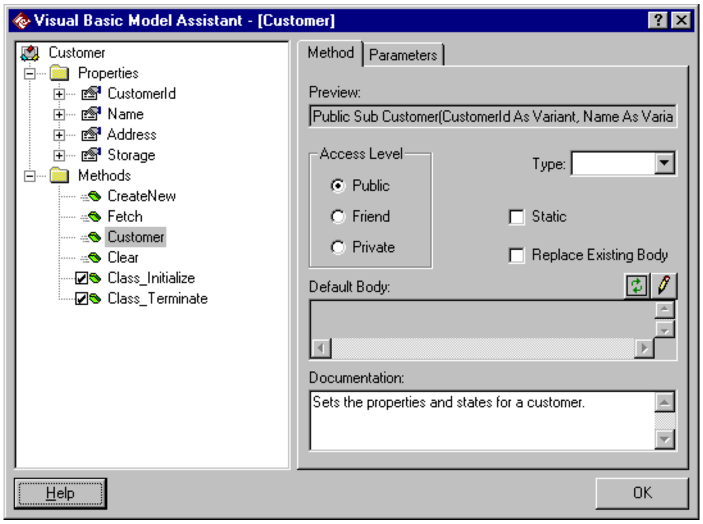
1. Open the **Visual Basic Properties** dialog box.
2. In the **Data member prefix** box enter the prefix that you want to add to data members.
3. Click **OK**. From now on, all properties or roles to which you assign property procedures will get the new prefix.

Creating Methods (Operations)

Creating a Method

To create a Sub or Function method on a class:

1. Open the Model Assistant for the class.
2. Right-click the class or the Methods folder in the left list, and click **New Method**.
3. The Model Assistant inserts a new method under the Methods folder. Edit the default name in the list.
4. Specify any implementation details on the displayed **Method** tab:



Preview: Public Sub Customer(CustomerId As Variant, Name As Varia

Access Level: Public [✓], Friend, Private

Static: ☐

Replace Existing Body: ☐

Documentation: Sets the properties and states for a customer.

Figure 61 Model Assistant—Method Tab

On the **Method** tab, you can specify the following implementation details about a method:

- **Access Level**—Defines whether to generate a Public, Friend, or Private method.
- **Type**—Specifies the return type of a Function declaration. If this box is empty, the method is mapped to a Sub declaration in Visual Basic.
- **Static**—Specifies if Rational Rose Visual Basic produces the method with the Static keyword. This indicates that the function procedure's local variables are preserved between calls. This option does not affect variables that are declared outside the function procedure, even if they are used in the procedure.
- **Replace Existing Body**—Specifies whether to always replace the body of a method with the contents of the Default Body box when updating the code for this class. Note that Rational Rose Visual Basic then overwrites the method's current body.
- **Default Body**—Provides a preview of the code to be inserted into the body of the method, for methods that are generated for the first time. (See also the **Replace Existing Body** option.) To modify the default body, see the section *Modifying the Default Body of a Method* in this chapter.
- **Documentation**—A textual description of the method. The text can also be found in the Documentation box of the Method Specification. When generating code for the class, the text is inserted as a code comment to the method. Also, for public classes, the text is used as a description of each method in the type library.

5. Create the method's parameters on the **Parameters** tab; see the section *Creating Method Parameters* below.

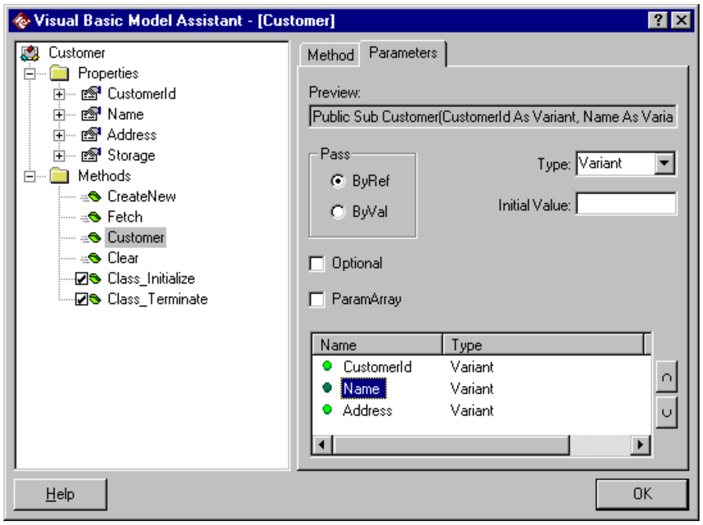
For more information about user-defined methods, please refer to *Methods* in chapter 2.

Creating Method Parameters

Method parameters are created with a name and a type. The type definition can be omitted, implying the use of a Variant data type. You can also define an initial value of each parameter. Those values are directly mapped to default parameter values in the Visual Basic code. Also, you can specify the parameter passing mechanism for each parameter.

