MDG Link for Visual Studio.NET

Welcome to the MDG link for Visual Studio.NET. The MDG link for Visual Studio.NET is designed to allow users to work simultaneously with both Enterprise Architect and Visual Studio.NET and merge the changes with minimal effort.

MDG Link For Visual Studio.NET

Introduction

by John Redfern

MDG Link for Visual Studio.NET Bridge provides integration between Enterprise Architect and Visual Studio.NET.

MDG Link for Visual Studio.NET

Printed: January 2005

Publisher

Sparx Systems

Managing Editor

Geoffrey Sparks

Technical Editors

John Redfern

Special thanks to:

All the people who have contributed suggestions, examples, bug reports and assistance in the development of EA over the last six years. The task of developing and maintaining this tool has been greatly enhanced by their contribution.

Table of Contents

Foreword

MDG Link for Visual Studio.NET Bridge provides integration between Enterprise Architect and Visual Studio.NET.

Part I

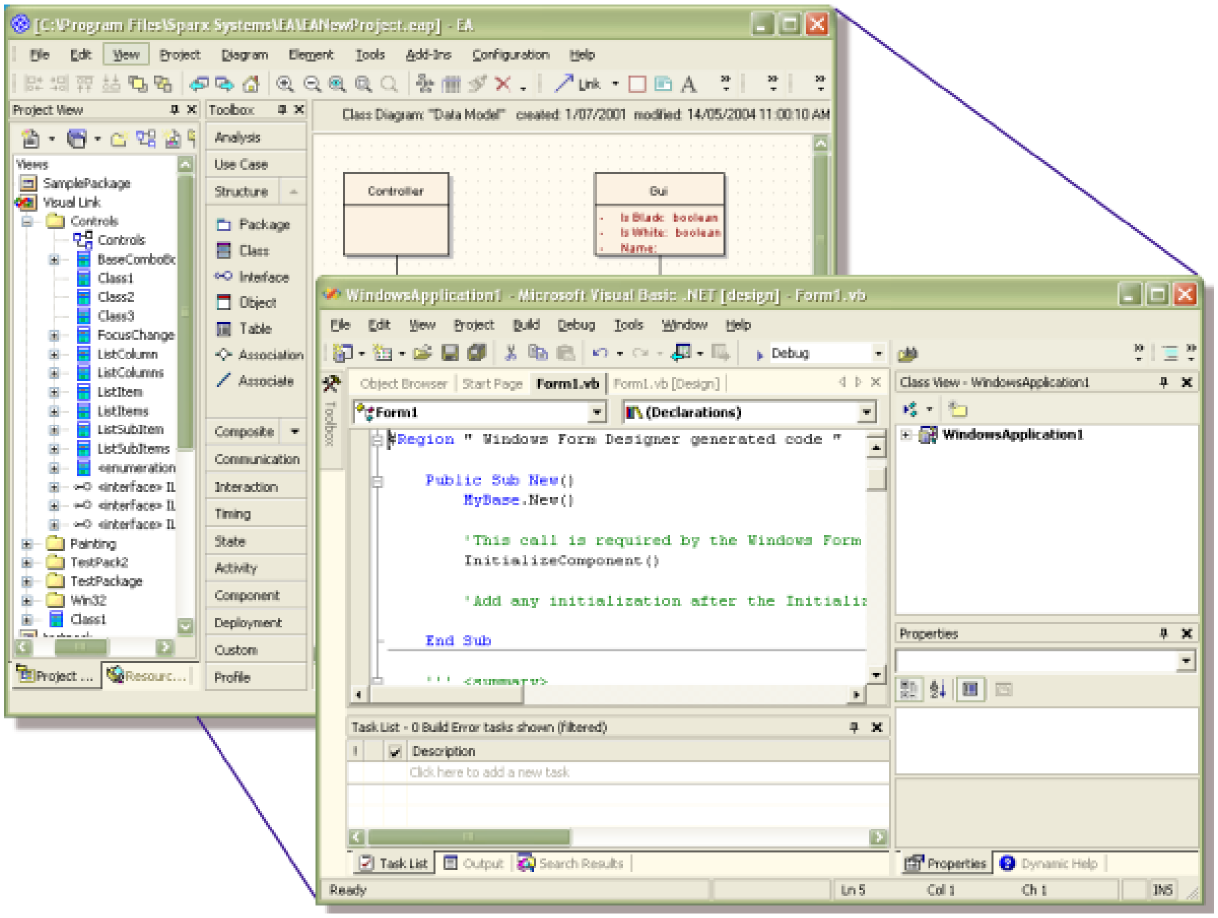
1 Introduction

1.1 Welcome

Welcome to the **Model Driven Generator Link for Visual Studio.NET™**. The MDG Link™ for Visual Studio.NET is designed to allow users to work simultaneously with both Enterprise Architect and Visual Studio.NET and merge the changes with minimal effort. The MDG Link for Visual Studio.NET works with both the **Professional** and **Corporate** editions of Enterprise Architect. The MDG Link for Visual Studio.NET provides a tight integration between Enterprise Architect and Visual Studio allowing you to either create UML in Enterprise Architect or to generate UML from Visual Studio.NET.

MDG Link for Visual Studio.NET has the following features:

- Allows the user to make use of a simple, easy to use connection between Enterprise Architect models to Visual Studio.NET projects.
- Merge an entire project with a with a simple, easy to use process.
- Support for different development configurations.
- Prompt user with proposed merge before changes are written.



```
#Region " Windows Form Designer generated code "

    Public Sub New()
        MyBase.New()

        'This call is required by the Windows Form
        InitializeComponent()

        'Add any initialization after the Initializ

    End Sub

    ''' <summary>
```

To get started now, see Getting Started.

See also:
License Agreement
Copyright Notice

1.2 Copyright Notice

Sparx Systems Pty. Ltd.
7 Curtis St,
Creswick, Victoria 3363,
AUSTRALIA

Phone: +61 (3) 5345 1140
Fax: +61 (3) 5345 1104

Support Email: support@sparxsystems.com.au
Sales Email: sales@sparxsystems.com.au

Website: www.sparxsystems.com.au

1.3 Trademarks

Acknowledgement of Trademarks

The following are Trademarks of Microsoft:

- Microsoft Visual Studio.NET
- Windows®

The following are Registered Trademarks of OMG:

- CORBA®
- the OMG Object Management Group logo
- The Information Brokerage®
- CORBA Academy®
- IIOP®
- XMI®

The following are Trademarks of the OMG:

- OMG™
- Object Management Group™
- the CORBA logo
- ORB™
- Object Request Broker™
- the CORBA Academy design
- OMG Interface Definition Language™
- IDL™
- CORBAservices™
- CORBAfacilities™
- CORBAmed™
- CORBAnet™
- Unified Modeling Language™
- UML™
- the UML Cube logo
- MOF™
- CWM™
- Model Driven Architecture™
- MDA™
- OMG Model Driven Architecture™

- OMG MDA™

1.4 EA Software Product License Agreement

SOFTWARE PRODUCT LICENSE AGREEMENT

MDG Link for Visual Studio.NET
Copyright (C) 1998-2004 Sparx Systems Pty Ltd. All Rights Reserved

IMPORTANT-READ CAREFULLY: This End User Licence Agreement ("EULA") is a legal agreement between YOU as Licensee and SPARX for the SOFTWARE PRODUCT identified above. By installing, copying, or otherwise using the SOFTWARE PRODUCT, YOU agree to be bound by the terms of this EULA. If YOU do not agree to the terms of this EULA, promptly return the unused SOFTWARE PRODUCT to the place of purchase for a full refund.

The copyright in the SOFTWARE PRODUCT and its documentation is owned by Sparx Systems Pty Ltd A.B.N 38 085 034 546. Subject to the terms of this EULA, YOU are granted a non-exclusive right for the duration of the EULA to use the SOFTWARE PRODUCT. YOU do not acquire ownership of copyright or other intellectual property rights in any part of the SOFTWARE PRODUCT by virtue of this EULA.

Your use of this software indicates your acceptance of this EULA and warranty.

DEFINITIONS
In this End User Licence Agreement, unless the contrary intention appears,
"EULA" means this End User Licence Agreement
"SPARX" means Sparx Systems Pty Ltd A.B.N 38 085 034 546
"Licensee" means YOU, or the organization (if any) on whose behalf YOU are taking the EULA.
"Registered Edition of MDG Link for Visual Studio.NET" means the edition of the SOFTWARE PRODUCT which is available for purchase from the web site:
"SOFTWARE PRODUCT" or "SOFTWARE" means MDG Link for Visual Studio.NET, which includes computer software and associated media and printed materials, and may include online or electronic documentation.
"Support Services" means email based support provided by SPARX, including advice on usage of MDG Link for Visual Studio.NET, investigation of bugs, fixes, repairs of models if and when appropriate and general product support.
"Trial edition of MDG Link for Visual Studio.NET" means the edition of the SOFTWARE PRODUCT which is available free of charge for evaluation purposes for a period of 30 days.
"SPARX support engineers" means employees of SPARX who provide on-line support services.

GRANT OF LICENSE
In accordance with the terms of this EULA YOU are granted the following rights:
a) to install and use one copy of the SOFTWARE PRODUCT, or in its place, any prior version for the same operating system, on a single computer. As the primary user of the computer on which the SOFTWARE PRODUCT is installed, YOU may make a second copy for your exclusive use on either a home or portable computer.
b) to store or install a copy of the SOFTWARE PRODUCT on a storage device, such as a network server, used only to install or run the SOFTWARE PRODUCT over an internal network. If YOU wish to increase the number of users entitled to concurrently access the SOFTWARE PRODUCT, YOU must notify SPARX and agree to pay an additional fee.
c) to make copies of the SOFTWARE PRODUCT for backup and archival purposes.

EVALUATION LICENSE
The Trial version of MDG Link for Visual Studio.NET is not free software. Subject to the terms of this agreement, YOU are hereby licensed to use this software for evaluation purposes without charge for a period of 30 days.
Upon expiration of the 30 days, the Software Product must be removed from the computer. Unregistered use of MDG Link for Visual Studio.NET after the 30-day evaluation period is in violation of Australian, U.S. and international copyright laws.
SPARX may extend the evaluation period on request and at their discretion.

If YOU choose to use this software after the 30 day evaluation period a licence must be purchased (as described at http://www.sparxsystems.com.au/ea_purchase.htm). Upon payment of the licence fee, YOU will be sent details on where to download the registered edition of MDG Link for Visual Studio.NET and will be provided with a suitable software 'key' by email.

ASSIGNMENT

YOU may only assign all your rights and obligations under this EULA to another party if YOU supply to the transferee a copy of this EULA and all other documentation including proof of ownership. Your licence is then terminated.

TERMINATION

Without prejudice to any other rights, SPARX may terminate this EULA if YOU fail to comply with the terms and conditions. Upon termination YOU or YOUR representative shall destroy all copies of the SOFTWARE PRODUCT and all of its component parts or otherwise return or dispose of such material in the manner directed by SPARX.

WARRANTIES AND LIABILITY

WARRANTIES

SPARX warrants that the SOFTWARE PRODUCT will perform substantially in accordance with the accompanying written materials for a period of ninety (90) days from the date of receipt, and any Support Services provided by SPARX shall be substantially as described in applicable written materials provided to YOU by SPARX, and SPARX support engineers will make commercially reasonable efforts to solve any problems associated with the SOFTWARE PRODUCT.

EXCLUSIONS

To the maximum extent permitted by law, SPARX excludes, for itself and for any supplier of software incorporated in the SOFTWARE PRODUCT, all liability for all claims , expenses, losses, damages and costs made against or incurred or suffered by YOU directly or indirectly (including without limitation lost costs, profits and data) arising out of:
YOUR use or misuse of the SOFTWARE PRODUCT;
YOUR inability to use or obtain access to the SOFTWARE PRODUCT;
Negligence of SPARX or its employees, contractors or agents, or of any supplier of software incorporated in the SOFTWARE PRODUCT, in connection with the performance of SPARX'S obligations under this EULA; or
Termination of this EULA by either party for any reason.

LIMITATION

The SOFTWARE PRODUCT and any documentation are provided "AS IS" and all warranties whether express, implied, statutory or otherwise, relating in any way to the subject matter of this EULA or to this EULA generally, including without limitation, warranties as to: quality, fitness; merchantability; correctness; accuracy; reliability; correspondence with any description or sample, meeting your or any other requirements; uninterrupted use; compliance with any relevant legislation and being error or virus free are excluded. Where any legislation implies in this EULA any term, and that legislation avoids or prohibits provisions in a contract excluding or modifying such a term, such term shall be deemed to be included in this EULA. However, the liability of SPARX for any breach of such term shall if permitted by legislation be limited, at SPARX'S option to any one or more of the following upon return of the SOFTWARE PRODUCT and a copy of the receipt:
If the breach relates to the SOFTWARE PRODUCT:
the replacement of the SOFTWARE PRODUCT or the supply of an equivalent SOFTWARE PRODUCT;
the repair of such SOFTWARE PRODUCT; or the payment of the cost of replacing the SOFTWARE PRODUCT or of acquiring an equivalent SOFTWARE PRODUCT; or
the payment of the cost of having the SOFTWARE PRODUCT repaired;
If the breach relates to services in relation to the SOFTWARE PRODUCT:
the supplying of the services again; or
the payment of the cost of having the services supplied again.

TRADEMARKS

All names of products and companies used in this EULA, the SOFTWARE PRODUCT, or the enclosed documentation may be trademarks of their corresponding owners. Their use in this EULA is intended to be in compliance with the respective guidelines and licenses. Windows, Windows 95, Windows 98, Windows NT, Windows ME, Windows XP and Windows 2000 are trademarks of Microsoft.

GOVERNING LAW

This agreement shall be construed in accordance with the laws of the Commonwealth of AUSTRALIA.

1.5 Ordering MDG Link for Visual Studio.NET

MDG Link for Visual Studio.NET is designed, built and published by Sparx Systems and is available from Sparx Systems.

The latest information on pricing and purchasing is available at: Sparx Systems Purchase/Pricing Website.

Purchase Options

- On-line using a secure credit-card transaction. See: Pricing and Purchase Options.
- Fax
- Check or equivalent
- Bank transfer

For more information, contact sales@sparxsystems.com.au.

1.6 Support

Support is available to Registered Users of MDG Link for Visual Studio.NET. All support issues are currently dealt with via email. Sparx Systems endeavor to provide a rapid response to all questions and concerns regarding the MDG Link for Visual Studio.NET.

You can contact the support team at support@sparxsystems.com.au.

An online user forum is also available for your questions and perusal, at www.sparxsystems.com.au/cgi-bin/yabb/YaBB.cgi.

Part II

2 Getting Started

2.1 Registering the MDG Link

Registering the MDG link for Visual Studio.NET

Follow these steps to activate the MDG Link for Visual Studio.NET:

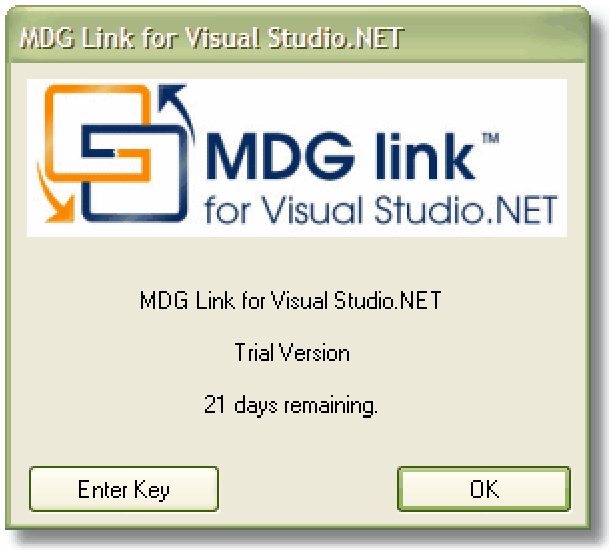
1. Purchase one or more licenses.

2. Once you have paid for a licensed version of the MDG Link for Visual Studio.NET, you will receive (via email or other suitable means)
   - a license key(s)
   - the address of the web site from which to download the full version

3. Save the license key and download the latest full install package from the address supplied.

4. Run the setup program to install the full version.

5. If this is the first time that the MDG Link for Visual Studio.NET has been installed a MDG Enter Key dialog box will prompt the user to register the MDG Link or to continue the trial.

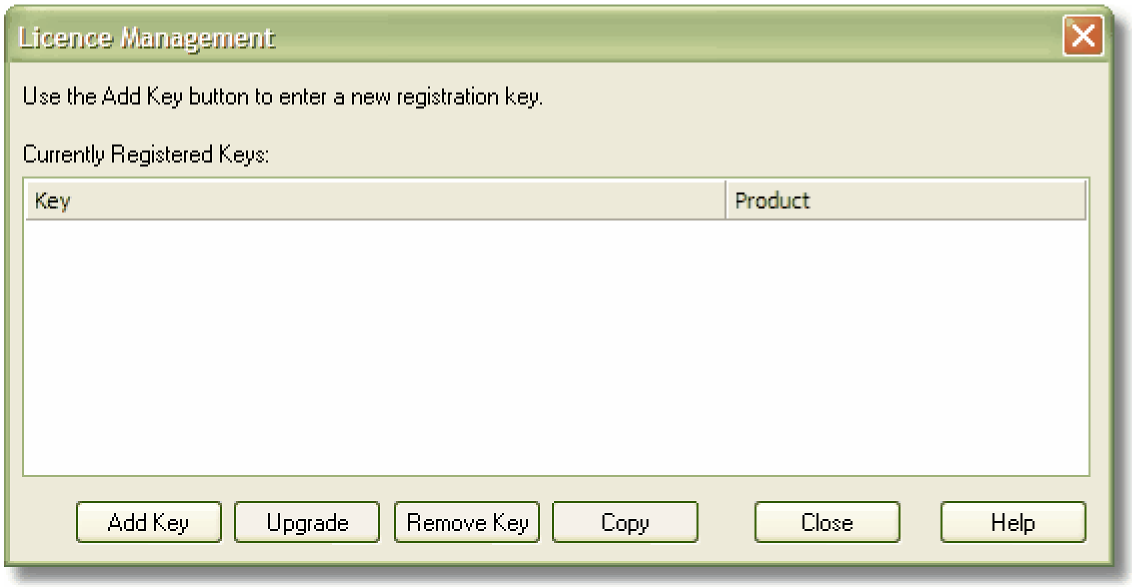


MDG Link for Visual Studio.NET

Trial Version

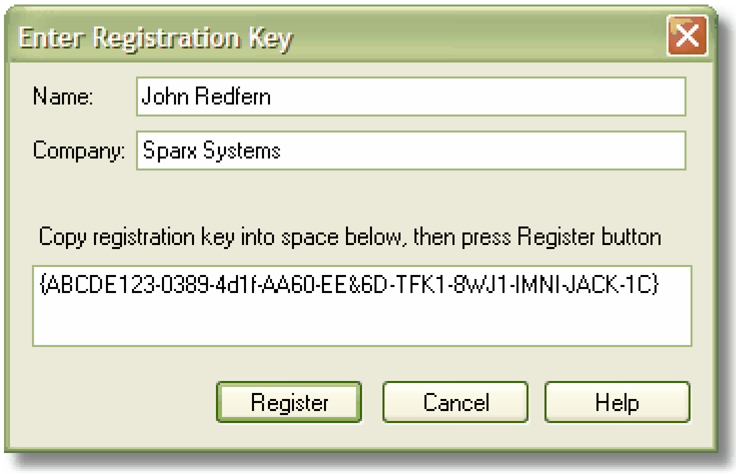
21 days remaining.

6. When the Licence Management dialog appears click on the Add Key button.


Licence Management
Use the Add Key button to enter a new registration key.
Currently Registered Keys:
Key
Product
Add Key
Upgrade
Remove Key
Copy
Close
Help

7. The Enter Registration dialog will then prompt the user to enter a license key (use copy and paste from an email to avoid typing mistakes) then Press OK on the MDG Enter Key Dialog.

8. The full version of the MDG Link for Visual Studio.NET is available for use with your version of Enterprise Architect.


Enter Registration Key

Name: John Redfern

Company: Sparx Systems

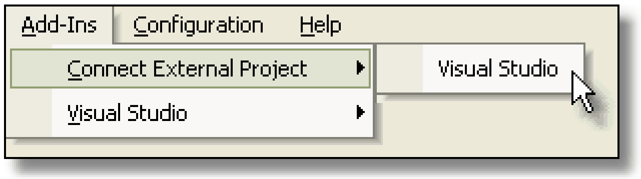
Copy registration key into space below, then press Register button

{ABCDE123-0389-4d1f-AA60-EE&6D-TFK1-8WJ1-IMNI-JACK-1C}

2.2 Setting Up the MDG Link

Before the MDG Link can perform its main operations, the EA Package must be configured to link to a particular Visual Studio.NET project. For more information on how to create a link to a Visual Studio Link go to the Create a link to a Visual Studio Project page.

Once the install program has been run, the MDG Link for Visual Studio.NET should be accessible through the *Add-Ins* item of the menu bar in EA as shown below.

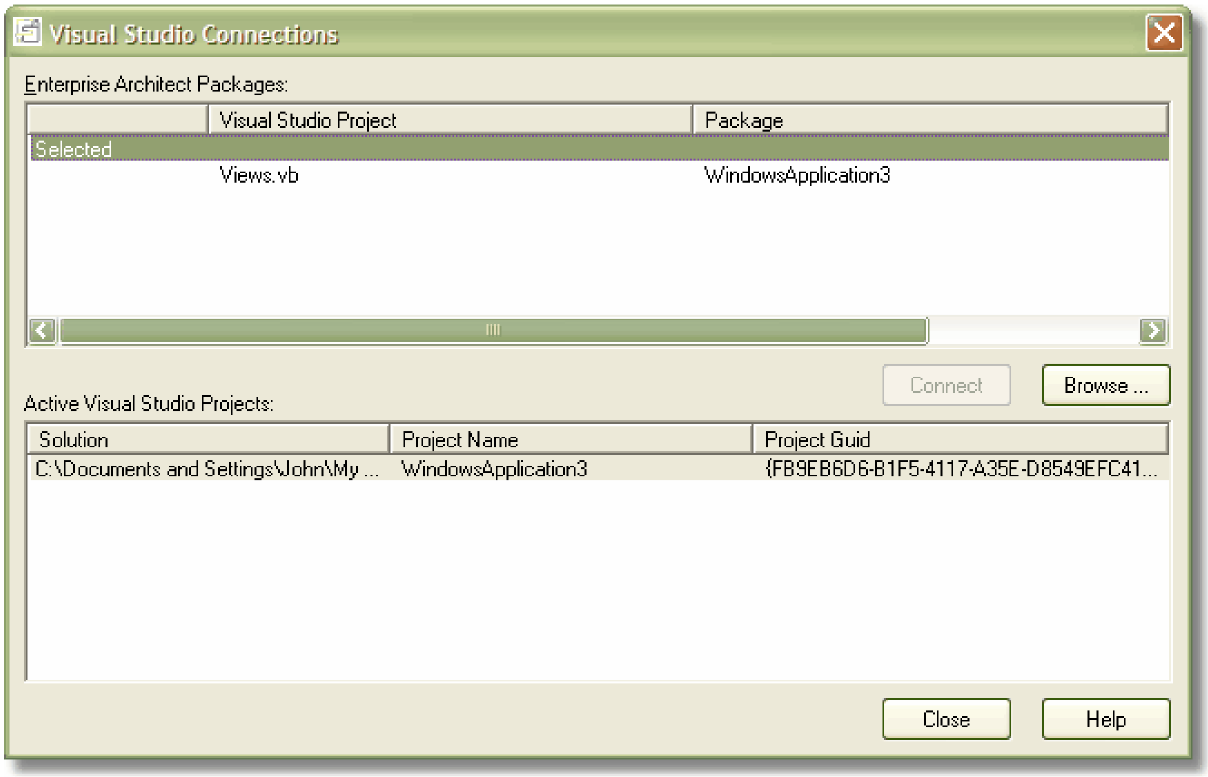


If this menu doesn't appear, check the System Requirements.

2.3 Create a link to a Visual Studio Project

To link an Enterprise Architect package to a particular Visual Studio.NET project, follow the instructions detailed below.

1. From Visual Studio.NET, open the solution containing the project that you wish to link to.
2. Ensure that the project is the active project within the solution.
3. Open an EA model and in the Tree View of the Project View select the package which is to represent your Visual Studio project.
4. Right click on the package to bring up its context menu, go to the *Add-In* | *Connect External Project* | *Visual Studio*, this will bring up a dialog box like the one below.



Enterprise Architect Packages:

	Visual Studio Project	Package
Selected		
	Views.vb	WindowsApplication3

Active Visual Studio Projects:

Solution	Project Name	Project Guid
C:\Documents and Settings\John\My ...	WindowsApplication3	{FB9EB6D6-B1F5-4117-A35E-D8549EFC41...

This dialog box allows you to review and configure connections to Visual Studio.Net from this project.

Text Fields

Existing Connections	Shows the EA packages in the current model which are connected to Visual Studio.net projects.
Selected Package	If the currently selected package in the EA Tree View has a new screen.
Visual Studio Projects	The Visual Studio.NET package that you may connect to.

Buttons

Connect	Connects the EA package to the selected Visual Studio project.
Browse	Click this button to select a Visual Studio solution via Windows Explorer. Once selected, the solution and its projects will appear in the list of "Visual Studio Projects".
Close	Close this form.

Note: If you are using an EA model that has already been configured on another machine, you will still need to tell the MDG Link where the solution lies. This is done by opening the solution through Visual Studio.NET then clicking any of the menu items.

2.4 Visual Studio Connections Dialog Options

The Visual Studio Connections dialog allows the user to connect and disconnect to a single Visual Studio.Net project.