To create parameters to a method in the Model Assistant:

1. Select the method in the left list and click on the **Parameters** tab.
2. For each parameter, right-click in the parameters list at the bottom of the tab and click **Add Parameter** on the displayed menu.
3. Specify any implementation details about each parameter, by selecting the parameter and using the available options on the **Parameters** tab:



Preview: Public Sub Customer(CustomerId As Variant, Name As Varia

Pass: [✓ ByRef] ByVal

Type: Variant

Initial Value:

Optional

ParamArray

Name	Type
CustomerId	Variant
Name	Variant
Address	Variant

Figure 62 Model Assistant—Method Parameters Tab

On the **Parameters** tab, you can specify the following implementation details about each parameter:

- **Pass**—Specifies the parameter passing mechanism for the selected parameter. You do not have to specify the passing mechanism in Rational Rose. You can do it later, in Visual Basic, after you have generated the code. For more information about parameter passing, see the section *Method Parameter Passing* in chapter 2.
- **Type**—Specifies the type of the selected parameter.
- **Initial Value**—Specifies that the selected parameter is Optional and has the given initial value.
- **Optional**—Specifies if the selected parameter is optional. (This option is unavailable if the **ParamArray** option is selected.)

- **ParamArray** — Specifies if the selected parameter represents an indefinite number of parameters. This option cannot be selected if the **ByVal**, **ByRef**, or **Optional** option is selected. Also, this option can only be used when the parameter is a Variant type and only for the last parameter.

4. To rearrange the order between the parameters, use the arrow buttons.

Adding Error-Handling Code

Module methods often define exception handling in a macro-like way. The exception handler is declared in the first line of the method and the exception handler, including calls to an error or event handler, are added at the end of the method.

You can instruct Rational Rose to automatically insert error-handling code into all new methods. Then each new method will get a default body with the following error-handling code when you generate code:

```
   On Error GoTo ErrorHandler
...
ErrorHandler:
   Call RaiseError(MyUnhandledError,"method_name
       method_type")
```

Also, Rational Rose creates a logical package in the model, called Debug, containing a class utility, modErrorHandling, which is generated as a module in the code. All classes containing debug code will have a dependency relationship with that class utility.

If you have several Visual Basic components in the same model, the same modErrorHandling module will automatically be assigned to all those components, and the corresponding Visual Basic projects will include the same .bas file.

To add error-handling code to all new methods:

1. Open the **Visual Basic Properties** dialog box.
2. Select the **Generate error handling code** option and click **OK**.

To add error-handling code to new methods on a specific class:

1. Open the Model Assistant for the class and click the **Template** tab.
2. Select **True** in the value field for the **ErrorHandling** parameter.

This overrides the value of the **Generate error handling code** option in the **Visual Basic Properties** dialog box.

To remove generated error-handling code for a class:

Remove the code manually in Visual Basic.

Adding "Your Code Goes Here..." Comments

You can instruct the code generator to generate comments indicating where you need to add code in a method. The following code comment is then inserted into the default body of the method:

```
Public Sub NewMethod()
    'your code goes here...
End Sub
```

The comment is inserted only on methods that are generated for the first time.

To add "Your code goes here..." comments to new methods on all classes:

1. Open the **Visual Basic Properties** dialog box.
2. Select the **Generate "Your code goes here..." comments** option and click **OK**.

To add "Your code goes here..." comments to new methods on a specific class:

1. Open the Model Assistant for the class and click the **Template** tab.
2. Select **True** in the value field for the **Comments** parameter.

This overrides the value of the **Generate "Your code goes here..." comments** option in the **Visual Basic Properties** dialog box.

Using Friend Methods in Sequence Diagrams

Friend methods in Visual Basic correspond to protected methods in Rational Rose. To be able to select a protected method when assigning a method to a message in a sequence diagram, the calling class must be defined as a "friend" to the class with the protected method.

To define a class as a friend using a dependency relationship:

1. Create a dependency relationship between the two classes, pointing from the calling class to the other class.
2. Open the Dependency Relationship Specification.
3. On the **General** tab, select the **Friendship required** option.

Note: *If there is an association relationship between the two classes already, open the Association Relationship Specification and select the* **Friend** *option on the* **Detail** *tab of the called class's role. If you do not want to create any relationship, you can simply type the name of the protected method on the message in the sequence diagram.*

Modifying the Default Body of Methods

When you create a new method (operation) on a Visual Basic class in the model, the Model Assistant automatically attaches a method body with default code to that method. You can preview and modify the suggested method body in the **Default Body** box on the **Method** tab in the Model Assistant.

The Default Body

The default body contents are defined by the template that is currently attached to the class. The contents are different for different kinds of methods. For example, the default body for a Public Sub method, mySub, of a Class Module contains the following code comments and source code:

```
' <Documentation box>
Public Sub MySub()

   ' <Preconditions tab>
   ' <Semantics tab>

   On Error GoTo ErrorHandler
   ' ## Your code goes here…
   ' <Postconditions tab>
   Exit Sub

ErrorHandler:
   Call RaiseError(MyUnhandledError, "MySub Sub")
End Sub
```

The exact contents for a specific method are determined by the value of the class's template parameters and the information in the Method Specification. For example:

- The "On Error..." and "ErrorHandler:..." statements are inserted only if the **ErrorHandling** template parameter for the class is **True**; see *Adding Error-Handling Code* in this chapter.
- The "Your code goes here..." comment is inserted only if the **Comments** template parameter is True; see *Adding "Your Code Goes Here..." Comments* in this chapter.
- The code comments and source code in `<...>` are only present if the corresponding Class Specification fields—**Documentation** box, **Semantics** tab, **Postconditions** tab, and **Preconditions** tab—contain any information.