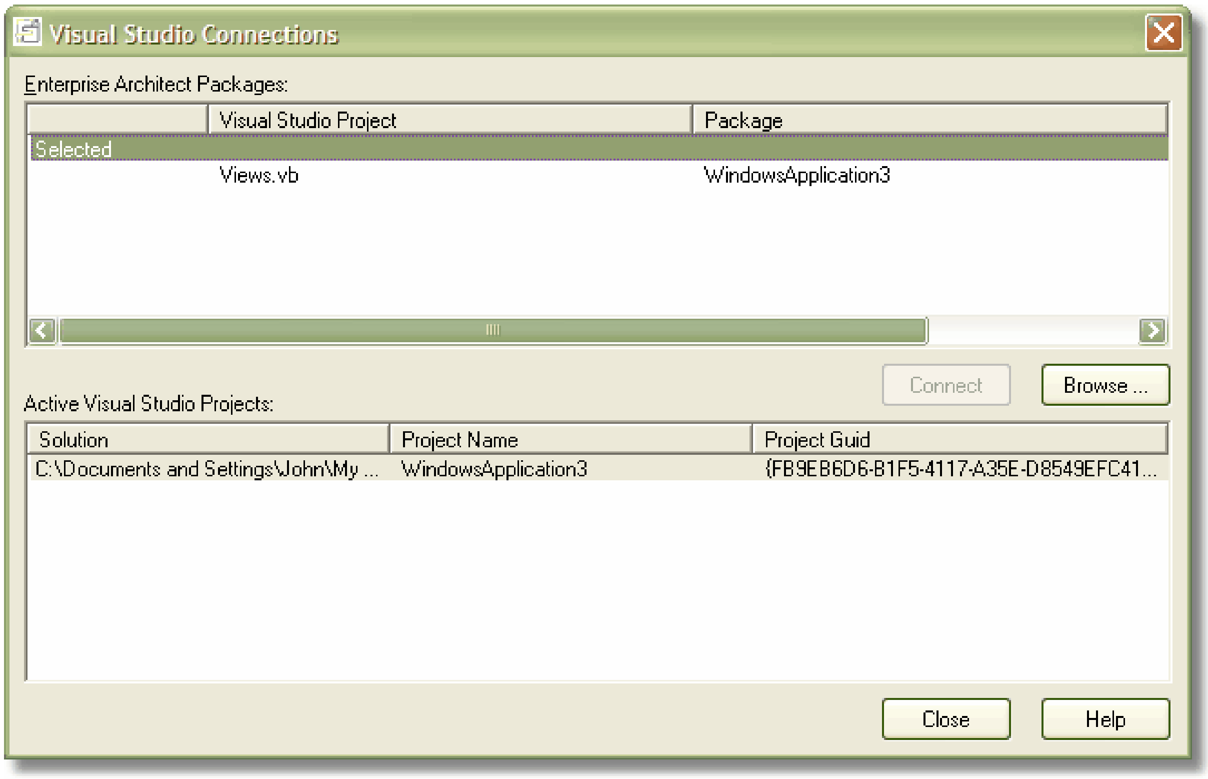
Text Fields

Existing Connections	Shows the EA packages in the current model which are connected to Visual Studio.NET projects.
Selected Package	If the currently selected package in the EA Tree View has a new screen.
Visual Studio Projects	The Visual Studio.NET package that you may connect to.

Buttons

Connect	Connects the EA package to the selected Visual Studio.NET project.
Browse	Click this button when to select a Visual Studio solution via Windows Explorer. Once selected, the solution and its projects will appear in the list of "Visual Studio Projects".
Close	Close this form.

Note: If you are using an EA model that has already been configured on another machine, you will still need to tell the bridge where the solution lies. This is done by opening the solution through Visual Studio.NET then clicking any of the menu items.

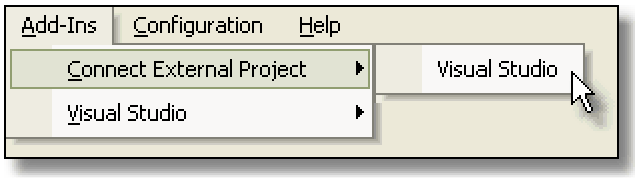

Visual Studio Connections
Enterprise Architect Packages:
Connect
Browse ...
Active Visual Studio Projects:
Close
Help

	Visual Studio Project	Package
Selected		
	Views.vb	WindowsApplication3

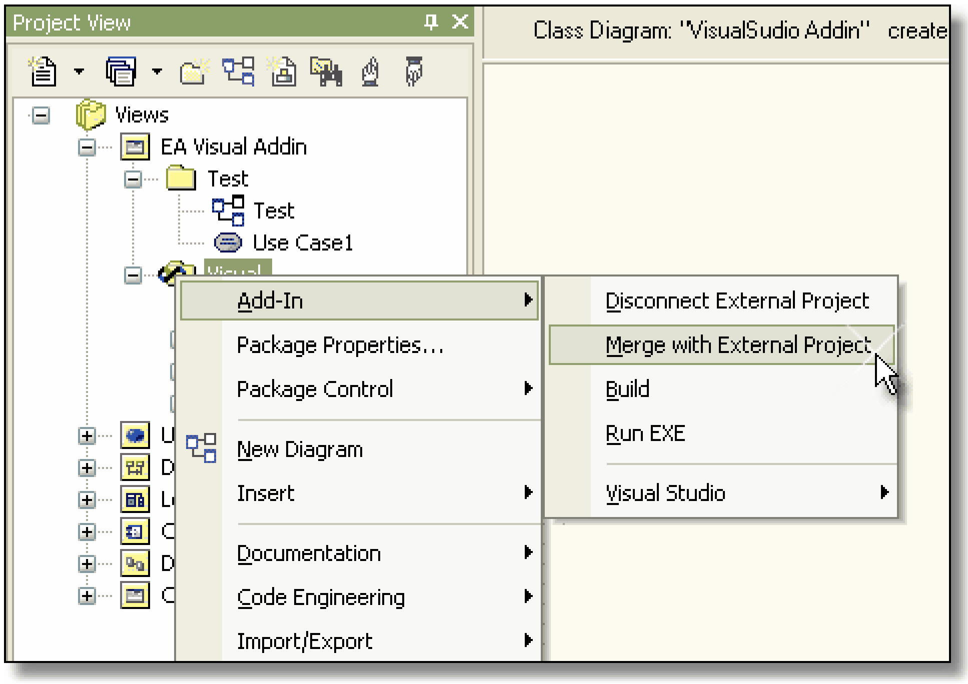
Solution	Project Name	Project Guid
C:\Documents and Settings\John\My ...	WindowsApplication3	{FB9EB6D6-B1F5-4117-A35E-D8549EFC41...

2.5 Merging for the First Time

Merging for the first time provides the user the opportunity to reverse engineer code from a Visual Studio.NET project or to generate code from an Enterprise Architect model into a Visual Studio.NET project. Merging the model is a simple task once a link has been created to a Visual Studio Project, you may then perform a merge from the Project View or from the Add-Ins item on the menu bar. To Merge from the *Add-Ins* Item on the Menu Bar select the *Merge with Visual Studio* item.



To perform a merge from the Project View by selecting an item from tree view item inside your Project View and right clicking on the connected package, this will open up the context menu for the item. Select *Add-In* | *Merge* with External Project.



For more information on the options that are available for Merging go to the Synchronizing Code with a Model Page.

Part III

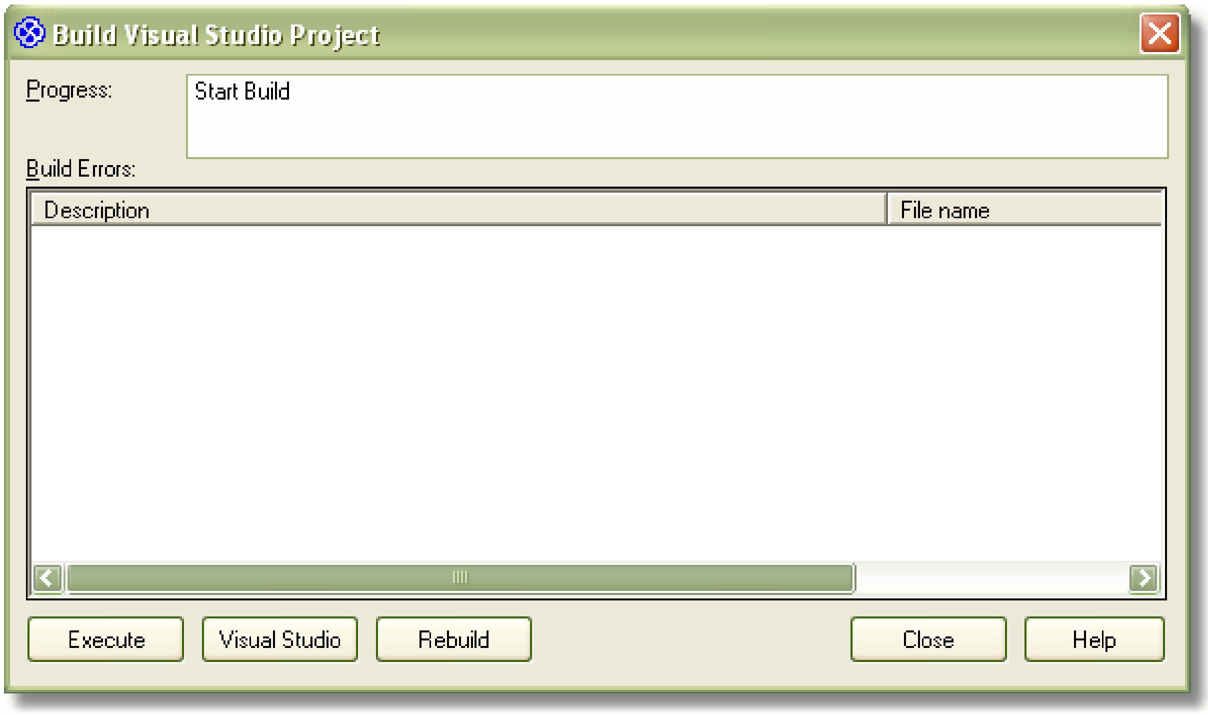
3 Performing Tasks with MDG Link for Visual Studio.NET

3.1 Build Project

It is possible to build and execute a Visual Studio.NET project from within Enterprise Architect. Building the project from within Enterprise Architect allows the user to make changes to the code from the model and to determine if the changes to the code has been successful. Selecting the Build Project option gives the user the choice of building the project and executing the project.

3.1.1 Building and Running a Project

To build a Visual Studio.NET project from within Enterprise Architect select the *Add-Ins* | *Build* menu item. This will open up a dialog box as shown below:

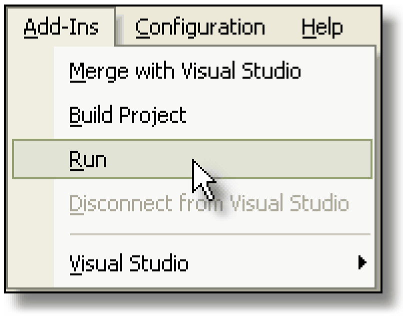


Start Build

To begin a build click on the *Execute* button, when the build is successful the Progress text field will display the message "Build Successful". If any errors have been encountered a list of the errors will be generated in the Build Errors text field. For more information relating to build errors view the Build Project Errors topic.

For the options that available on this dialog box go to the Build Dialog Options topic.

To run a project from EA select the *ADD-Ins* | *Run* menu item. This execute the project from within Enterprise Architect.



3.1.2 Build Dialog Options

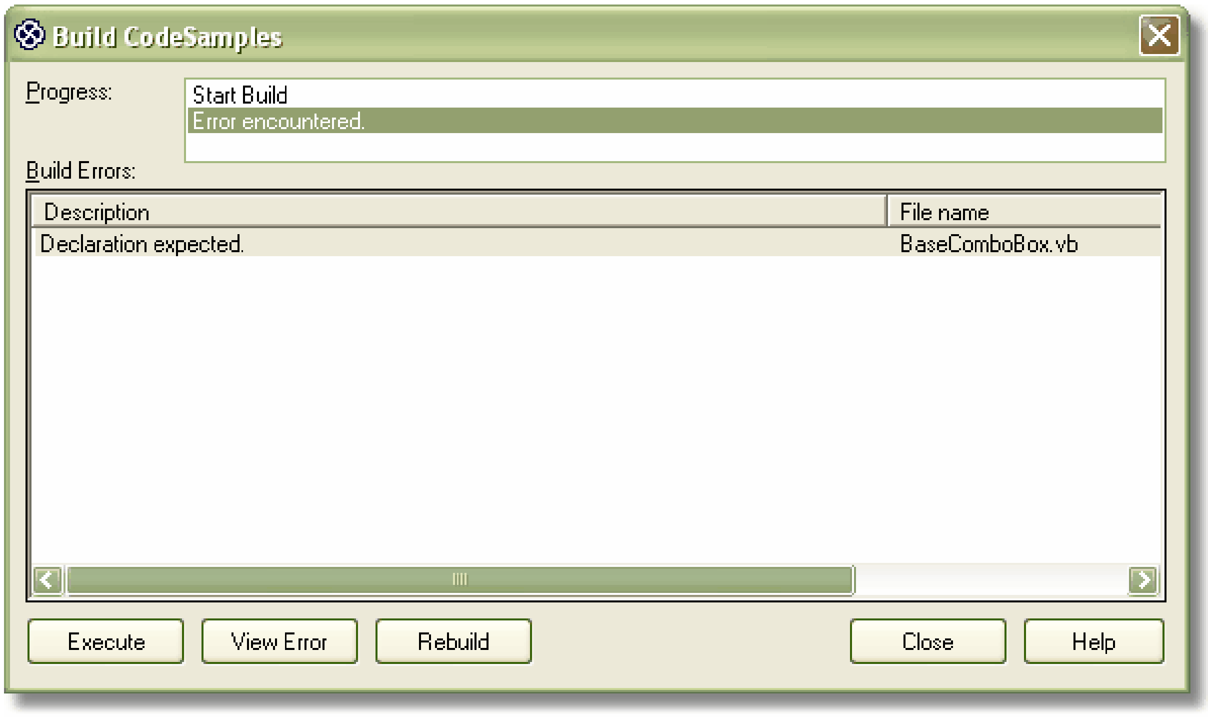
The Build dialog allows the user build and execute a Visual Studio.NET project from within Enterprise Architectl.

Text Fields

Progress	This gives the user the option of selecting classes for export, if a class is not selected it will not be included for the export. The All option selects all of the classes in the list. None selects none of the classes for export.
Build Errors	The build element displays information relating to the error/s that may be encountered during a build, this section gives information relating to the error description as well as the filename associated with the error.

Buttons

Execute	Executes the project.
Visual Studio	Switches to Visual Studio.NET
Rebuild	Rebuild the project.
Close	Closes the Dialog Box.
Help	Opens the Help Contents for this operation.
View Error	Takes the user directly to the line of code with the error (This button will only appear when the Build has encountered errors).

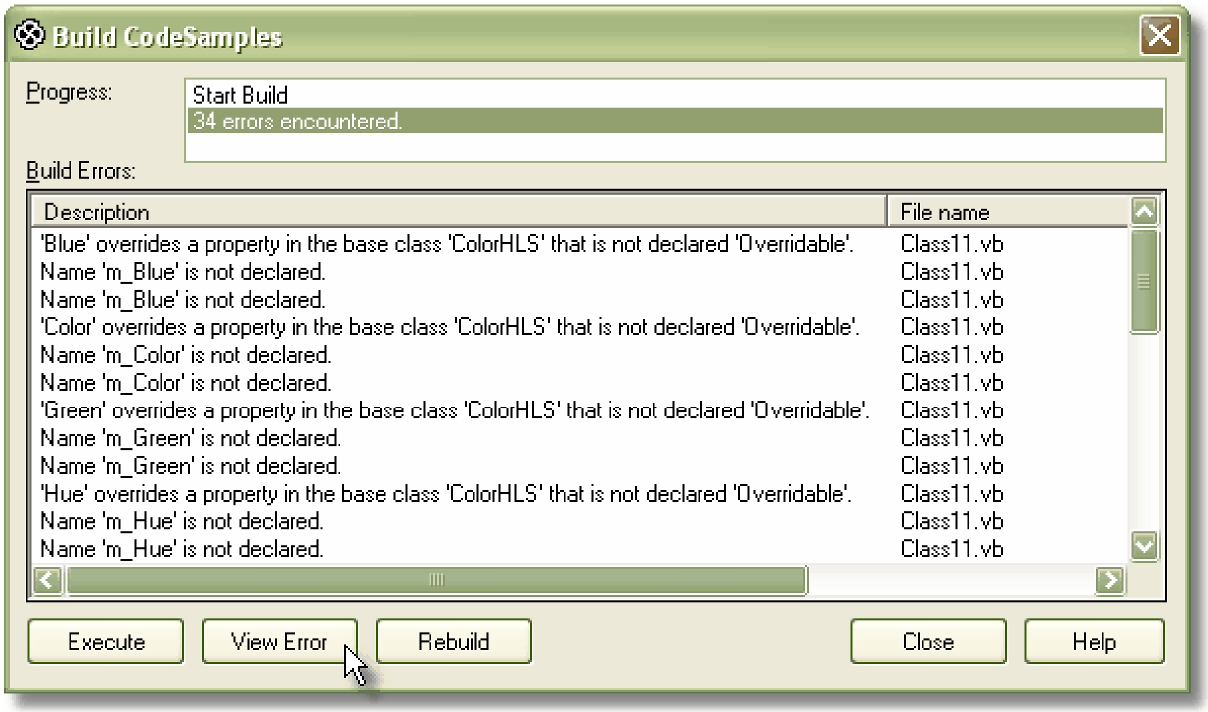


Start Build
Error encountered.

Description	File name
Declaration expected.	BaseComboBox.vb

3.1.3 Build Project Errors

When errors have been encountered when building a project a list of errors will be generated in the Build Errors text field. This will detail the type of error as well as the name of the class. To inspect the error in Visual Studio.NET highlight the class from the Builds Errors text field and click on the *Visual Studio* button, or alternatively highlight the class in the text window and double click the mouse.

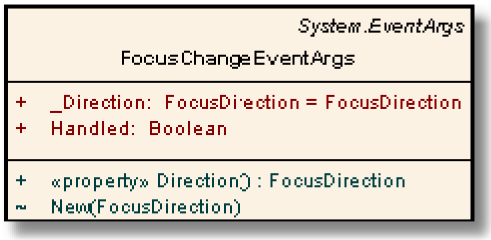


Start Build
34 errors encountered.

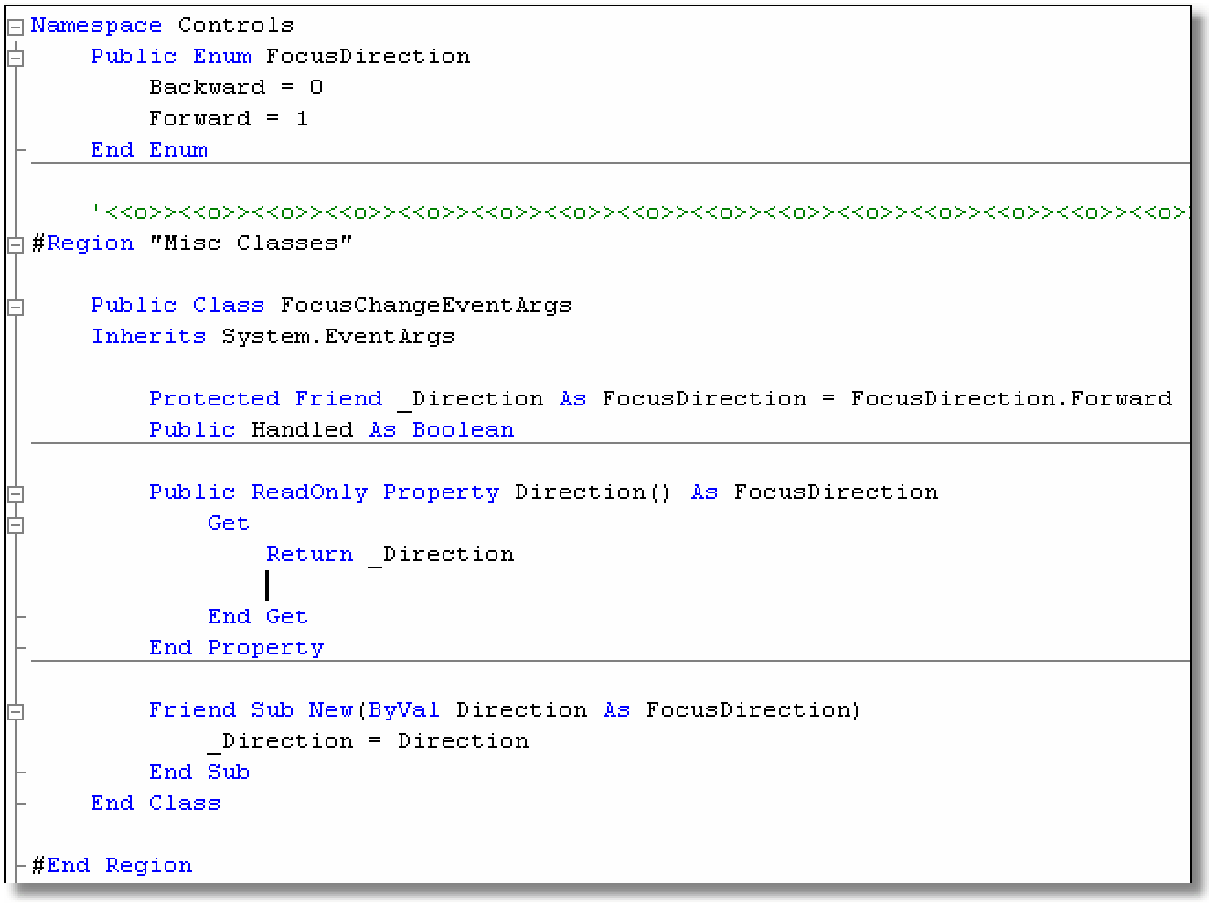
Description	File name
'Blue' overrides a property in the base class 'ColorHLS' that is not declared 'Overridable'.	Class11.vb
Name 'm_Blue' is not declared.	Class11.vb
Name 'm_Blue' is not declared.	Class11.vb
'Color' overrides a property in the base class 'ColorHLS' that is not declared 'Overridable'.	Class11.vb
Name 'm_Color' is not declared.	Class11.vb
Name 'm_Color' is not declared.	Class11.vb
'Green' overrides a property in the base class 'ColorHLS' that is not declared 'Overridable'.	Class11.vb
Name 'm_Green' is not declared.	Class11.vb
Name 'm_Green' is not declared.	Class11.vb
'Hue' overrides a property in the base class 'ColorHLS' that is not declared 'Overridable'.	Class11.vb
Name 'm_Hue' is not declared.	Class11.vb
Name 'm_Hue' is not declared.	Class11.vb

3.2 Classes

The MDG Link for Visual Studio.NET allows for the flexible creation, editing and UML modeling of class diagrams. In UML a class is represented by a rectangle with three sperate compartments. The upper compartment is used to show the name of the class and if it has one the stereotype of the class. The middle compartment is used to display the attributes of the class while the final compartment details the methods or operations that are available for the class. An example of a UML class is shown below.


System.EventArgs
FocusChangeEventArgs
+ _Direction: FocusDirection = FocusDirection
+ Handled: Boolean
+ «property» Direction() : FocusDirection
~ New(FocusDirection)

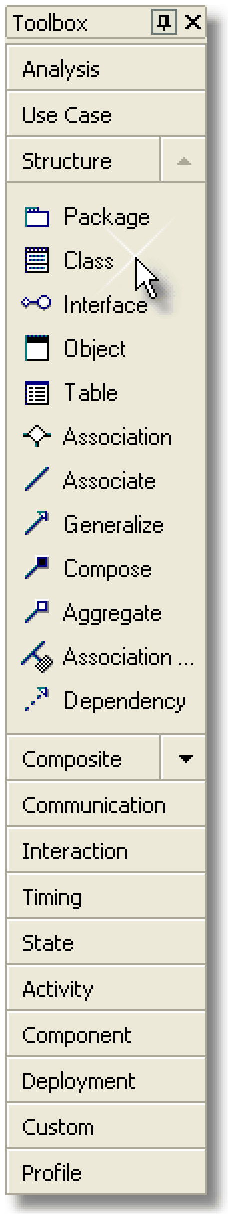
The Visual Basic.NET code that corresponds to this Enterprise Architect Class Diagram will appear in Visual Studio.Net for this class like the diagram below.



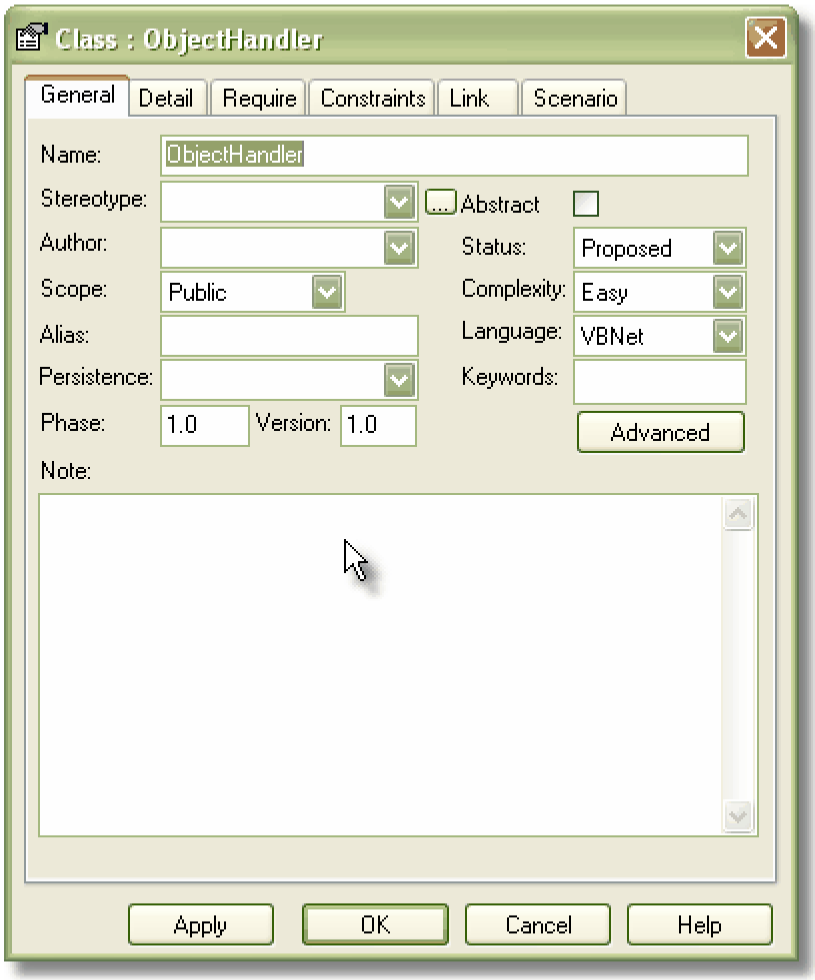
```
Namespace Controls
    Public Enum FocusDirection
        Backward = 0
        Forward = 1
    End Enum

    '<<o>><<o>><<o>><<o>><<o>><<o>><<o>><<o>><<o>><<o>><<o>><<o>><<o>><<o>><<o>
#Region "Misc Classes"

    Public Class FocusChangeEventArgs
    Inherits System.EventArgs

        Protected Friend _Direction As FocusDirection = FocusDirection.Forward
        Public Handled As Boolean

        Public ReadOnly Property Direction() As FocusDirection
            Get
                Return _Direction

            End Get
        End Property

        Friend Sub New(ByVal Direction As FocusDirection)
            _Direction = Direction
        End Sub
    End Class

#End Region
```

3.2.1 Create Class

With the MDG Link for Visual Studio.NET it is possible to create a class either in Visual Studio.NET or via Enterprise Architect. To create a class in Enterprise Architect go to the *Toolbox* toolbar | *Structure* | *Class* and drag the class onto the workspace of the diagram pertaining to the current package (namespace).



Once the class has been dragged onto the diagram a dialog box will be presented to the user to set the properties of the Class.



Name: ObjectHandler
Stereotype:
Abstract
Author:
Status: Proposed
Scope: Public
Complexity: Easy
Alias:
Language: VBNet
Persistence:
Keywords:
Phase: 1.0
Version: 1.0
Note:

This dialog box offers a range of options, from the General tab the following options are available:

Name	The name of the Class
Stereotype	A Stereotype is an element that extends the semantics of the metamodel. Stereotypes must be based on certain existing types or classes in the metamodel. Stereotypes may extend the semantics, but not the structure of pre-existing types and classes.
Abstract	The checkbox determines if the class is an abstract class or a concrete class.
Author	The name of the Author of the class.
Status	Flags the status of the class.
Scope	Scope is used to determine the visibility of the class (public, private, protected and package).
Complexity	Complexity is used for project estimation (easy, medium, hard).
Persistence	The persistence that is associated with the class, it may be either persistent or transient.
Language	Determines or displays the .NET class type the class belongs to.
Alias	Enter an alias (alternate display name) for the object.
Keywords	A free text area that may be filtered in Use Case metrics and search dialogs - typically used for keywords, context information, etc.
Phase	Indicate the phase this element will be implemented in)e.g.1, 1.1, 2.0....).
Version	Version of the Class

3.2.2 Edit Class

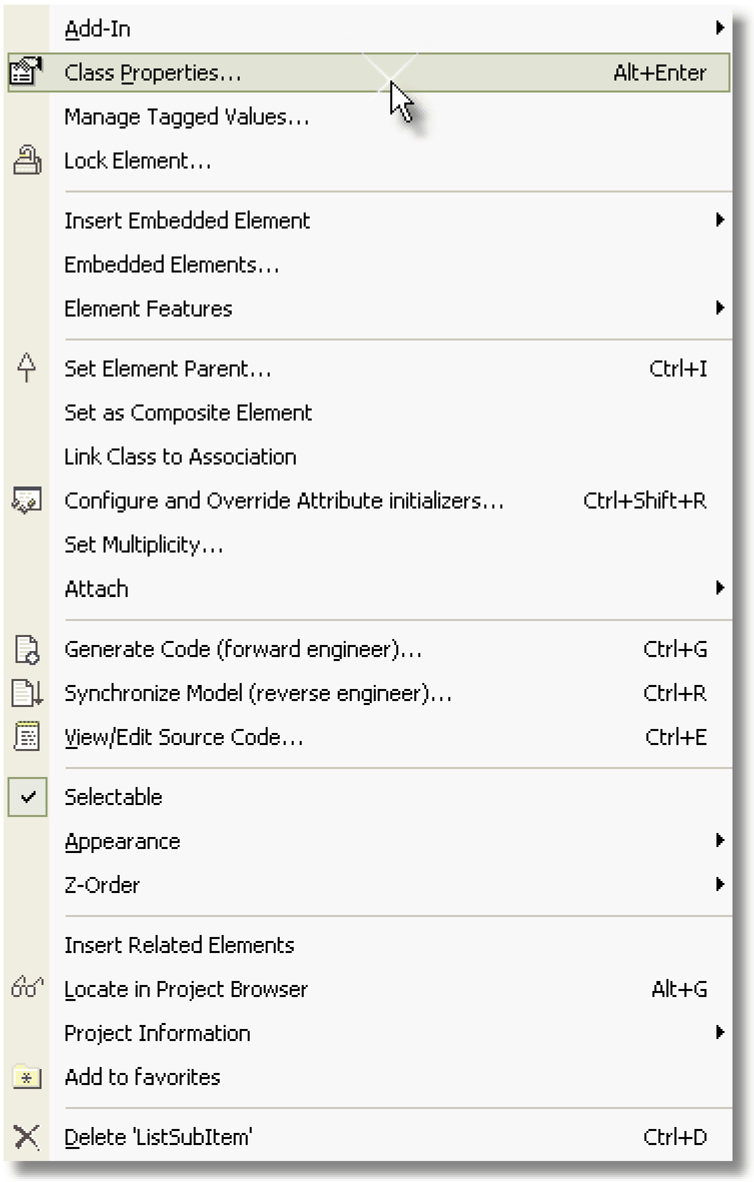
With the MDG Link for Visual Studio.NET it is possible to edit the class from within Enterprise Architect and from Visual Studio.NET. When Editing the class in Enterprise Architect it is feasible to add and delete both attributes, and operations as well as defining inheritance, class dependencies and uses. For more information relating to adding inheritance to classes go to the Adding Inheritance to Classes topic.

To edit the properties of a class in Enterprise Architect use the following instructions. To access the class in Enterprise Architect and to perform the editing in Visual Studio.NET go to the Edit Class, Switching to Visual Studio topic.

1. Select the class which you intend to modify from either a class diagram or from the hierarchical tree in the Project View.

2. Right click on the class to bring up its context view.

3. Select the menu item *Class Properties*, or alternatively press the *ALT*+*ENTER* shortcut key combination to access the class properties dialog box. This will bring up the Classes's property page which has a series of options as detailed in the Create Class topic.

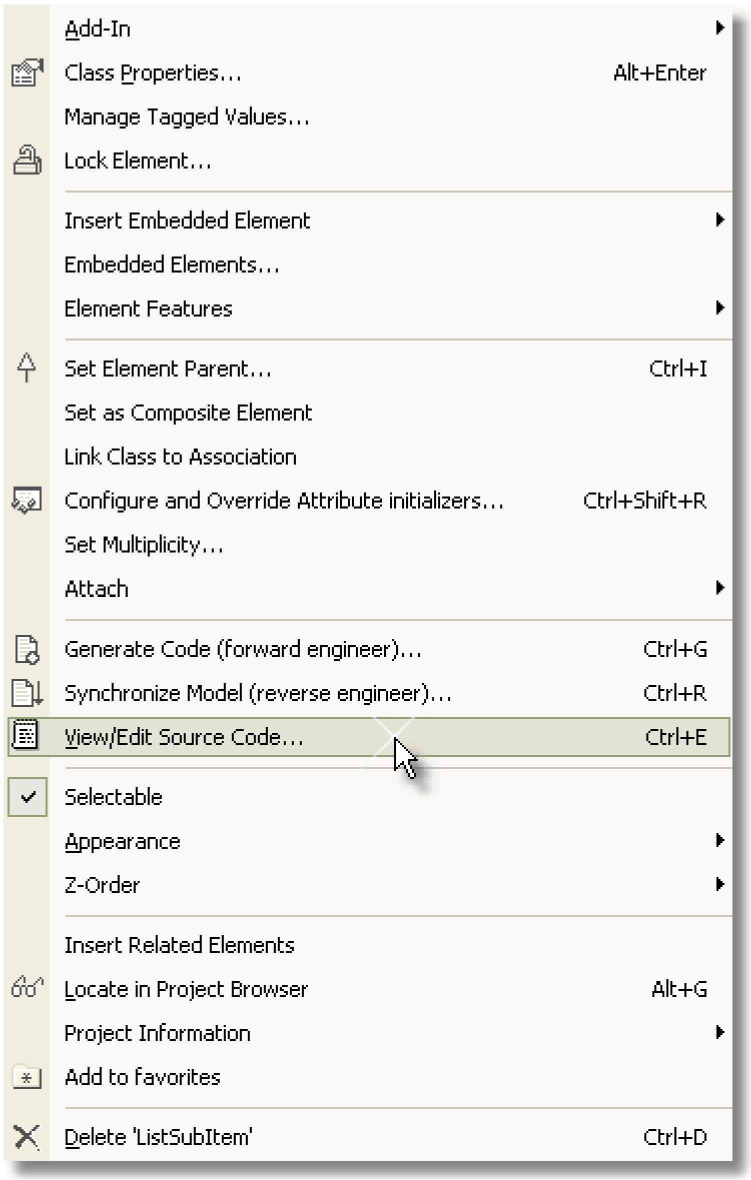

Add-In
Class Properties... Alt+Enter
Manage Tagged Values...
Lock Element...
Insert Embedded Element
Embedded Elements...
Element Features
Set Element Parent... Ctrl+I
Set as Composite Element
Link Class to Association
Configure and Override Attribute initializers... Ctrl+Shift+R
Set Multiplicity...
Attach
Generate Code (forward engineer)... Ctrl+G
Synchronize Model (reverse engineer)... Ctrl+R
View/Edit Source Code... Ctrl+E
Selectable
Appearance
Z-Order
Insert Related Elements
Locate in Project Browser Alt+G
Project Information
Add to favorites
Delete 'ListSubItem' Ctrl+D

The Class Properties Dialog also gives the user access to the classes's attributes and operations, to edit these items go to the Edit Class Attributes and Operations topic.

3.2.3 Edit Class, Switching to Visual Studio

To edit a class in Visual Studio.NET from Enterprise Architect follow the steps outlined below:

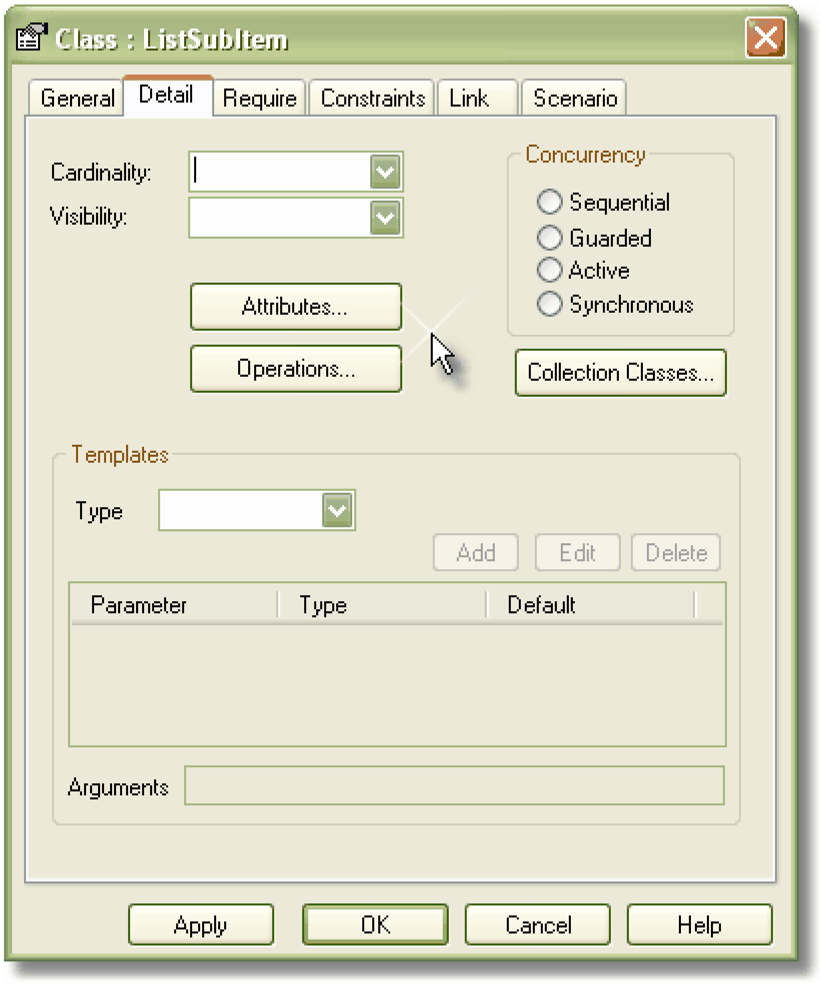
1. To edit the class in Visual Studio.NET select a class from a diagram.
2. Right click on the class to bring up its context menu.
3. Select the menu item *View* / *Edit Source Code*, or alternatively press the *CTRL+E* shortcut key. This will open up the class at the start of the code in Visual Studio.NET ready for editing.


Add-In
Class Properties... Alt+Enter
Manage Tagged Values...
Lock Element...
Insert Embedded Element
Embedded Elements...
Element Features
Set Element Parent... Ctrl+I
Set as Composite Element
Link Class to Association
Configure and Override Attribute initializers... Ctrl+Shift+R
Set Multiplicity...
Attach
Generate Code (forward engineer)... Ctrl+G
Synchronize Model (reverse engineer)... Ctrl+R
View/Edit Source Code... Ctrl+E
Selectable
Appearance
Z-Order
Insert Related Elements
Locate in Project Browser Alt+G
Project Information
Add to favorites
Delete 'ListSubItem' Ctrl+D

3.2.4 Edit Class, Attributes and Operations

It is possible to edit the attributes and operations of a class from within Enterprise Architect.

1. Open the *Detail* tab from the *Class Properties* Dialog. To access the *Class Properties* follow the steps outlined in the Edit Class topic. This will bring up the dialog as shown below, this dialog allows the user to edit the attributes or the operations of the class, by selecting either the *Attributes* button or the *Operations* button.



2. To set the Attributes of a class select the *Attribute* button, for more information on the options related to editing attributes go to the Edit Attributes topic.

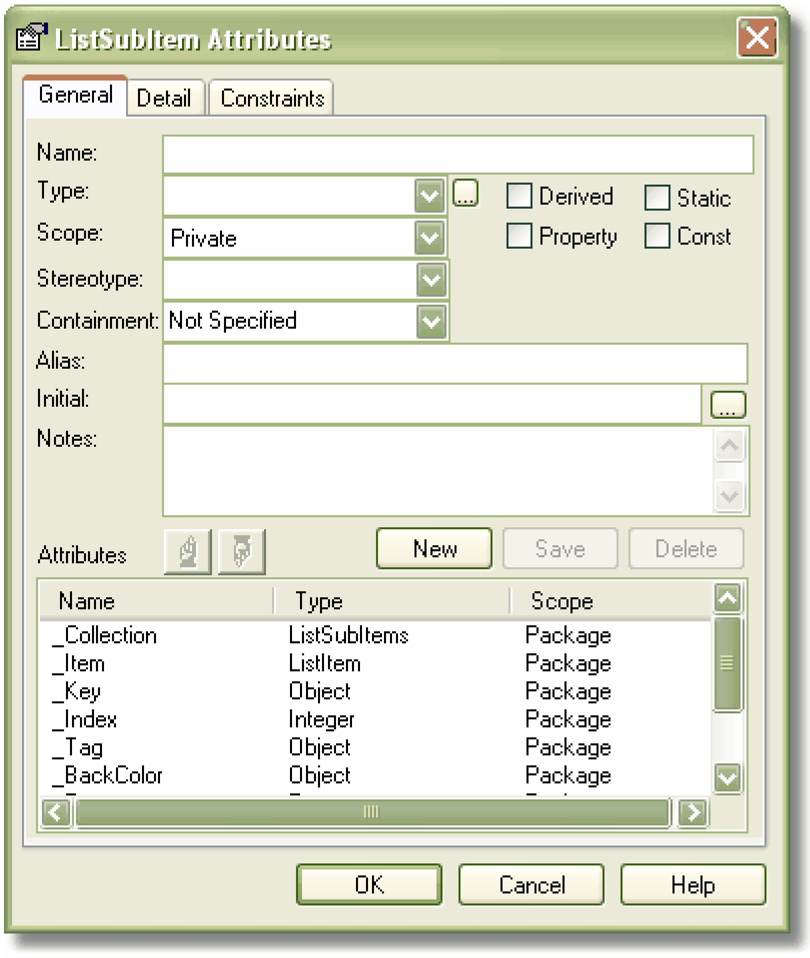
3. To set the Operations of a class select the *Operations* button, for more information on the options related to editing operations go to the Edit Operations topic.

3.2.4.1 Edit Attributes

Attributes are features of a class that represents the properties or internal data elements of that class. For a Customer class, CustomerName and CustomerAddress may be attributes. Attributes have several important characteristics, such as type, scope (visibility), static, derived and notes.

To access the attributes of a class in Enterprise architect use the following instructions:

1. Open the *Detail* tab from the *Class Properties* Dialog. To access the *Class Properties* follow the steps outlined in the Edit Class topic, then click on the *Attributes* button. This will bring up the Attributes dialog as shown below. Alternatively click on the class in the class diagram and use th *F9* key to bring up the Attributes dialog.



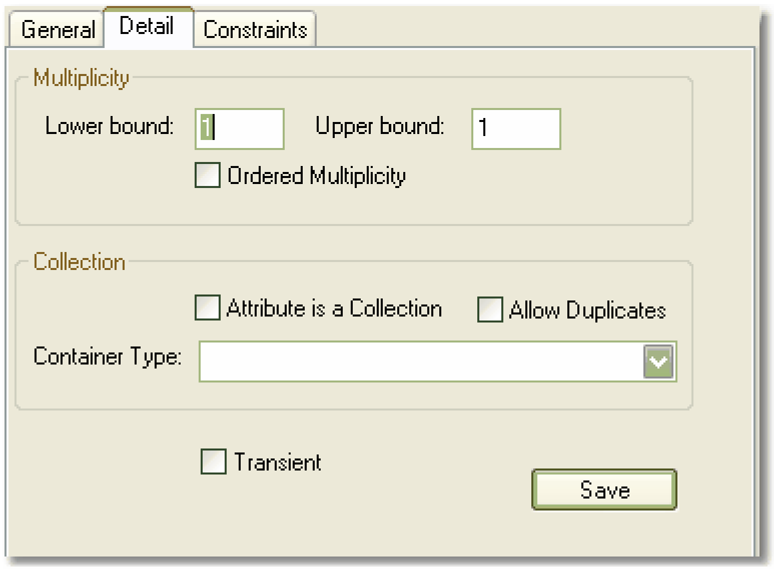
Name	Type	Scope
_Collection	ListSubItems	Package
_Item	ListItem	Package
_Key	Object	Package
_Index	Integer	Package
_Tag	Object	Package
_BackColor	Object	Package

This dialog box offers a range of options, from the General tab the following options are available:

Control	Description
Name	Attribute name
Type	Data type of attribute - select from the drop down list
Build button	Opens the *Select Attribute Type* dialog
Scope	Public/Protected/Private/Package
Stereotype	Optional Stereotype of the attribute
Containment	Containment type(by reference/value)
Derived	Indicates attribute is a calculated value
Static	Attribute is a static member
Property	Select automatic property creation
Const	Attribute is a constant
Alias	An optional alias for the attribute
Initial	An optional initial value
Notes	Free text notes
Attribute List	List of defined attributes. Select an attribute to make it current
Up/Down buttons	Use to change the order of attributes in the list
New	Create new attribute
Save	Save new attribute, or save modified details for existing attribute
Delete	Delete currently selected attribute

3.2.4.1.1 Attribute Details

The *Detail* tab of the *Attributes* dialog has some additional details relating to collections.

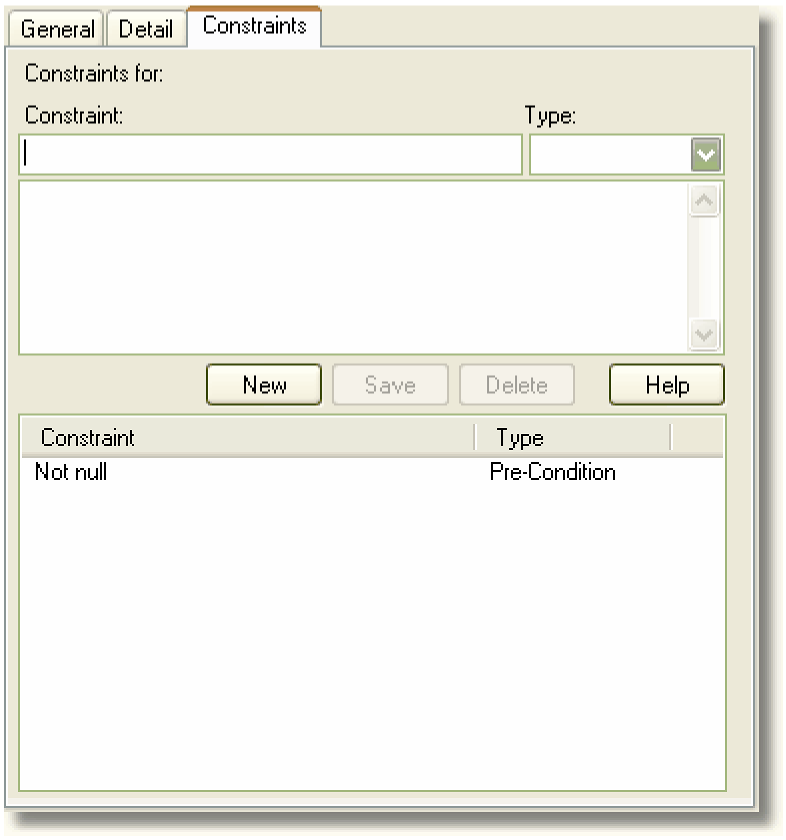


Lower bound: 1
Upper bound: 1
Ordered Multiplicity
Attribute is a Collection
Allow Duplicates
Container Type:
Transient

Control	Description
Lower Bound	A lower limit
Upper Bound	An upper limit to the number of elements in the collection
Ordered Multiplicity	Set if the collection is ordered
Attribute is a Collection	Check if the attribute is a collection
Allow Duplicates	Set if duplicates are allowed
Container Type	The container type
Save	Save changes

3.2.4.1.2 Attribute Constraints

Attributes may also have *Constraints* associated with them. Typically this will indicate such things as maximum value, minimum value, length of field etc.



Constraint	Type
Not null	Pre-Condition

Control	**Description**
Constraint	Constraint name
Type	Constraint type
Notes	Constraint details
Constraint list	A list of constraints already defined
New	Create new attribute constraint
Save	Save new constraint details
Delete	Delete currently selected constraint
Help	Opens this help document

3.2.4.1.3 Attribute Tagged Values

An attribute may have *Tagged Values* defined for it. Tagged values are a convenient means of extending the properties a model element supports. This in turn can be used by code generators and other utilities to transform UML models into other forms.

Tip: *Tagged values are supported for Attributes, Operations, Objects and Connectors.*

An attribute may have *Tagged Values* defined for it. Tagged values are a convenient means of extending the properties a model element supports. This in turn can be used by code generators and other utilities to transform UML models into other forms.

Tip: *Tagged values are supported for Attributes, Operations, Objects and Connectors.*

Add a Tagged Value

To add a tagged value for an attribute,use the following the steps :

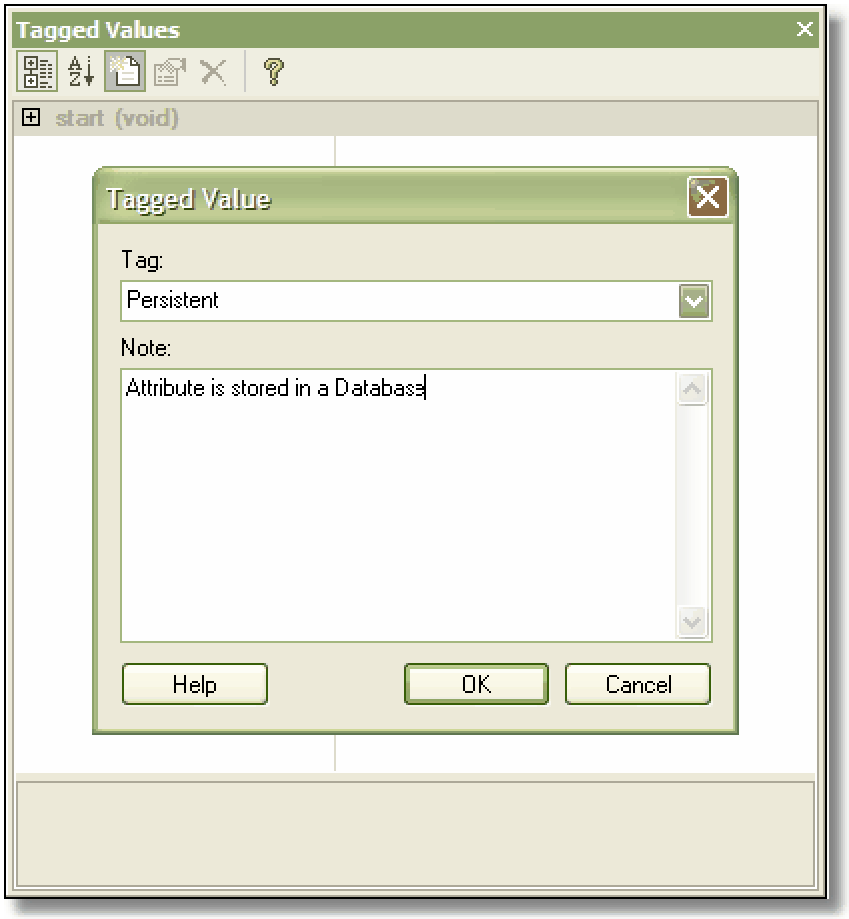
1. Ensure the *Tagged Values* window is open by selecting *View* | *Other Windows* | *Tagged Values* (or press the *Ctrl* + *Shift* + *6* hotkey combination).

2. Select the attribute by double clicking on the attribute in a diagram or on the attribute in the Project View.

3. The *Tagged Values* window will now have the attribute selected, press either the *New Tags* button or the *Ctrl + N* hotkey combination.

4. Define the tag in the *Tag* field (or select a custom defined tag from the drop down list), then add notes as appropriate to the *Note* text entry field.