Thus, if the **Preconditions**, **Semantics**, and **Postconditions** tabs are empty in the Method Specification, but **ErrorHandling** and **Comments** are **True**, the default body for MySub above would look like:

```
' The MySub method...
Public Sub MySub()

   On Error GoTo ErrorHandler
   ' ## Your code goes here ...
   Exit Sub

ErrorHandler:
   Call RaiseError(MyUnhandledError, "MySub Sub")
End Sub
```

A template can have additional template parameters, which insert different default body code depending on the parameter values; see the documentation of each template. The template parameters for a class are available on the **Template** tab in the Model Assistant for the class. For information on how to change the value of a template parameter, see *Changing the Value of a Template Parameter* in this chapter.

The default body is inserted only the first time code is generated for a method. However, if the **Replace Existing Body** option on the **Method** tab in the Model Assistant is selected, Rational Rose Visual Basic inserts the default body each time the method is generated.

If you are not satisfied with the method body suggested by default for a method, you can edit the body before generating the Visual Basic code; see *Modifying the Default Body of a Method* in this chapter.

Note: *The code generator does not wrap code comments when inserting them into the Visual Basic project.*

Modifying the Default Body of a Method

If you are not satisfied with the default body for a method (operation), you can modify the code in the model. This section explains how to modify the default body before or after generating Visual Basic code for the method.

Some templates also provide parameters that control the contents of the default bodies. For more information, see the following sections in this chapter:

- *Adding Error-Handling Code*
- *Adding "Your code goes here..." Comments*
- *Changing the Value of a Template Parameter for a Class*

To modify the method body BEFORE generating any code for the method:

1. Open the Model Assistant for the class in which the method is defined.
2. Select the method in the left list.
3. On the **Method** tab, click the button above the **Default Body** box. The default body becomes editable.
4. Modify the default body by removing, adding, or changing the code.
5. When you generate code for the class, your code is inserted into the method's body in Visual Basic.

To modify the method body AFTER code has been generated for the method:

You can edit a generated method body directly in the Visual Basic project, but in order to generate a new default body from Rational Rose, follow these steps:

1. Open the Model Assistant for the class in which the method is defined.
2. Select the method in the left list.
3. On the **Method** tab, click the button above the **Default Body** box. The default body becomes editable.

4. Modify the default body by removing, adding, or changing the code.
5. Select the **Replace Existing Body** option and click **OK**.
6. Generate code for the class. Rational Rose replaces the code body for the method with the new default body.
7. If you leave the **Replace Existing Body** option selected, the code body for the method is replaced each time you generate code for the class. If you do not want this to happen—for example, if you want to edit the method body in Visual Basic—clear **Replace Existing Body** for the method before generating code again.

To revert to the default body defined by the template:

1. Open the Model Assistant for the class in which the method is defined.
2. Select the method in the left list.
3. On the **Method** tab, click the button above the **Default Body** box. The Model Assistant overwrites the default body with the code body defined by the template currently attached to the class.

Changing the Value of a Template Parameter for a Class

When you select the class node in the Model Assistant, a **Template** tab is displayed. This tab shows the class's template parameters, which control:

- The contents of the generated default body for the class's methods.

  For example, the **ErrorHandling** parameter defines whether error-handling code should be added to the method bodies.
- What standard members the class should have.

  For example, the **DebugCode** parameter defines whether to add members that are needed to enable debugging of the generated class.

The template parameters available for a specific class depend on the template that is attached to the class. The parameters in Table 14 are available for most classes. For information about other template parameters, refer to the documentation of the template.

Table 14 Default Body Variables

Variable	Meaning and value
CollectionOf	This template parameter is used only for collection classes. It specifies the name of the item class. The name is used in the collection class's standard methods to refer to an item of the correct type. The default value is the name of the class with which the collection class has a dependency relationship. If there is no dependency relationship, the default value is Variant. The value can be modified in the Model Assistant on the **Template** tab for the class.
Comments	This template parameter, which can be **True** or **False**, specifies whether to add "Your code goes here..." comments to the class' methods. The default value is determined by the **Generate "Your code goes here..." comments** option in the **Visual Basic Properties** dialog box, but can be changed for a specific class on the **Template** tab in the Model Assistant.
DebugCode	This template parameter, which can be **True** or **False**, specifies whether to add debug code to the class's Initialize and Terminate events. The default value is determined by the **Generate debug code** option in the **Visual Basic Properties** dialog box, but can be changed for a specific class on the **Template** tab in the Model Assistant.
ErrorHandling	This template parameter, which can be **True** or **False**, specifies whether to add error-handling code to the class's methods. The default value is determined by the **Generate error handling code** option in the **Visual Basic Properties** dialog box, but can be changed for a specific class on the **Template** tab in the Model Assistant.

To change the value of a template parameter for a class:

1. Open the Model Assistant for the class, and click the **Template** tab.
2. Type the new value for the parameter you want to change. For a conditional parameter, such as **Comments**, you should select either **True** or **False**.

Note that the new value affects the default body for all methods in the class. For example, if you change the **Comments** parameter from **False** to **True**, the default bodies for all methods in the class will include a "Your code goes here..." comment.

To revert to the default value of a parameter:

Right-click on the parameter and click **Revert** on the displayed menu.

To change the value of a template parameter for all classes:

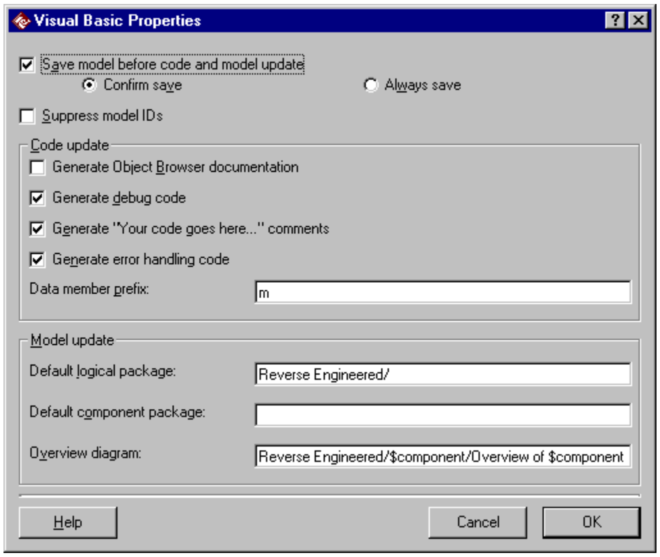
Some template parameters (**Comments**, **DebugCode** and **ErrorHandling**) have a default value, which can be changed in the **Visual Basic Properties** dialog box. The change affects all classes and methods for which you have not yet generated code. For more information, see the following sections in this chapter:

- *Inserting Debug Code*
- *Adding Error-Handling Code*
- *Adding "Your code goes here..." Comments*

Customizing the Default Behavior of the Model Assistant

The Visual Basic Properties Dialog Box

You can customize the code that the Model Assistant inserts by default in the **Visual Basic Properties** dialog box. To open the **Visual Basic Properties** dialog box, click **Tools > Visual Basic > Properties**.



Visual Basic Properties

☑ Save model before code and model update
● Confirm save ○ Always save

☐ Suppress model IDs

Code update
☐ Generate Object Browser documentation
☑ Generate debug code
☑ Generate "Your code goes here..." comments
☑ Generate error handling code

Data member prefix: m

Model update
Default logical package: Reverse Engineered/
Default component package:
Overview diagram: Reverse Engineered/$component/Overview of $component

Figure 63 Visual Basic Properties Dialog Box—Model Assistant Properties

The following options customize the behavior of the Model Assistant:

Generate debug code

Specifies whether to include debug code and debug members for Initialize and Terminate events for new class modules. See the section *Inserting Debug Code for all New Classes* in this chapter.

Generate “Your code goes here...” comments

Specifies whether to generate code comments indicating where you need to add code for new methods. See the section *Adding “Your Code Goes Here...” Comments* in this chapter.

Generate error handling code

Specifies whether to include error-handling code for new methods. See the section *Adding Error-Handling Code* in this chapter.

Data member prefix

Specifies the prefix to add to the property names for properties that have Get, Set, or Let procedures associated with them. The specified prefix is used in the data member names to avoid name collisions with the property procedures. The default value is “m”, which means that if you associate a Property Get procedure to a property called Order, the Model Assistant changes the name of the property to mOrder. See *Modifying the Default Data Member Prefix* later in this chapter.

Appendix A

Model Properties Reference

The Visual Basic Language Support add-in provides model properties for the following kinds of model elements:

- Component Properties
- Class Properties
- Role and Property (Attribute) Properties
- Method (Operation) Properties
- Generalization Properties

For model elements without a code mapping, such as packages, there are no model properties.

The Model Assistant lets you modify the Visual Basic model properties, and ensures that the settings are consistent and that they will generate correct code. The model properties of a model element are also available on the **Visual Basic** tab of its specification. Note, however, when editing the model properties in a specification, no consistency checks are being made.

This appendix explains each model property that is provided by the Visual Basic Language Support add-in in Rational Rose. Use this appendix as a reference when you want to know the meaning of a specific model property, how to use it, and the meaning of each of its values.

Model Properties for Components

Model properties for components control the correspondence between a component and a Visual Basic project. A component has the following model properties:

- ImportBinary
- ImportReferences
- ProjectFile
- QuickImport
- UpdateCode
- UpdateModel

ImportBinary (Component Property)

The ImportBinary model property specifies whether to import the type library of the binary component, for example, a DLL, that is compiled from this component. The type library is imported the next time you update the model from the Visual Basic project that is associated with this component. This option is not relevant for Standard EXE components.

The value of the ImportBinary model property can be modified on the **References** tab of the **Visual Basic Component Properties** dialog box, as well as on the **Visual Basic** tab in the Component Specification.

Table 15 ImportBinary Model Property Values

Value	Result
TRUE	(Default) Rose imports the type library of the binary component that is compiled from this component.
FALSE	Rose does not import the type library of the binary component.

ImportReferences (Component Property)

The ImportReferences model property specifies whether to import the type library of the COM components that are referred to from the Visual Basic project associated to this component. By default, the Model Update Tool imports only the interface classes of the COM components, and no properties and methods, but you can use the QuickImport model property to specify how much of the type library you want to import.

The value of the ImportReferences model property can be modified on the **References** tab of the **Visual Basic Component Properties** dialog box, as well as on the **Visual Basic** tab in the Component Specification.