5. Then press *OK* the button to confirm the operation.



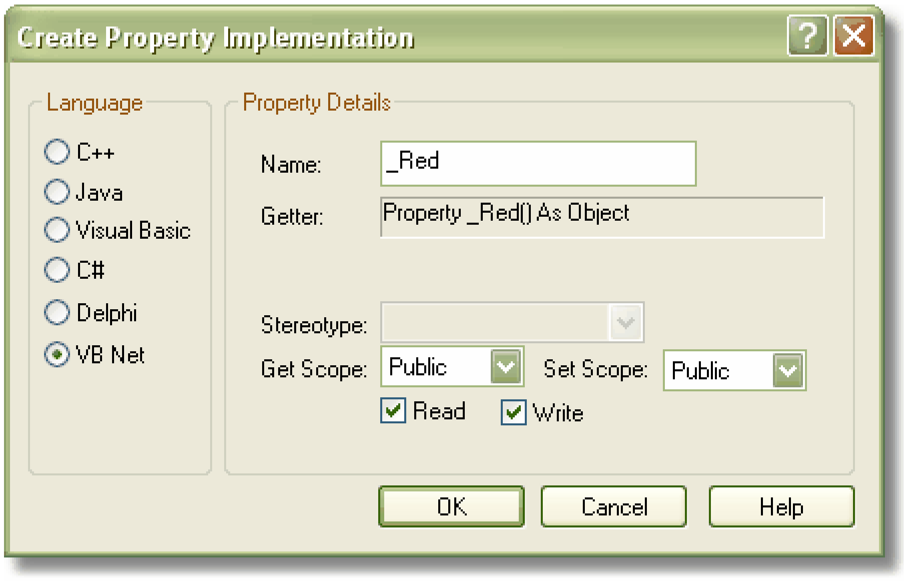
Tag: Persistent

Note: Attribute is stored in a Database

Tip: *Use the Reference/Property Types dialog to add common Tag types to the model. These will appear in the drop down list.*

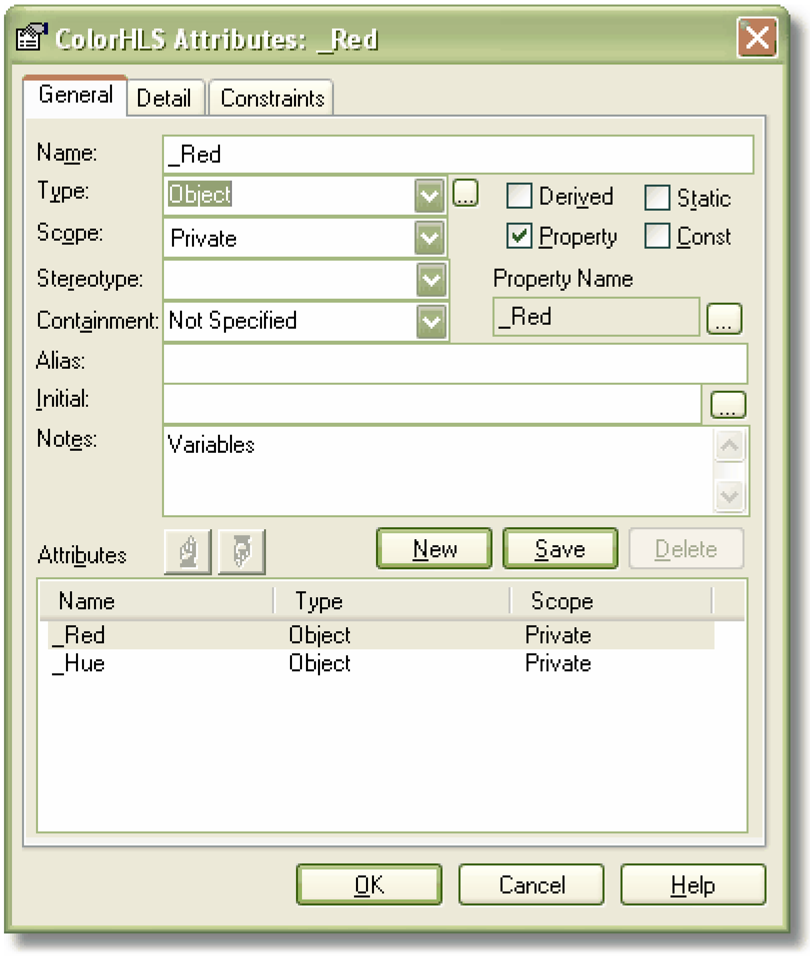
3.2.4.1.4 Creating Properties

Enterprise Architect has some capabilities for automatically creating properties in various languages. Property creation is controlled from the *General* tab of the *Attribute* dialog. Select the *Property* option to activate this feature.



Name: _Red
Getter: Property _Red() As Object
Stereotype:
Get Scope: Public
Set Scope: Public
[✓ Read] [✓ Write]
Language: [✓ VB Net]

This opens the *Create Property Implementation* dialog (shown below). By default the class language is picked up as the default, however you may change this and generate for any language. Each language has slightly different syntax and generates slightly different results. For example, C++ generates get and set functions, C# and VB.Net create property functions.

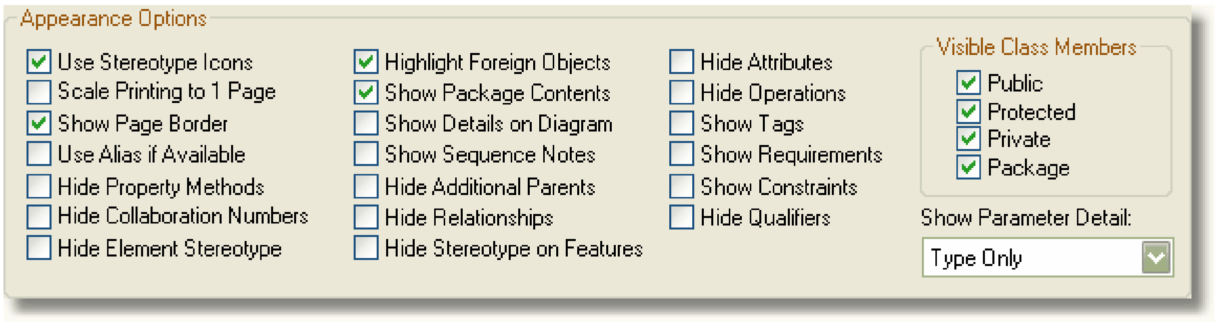


Name: _Red
Type: Object
Derived: ☐
Static: ☐
Property: ☑
Const: ☐
Scope: Private
Stereotype:
Property Name: _Red
Containment: Not Specified
Alias:
Initial:
Notes: Variables

Attributes

Name	Type	Scope
_Red	Object	Private
_Hue	Object	Private

Enter your required details and press *OK*. EA will generate the required operations and/or properties to comply with the selected language. Note that get and set functions will be stereotypes with <<property get>> <<property set>> etc. making it easy to recognize property functions. You may also hide these specialized functions by checking the *Hide Properties* check box in the *Diagram Properties* dialog for a specific diagram. This makes it easier to view a class, uncluttered by many get and set methods.


Appearance Options

- ☑ Use Stereotype Icons
- ☐ Scale Printing to 1 Page
- ☑ Show Page Border
- ☐ Use Alias if Available
- ☐ Hide Property Methods
- ☐ Hide Collaboration Numbers
- ☐ Hide Element Stereotype
- ☑ Highlight Foreign Objects
- ☑ Show Package Contents
- ☐ Show Details on Diagram
- ☐ Show Sequence Notes
- ☐ Hide Additional Parents
- ☐ Hide Relationships
- ☐ Hide Stereotype on Features
- ☐ Hide Attributes
- ☐ Hide Operations
- ☐ Show Tags
- ☐ Show Requirements
- ☐ Show Constraints
- ☐ Hide Qualifiers

Visible Class Members

- ☑ Public
- ☑ Protected
- ☑ Private
- ☑ Package

Show Parameter Detail: Type Only

3.2.4.2 Edit Operations

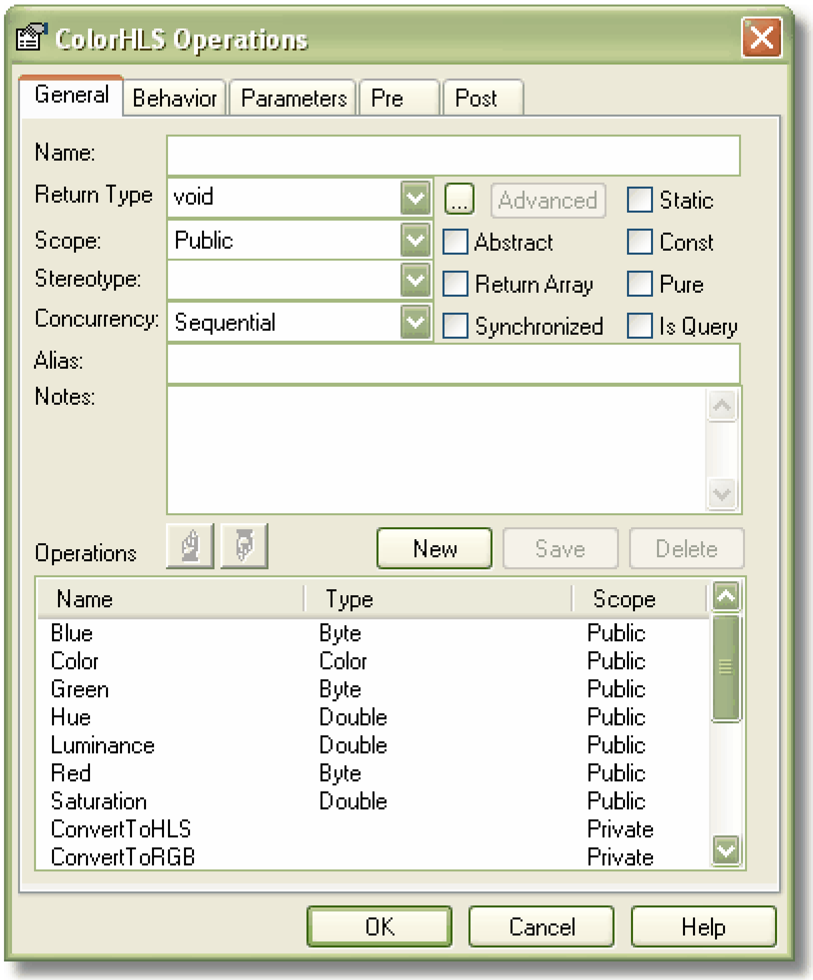
Operations are features of a class that represents the behavior or services that the class supports. For a Customer class, UpdateCustomerName and GetCustomerAddress may be operations. Operations have several important characteristics, such as type, scope (visibility), static, abstract and notes.

To access the operations of a class in Enterprise architect use the following instructions:

1. Open the *Detail* tab from the *Class Properties* Dialog.

2. To access the *Class Properties* follow the steps outlined in the Edit Class topic, then click on the *Operations* button.

3. This will bring up the Operations dialog as shown below. Alternatively click on the Class in the Class diagram and use the *F10* key to bring up the operations dialog.



Name:
Return Type: void
Scope: Public
Stereotype:
Concurrency: Sequential
Alias:
Notes:

[ ] Static
[ ] Abstract
[ ] Const
[ ] Return Array
[ ] Pure
[ ] Synchronized
[ ] Is Query

Operations

Name	Type	Scope
Blue	Byte	Public
Color	Color	Public
Green	Byte	Public
Hue	Double	Public
Luminance	Double	Public
Red	Byte	Public
Saturation	Double	Public
ConvertToHLS		Private
ConvertToRGB		Private

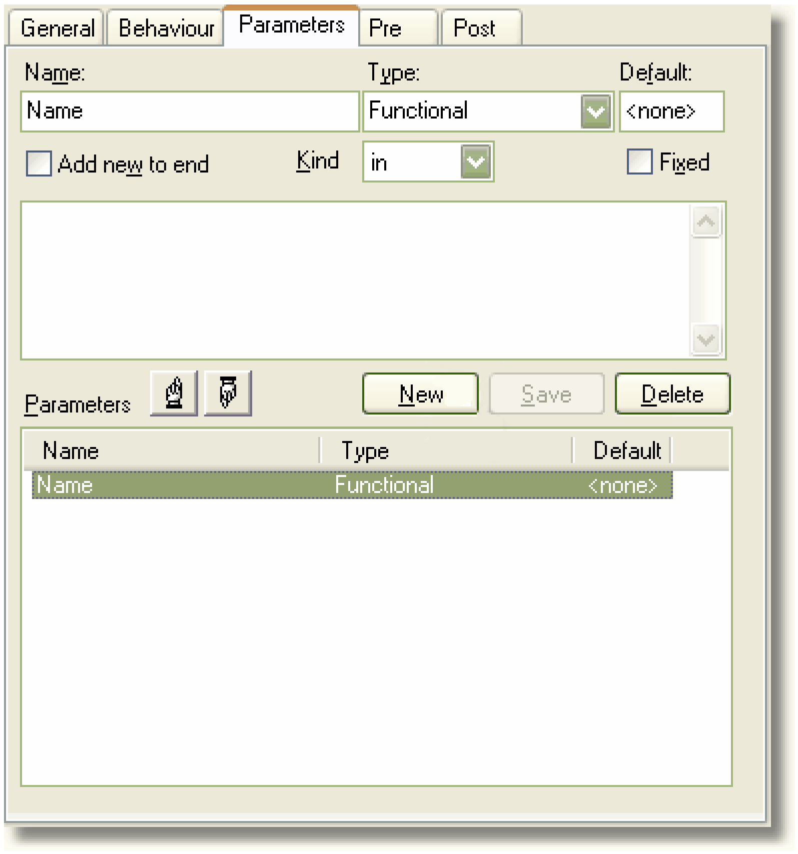
This dialog box offers a range of options, from the *General* tab the following options are available:

Control	Description
Name	Operation name
Type	Data type returned by operation
Build button	Opens the *Set Element Classifier* dialog
Scope	Public/Protected/Private/Package
Stereotype	An optional stereotype for this operation
Concurrency	Concurrency of operation
Virtual/Abstract	If the operation's language is set to C++, this option maps to the C++ Virtual keyword. Otherwise this option is Abstract, pertaining to an abstract function.
Return Array	The return value is an array
Synchronized	A code engineering flag which relates to multithreading in Java
Static	Operation is a static member
Const	Operation is a constant
Pure	Relates to C++ pure virtual syntax - eg. virtual void myFunction( ) = 0;
IsQuery	Operation is a database query
Alias	An optional alias for the operation
Notes	Free text notes
Operation List	List of defined operations
Up/Down Buttons	Use to change the order of operations in the list
New	Create new operation
Save	Save new operation, or save modified details for existing operation
Delete	Delete currently selected operation

3.2.4.2.1 Operation Parameters

The *Parameters* tab in the *Operations* dialog lets you define the parameters that an operation will have. The parameter list will be reproduced in code in the order they appear in the parameters list - so use the up and down arrows to move parameters into their required positions. Additionally, you may select the *Add new to end* option to force new parameters to appear at the end of the list instead of the head.

Tip: *Set the amount of parameter detail to display in a specific diagram using the Show Parameter Detail drop down list on the Diagram Properties dialog. The setting applies only to the current diagram. The default is to show the type only.*



Name: Name
Type: Functional
Default: <none>
[ ] Add new to end
Kind: in
[ ] Fixed

Parameters

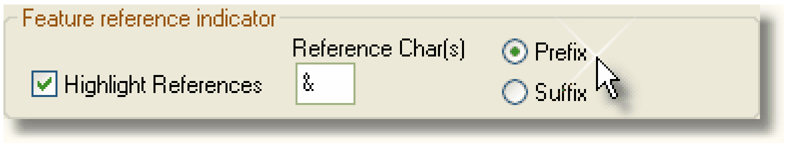
Name	Type	Default
Name	Functional	<none>

Control	Description
Name	Parameter name
Type	Data type of parameter
Default	Optional default value
Kind	Indicates the way a parameter is passed to a function • In = By Value • InOut = By Reference • Out is passed by Reference - but only the return value is significant
Fixed	The parameter is 'const' - even if passed by reference
Add new to end	Place new parameters at the end of the list instead of the start
Notes	Free text

3.2.4.2.2 Operation Parameters by Reference

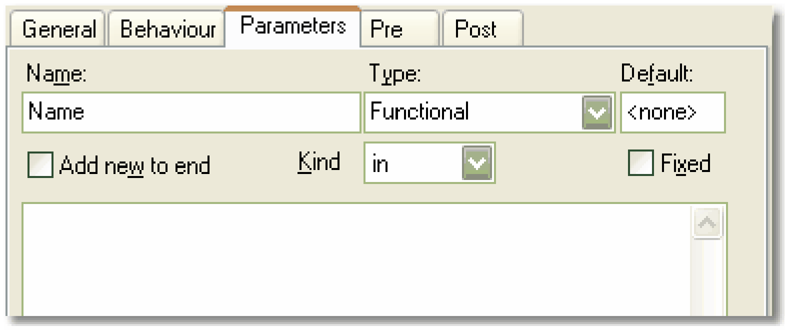
You can elect to highlight parameters declared as type 'inout' with an additional user-defined prefix or suffix. In the *Objects* section of the *Local Options* dialog (*Tools* | *Options*), there is a segment which allows you to set whether references are highlighted or not.

If you select the *Highlight References* option, you can also indicate whether a prefix or suffix will be used, and the actual character to use. In the example below, the '&' character as a prefix has been selected.



Feature reference indicator: Highlight References [✓]; Reference Char(s): &; [✓ Prefix] Suffix

When you declare a parameter of type 'inout', it is assumed you are passing the parameter by reference, rather than by value. If you have elected to highlight references, then this will be displayed in the diagram view.

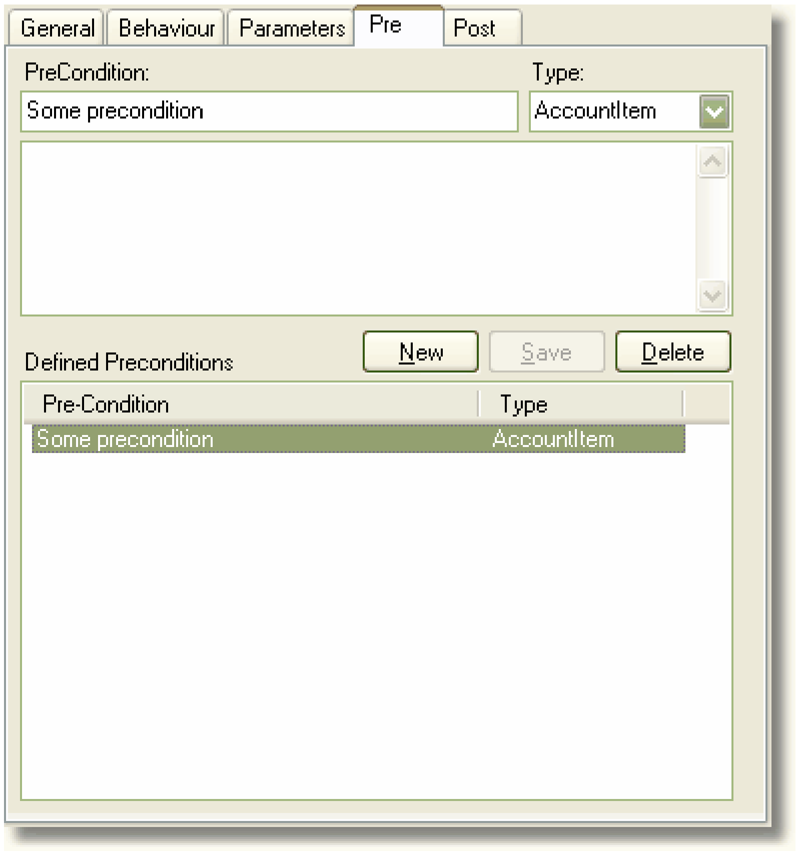


Name: Name
Type: Functional
Default: <none>
Kind: in

3.2.4.2.3 Operation Constraints

Operations may have pre- and post- conditions defined. For each type, give the condition a name, a type and enter notes.

Constraints define the contractual behavior of an operation - what must be true before they are called and what is true after. In this respect they are related to the state model of a class and can also relate to the guard conditions that apply to a transition.



PreCondition: Some precondition

Type: AccountItem

Defined Preconditions

Pre-Condition	Type
Some precondition	AccountItem

3.2.4.2.4 Operation Tagged Values

Operations may have tagged values associated with them. Tagged values offer a convenient extension mechanism for UML elements - so you can define any tags you like - and then assign them values using this form.

Tagged values will be written to the XMI output, and may be input to other third party tools for code generation or other activity.

Tip: *Tagged values are supported for Attributes, Operations, Objects and Connectors.*

Add a Tagged Value

To add a tagged value for an operation,use the following the steps :

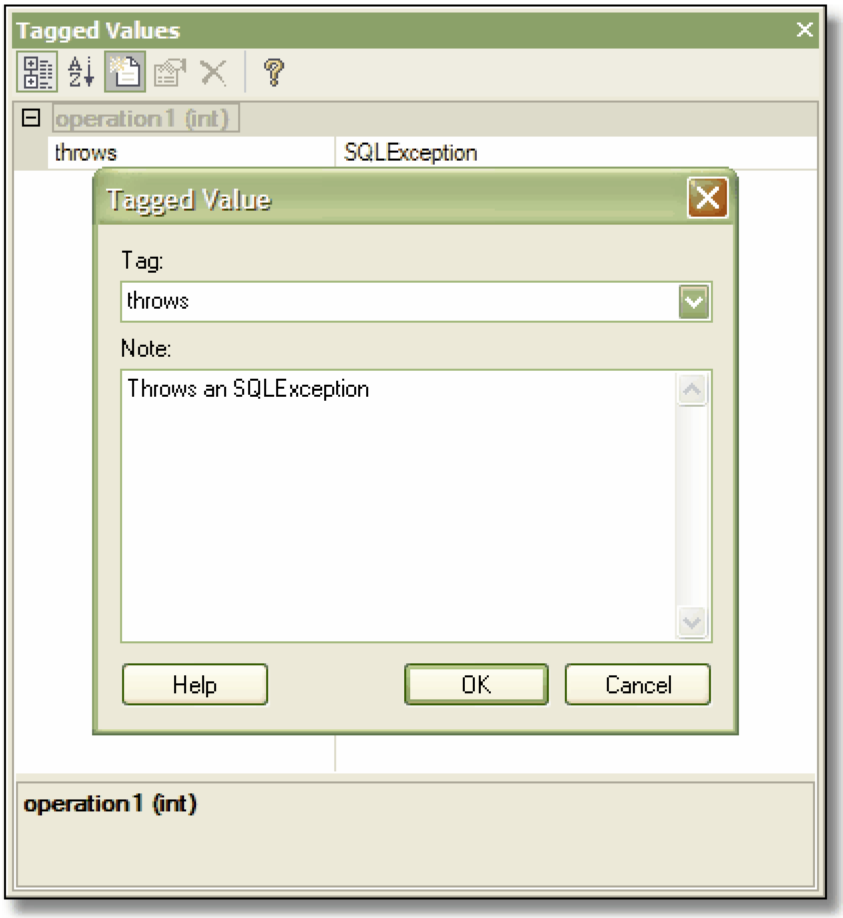
1. Ensure the *Tagged Values* window is open by selecting *View* | *Other Windows* | *Tagged Values* (or press the *Ctrl* + *Shift* + *6* hotkey combination).

2. Select the operation by double clicking on the operation in a diagram or on the operation in the Project View.

3. The *Tagged Values* window will now have the operation selected, press either the *New Tags* button or the *Ctrl + N* hotkey combination.

4. Define the tag in the *Tag* field (or select a custom defined tag from the drop down list), then add notes as appropriate to the *Note* text entry field.

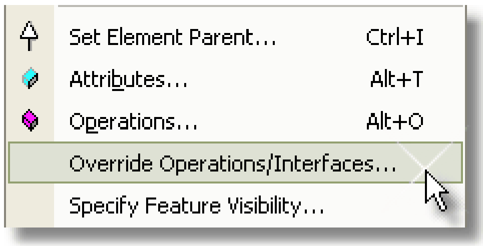
5. Then press *OK* the button to confirm the operation.



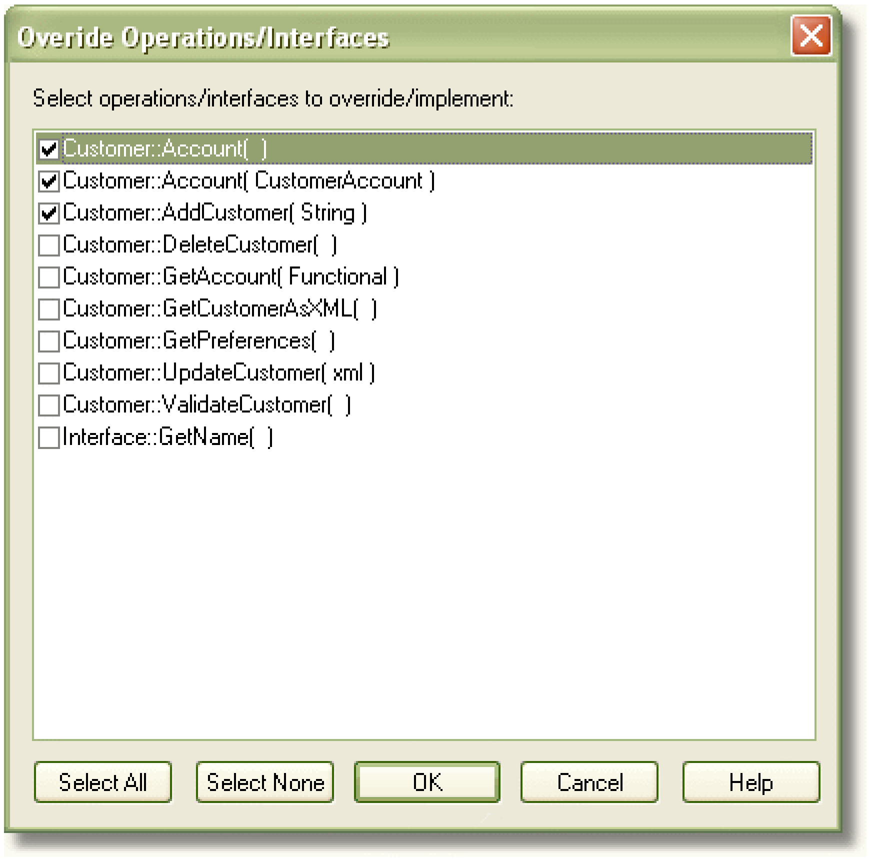
3.2.4.2.5 Override Parent Operations

It is possible in Enterprise Architect to automatically override methods from parent classes and from realized interfaces.

Select a class that has a parent or realized interface and choose *Override Implementation* from the *Element* menu.


Set Element Parent... Ctrl+I
Attributes... Alt+T
Operations... Alt+O
Override Operations/Interfaces...
Specify Feature Visibility...

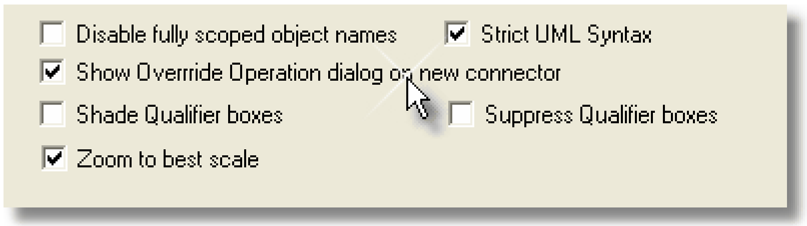
In the *Override Operations* / *Interfaces* dialog check the operations/interfaces that you wish to automatically override and press *OK*. EA will generate the equivalent function definitions in your child class.



Select operations/interfaces to override/implement:

- [✓] Customer::Account( )
- [✓] Customer::Account(CustomerAccount)
- [✓] Customer::AddCustomer(String)
- [ ] Customer::DeleteCustomer( )
- [ ] Customer::GetAccount(Functional)
- [ ] Customer::GetCustomerAsXML( )
- [ ] Customer::GetPreferences( )
- [ ] Customer::UpdateCustomer(xml)
- [ ] Customer::ValidateCustomer( )
- [ ] Interface::GetName( )

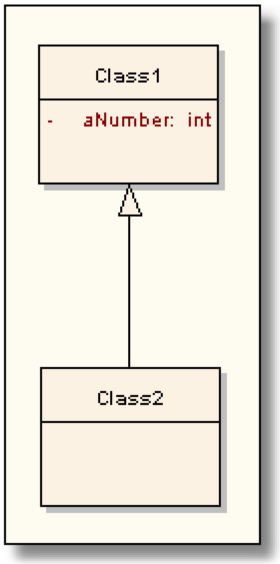
It is possible to configure EA to display this dialog each time you add a Generalization or Realization link between classes and their possible operations/interfaces to override/implement. Do this from the *Diagram* section of the *Local Options* dialog (*Tools* | *Options*).


- [ ] Disable fully scoped object names
- [✓] Strict UML Syntax
- [✓] Show Override Operation dialog on new connector
- [ ] Shade Qualifier boxes
- [ ] Suppress Qualifier boxes
- [✓] Zoom to best scale

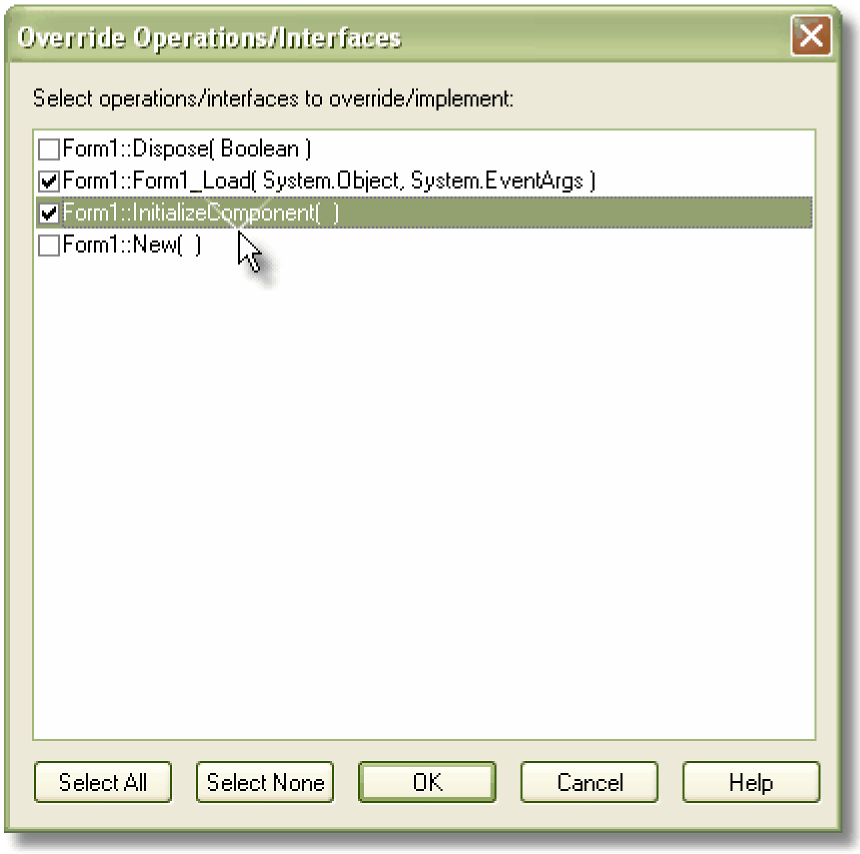
3.2.5 Adding Inheritance to Classes

Adding inheritance between classes in the MDG Link for Visual Studio.NET from Enterprise Architect is a simple procedure, to achieve this follow the steps detailed below.

1. Locate the Classes involved in the operation.
2. Select the *Generalize* connection from the *Toolbox*.
3. Connect the child class to the parent class.



4. Then select the operations/interfaces that you want to override/implement.



Select operations/interfaces to override/implement:

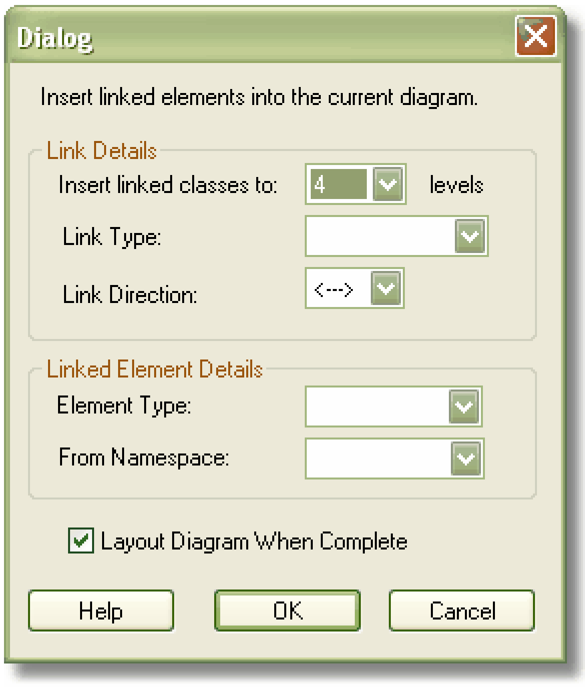
- [ ] Form1::Dispose(Boolean)
- [✓] Form1::Form1_Load(System.Object, System.EventArgs)
- [✓] Form1::InitializeComponent( )
- [ ] Form1::New( )

5. To update the model to the source code follow the steps outlined in the Merge Code with a Model.

3.2.6 Add Class and Find Association Links

One of the powerful options available in the MDG Link for Visual Studio.NET is the ability to add one class to a diagram and to find the relationships between classes that link to the original class. To achieve this follow the steps outlined below.

1. Create a new diagram and find the class in the Project View that you are interested in.
2. Drag this class onto the diagram workspace and paste it as a *Simple Link*.
3. Right click on the class to bring up its context menu, select the *Insert Related Elements.*
4. This will bring up the following Dialog Box.


Dialog

Insert linked elements into the current diagram.

Link Details

Insert linked classes to: 4 levels

Link Type:

Link Direction: <--->

Linked Element Details

Element Type:

From Namespace:

[✓] Layout Diagram When Complete

This will bring the related classes into the diagram, giving the user a picture of the relationships between the original class and other related classes.

For more information relating to the options of this dialog box got to the Insert Related Elements page.

3.3 Code

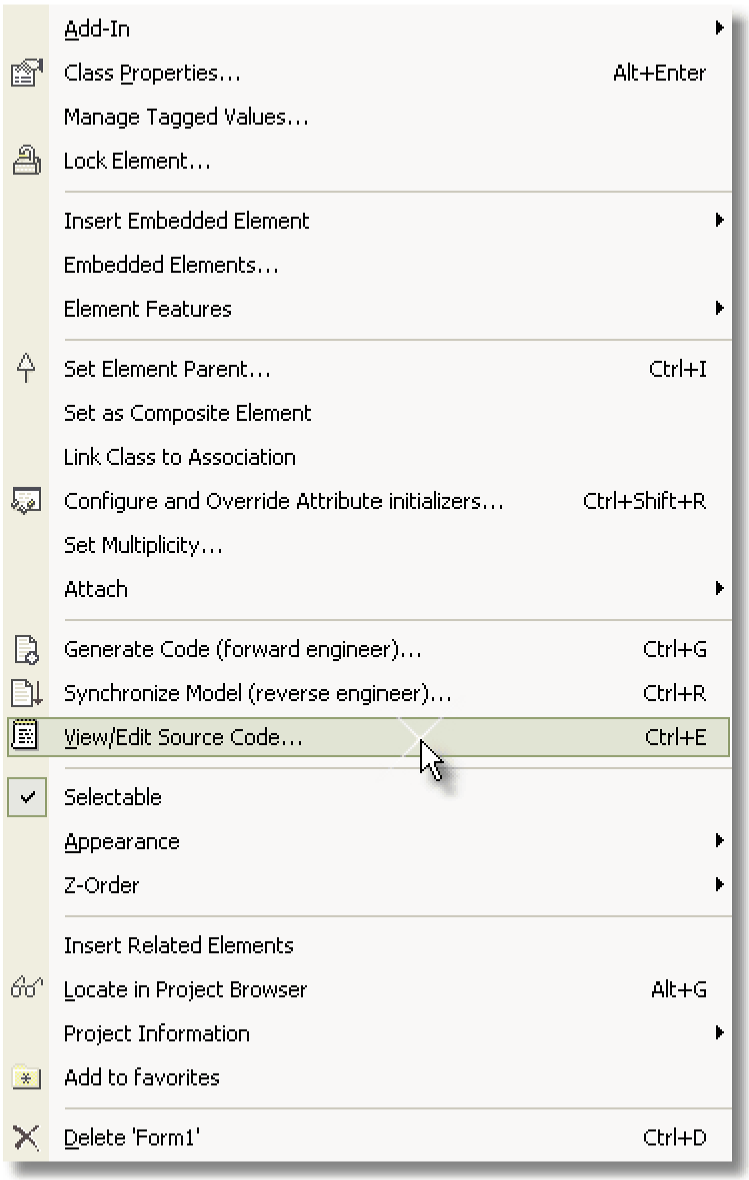
The MDG Link for Visual Studio.NET allows for the flexible creation, editing and UML modeling of class diagrams.

3.3.1 Edit Code

The MDG Link for Visual Studio.NET adds extra functionality to the code generation abilities of Enterprise Architect, in addition to generation of code (forward engineering) and synchronization of code (reverse engineering) the MDG Link for Visual Studio.NET offers the ability to quickly edit the source code in Visual Studio.NET. To achieve this use the following procedure:

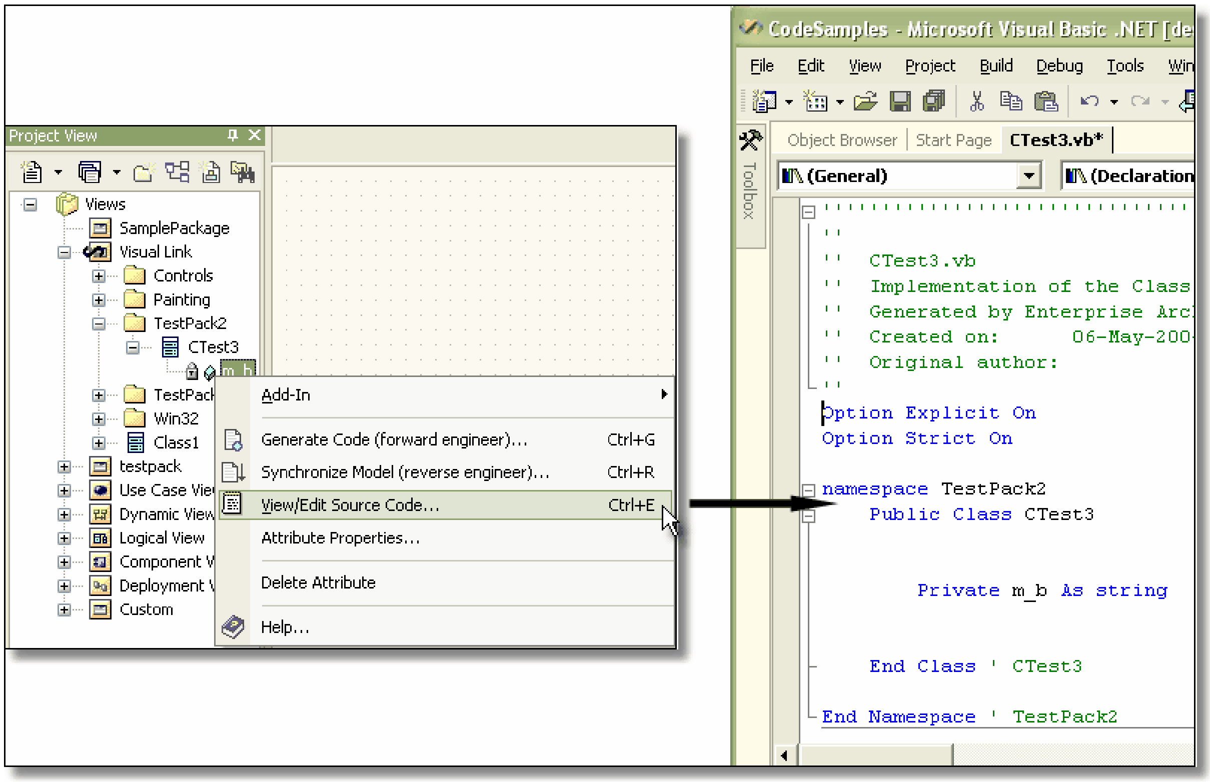
1. Right click on the class that you wish to edit in the diagram view.

2. Press *CTRL+E* when the class is selected to edit the class code. Or, alternatively select the *View / Edit Source Code* menu item and Visual Studio.NET will be opened to allow for the editing of the class's code.


Add-In
Class Properties... Alt+Enter
Manage Tagged Values...
Lock Element...
Insert Embedded Element
Embedded Elements...
Element Features
Set Element Parent... Ctrl+I
Set as Composite Element
Link Class to Association
Configure and Override Attribute initializers... Ctrl+Shift+R
Set Multiplicity...
Attach
Generate Code (forward engineer)... Ctrl+G
Synchronize Model (reverse engineer)... Ctrl+R
View/Edit Source Code... Ctrl+E
Selectable
Appearance
Z-Order
Insert Related Elements
Locate in Project Browser Alt+G
Project Information
Add to favorites
Delete 'Form1' Ctrl+D

3. Editing can also be achieved from the Project View by selecting the item of interest (which may be either a class or a method).

4. Right click on the item to bring up its context menu, then press *CTRL+E* or alternatively press *CTRL+E* when the class is selected to edit the code.



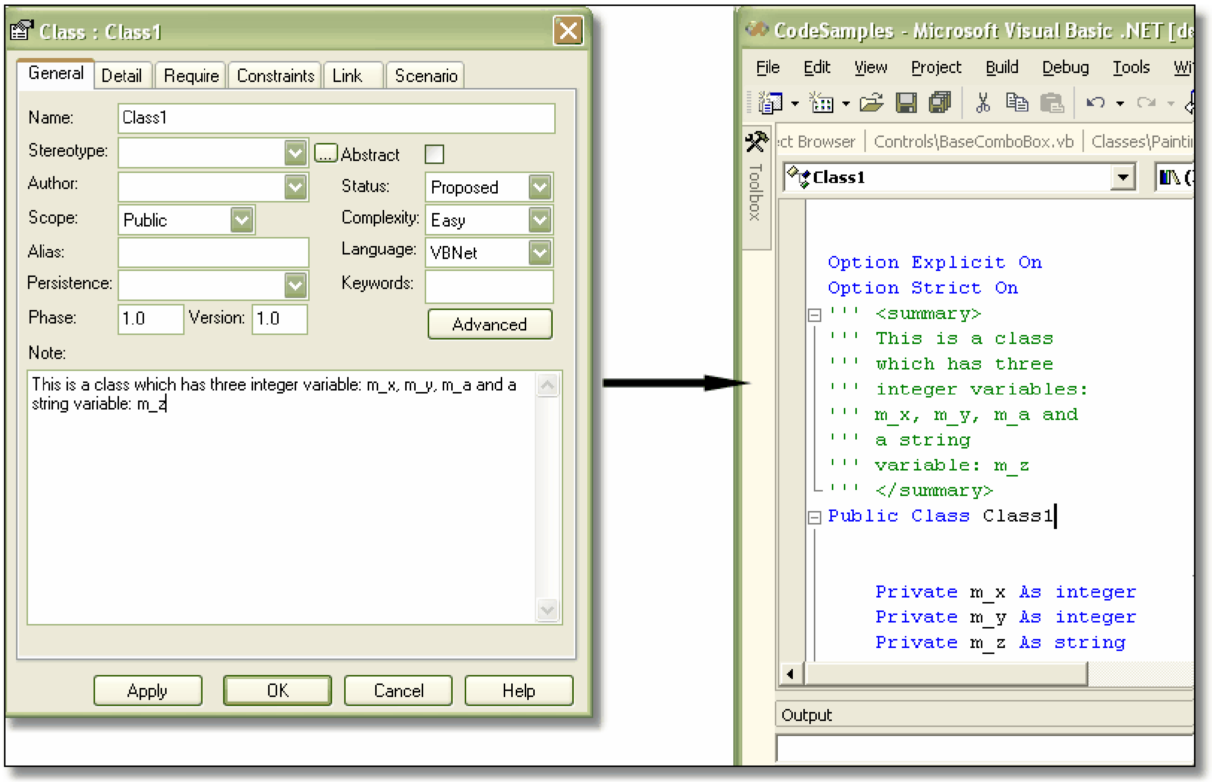
```
''''''''''''''''''''''''''''''''
''
''  CTest3.vb
''  Implementation of the Class
''  Generated by Enterprise Arc
''  Created on:      06-May-200
''  Original author:
''
Option Explicit On
Option Strict On

namespace TestPack2
    Public Class CTest3


        Private m_b As string


    End Class ' CTest3

End Namespace ' TestPack2
```

3.3.2 Adding Code Comments

To comment code from Enterprise Architect use the following procedure:

1. Open the context menu of the class by right clicking on the class and select *Class Properties*.

2. Locate the *Note* text field and enter the comments here.



Name:	Class1		
Stereotype:		Abstract	☐
Author:		Status:	Proposed
Scope:	Public	Complexity:	Easy
Alias:		Language:	VBNet
Persistence:		Keywords:	
Phase:	1.0	Version:	1.0

Note:

This is a class which has three integer variable: m_x, m_y, m_a and a string variable: m_z

```
Option Explicit On
Option Strict On
''' <summary>
''' This is a class
''' which has three
''' integer variables:
''' m_x, m_y, m_a and
''' a string
''' variable: m_z
''' </summary>
Public Class Class1


    Private m_x As integer
    Private m_y As integer
    Private m_z As string
```

OR

3. Open up the Project View and locate the class or method of choice.

4. Right click on the class to bring up its context menu and select *Properties.*

5. Locate the *Note* text field and enter the comments here

6. For methods double click on the method to bring up its operations dialog.

7. Then enter the comments into the *Notes* text field.

Note: The comments will be placed in the target class when the model synchronized or when the code is generated (forward engineered).

3.4 Diagrams

UML Diagrams are collections of project elements laid out and inter-connected as required. Enterprise Architect supports several kinds of UML diagrams as well as custom extensions.

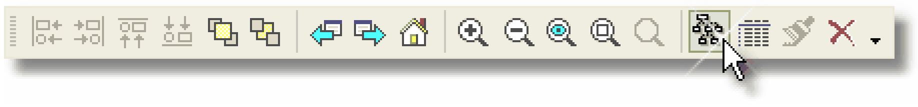
3.4.1 Formatting a Diagram

Formatting a UML class diagram does not change the functionality of your classes, but instead are used to create more readable diagrams. A facility is provided by Enterprise Architect to layout diagrams automatically. This will attempt to create a reasonable tree based structure from the class diagram elements and

relationships in a diagram. Owing to the complexity of many class diagrams, the results may need some manual 'tweaking'.
To format your UML class Diagrams:

1. Select a diagram.

2. From the *Diagram* menu, select *Layout Diagram* -OR- use the *Auto Layout* button on the diagram toolbar .



For more information on the manual options for laying out a UML class diagram go to the Layout a Diagram Page.

3.5 Round Trip Engineering

The MDG Link for Visual Studio.NET round-trip engineering process enables you to model your application in UML 2.0 notation, then generate (forward engineer) the code elements to Visual Studio.NET based on the model, perform modifications and to implement the code as necessary, and then synchronize (reverse engineer) that code back into the Enterprise Architect model.

This allows for consistency between the model and the external code base and may be performed with a merge at the touch of a button. The MDG Link for Visual Studio.NET also allows the user the option of merging the project. The merge options include both forward and reverse engineering as well as the option to both forward and reverse engineer classes at the same time to completely synchronize the code with the model.

3.5.1 Merge Project Dialog Options

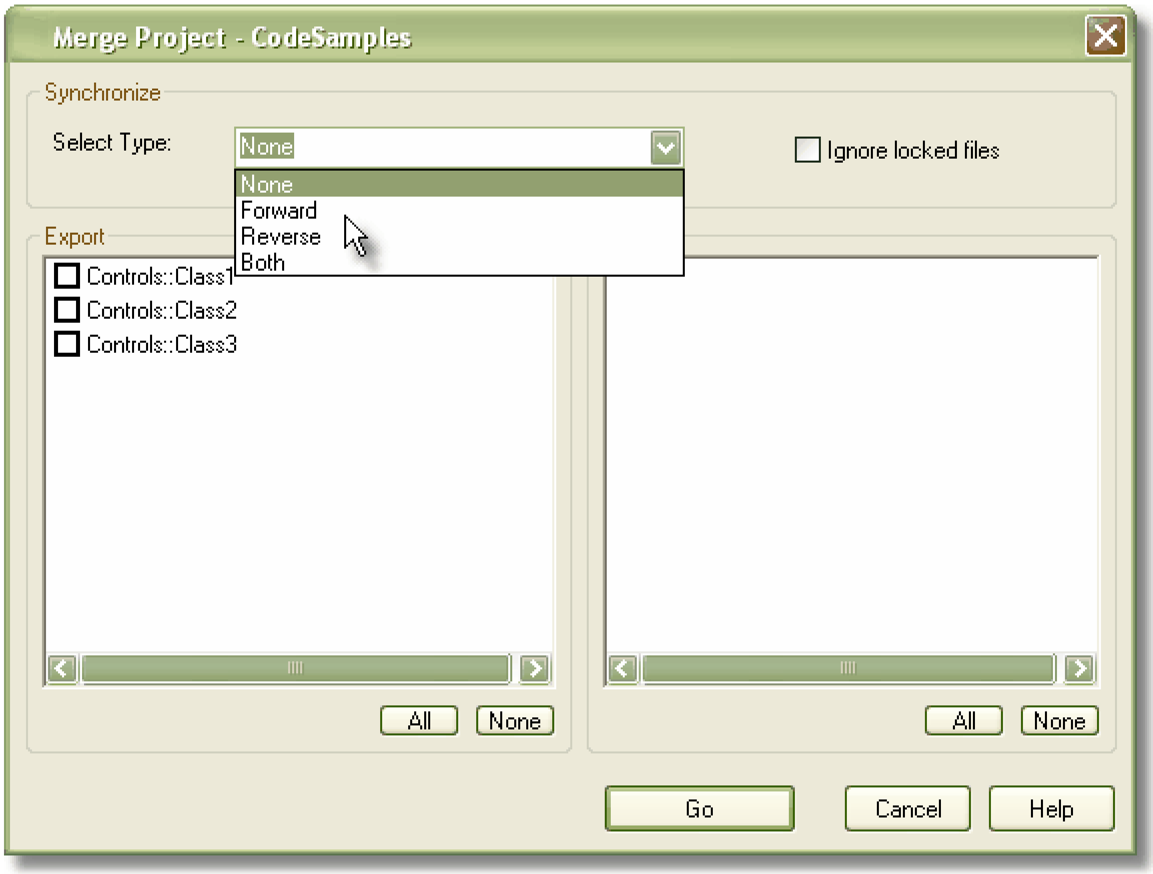
The Visual Studio Connections dialog allows the user to connect and disconnect to a single Visual Studio.Net project

Text Fields

Export	This gives the user the option of selecting classes for export, if a class is not selected it will not be included for the export. The *All* option selects all of the classes in the list. *None* selects none of the classes for export. The list represents classes are present only within the model and are not currently included in the code in Visual Studio.NET
Import	This gives the user the option of selecting classes for import, if a class is not selected it will not be included for the import. The *All* option selects all of the classes in the list. *None* selects none of the classes for import. The list represents classes are present only within the code in Visual Studio.NET and are not currently included in the model.

Buttons

Synchronize	The synchronize drop down menu gives the user four different options for merging. 1. *None*, Selecting this option means that on a project merge, if a class exists in both EA and Visual Studio.NET neither class will be updated. 2. *Forward*, Selecting this option means that when a project merge is performed and a class exists in both EA and Visual Studio.NET then the Visual Studio.NET file will be updated by the merge operation. 3. *Reverse*, Selecting this option means that when a Project merge is performed and a class exists in both EA and Visual Studio.NET then the EA element will be updated by the merge operation. 4. *Both*, Selecting this option means that on a project merge, if a class exists in both EA and Visual Studio.NET then full-round trip code generation will occur with the forward generation procedure executed followed by the reverse engineering execution.
Ignore Locked files	Ignores locked files.
Run	The *Run* button runs the merge.
Cancel	The *Cancel* button cancels the operation.
Help	The *Help* button opens up the help file.



3.5.2 Merge Options

Merging gives the user the opportunity to reverse engineer code from a Visual Studio.NET project or to generate code from an Enterprise Architect model into a Visual Studio.NET project. Merging interrupts the normal processes involved in forward and reverse engineering allowing for a greater level of control than is available in the standard versions of Enterprise Architect. Performing a merge allows the user to:

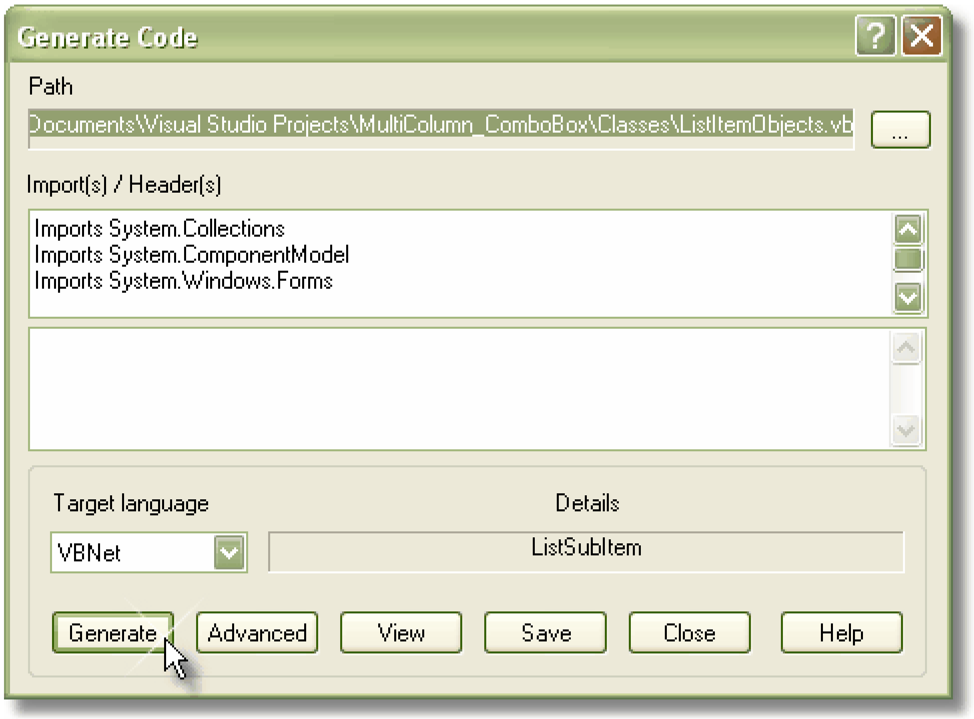
- Choose the filename for new classes created in Enterprise Architect, allowing the user to assign more than one class to the same file name.
- Export selected classes. Allowing the user to perform an export of code only on selected classes.
- Import selected classes. Allowing the user to perform an import of code only on selected classes.
- Synchronize the Model and the source code in one simple step. A synchronized merge reverse engineers the code from Visual Studio.NET into the Enterprise Architect model and then forward engineers the model from Enterprise Architect into Visual Studio.NET in one simple step, allowing the model and the code to accurately represent each other.
- Optionally ignore locked files.