Table 16 ImportReferences Model Property Values

Value	Result
TRUE	(Default) Rose imports the type library of the COM components that are referenced by the Visual Basic project for this component.
FALSE	Rose ignores any project references.

ProjectFile (Component Property)

The ProjectFile property specifies the project file (.vbp) for the Visual Basic project associated with the component. When generating code for a component, Rational Rose generates the code into the project file specified by the ProjectFile property.

Rational Rose sets this model property automatically when generating or reverse engineering Visual Basic code. You can assign a project to a component in the **Visual Basic Component Properties** dialog box, as well as on the **Visual Basic** tab in the Component Specification.

Table 17 ProjectFile Model Property Values

Value	Result
An absolute file path	Rose generates code into the specified project file, for example: **`c:\my_project\project1.vbp`**.

Table 17 ProjectFile Model Property Values

Value	Result
A virtual file path	Rose transforms the virtual path (for example `$MYPATH\project1.vbp`) into the corresponding absolute path (for example `c:\my_project\project1.vbp`) and generates code into that project.
project.vbp	Rose generates code into the specified project file, for example `project1.vbp`, in the same folder as where the model file is located.
Nothing	Rose generates code into the current open Visual Basic project: If the open project has the same name as the component or If the component has been generated from/into that project before. Otherwise, Rose starts the Visual Basic application and generates code into a new project.

QuickImport (Component Property)

If the ImportReferences model property is True, the QuickImport property specifies whether to import properties and methods of the imported interface classes into the model. If ImportReferences is False, the Model Update Tool ignores the value of the QuickImport model property.

The value of the QuickImport model property can be modified on the **References** tab of the **Visual Basic Component Properties** dialog box, as well as on the **Visual Basic** tab in the Component Specification.

Table 18 QuickImport Model Property Values

Value	Result
TRUE	(Default) If ImportReferences is True, Rose imports only the interface classes, but not properties and methods, of referenced COM components.
FALSE	If ImportReferences is True, Rose imports the interface classes, including their properties and methods, of referenced COM components.

UpdateCode (Component Property)

The UpdateCode property specifies whether code can be generated for this component. If UpdateCode is set to False, the component cannot be selected for code generation.

The value of the UpdateCode model property can be modified in the **Visual Basic Component Properties** dialog box (**Should be generated** option), as well as on the **Visual Basic** tab in the Component Specification.

Table 19 UpdateCode Model Property Values

Value	**Result**
TRUE	(Default) You can select this component for code generation.
FALSE	You cannot select this component for code generation.

UpdateModel (Component Property)

The UpdateModel property specifies whether it is possible to update the classes assigned to this component from code changes. If UpdateModel is set to False, the component cannot be selected for model update.

The value of the UpdateModel model property can be modified in the **Visual Basic Component Properties** dialog box (**Should be updated from Ccde** option), as well as on the **Visual Basic** tab in the Component Specification.

Table 20 UpdateModel Model Property Values

Value	**Result**
TRUE	(Default) You can update this component with code changes.
FALSE	You cannot update this component from code changes.

Model Properties for Classes

Model properties for classes control the Visual Basic code that Rational Rose Visual Basic produces for a class.

The following properties are available for classes:

- Instancing
- OptionBase
- OptionCompare
- OptionExplicit
- UpdateCode
- UpdateModel

Instancing (Class Property)

The Instancing model property is used to determine how classes you define are exposed to other applications. It specifies, for example, if you can create instances of a public class outside a Visual Basic project. The Instancing property is only relevant for classes with the stereotype set to Class Module.

The value of the Instancing model property can be changed on the **Class** tab of the Model Assistant and on the **Visual Basic** tab in the Class Specification.

Table 21 Instancing Model Property Values

Value	**Result**
Private	Rose makes the class visible only within a Visual Basic project.
PublicNotCreatable	Rose makes the class visible to applications outside the project, but instantaible inside the project only.
SingleUse	Refer to your Microsoft Visual Basic documentation.
GlobalSingleUse	Please refer to your Microsoft Visual Basic documentation.
MultiUse	(Default) Refer to your Microsoft Visual Basic documentation.
GlobalMultiUse	Refer to your Microsoft Visual Basic documentation.

Restrictions on class instancing may be applied by the stereotype of the component where a class belongs. For example, a Standard EXE component can have only private classes.

For detailed explanations of the meaning of each value and any restrictions, please refer to the Visual Basic documentation.

OptionBase (Class Property)

The Option Base statement in Visual Basic declares the default lower bound for array subscripts and is used to override the default base array subscript value of 0. The value must be either 0 or 1. The Option Base statement has no effect on arrays within user-defined types. The code generator inserts an Option Base statement in the Declarations section, before any array dimension declaration.

The OptionBase model property lets you define the value of the Option Base statement for each class. The value of the OptionBase model property can be changed on the **Options** tab of the Model Assistant and on the **Visual Basic** tab in the Class Specification.

Table 22 OptionBase Model Property Values

Value	**Result**
(none)	(Default) Rose does not generate an Option Base statement for the class. Thus, the Option Base for this class becomes the default value in Visual Basic, which is 0.
0	Rose generates an Option Base 0 statement for the class.
1	Rose produces an Option Base 1 statement for the class.

OptionCompare (Class Property)

The Option Compare statement in Visual Basic is used to declare the default comparison method to use when string data is compared.

The OptionCompare model property lets you define the value of the Option Compare statement for each class. You can change the OptionCompare model property on the **Options** tab of Model Assistant and on the **Visual Basic** tab in the Class Specification.

Table 23 OptionCompare Model Property Values

Value	**Result**
(none)	(Default) Rose does not produce an Option Compare statement for the class. Thus, the Option Compare becomes the default value in Visual Basic, which is Binary.
Text	Rose produces an Option Compare Text statement for the class.
Binary	Rose produces an Option Compare Binary statement for the class.

OptionExplicit (Class Property)

The Option Explicit statement is used in Visual Basic to force explicit declaration of all variables in that module. If you attempt to use an undeclared variable name, an error occurs at Visual Basic compile time. Rational Rose inserts an Option Explicit statement in the Declarations section.

The OptionExplicit model property lets you define the value of the Option Explicit statement for each class. The value of the OptionExplicit model property can be changed on the **Options** tab of the Model Assistant and on the **Visual Basic** tab in the Class Specification.

Table 24 OptionExplicit Model Property Values

Value	Result
TRUE	Rose produces an Option Explicit statement for the module.
FALSE	Rose does not produce an Option Explicit statement for the module.

UpdateCode (Class Property)

The UpdateCode property specifies if code can be generated for this class. If UpdateCode is set to FALSE, the class cannot be selected for code generation. However, all association relationships referring to such a class result in type or object variable declarations.

The value of the UpdateCode model property can be modified on the **Class** tab of the Model Assistant (**Should be Generated** option), as well as on the **Visual Basic** tab in the Class Specification.

Table 25 UpdateCode Model Property Values

Value	Result
TRUE	(Default) You can generate code for this class.
FALSE	You cannot select this class for code generation.

UpdateModel (Class Property)

The UpdateModel property specifies if it is possible to update this class from changes in the corresponding project item. If UpdateModel is set to False, the class cannot be selected for model update.

The value of the UpdateModel model property can be modified on the **Class** tab of the Model Assistant (**Should be Updated from Code** option), as well as on the **Visual Basic** tab in the Class Specification.

Table 26 UpdateModel Model Property Values

Value	**Result**
TRUE	(Default) You can update this class with code changes.
FALSE	You cannot select this class to be updated from code changes.

Model Properties for Roles and Properties (Attributes)

When generating code for a class, Rational Rose Visual Basic produces a data member for each role and property. Use the following model properties to control the code generated for data members:

- FullName (roles only)
- New
- ProcedureID
- PropertyName
- Subscript
- UpdateCode (roles only)
- WithEvents

If the code generator discovers conflicts between these model properties, the following priority rules are used when generating code for a property:

Priority 1- WithEvents

Priority 2- New and Subscript

This means, for example, that if both WithEvents and New are set to True, the property is generated as a WithEvents, and the value of the New model property is ignored.

FullName (Role Property)

The FullName model property specifies if the full name of the referenced class, including the name of the component to which the class is assigned, should be used in the property declaration for a role. The full name is needed to generate correct code for class names that are defined in several components.

The value of the FullName model property can be modified on the **Data Member** tab of the Model Assistant and on the **Visual Basic** tab in the Association Specification.

Table 27 FullName Model Property Values

Value	Result
TRUE	Rose uses the full name of the referenced class in the property declaration for this role. For example, if the class Customer is assigned to a component called OrderSys, Rose generates the following code for the private role Purchaser on an association between an Order and Customer class: `Private Purchaser As OrderSys.Customer`
FALSE	(Default) Rose does not use the full name of the referenced class in the property declaration for this role. In the above example, Rose would generate the following declaration: `Private Purchaser As Customer`

New (Property or Role Property)

The New model property is used to specify whether Rational Rose Visual Basic adds the `New` keyword to the module variable declaration.

The value of the New model property can be changed on the **Data Member** tab of the Model Assistant and on the **Visual Basic** tab in the Property and Association Specification.

Table 28 New Model Property Values

Value	Result
TRUE	Rose adds the `New` keyword to the module variable declaration.
FALSE	(Default) Rose creates module variable declaration without implicit object allocation.

ProcedureID (Property or Role Property)

The ProcedureID property corresponds to the **Procedure ID** box in the **Procedure Attributes** dialog box for the property's or role's property procedures in Microsoft Visual Basic. The value of this model property is set when the property or role in the model is updated from code.

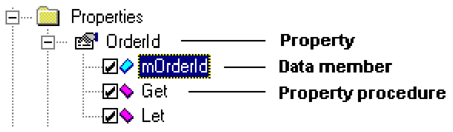
The value of the ProcedureID model property for a property or role is only available on the **Visual Basic** tab in the Attribute or Association Specification.

Table 29 ProcedureID Model Property Values

Value	Corresponds to value
nothing	(None)
0	(Default)
-552	AboutBox
-550	Refresh

PropertyName (Property or Role Property)

The PropertyName property is the name of the property to which a data member and the associated property procedure belong. The Model Assistant needs this information to display a data member and its property procedures together, as illustrated below:



Subscript (Property or Role Property)

The Subscript model property specifies the array subscript that Rational Rose Visual Basic uses when generating data members for the role or property.

The subscript must include the subscript parenthesis, for example **`(1 To MaxLen)`** or **`(mnuOpen to mnuQuit)`**.