3.5.3 Forward Engineering

Code Generation (forward engineering) generates code from the UML model and places it into Visual Studio.NET. When used to generate a class created purely in Enterprise Architect the code that will be crated in Visual Studio.NET will consist of constructors, destructors as well as get and set methods, this leaves the generation of the business operations of the code up to the user. The Code generation operation can be performed in several ways with the Enterprise Architect MDG Link for Visual Studio.NET. It can be performed by using the Merge operation from the Add-In menu, as well as from the context menu of a class.

3.5.3.1 Forward Engineering from a Class

The *Code Generation* dialog allows you to control how your source code is generated. Normally you will access this dialog from the context menu of a single class or interface. Right click on the class or interface and select *Generate Code* from the context menu. Alternatively, select the class or interface and press *Ctrl+G.*



Path: Documents\Visual Studio Projects\MultiColumn_ComboBox\Classes\ListItemObjects.vb

Import(s) / Header(s):

```
Imports System.Collections
Imports System.ComponentModel
Imports System.Windows.Forms
```

Target language: VBNet

Details: ListSubItem

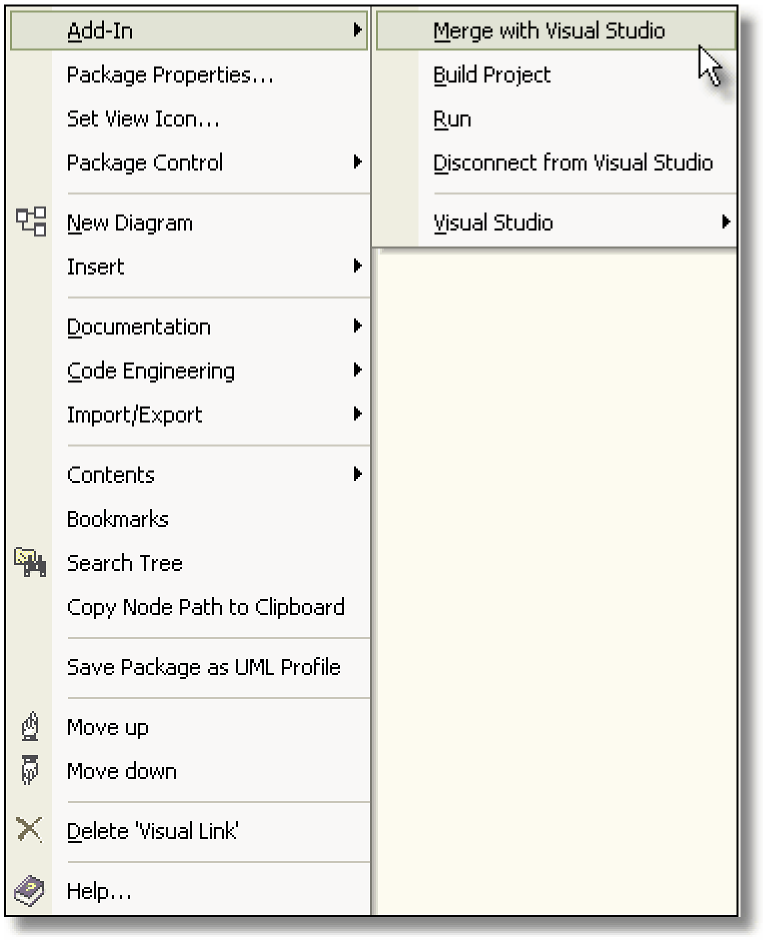
This dialog allows you to set

- The *Path* where the source will be generated. Press the *Browse [...]* button to bring up a file browser dialog, this will default to the path of the current Visual Studio.NET Solution.
- The *Target Language* for generation - select the language to generate - this will then become the permanent option for that class - so change it back if you only want to do one pass in another language.
- *Advanced* settings. Note that the settings you make here only apply to the current class.
- *Import statements #1*. An area for you to enter any special import statements (or #include in Visual C++). For Visual C++ this area is placed in the header file.
- *Import statements #2*. An area to define additional import or include statements (or even macros and #defines in Visual C++).
- *Generate*. Press this to generate your source code - you will be advised of progress as the generation proceeds.
- *View*. Press this top view the generated source code in Visual Studio.NET.

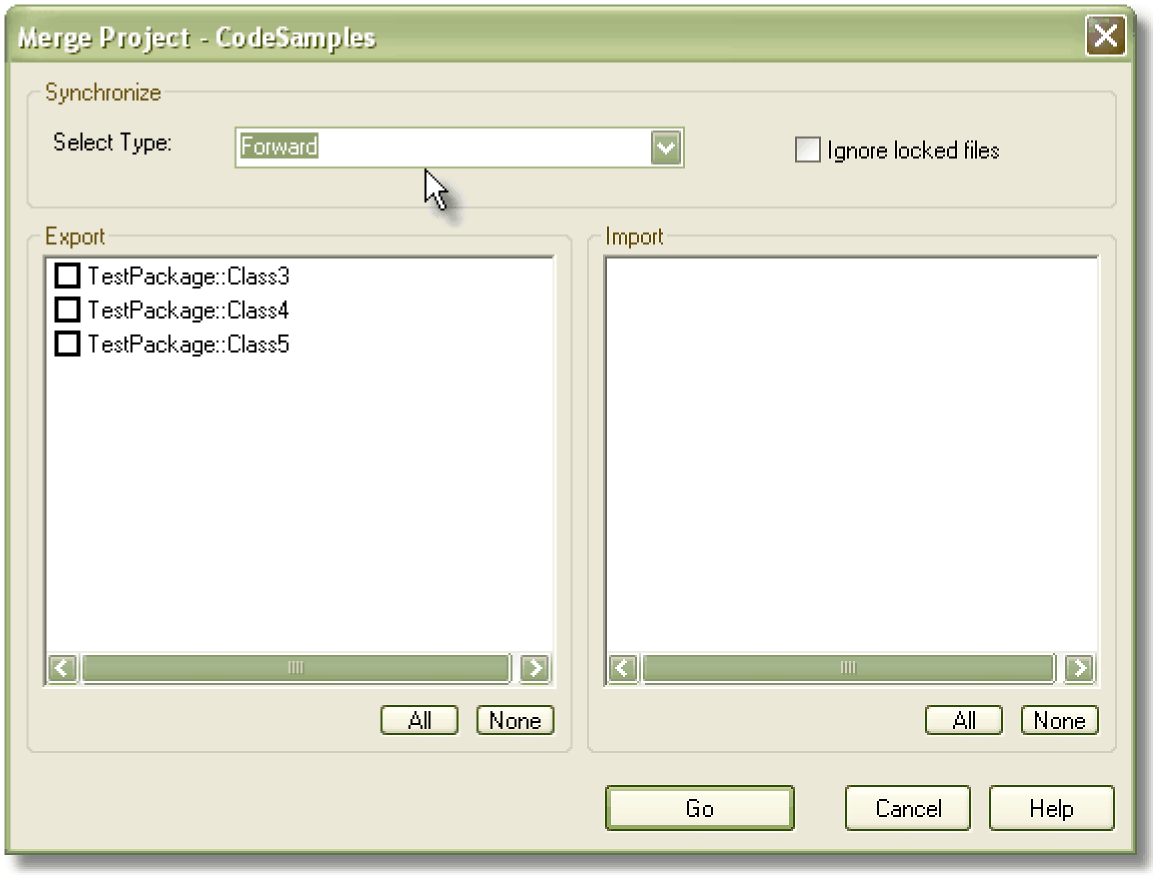
3.5.3.2 Forward Enginnering with a Merge

To generate code with a merge use the following steps once a link has been created to a Visual Studio.NET Project.

1. Select *Add-In* | *Merge with Visual Studio*.



2. This will bring up the following dialog:



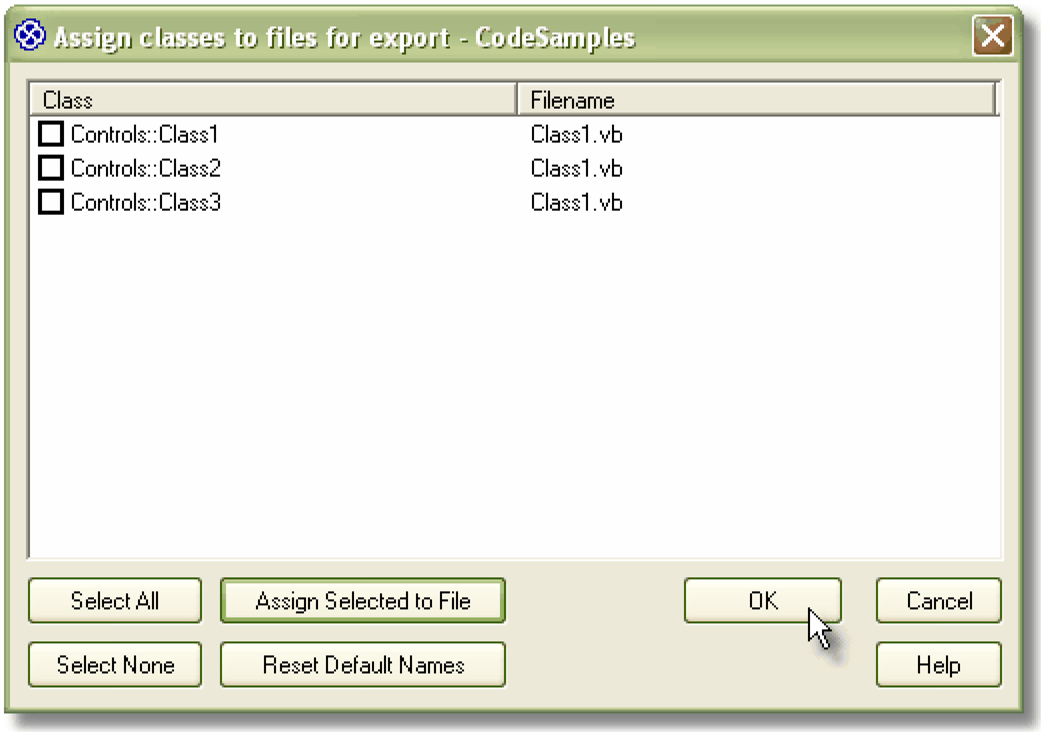
3. In the Synchronize section of the dialog select forward to update classes contained in the code from corresponding elements contained in the model.

4. The items in the Export section apply to element that currently exist in the model but do not exist in the code, select the appropriate classes to be included in for export into the code. Select the appropriate classes by using the checkbox or press *All* to select all of the classes.

5. Press the *Go* button to forward engineer the code. If the forward engineer includes new classes, the user will be prompted to assign a filename for the new classes. For more information relating to assigning new classes go to the Performing a Merge: Export New Class topic.

3.5.3.3 Performing a Merge: Export New Class

When a new class is created in Enterprise Architect and a Merge is performed, the user is given the option to assign the filename for the classes to allow multiple classes to be assigned to the same filename.

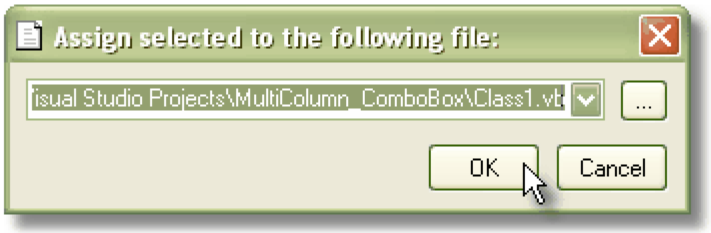


Class	Filename
☐ Controls::Class1	Class1.vb
☐ Controls::Class2	Class1.vb
☐ Controls::Class3	Class1.vb

To assign a filename to a class use the following instructions:

1. Tick the check boxes next to the classes that you wish to assign the filename (to select of the available classes press *Select All*, to deselect all of the selected classes press *Select None*).

2. To assign the file name press the *Assign Selected to File* button, if you wish to cancel the assigned filename and return to the default filename press the *Reset Default Names* button. .



isual Studio Projects\MultiColumn_ComboBox\Class1.vb

3. This will prompt the user for a file path for the class to be saved.

4. Click on the *OK* button to proceed.

3.5.3.4 Assign Classes to Files for Export

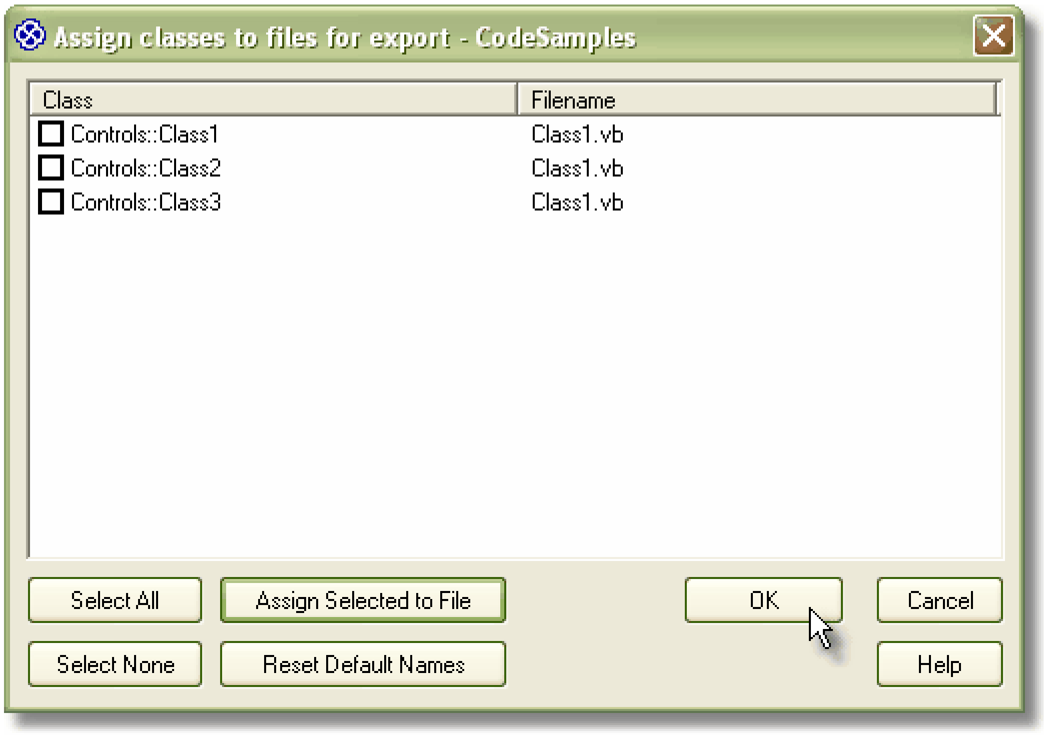
The Select the files in which the new classes are to export to dialog allows the user select the class/s to add into file/s.

Text Fields

Class	This is the name/s of the new classes that the user has the opportunity to export into Visual Studio.NET.
Filename	Filename is the destination of the selected class/s. It is possible to assign more than one class to a filename.

Buttons

Select All	Selects all of the new class files for export.
Select None	Deselects all of the classes.
Assign to Selected to File	Assigns the selected file to a specified destination, this will bring up a new dialog, which will prompt the user for a file destination location.
Reset Default Names	Assigns a default destination and class name for the classes that are to be exported.
OK	Confirms and executes the export of classes.
Cancel	Cancels the export of the new classes.
Help	Opens up the help contents for this operation.



Class	Filename
Controls::Class1	Class1.vb
Controls::Class2	Class1.vb
Controls::Class3	Class1.vb

3.5.4 Reverse Engineering

Synchronization of a model (reverse engineering) updates the UML model from the Visual Studio.NET source code. This action can be used to allow the user to reverse engineer a legacy system and to examine the architecture of the existing code. Synchronizing the model can be performed in several ways with the

Enterprise Architect / MDG Link for Visual Studio.NET. It can be performed in Enterprise Architect from a Merge operation, or from the context menu of a class.

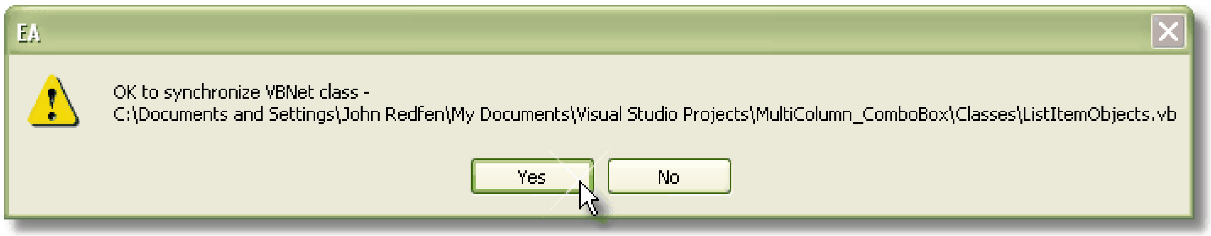
3.5.4.1 Reverse Engineering from a Visual Studio.NET source class

To import source code (reverse engineer) you will usually do the following:

1. In the Project View, select (or add) a diagram into which the classes will be imported.
2. Right click on the diagram background to open the context menu. Select *Synchronize the Model (reverse engineer)*.

   -OR-

   Left click on the diagram background to select the diagram and press *CTRL*+*R*.
3. A message will be displayed to confirm the synchronization of the model and the code select *Yes* to continue or *No* to quit the operation.



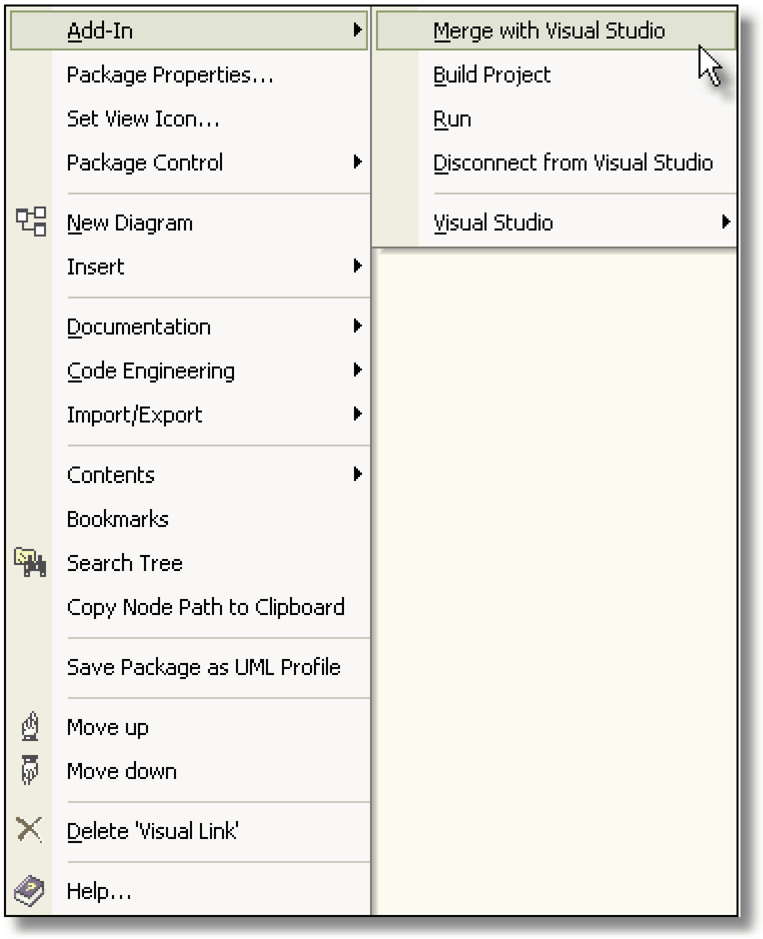
OK to synchronize VBNet class -
C:\Documents and Settings\John Redfen\My Documents\Visual Studio Projects\MultiColumn_ComboBox\Classes\ListItemObjects.vb

As the import proceeds, EA will provide progress information. When all files are imported, EA will make a second pass to resolve and associations and inheritance relationships between the imported classes.

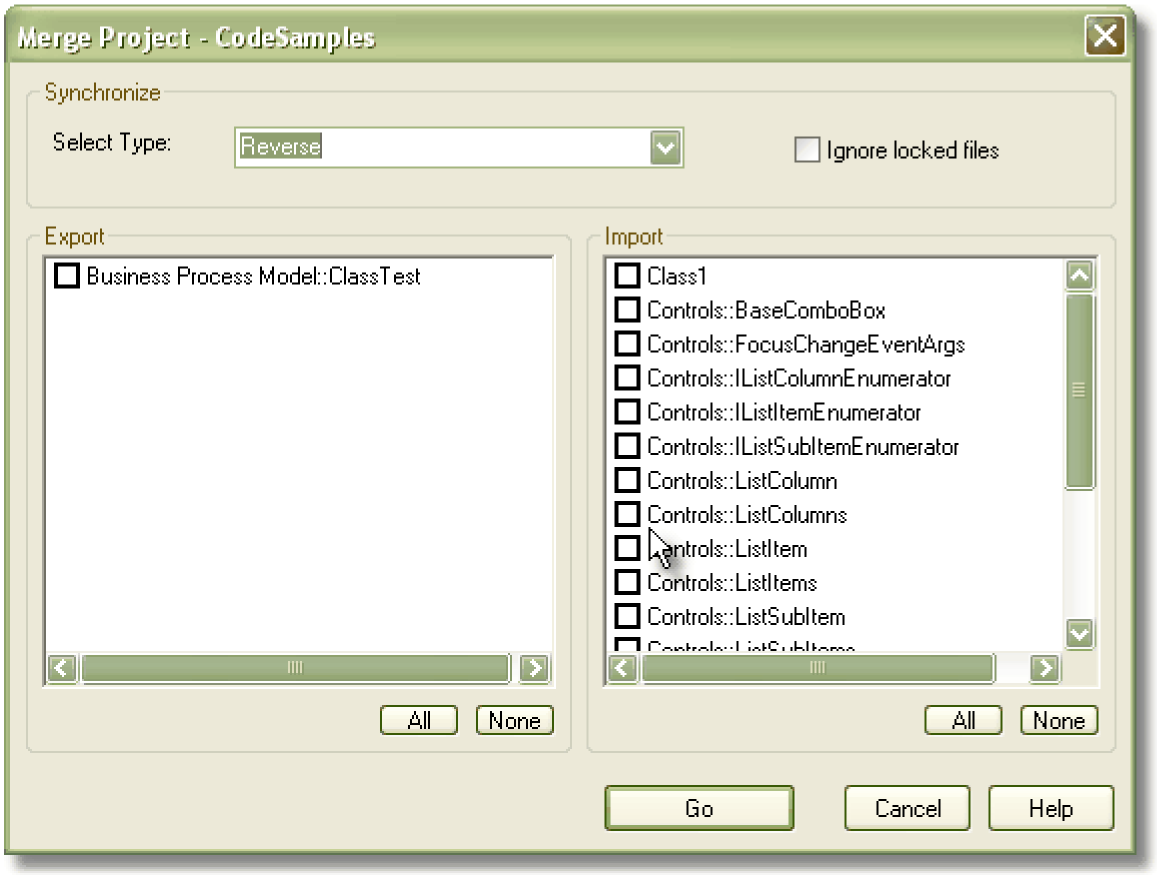
3.5.4.2 Reverse Engineering with a Merge

To generate code with a merge use the following steps once a link has been created to a Visual Studio.NET Project.

1. Select *Add-In* | *Merge with Visual Studio*.



2. This will bring up the following dialog:



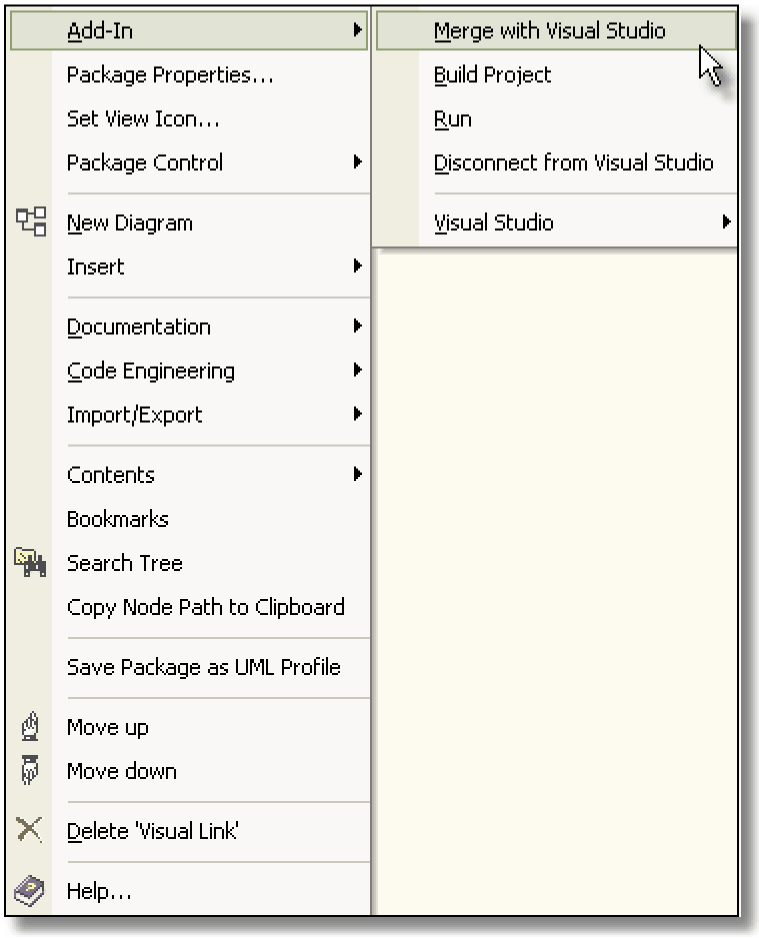
3. In the Synchronize section of the dialog select forward to update classes contained in the code from corresponding elements contained in the model.

4. The items in the Import section apply to element that currently exist in the model but do not exist in the code, select the appropriate classes to be included in for export into the code. Select the appropriate classes by using the checkbox or press *All* to select all of the classes.

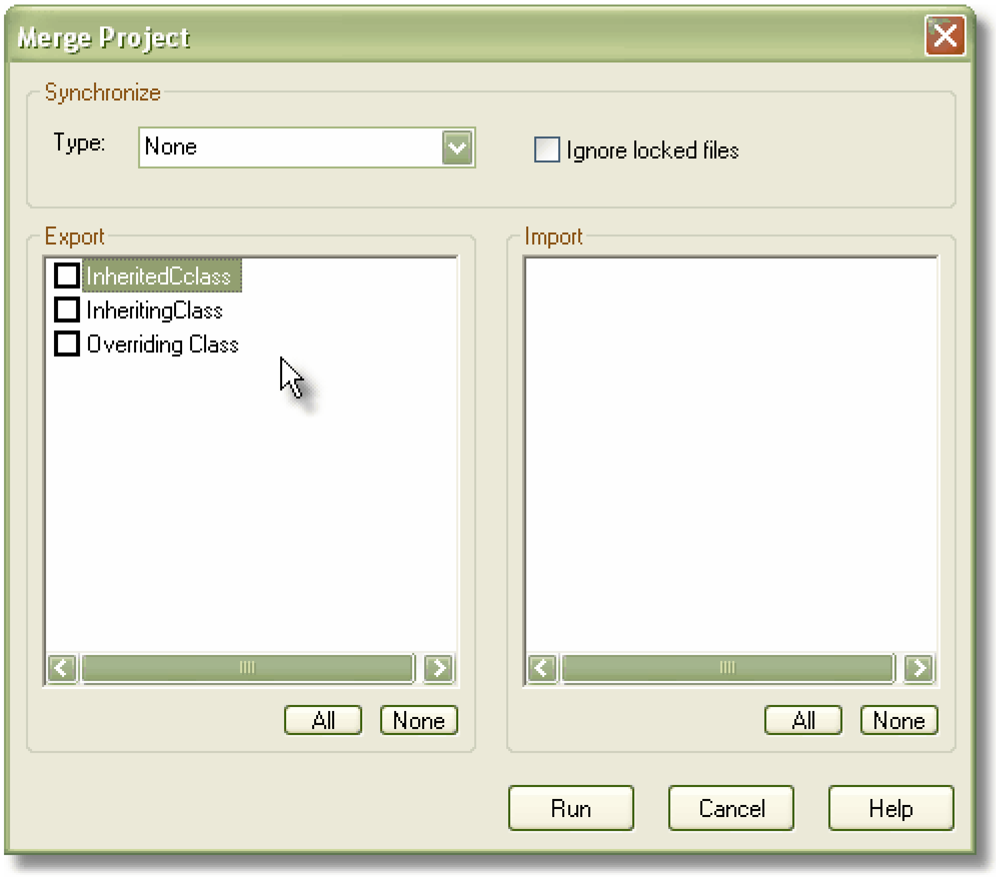
5. Press the *Go* button to reverse engineer the code.

3.5.5 Synchronizing Code with a Model

Synchronizing the code with the model is a simple task once a link has been created to a Visual Studio.NET Project, you may perform a merge at any time by selecting an item from tree view inside the Project View and right clicking on the connected package, this will open up the context menu for the item. Select *Add-In* | *Merge* with External Project.



This will then open up the following Menu:



The Merge project screen gives the following options:

Synchronize	The synchronize drop down menu gives the user four different options for merging. 1. *None*, selecting the None option does not perform synchronization. 2. *Forward*, the forward option generates code from the model into Visual Studio. 3. *Reverse*, the reverse option brings code out of Visual Studio.Net and puts it into the model. 4. *Both*, This option performs the operations of reverse engineering and then the operation of forward engineering which fully synchronizes the model and the code.
Ignore Locked files	Ignores locked files.
Export	This gives the user the option of selecting classes for export, if a class is not selected it will not be included for the export. The *All* option selects all of the classes in the list. *None* selects none of the classes for export.
Import	This gives the user the option of selecting classes for import, if a class is not selected it will not be included for the import. The *All* option selects all of the classes in the list. *None* selects none of the classes for import.
Run	The *Run* button runs the merge.
Cancel	The *Cancel* button cancels the operation.
Help	The *Help* button opens up the help file.

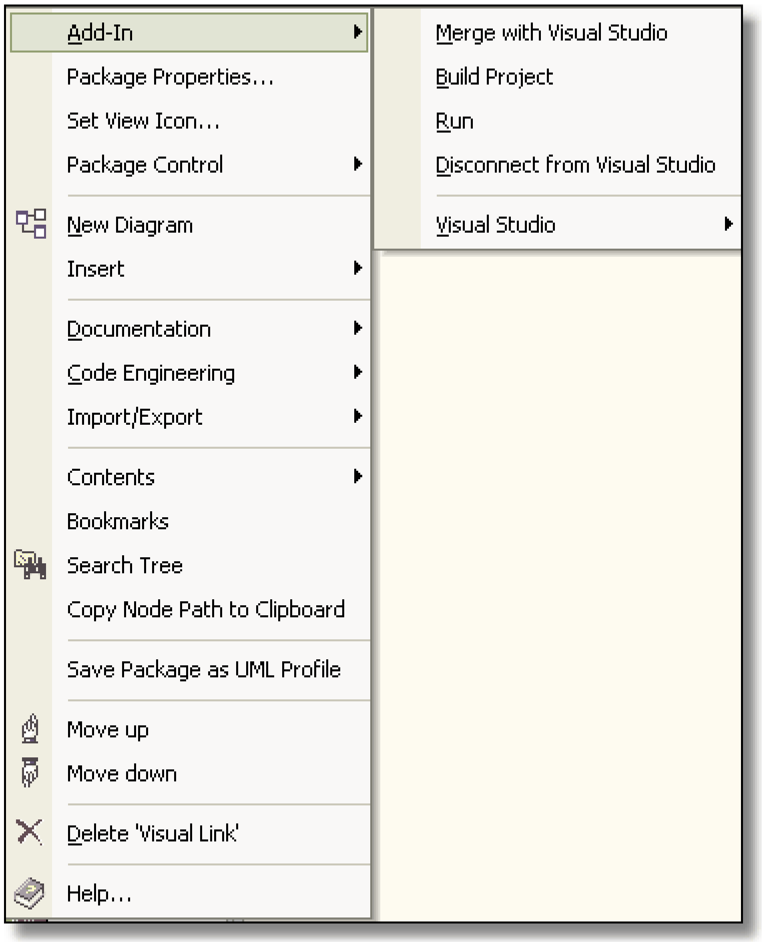
3.6 Add-In Options from the Project View

The *Project Browser* allows you to navigate through the Enterprise Architect project space. It displays packages, diagrams, elements and element properties.

You can drag and drop elements between folders, or even drop elements from the Project Browser directly into the current diagram. With the MDG Link for Visual Studio.NET additional functionality is given to the Project View. This includes the ability to access the Add-In menu, locate class diagrams and to provide the direct link to editing both classes and methods in Visual Studio.NET

3.6.1 Add-In Menu Items

To access the *Add-In* menu from the *Project View* right click on an object in the *Project View* to bring up the context menu. The *Add-In* menu Item is the first entry, when you mouse over the *Add-In* entry the following window will be displayed:



This menu offers several navigation options:

Merge with Visual Studio	Opens the Merge with Visual Studio dialog box to provide Merging options.
Build Project	Builds the current project.
Run	Runs the project.
Disconnect from Visual Studio	Disconnecting an EA package from a Visual Studio.NET solution will free that package so that it may be connected to other solutions.
Visual Studio	This option opens up a dialog box with details of the Visual Studio.NET connections.

3.6.2 Locate Diagrams

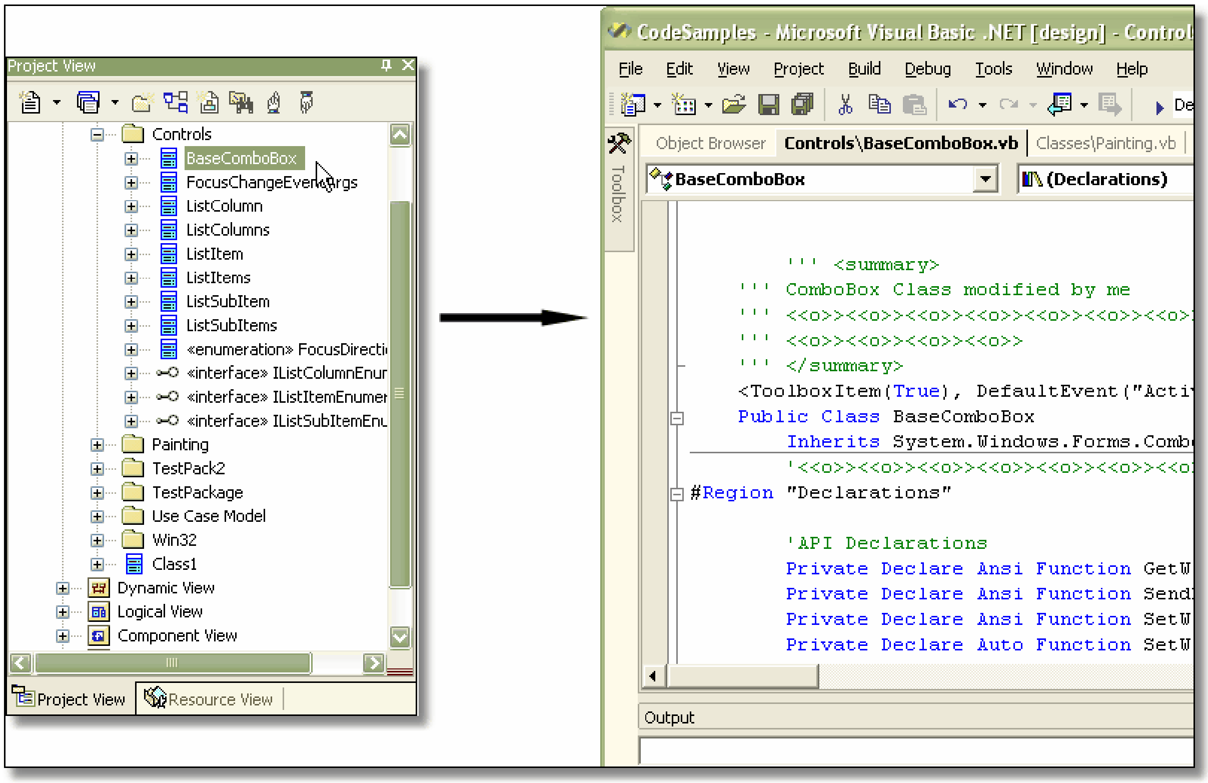
Locating a diagram in the Project Browser can be a difficult task especially when the size of a package has increased to include many classes. To locate a class in the Project View from a Class displayed in a class diagram select the class by clicking on the diagram to bring up class's context menu. Then select the *Locate in Browser* option. Alternatively select the Class in the diagram and press the *ALT*+*G* key combination.

3.6.3 Editing Classes

The Project View allows the user of the MDG Link for Visual Studio.NET to easily access the details of a class. This access allows the user to edit the class properties directly from Enterprise Architect or to edit the class in Visual Studio.NET. To select a specific operation follow the instructions detailed below:

1. From the Project View navigate to the location in the tree hierarchy, to the package containing the class of interest.

2. Expand the details of the class by clicking on the + symbol next to the class details.



```
    ''' <summary>
''' ComboBox Class modified by me
''' <<o>><<o>><<o>><<o>><<o>><<o>><<o>
''' <<o>><<o>><<o>><<o>>
''' </summary>
<ToolboxItem(True), DefaultEvent("Acti
Public Class BaseComboBox
    Inherits System.Windows.Forms.Comb
    '<<o>><<o>><<o>><<o>><<o>><<o>><<o
#Region "Declarations"

    'API Declarations
    Private Declare Ansi Function GetW
    Private Declare Ansi Function Send
    Private Declare Ansi Function SetW
    Private Declare Auto Function SetW
```

3. Right click on the class to bring up its context menu.

4. Select *Properties* to gain access to the Operation, from within Enterprise Architect.

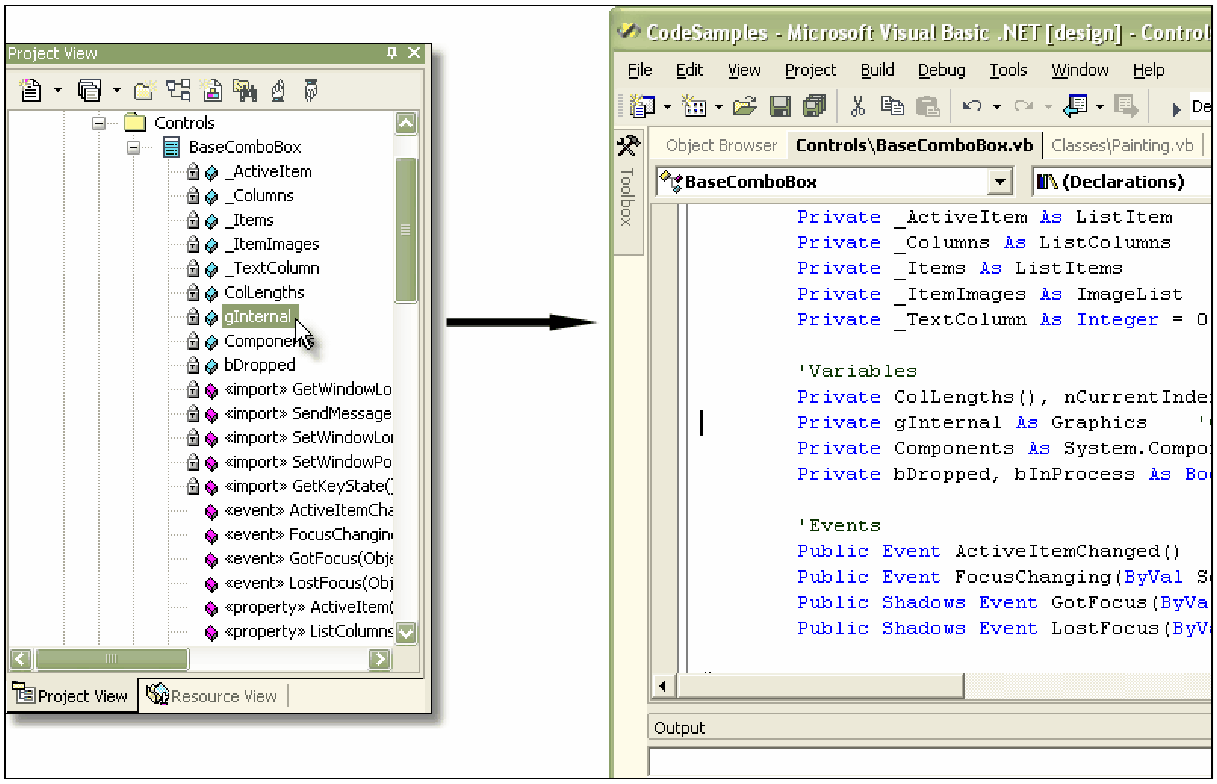
5. Select *View* / *Edit Source Code* to edit the operation in Visual Studio.NET, selecting this option will take the user straight to the beginning of the class in the code.

3.6.4 Editing Operations

The Project View allows the user of the MDG Link for Visual Studio.NET to easily access the operations of a class. This access allows the user to edit the operations directly from Enterprise Architect or to edit the operations in Visual Studio.NET. To select a specific operation follow the instructions detailed below:

1. From the Project View navigate to the location in the tree hierarchy to the package containing the class of interest.

2. Expand the details of the class by clicking on the + symbol next to the class details.



```
Private _ActiveItem As ListItem
Private _Columns As ListColumns
Private _Items As ListItems
Private _ItemImages As ImageList
Private _TextColumn As Integer = 0

'Variables
Private ColLengths(), nCurrentInde
Private gInternal As Graphics    '
Private Components As System.Compo
Private bDropped, bInProcess As Bo

'Events
Public Event ActiveItemChanged()
Public Event FocusChanging(ByVal S
Public Shadows Event GotFocus(ByVa
Public Shadows Event LostFocus(ByV
```

3. Right click on the operation to bring up its context menu.

4. Select *Operation Properties* to gain access to the Operation, from within Enterprise Architect.

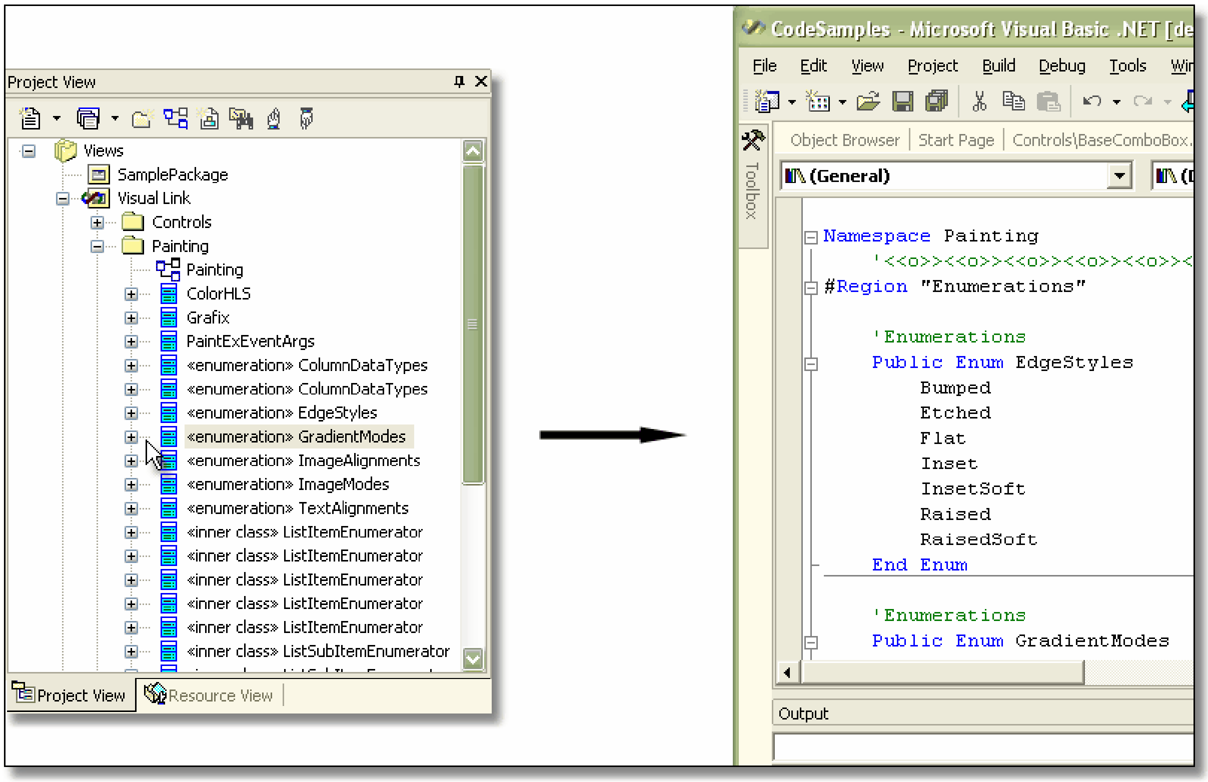
5. Select *View / Edit Source Code* to edit the operation in Visual Studio.NET, selecting this option will take the user straight to the operation in the code.

3.6.5 Editing Attributes

The Project View allows the user of the MDG Link for Visual Studio.NET to easily access the attributes of a class. This access allows the user to edit the attribute directly from Enterprise Architect or to edit the attributes in Visual Studio.NET. To select a specific attribute follow the instructions detailed below:

1. From the *Project View* navigate to the location in the tree hierarchy to the package containing the class of interest.

2. Expand the details of the class by clicking on the + symbol next to the class details.


Project View
Views
SamplePackage
Visual Link
Controls
Painting
Painting
ColorHLS
Grafix
PaintExEventArgs
«enumeration» ColumnDataTypes
«enumeration» ColumnDataTypes
«enumeration» EdgeStyles
«enumeration» GradientModes
«enumeration» ImageAlignments
«enumeration» ImageModes
«enumeration» TextAlignments
«inner class» ListItemEnumerator
«inner class» ListItemEnumerator
«inner class» ListItemEnumerator
«inner class» ListItemEnumerator
«inner class» ListItemEnumerator
«inner class» ListSubItemEnumerator
Project View
Resource View
CodeSamples - Microsoft Visual Basic .NET [de
File
Edit
View
Project
Build
Debug
Tools
Wir
Toolbox
Object Browser
Start Page
Controls\BaseComboBox.
(General)
Output

```
Namespace Painting
    '<<o>><<o>><<o>><<o>><<o>><
#Region "Enumerations"

    'Enumerations
    Public Enum EdgeStyles
        Bumped
        Etched
        Flat
        Inset
        InsetSoft
        Raised
        RaisedSoft
    End Enum

    'Enumerations
    Public Enum GradientModes
```

3. Right click on the attribute to bring up its context menu.

4. Select *Attribute Properties* to gain access to the attribute, from within Enterprise Architect.

5. Select *View* / *Edit Source Code* to edit the Attribute in Visual Studio.NET, selecting this option will take the user straight to the attribute in the code.

Part IV

4 Reference

4.1 System Requirements

The following software needs to be installed to use the Enterprise Architect to MDG Link for Visual Studio.NET:

1. Operating System:
   Microsoft Windows® Server 2003
   Windows XP Professional
   Windows XP Home Edition
   Windows XP Media Center Edition
   Windows XP Tablet PC Edition
   Windows 2000 Professional (SP3 or later **required** for installation)
   Windows 2000 Server (SP3 or later **required** for installation)
2. Enterprise Architect Version 4.1 (or higher) Professional or Corporate Editions
3. Visual Studio.NET version 2003

4.2 Glossary

This section provides a detailed glossary for MDG Link for Visual Studio.NET.

A B C D E F G I L M N O P Q R S T U V

4.2.1 Glossary (A)

~A~

abstract class
A class that cannot be directly instantiated.

Contrast: concrete class

abstraction
The essential characteristics of an entity that distinguish it from all other kinds of entities. An abstraction defines a boundary relative to the perspective of the viewer.

action
The specification of an executable statement that forms an abstraction of a computational procedure. An action typically results in a change in the state of the system, and can be realized by sending a message to an object or modifying a link or a value of an attribute. .

action sequence
An expression that resolves to a sequence of actions.

action state
A state that represents the execution of an atomic action, typically the invocation of an operation.

activation

The execution of an action.

active class
A class whose instances are active objects. When instantiated, an active class will control its execution. Rather than being invoked or activated by other objects, it can operate standalone, and define its own thread of behavior.

See also: active object

activation
An object that owns a thread and can initiate control activity. An instance of active class.

See also: Active class, thread

activity
An activity defines the bounds for the structural organization that contains a set of basic or fundamental behaviors. It can used to model procedural type application development for system design through to modeling business processes in organizational structures and workflow.

activity diagram
An activity diagram can used to model procedural type application development for system design through to modeling business processes in organizational structures and workflow.

activity graph
A special case of a state machine that is used to model processes involving one or more classifiers.

Contrast: state chart diagram

actor [class]
A coherent set of roles that users of use cases play when interacting with these use cases. An actor has one role for each use case with which it communicates.

actual parameter
Synonym: argument

aggregate [class]
A class that represents the "whole" in an aggregation (whole-part) relationship.

See also: aggregation

aggregation
A special form of association that specifies a whole-part relationship between the aggregate (whole) and a component part.

See also: composition

analysis
The part of the software development process whose primary purpose is to formulate a model of the problem domain. Analysis focuses what to do, design focuses on how to do it.

Contrast: design

analysis diagram
An *Analysis diagram* is used to capture high level business processes and early models of system behavior and elements. It is less formal than some other diagrams, but provides a good means of capturing the essential business characteristics and needs.

analysis time
Refers to something that occurs during an analysis phase of the software development process.

See also: design time, modeling time

architecture
The organizational structure and associated behavior of a system. An architecture can be recursively decomposed into parts that interact through interfaces, relationships that connect parts, and constraints for assembling parts. Parts that interact through interfaces include classes, components and subsystems.

argument
A binding for a parameter that resolves to a run-time instance.

Synonym: actual parameter

Contrast: parameter

artifact
A physical piece of information that is used or produced by a software development process. Examples of Artifacts include models, source files, scripts, and binary executable files. An artifact may constitute the implementation of a deployable component.

Synonym: product

Contrast: component

assembly
An assembly connector bridges the required interface of a component with the provided interface of a second component.

association
The semantic relationship between two or more classifiers that specifies connections among their instances.

association class
A model element that has both association and class properties. An association class can be seen as an association that also has class properties, or as a class that also has association properties.

association end
The endpoint of an association, which connects the association to a classifier.

attribute
A feature within a classifier that describes a range of values that instances of the classifier may hold.

auxiliary class
A stereotyped class that supports another more central or fundamental class, typically by implementing secondary logic or control flow. Auxiliary classes are typically used together with focus classes, and are particularly useful for specifying the secondary business logic or control flow of components during

design.

See also: focus

4.2.2 Glossary (B)

~B~

binary association
An association between two classes. A special case of an n-ary association.

binding
The creation of a model element from a template by supplying arguments for the parameters of the template.

boolean
An enumeration whose values are true and false.

boolean expression
An expression that evaluates to a boolean value.

4.2.3 Glossary (C)

~C~

C++
An object-oriented programming language based on the earlier 'C' language.

call
An action state that invokes an operation on a classifier.

cardinality
The number of elements in a set.

Contrast: multiplicity

CASE
Computer Aided Software Engineering. A tool designed for the purpose of modeling and building software systems.

child
In a generalization relationship, the specialization of another element, the parent.

See also: subclass, subtype.

Contrast: parent

choice
The choice pseudo-state is used to compose complex transitional paths, where the outgoing transition path is decided by dynamic, run-time conditions determined by the actions performed by the state machine on the path leading to the choice.

class
A description of a set of objects that share the same attributes, operations, methods, relationships, and semantics. A class may use a set of interfaces to specify collections of operations it provides to its environment.

See also: interface

class diagram
A diagram that shows a collection of declarative (static) model elements, such as classes, types, and their contents and relationships.

classification
The assignment of an object to a classifier. See dynamic classification, multiple classification and static classification.

classifier
A mechanism that describes behavioral and structural features. Classifiers include interfaces, classes, datatypes, and components.

client
A classifier that requests a service from another classifier.

Contrast: supplier

collaboration
The specification of how an operation or classifier, such as a use case, is realized by a set of classifiers and associations playing specific roles used in a specific way. The collaboration defines an interaction.

See also: interaction

collaboration diagram
Used pre - UML 2.0.

collaboration occurrence
Use an Occurrence to apply a pattern defined by a collaboration to a specific situation.

comment
An annotation attached to an element or a collection of elements. A note has no semantics.

Contrast: constraint

compile time
Refers to something that occurs during the compilation of a software module.

See also: modeling time, run time

component
A modular, deployable, and replaceable part of a system that encapsulates implementation and exposes a set of interfaces. A component is typically specified by one or more classifiers (e.g., implementation classes) that reside on it, and may be implemented by one or more artifacts (e.g., binary, executable, or script files).

Contrast: artifact

composite [class]
A class that is related to one or more classes by a composition relationship.

See also: composition

composite state
A state that consists of either concurrent (orthogonal) substates or sequential (disjoint) substates.

See also: substate

composition
A form of aggregation which requires that a part instance be included in at most one composite at a time, and that the composite object is responsible for the creation and destruction of the parts. Composition may be recursive.

Synonym: composite aggregation

concrete class
A class that can be directly instantiated.

Contrast: abstract class

concurrency
The occurrence of two or more activities during the same time interval. Concurrency can be achieved by interleaving or simultaneously executing two or more threads.