The value of the Subscript model property can be changed on the **Data Member** tab of the Model Assistant and on the **Visual Basic** tab in the Property and Association Specification.

Table 30 Subscript Model Property Values

Value	Result
"(literal)"	Rose uses the given array subscript when generating the data member.
No value	(Default) No subscript is used when generating the data member.
"()"	Rose produces a dynamic array.

UpdateCode (Role Property)

The UpdateCode property specifies if code can be generated for this role.

Note: *The normal model and code synchronization is not applied to roles with the UpdateCode property set to FALSE.*

The value of the UpdateCode model property can be modified on the **Data Member** tab of the Model Assistant (**Should be Generated** option), as well as on the **Visual Basic** tab in the Association Specification.

Table 31 UpdateCode Model Property Values

Value	Result
TRUE	(Default) Rose generates code for this role.
FALSE	Rose does not generate code for this role.

WithEvents (Property or Role Property)

The WithEvents model property is used to enable events triggering from other objects.

The value of the WithEvents model property can be changed on the **Data Member** tab of the Model Assistant and on the **Visual Basic** tab in the Property and Association Specification.

Table 32 WithEvents Model Property Values

Value	Result
TRUE	Rose adds the **`WithEvents`** keyword to the module variable declaration.
FALSE	(Default) Rose creates module variable declaration without enables event triggering.

Model Properties for Methods (Operations)

When generating code for a class, Rational Rose Visual Basic produces a skeletal member function for each user-defined method that is detailed in the Class Specification. You use the following model properties to specify additional details that Rational Rose Visual Basic uses when generating the member functions:

- AliasName
- DefaultBody
- IsStatic
- LibraryName
- ProcedureID
- ReplaceExistingBody

AliasName (Method Property)

The AliasName property indicates that the procedure being called has another name in the DLL. This is useful when the external procedure name is the same as a Visual Basic reserved word. You can also use an alias when a DLL procedure has the same name as a global variable, or constant, or any other procedure in the same scope. An alias is also useful if any characters in the DLL procedure name are not allowed in Visual Basic names.

The AliasName property must be a string literal defining a valid alias name of the declared procedure.

The value of the AliasName model property for methods with stereotype Declare can be changed on the **Declare** tab of the Model Assistant. It can also be changed on the **Visual Basic** tab in the Method Specification.

DefaultBody (Method Property)

The DefaultBody model property is a text property that specifies the code and comments to insert into the method body when generating code for this method. If the DefaultBody model property is empty, Rational Rose uses the default body defined by the template currently attached to this class. However, if DefaultBody is not empty, its contents override the code defined by the template.

Depending on the value of the ReplaceExistingBody model property, Rational Rose inserts the contents of the DefaultBody in the method body either the first time or every time the method is generated.

Table 33 DefaultBody Model Property Values

If ReplaceExistingBody is	Result
TRUE	Rose always replaces the method body with the contents of DefaultBody when updating the code for this class. If DefaultBody is empty, Rose uses the default body defined by the template currently attached to this class.
FALSE	(Default) Rose inserts the contents of DefaultBody, but only the first time code is generated for this method. If DefaultBody is empty, Rose uses the default body defined by the template currently attached to this class.

The value of the DefaultBody model property for Sub methods, Function methods, and property procedures can be modified on the **Method** or **Property Get/Set/Let** tab of the Model Assistant. It can also be modified on the **Visual Basic** tab in the Method Specification.

Note: *Rose does not check the correctness of the code that you specify in the* DefaultBody *property.*

IsStatic (Method Property)

The IsStatic property specifies if Rational Rose Visual Basic produces the function procedure for a user-defined method with the Static keyword. This indicates that the function procedure's local variables are preserved between calls. The IsStatic property does not affect variables that are declared outside the function procedure, even if they are used in the procedure.

The value of the IsStatic model property for Sub methods, Function methods, and property procedures can be modified on the **Method** or **Property Get/Set/Let** tab of the Model Assistant. It can also be modified on the **Visual Basic** tab in the Method Specification.

The following table summarizes the possible values for the IsStatic property. In this table, **`result`** is the return type of the member function, **`fname`** is the name of the member function, and **`params`** is the formal parameter list.

Table 34 IsStatic Model Property Values

Value	**Result**

TRUE	Rose produces a static Function procedure declaration. For example:
	
	`Static Sub fname (params)`
	`Static Function fname (params) As result`

FALSE	(Default) Rose produces a Function procedure declaration. For example:
	
	`Sub fname (params)`
	`Function fname (params) As result`

LibraryName (Method Property)

The LibraryName property specifies the name of the DLL that contains the declared method.

The value of the LibraryName model property must be a string literal. It can be modified on the **Declare** tab of the Model Assistant and on the **Visual Basic** tab in the Method Specification.

ProcedureID (Method Property)

The ProcedureID property corresponds to the **Procedure ID** box in the **Procedure Attributes** dialog box for the method in Microsoft Visual Basic. The value of this model property is set when the method in the model is updated from code.

The value of the ProcedureID model property for a method is only available on the **Visual Basic** tab in the Method Specification.

Table 35 ProcedureID Model Property Values

Value	**Corresponds to value**
nothing	(None)
0	(Default)
-552	AboutBox
-550	Refresh

ReplaceExistingBody (Method Property)

The ReplaceExistingBody property specifies whether or not to always replace the body of a method with the default body in the model when updating the code for this class. The default body is the contents of the DefaultBody model property, or if that model property is empty, the default body is defined by the template currently attached to this class.