See also: thread

concurrent substate
A substate that can be held simultaneously with other substates contained in the same composite state.

See also: composite state

Contrast: disjoint substate

connection
A logical link between model elements. May be structural, dynamic or possessive.

constraint
A semantic condition or restriction. Certain constraints are predefined in the UML, others may be user defined. Constraints are one of three extensibility mechanisms in UML.

See also: tagged value, stereotype

constraint
A rule or condition that applies to some element. It is often modeled as a pre- or post- condition.

container
1. An instance that exists to contain other instances, and that provides operations to access or iterate over its contents.(for example, arrays, lists, sets).

2. A component that exists to contain other components.

containment hierarchy
A namespace hierarchy consisting of model elements, and the containment relationships that exist between them. A containment hierarchy forms a graph.

context
A view of a set of related modeling elements for a particular purpose, such as specifying an operation.

control
A Control is a stereotyped class that represents a controlling entity or manager. A control organizes and schedules other activities and elements. It is the controller of the Model-View-Controller pattern.

control flow
The control flow is a connector linking two nodes in an activity diagram. Control Flow connectors start a nodes activity when the preceding nodes action is finished.

4.2.4 Glossary (D)

~D~

datatype
A descriptor of a set of values that lack identity and whose operations do not have side effects. Datatypes include primitive pre-defined types and user-definable types. Pre-defined types include numbers, string and time. User-definable types include enumerations.

decision
A Decision is an element of an Activity diagram that indicates a point of conditional progression: if a condition is true, then processing continues one way, if not, then another.

defining model [MOF]
The model on which a repository is based. Any number of repositories can have the same defining model.

delegate
A delegate connector defines the internal assembly of a component's external ports and interfaces. Using a delegate connector wires the internal workings of the system to the outside world, by a delegation of the external interfaces' connections.

delegation
The ability of an object to issue a message to another object in response to a message. Delegation can be used as an alternative to inheritance.

Contrast: inheritance

dependency
A relationship between two modeling elements, in which a change to one modeling element (the independent element) will affect the other modeling element (the dependent element).

deployment
A deployment is a type of dependency relationship that indicates the deployment of an artifact onto a node or executable target.

4.2.5 Glossary (E)

~E~

element
An atomic constituent of a model.

element
A model object of any type - class, component, node, object or etc.

entity
An Entity is a store or persistence mechanism that captures the information or knowledge in a system. It is the Model in the Model-View-Controller pattern.

entry action
An action executed upon entering a state in a state machine regardless of the transition taken to reach that state.

entry point
Entry points are used to define where external states can enter a submachine.

enumeration
A list of named values used as the range of a particular attribute type. For example, RGBColor = {red, green, blue}. Boolean is a predefined enumeration with values from the set {false, true}.

event
The specification of a significant occurrence that has a location in time and space. In the context of state diagrams, an event is an occurrence that can trigger a transition.

exception handler
The exception handler element defines the group of operations to carry out when an exception occurs.

exit action
An action executed upon exiting a state in a state machine regardless of the transition taken to exit that state.

exit point
Exit points are used in submachine states and state machines to denote the point where the machine will be exited and the transition sourcing this exit point, for submachines, will be triggered. Exit points are a type of pseudo-state used in the state machine diagram.

export
In the context of packages, to make an element visible outside its enclosing namespace.

See also: visibility

Contrast: export [OMA], import

expose interface
The expose interface toolbox element is a graphical way to depict the required and supplied interfaces of a component, class, or part.

expression

A string that evaluates to a value of a particular type. For example, the expression "(7 + 5 * 3)" evaluates to a value of type number. A relationship from an extension use case to a base use case, specifying how the behavior defined for the extension use case augments (subject to conditions specified in the extension) the behavior defined for the base use case. The behavior is inserted at the location defined by the extension point in the base use case. The base use case does not depend on performing the behavior of the extension use case. See extension point, include.

extend

An Extend connection is used to indicate an element extends the behavior of another. Extensions are used in use case models to indicate one use case (optionally) extends the behavior of another.

4.2.6 Glossary (F)

~F~

facade

A stereotyped package containing only references to model elements owned by another package. It is used to provide a 'public view' of some of the contents of a package.

feature

A property, like operation or attribute, which is encapsulated within a classifier, such as an interface, a class, or a datatype.

fire

To execute a state transition.

See also: transition

focus class

A stereotyped class that defines the core logic or control flow for one or more auxiliary classes that support it. Focus classes are typically used together with one or more auxiliary classes, and are particularly useful for specifying the core business logic or control flow of components during design.

See also: auxiliary

focus of control

A symbol on a sequence diagram that shows the period of time during which an object is performing an action, either directly or through a subordinate procedure.

forward engineering

The process of generating source code from the UML model.

framework

A stereotyped package that contains model elements which specify a reusable architecture for all or part of a system. Frameworks typically include classes, patterns or templates. When frameworks are specialized for an application domain, they are sometimes referred to as application frameworks.

See also: pattern

4.2.7 Glossary (G)

~G~

generalizable element
A model element that may participate in a generalization relationship.

See also: generalization

generalization
A taxonomic relationship between a more general element and a more specific element. The more specific element is fully consistent with the more general element and contains additional information. An instance of the more specific element may be used where the more general element is allowed.

See also: inheritance

guard condition
A condition that must be satisfied in order to enable an associated transition to fire.

4.2.8 Glossary (I)

~I~

implementation
A definition of how something is constructed or computed. For example, a class is an implementation of a type, a method is an implementation of an operation.

implementation class
A stereotyped class that specifies the implementation of a class in some programming language (e.g., C++, Smalltalk, Java) in which an instance may not have more than one class. An Implementation class is said to realize a type if it provides all of the operations defined for the type with the same behavior as specified for the type's operations.

See also: type

implementation inheritance
The inheritance of the implementation of a more general element. Includes inheritance of the interface.

Contrast: interface inheritance

import
In the context of packages, a dependency that shows the packages whose classes may be referenced within a given package (including packages recursively embedded within it).

Contrast: export

include
A relationship from a base use case to an inclusion use case, specifying how the behavior for the base use case contains the behavior of the inclusion use case. The behavior is included at the location which is defined in the base use case. The base use case depends on performing the behavior of the inclusion use case, but not on its structure (ie., attributes or operations).

See also: extend

inheritance
The mechanism by which more specific elements incorporate structure and behavior of more general

elements related by behavior.

See also: generalization

initial state
The Initial pseudo-state is used to denote the default state of a composite state; there can be one initial vertex in each region of the composite state.

interaction diagram
Interaction diagrams can be sequence diagrams, communication diagrams, interaction overview diagrams, and timing diagrams. Interaction diagrams include Timing Diagrams, Sequence Diagrams, Interaction Overview Diagrams and Communication Diagrams.

instance
An entity that has unique identity, a set of operations that can be applied to it, and state that stores the effects of the operations.

See also: object

interaction
A specification of how stimuli are sent between instances to perform a specific task. The interaction is defined in the context of a collaboration.

See also: collaboration

interaction diagram
A generic term that applies to several types of diagrams that emphasize object interactions. These include collaboration diagrams and sequence diagrams.

interaction occurrence
An interaction occurrence is a reference to an existing interaction element. Interaction occurrences are visually represented by a frame, with "ref" in the frame's title space. The diagram name is indicated in the frame contents.

interaction overview diagram
Interaction Overview diagrams visualize the cooperation between other interaction diagrams to illustrate a control flow serving an encompassing purpose. As interaction overview diagrams are a variant of activity diagrams, most of the diagram notation is similar, as is the process in constructing the diagram.

interface
A named set of operations that characterize the behavior of an element.

interface inheritance
The inheritance of the interface of a more general element. Does not include inheritance of the implementation.

Contrast: implementation inheritance

internal transition
A transition signifying a response to an event without changing the state of an object.

interrupt flow

A EA defined toolbox element used to define the exception handler and interruptible activity region concepts.

4.2.9 Glossary (L)

~L~

layer
The organization of classifiers or packages at the same level of abstraction. A layer represents a horizontal slice through an architecture, whereas a partition represents a vertical slice.

Contrast: partition

lifeline
A lifeline is an individual participant in an interaction (i.e., lifelines cannot have multiplicity). A lifeline represents a distinct connectable element.

link
A semantic connection among a tuple of objects. An instance of an association.

See also: association

link end
An instance of an association end.

See also: association end

local path
A relative path on a local machine. Allows developers to store shared source code in machine specific directories, but still generate and synchronize code.

4.2.10 Glossary (M)

~M~

metaclass
A class whose instances are classes. Metaclasses are typically used to construct metamodels.

metafile
A vector based image format native to Windows. Supports high detail and excellent scaling. Typically used for saving diagram images for placement in documents. Comes in Placeable (an older format) and Enhanced (current standard format).

meta-metamodel
A model that defines the language for expressing a metamodel. The relationship between a meta-metamodel and a metamodel is analogous to the relationship between a metamodel and a model.

metamodel
A model that defines the language for expressing a model.

metaobject
A generic term for all metaentities in a metamodeling language. For example, metatypes, metaclasses, metaattributes, and metaassociations.

method
The implementation of an operation. It specifies the algorithm or procedure associated with an operation.

model [MOF]
An abstraction of a physical system with a certain purpose.

See also: physical system

Usage note: In the context of the MOF specification, which describes a meta-metamodel, for brevity the meta-metamodel is frequently to as simply the model.

model aspect
A dimension of modeling that emphasizes particular qualities of the metamodel. For example, the structural model aspect emphasizes the structural qualities of the metamodel.

model elaboration
The process of generating a repository type from a published model. Includes the generation of interfaces and implementations which allows repositories to be instantiated and populated based on, and in compliance with, the model elaborated.

model element [MOF]
An element that is an abstraction drawn from the system being modeled.

Contrast: view element. In the MOF specification model elements are considered to be metaobjects.

model library
A stereotyped package that contains model elements which are intended to be reused by other packages. A model library differs from a profile in that a model library does not extend the metamodel using stereotypes and tagged definitions. A model library is analogous to a class library in some programming languages.

modeling time
Refers to something that occurs during a modeling phase of the software development process. It includes analysis time and design time. Usage note: When discussing object systems, it is often important to distinguish between modeling-time and run-time concerns.

See also: analysis time, design time

Contrast: run time

module
A software unit of storage and manipulation. Modules include source code modules, binary code modules, and executable code modules.

See also: component

multiple classification
A semantic variation of generalization in which an object may belong directly to more than one classifier.

See also: static classification, dynamic classification

multiple inheritance
A semantic variation of generalization in which a type may have more than one supertype.

Contrast: single inheritance

multiplicity
A specification of the range of allowable cardinalities that a set may assume. Multiplicity specifications may be given for roles within associations, parts within composites, repetitions, and other purposes. Essentially a multiplicity is a (possibly infinite) subset of the non-negative integers.

Contrast: cardinality

multi-valued [MOF]
A model element with multiplicity defined whose Multiplicity Type:: upper attribute is set to a number greater than one. The term multi-valued does not pertain to the number of values held by an attribute, parameter, etc. at any point in time.

Contrast: single-valued

4.2.11 Glossary (N)

~N~

name
A string used to identify a model element.

namespace
A part of the model in which the names may be defined and used. Within a namespace, each name has a unique meaning.

See also: name

n-ary association
An association among three or more classes. Each instance of the association is an n-tuple of values from the respective classes.

Contrast: binary association

nesting
The nesting connector is an alternative membership notation used to indicate nested members within an element, for example, a package which has nested members. The nested members of a package could also be shown inside the packaged rather than linked by the nesting connection.

node
A node is classifier that represents a run-time computational resource, which generally has at least a memory and often processing capability. Run-time objects and components may reside on nodes.

4.2.12 Glossary (O)

~O~

object
An entity with a well-defined boundary and identity that encapsulates state and behavior. State is represented by attributes and relationships, behavior is represented by operations, methods, and state machines. An object is an instance of a class.

See also: class, instance

object diagram

A diagram that encompasses objects and their relationships at a point in time. An object diagram may be considered a special case of a class diagram or a collaboration diagram.

See also: class diagram, collaboration diagram

object flow

An Object Flow is a sub type of the State Flow or Transition. It implies the passing of an object instance between elements at run-time.

object flow state

A state in an activity graph that represents the passing of an object from the output of actions in one state to the input of actions in another state.

object lifeline

A line in a sequence diagram that represents the existence of an object over a period of time.

See also: sequence diagram

Object Management Group

The standards body responsible for the UML specification and management. Their website is www.omg.org - follow the links to the UML pages.

object toolbar

The main toolbar running down the center of EA from which you can select model elements to insert into diagrams. This is also known as the UML Toolbox and the Toolbox.

occurrence

An occurrence relationship indicates that a collaboration represents a classifier. An occurrence connector is drawn from the collaboration to the classifier.

operation

A service that can be requested from an object to effect behavior. An operation has a signature, which may restrict the actual parameters that are possible.

4.2.13 Glossary (P)

~P~

package

1. A package is a namespace as well as an element that can be contained in other package's namespaces. Packages can own or merge with other packages, and its elements can be imported into a package's namespace.

2. A logical container of model elements. Groups elements and may also contain other packages.

   The OMG UML specifications states:

   "*A package is a grouping of model elements. Packages themselves may be nested within other packages. A package may contain subordinate packages as well as other kinds of model elements. All kinds of UML model elements can be organized into packages.*"

   Note that packages own model elements and are the basis for configuration control, storage, and access control. Each element can be directly owned by a single package, so the package hierarchy is a strict tree. However, packages can reference other packages, modeled by using one of the stereotypes «import» and «access» of Permission dependency, so the usage network is a graph. Other kinds of dependencies between packages usually imply that one or more dependencies among the elements exists.

A package is shown as a large rectangle with a small rectangle (a "tab") attached to the left side of the top of the large rectangle. It is the common folder icon.

package diagram
Package diagrams are used to reflect the organization of packages and their elements, and provide a visualization of their corresponding namespaces.

package import
A package import relationship is drawn from a source package to a package whose contents will be imported. Private members of a target package cannot be imported.

package merge
A package merge indicates a relationship between two packages whereby the contents of the target package are merged with those of the source package. Private contents of a target package are not merged.

parameter
The specification of a variable that can be changed, passed, or returned. A parameter may include a name, type, and direction. Parameters are used for operations, messages, and events.

Synonym: formal parameter

Contrast: argument

parameterized element
The descriptor for a class with one or more unbound parameters.

Synonym: template, parameterized class

parent
In a generalization relationship, the generalization of another element, the child.

See also: subclass, subtype

Contrast: child

part
Parts are run-time instances of classes or interfaces.

participate
The connection of a model element to a relationship or to a reified relationship. For example, a class participates in an association, an actor participates in a use case.

partition
1. activity graphs: A portion of an activity graphs that organizes the responsibilities for actions.

   See also: swim lane

2. architecture: A set of related classifiers or packages at the same level of abstraction or across layers in a layered architecture. A partition represents a vertical slice through an architecture, whereas a layer represents a horizontal slice.

Contrast: layer

pattern
A template collaboration.

persistent object
An object that exists after the process or thread that created it has ceased to exist.

physical system
1. The subject of a model.
2. A collection of connected physical units, which can include software, hardware and people, that are organized to accomplish a specific purpose. A physical system can be described by one or more models, possibly from different viewpoints.

Contrast: system

postcondition
A constraint that must be true at the completion of an operation.

precondition
A constraint that must be true when an operation is invoked.

primitive type
A pre-defined basic datatype without any substructure, such as an integer or a string.

process
1. A heavyweight unit of concurrency and execution in an operating system.

Contrast: thread, which includes heavyweight and lightweight processes. If necessary, an implementation distinction can be made using stereotypes.

2. A software development process - the steps and guidelines by which to develop a system.
3. To execute an algorithm or otherwise handle something dynamically.

profile
A profile is a stereotyped package that contains model elements which have been customized for a specific domain or purpose using extension mechanisms, such as stereotypes, tagged definitions and constraints. A profile may also specify model libraries on which it depends and the metamodel subset that it extends.

project view
The workspace window (top left) where the model contents are displayed in 'tree' format. Displays packages, diagrams, model elements, etc.

property
A named value denoting a characteristic of an element. A property has semantic impact. Certain properties are predefined in the UML; others may be user defined.

See also: tagged value

4.2.14 Glossary (Q)

~Q~

qualifier
An association attribute or tuple of attributes whose values partition the set of objects related to an object across an association.

4.2.15 Glossary (R)

~R~

realize
A source object realizes the destination object. Realize is used to express traceability and completeness in the model – a business process or requirement is realized by one or more use cases which are in turn realized by some classes which in turn are realized by a component, etc.

receive [a message]
The handling of a stimulus passed from a sender instance.

See also: sender, receiver

receive
A Receive element is used to define the acceptance or receipt of a request. Movement on to next action does occur until it has received what is defined.

receiver [object]
The object handling a stimulus passed from a sender object.

Contrast: sender

reception
A declaration that a classifier is prepared to react to the receipt of a signal.

recursion
A recursion is a type of message used in sequence diagrams to indicate a recursive function.

reference
1. A denotation of a model element.

2. A named slot within a classifier that facilitates navigation to other classifiers.

Synonym: pointer

region
UML 2 supports both expansion regions and interruptible activity regions. An Expansion Region defines the bounds of an region consisting of one or more sets of input collections, where an input collection is a set of elements of the same type. An interruptible region contains activity nodes - when a token leaves an interruptible region, this terminates all of the regions tokens and behaviors.

refinement
A relationship that represents a fuller specification of something that has already been specified at a certain level of detail. For example, a design class is a refinement of an analysis class.

relationship
A semantic connection among model elements. Examples of relationships include associations and generalizations.

represents
The Represents connector indicates a collaboration is used in a classifier. The connector is drawn from the collaboration to its owning classifier.

requirement
A desired feature, property, or behavior of a system.

responsibility
A contract or obligation of a classifier.

reuse
The use of a pre-existing artifact.

reverse engineering
The process of importing source code into the model as standard UML model elements (classes, attributes, operations, etc.).

rich text format
A standard mark-up language for creating word processor documents, frequently associated with Microsoft Word.

run time
The period of time during which a computer program executes.

Contrast: modeling time

4.2.16 Glossary (S)

~S~

schema [MOF]
In the context of the MOF, a schema is analogous to a package which is a container of model elements. Schema corresponds to an MOF package.

Contrast: metamodel, package

self-message
A self-message reflects a new process or method invoked within the calling lifeline's operation. It is a specification of a message.

send [a message]
The passing of a stimulus from a sender instance to a receiver instance.

See also: sender, receiver

sender [object]
The object passing a stimulus to a receiver object.

Contrast: receiver

sequence diagram
A diagram that shows object interactions arranged in time sequence. In particular, it shows the objects participating in the interaction and the sequence of messages exchanged. Unlike a collaboration diagram, a sequence diagram includes time sequences but does not include object relationships. A sequence diagram can exist in a generic form (describes all possible scenarios) and in an instance form (describes one actual scenario). Sequence diagrams and collaboration diagrams express similar information, but show it in different ways.

See also: collaboration diagram

signal
The specification of an asynchronous stimulus communicated between instances. Signals may have parameters.

signature
The name and parameters of a behavioral feature. A signature may include an optional returned parameter.

single inheritance
A semantic variation of generalization in which a type may have only one supertype.

Synonym: multiple inheritance [OMA]

Contrast: multiple inheritance

single valued [MOF]
A model element with multiplicity defined is single valued when its Multiplicity Type: upper attribute is set to one. The term single-valued does not pertain to the number of values held by an attribute, parameter, etc., at any point in time, since a single-valued attribute (for instance, with a multiplicity lower bound of zero) may have no value.

Contrast: multi-valued

specification
A declarative description of what something is or does.

Contrast: implementation

state
A condition or situation during the life of an object during which it satisfies some condition, performs some activity, or waits for some event.

Contrast: state [OMA]

state invariant
A State Invariant is a condition applied to a lifeline, which must be fulfilled for the lifeline to exist.

state machine
A behavior that specifies the sequences of states that an object or an interaction goes through during its life in response to events, together with its responses and actions.

state machine diagram
A State Machine diagram illustrates how an element, often a class, can move between states classifying its behavior, according to transition triggers, constraining guards, and other aspects of state machine

diagrams that depict and explain movement and behavior.

state chart
diagram A diagram that shows a state machine.

See also: state machine

state continuation
The State/Continuation symbol serves two different purposes for interaction diagrams, as state invariants and as continuations. A State Invariant is a condition applied to a lifeline, which must be fulfilled for the lifeline to exist. A Continuation is used in seq and alt combined fragments, to indicate the branches of continuation an operand follows.

state lifeline
A State Lifeline follows discrete transitions between states, which are defined along the y-axis of the time line. Any transition has optional attributes of timing constraints, duration constraints and observations.

static classification
A semantic variation of generalization in which an object may not change classifier.

Contrast: dynamic classification

stereotype
A new type of modeling element that extends the semantics of the metamodel. Stereotypes must be based on certain existing types or classes in the metamodel. Stereotypes may extend the semantics, but not the structure of pre-existing types and classes. Certain stereotypes are predefined in the UML, others may be user defined. Stereotypes are one of three extensibility mechanisms in UML.

See also: constraint, tagged value

stimulus
The passing of information from one instance to another, such as raising a signal or invoking an operation. The receipt of a signal is normally considered an event.

See also: message

string
A sequence of text characters. The details of string representation depend on implementation, and may include character sets that support international characters and graphics.

structural diagram
Structural diagrams depict the structural elements composing a system or function. These diagrams can reflect the static relationships of a structure, as do class or package diagrams, or run-time architectures, such as object or composite structure diagrams. Structural diagrams include Class diagrams, Composite Structure diagrams, Component diagrams, Deployment diagrams, Object diagrams and Package diagrams.

structural feature
A static feature of a model element, such as an attribute.

structural model aspect
A model aspect that emphasizes the structure of the objects in a system, including their types, classes, relationships, attributes, and operations.

subactivity state
A state in an activity graph that represents the execution of a non-atomic sequence of steps that has some duration.

subclass
In a generalization relationship, the specialization of another class; the superclass.

See also: generalization

Contrast: superclass

submachine state
A state in a state machine which is equivalent to a composite state but its contents is described by another state machine.

subpackage
A package that is contained in another package.

substate
A state that is part of a composite state.

See also: concurrent state, disjoint state

subsystem
A grouping of model elements that represents a behavioral unit in a physical system. A subsystem offers interfaces and has operations. In addition, the model elements of a subsystem can be partitioned into specification and realization elements.

See also: package, physical system

subtype
In a generalization relationship, the specialization of another type; the supertype.

See also: generalization

Contrast: supertype

superclass
In a generalization relationship, the generalization of another class; the subclass.

See also: generalization

Contrast: subclass

supertype
In a generalization relationship, the generalization of another type; the subtype.

See also: generalization

Contrast: subtype

supplier
A classifier that provides services that can be invoked by others.

Contrast: client

swimlane
A partition on a activity diagram for organizing the responsibilities for actions. Swimlanes typically correspond to organizational units in a business model.

See also: partition

synch
A synch state is useful for indicating concurrent paths of a state machine will be synchronized. After bringing the paths to a synch state, the emerging transition will indicate unison.

synchronize code
The process of importing and exporting code changes to ensure the model and source code match

system
A top-level subsystem in a model.

Contrast: physical system

system boundary
A System Boundary element is used to delineate a particular part of the system. For example in the diagram below, the actor is outside the system and the use case within.

4.2.17 Glossary (T)

~T~

table
A relational table (composed of columns).

tagged value
The explicit definition of a property as a name-value pair. In a tagged value, the name is referred as the tag. Certain tags are predefined in the UML; others may be user defined. Tagged values are one of three extensibility mechanisms in UML.

See also: constraint, stereotype

template
Synonym: parameterized element

terminate
The terminate pseudostate indicates that upon entry of its pseudostate, the state machine's execution will end.

thread [of control]
A single path of execution through a program, a dynamic model, or some other representation of control flow. Also, a stereotype for the implementation of an active object as lightweight process.

See also: process

time event
An event that denotes the time elapsed since the current state was entered.

See also: event

time expression
An expression that resolves to an absolute or relative value of time.

toolbox
The main toolbar running down the center of EA from which you can select model elements to insert into diagrams. This is also known as the UML Toolbox and the Object Toolbar.

top level
A stereotype of package denoting the top-most package in a containment hierarchy. The topLevel stereotype defines the outer limit for looking up names, as namespaces "see" outwards. For example, opTopLevelubsystem represents the top of the subsystem containment hierarchy.

trace
A dependency that indicates a historical or process relationship between two elements that represent the same concept without specific rules for deriving one from the other.

transient object
An object that exists only during the execution of the process or thread that created it.

transition
A relationship between two states indicating that an object in the first state will perform certain specified actions and enter the second state when a specified event occurs and specified conditions are satisfied. On such a change of state, the transition is said to fire.

type
type A stereotyped class that specifies a domain of objects together with the operations applicable to the objects, without defining the physical implementation of those objects. A type may not contain any methods, maintain its own thread of control, or be nested. However, it may have attributes and associations. Although an object may have at most one implementation class, it may conform to multiple different types.

See also: implementation class

Contrast: interface

type expression
An expression that evaluates to a reference to one or more types.

4.2.18 Glossary (U)

~U~

UML
The Unified Modeling Language, a notation and specification for modeling software systems in an Object-Oriented manner. You can read more about UML at the OMG home page or at our UML Tutorial

UML diagrams
UML diagrams are used to model different aspects of the system under development. They include various elements and connections, all of which have their own meanings and purposes. UML 2.0 includes 13 diagrams: Use Case diagram, Activity diagram, State Machine diagram, Timing diagram, Sequence diagram, Interaction Overview diagram, Communication diagram, Package diagram, Class diagram,

Object diagram, Composite Structure diagram, Component diagram and Deployment diagram.

UML toolbox
The main toolbar running down the center of EA from which you can select model elements to insert into diagrams. This is also known as the Toolbox and the Object Toolbar.

usage
A dependency in which one element (the client) requires the presence of another element (the supplier) for its correct functioning or implementation.

utility
A stereotype that groups global variables and procedures in the form of a class declaration. The utility attributes and operations become global variables and global procedures, respectively. A utility is not a fundamental modeling construct, but a programming convenience.

4.2.19 Glossary (V)

~V~

value
An element of a type domain.

value lifeline
The Value lifeline shows the lifeline's state across the diagram, within parallel lines indicating a steady state. A cross between the lines indicates a transition or change in state.

view
A projection of a model, which is seen from a given perspective or vantage point and omits entities that are not relevant to this perspective.

view element
A view element is a textual and/or graphical projection of a collection of model elements.

view projection
A projection of model elements onto view elements. A view projection provides a location and a style for each view element.

visibility
An enumeration whose value (public, protected, package or private) denotes how the model element to which it refers may be seen outside its enclosing namespace.

Visual Basic
A rapid application development programming language. Windows only scripting language based on COM.

Index

MDG Link for Visual Studio.NET User Guide

www.sparxsystems.com.au