The value of the ReplaceExistingBody model property for Sub methods, Function methods, and property procedures can be modified on the **Method** or **Property Get/Set/Let** tab of the Model Assistant. It can also be modified on the **Visual Basic** tab in the Method Specification.

Table 36 ReplaceExistingBody Model Property Values

Value	Result
TRUE	Rose always replaces the method body with the default body when updating the code for this class. Thus, the current method body is overwritten with the default body.
FALSE	(Default) Rose inserts a default body only the first time code is generated for this method.

Model Properties for Generalization Relationship

The following model properties can be used on generalization relationships:

- FullName
- ImplementsDelegation

FullName (Generalization Property)

The FullName model property specifies if the full name of the implemented class, including the name of the component to which the class is assigned, should be used in the Implements statement. The full name is needed to generate correct code for class names that are defined in several components.

The value of the FullName model property can be modified on the **Implements Class** tab of the Model Assistant and on the **Visual Basic** tab in the Generalization Specification.

Table 37 FullName Model Property Values

Value	**Result**
TRUE	Rose uses the full name of the referenced class in the Implements statement. For example, if the class Customer is assigned to a component called OrderSys, Rose generates the following code for the generalization relationship between an Order and Customer class: Implements OrderSys.Customer
FALSE	(Default) Rose does not use the full name of the referenced class in the Implements statement. In the above example, Rose would generate the following declaration: Implements Customer

ImplementsDelegation (Generalization Property)

The ImplementsDelegation model property is used to control whether to generate code that delegates the implemented public methods to the delegation object.

The default value of this model property is True. If the delegation object is defined in a type library without code, or if the current class is supposed to provide its own variants of the implemented methods, you should use the value False.

The value of the ImplementsDelegation property can be changed in the Model Assistant under the Implements Classes folder, by selecting or clearing the check box next to the delegation object declaration, and on the **Visual Basic** tab in the Generalization Specification.

Table 38 ImplementsDelegation Model Property Values

Value	Result

TRUE	(Default) Rose generates an object declaration and dispatching declarations of the delegation object into the current class. For example, if B inherits from A, which has a public method called methodA(), the following code is generated into B:
	
	`Implements A`
	`Private mAObject As New A`
	`Private Sub A_MethodA()`
	`    mAObject.MethodA`
	`End Sub`

FALSE	Rose does not generate an object declaration and dispatching declarations of the delegation object into the current class. For example, if B inherits from A, which has a public method called methodA(), the following code is generated into B:
	
	`Implements A`
	`Private Sub A_MethodA()`
	`End Sub`

Appendix B

UML to Visual Basic Mapping Quick Reference

This appendix provides quick reference tables that map the Visual Basic programming language and UML constructs for code generation or reverse engineering.

UML to Visual Basic Mapping

The following table summarizes the mapping between model element types in UML and the Visual Basic programming language.

Table 39 UML to Visual Basic Mapping

Model element	**Becomes in Visual Basic**
Logical View	
Class	Project item of the same type as the stereotype of the class in the model
Class utility	Module
Interface	Class module
Nested class	Enum declaration, Type declaration, or web item, depending on the stereotype of the nested class in the model
Logical package	Nothing
Property (attribute)	Property or constant, depending on the stereotype of the property in the model
Method (operation)	Method, Property Get, Set, or Let Procedure, Declare declaration, or Event declaration, depending on the stereotype of the method in the model

Table 39 UML to Visual Basic Mapping

Model element	Becomes in Visual Basic
Role of an association	Property (if navigable) Also, a reference to a COM component (if an imported interface is associated)
Generalization	Implements construct with delegation code
Realize	Implements construct without delegation code
Dependency	If a class is dependent upon a module (class utility), the module is automatically added to the same project as the class. If a class is dependent upon a class that belongs to another component, a project reference to that component is added to the project. If a class is dependent upon a class in the same component, nothing is generated.
Component View	
Component	Visual Basic project of the type defined by the component's stereotype
Component package	Nothing
Dependency	A reference in the project

Visual Basic to UML Mapping

The following table summarizes the mapping between the Visual Basic programming language and UML.

Table 40 Visual Basic to UML Mapping

Project item	**Becomes in the model**
Visual Basic project	Component with the same stereotype as the type of the project
COM component compiled from the Visual Basic project	A component with the language and stereotype COM, and a logical package with the component's type library
Project reference	A component and a logical package with the referenced type library, as well as a dependency relationship with the project's component
Project item	Class with the same stereotype as the type of the project item
Module	Class utility
Code comment	Text in the **Documentation** box in the corresponding model element's specification
Implements statement	Realize relationship
Constant declaration	Property (attribute) with default value
Enum or Type declaration	Nested class with stereotype Enum or Type, which is contained by the reverse engineered class
Web item in web class	Nested class with stereotype Custom WebItem or Template WebItem, which is contained by the reverse engineered web class
Data member	Property (attribute)
	Navigable association to object type
Method	Method (operation), stereotype= empty

Table 40 Visual Basic to UML Mapping

Project item	**Becomes in the model**
Event declaration	Method, stereotype = Event
Declare declaration	Method, stereotype = Declare
Property procedure	Method, stereotype = Set, Get, or Let

Index