UML for C#

C# is a modern object-oriented language for application development. In addition to object-oriented constructs, C# supports component-oriented programming with properties, methods and events.

UML defines graphical notations for describing and designing object-oriented software systems. It's an open standard controlled by the Object Management Group (OMG). Although UML has many diagram types, we'll focus on class models that show static class structure and relationships.

WinA&D is a complete UML modeling tool enriched with C# language specific details used to generate source code. WinTranslator is a reverse engineering tool that scans code to extract design information into WinA&D models. Diagrams created in WinA&D are used to illustrated C# programs represented in the UML notation.

This paper assumes a working knowledge of C# and UML. It briefly describes how C# constructs are represented by UML for forward and reverse engineering.

Modeling Basics

In WinA&D, a class model is drawn from a palette of tools. As each class instance is placed on the diagram, it's named in the Class Properties dialog. Each class has a corresponding dictionary entry of the same name in the data dictionary. Many diagrams within the class model and in other types of diagram documents share data that is stored in the same global data dictionary.

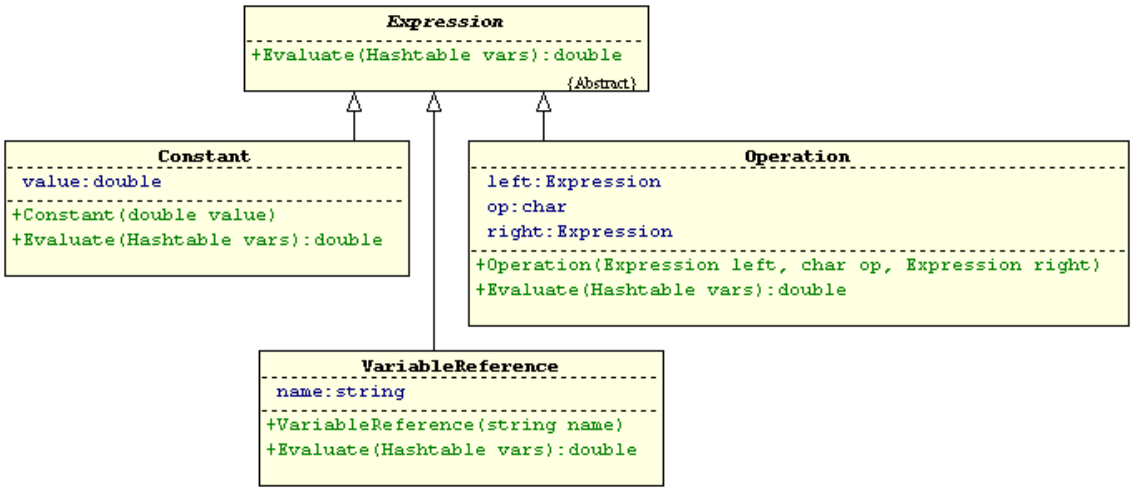
For a selected class object on the diagram, the Details button presents the Class Attributes & Operations dialog. This dialog is used to define members of the class. In WinA&D terminology, a class can have Attribute, Operation, Property and Event members. Behind the scenes, WinA&D adds a dictionary entry for each class member with a name of the form Class'Attribute, Class.Operation, Class$Property and Class-Event.

Each class member has a details dialog for defining language specific information for that class member. WinA&D supports many programming languages for code generation including C#. Depending on which language is currently selected, the Attribute Details, Operation Details, Property Details and Event Details dialog will vary slightly based on specific characteristics of the selected language.

WinA&D can concurrently store language specific details for multiple languages for each modeling element. When instances of a class are presented on different diagrams, detailed information is stored once in the global dictionary. WinA&D uses this information to generate source code.

Class Model

A UML class diagram shows the static class structure of a C# application. Different types of the objects on the class diagram represent C# classes, interfaces, structs and their generic counterparts.



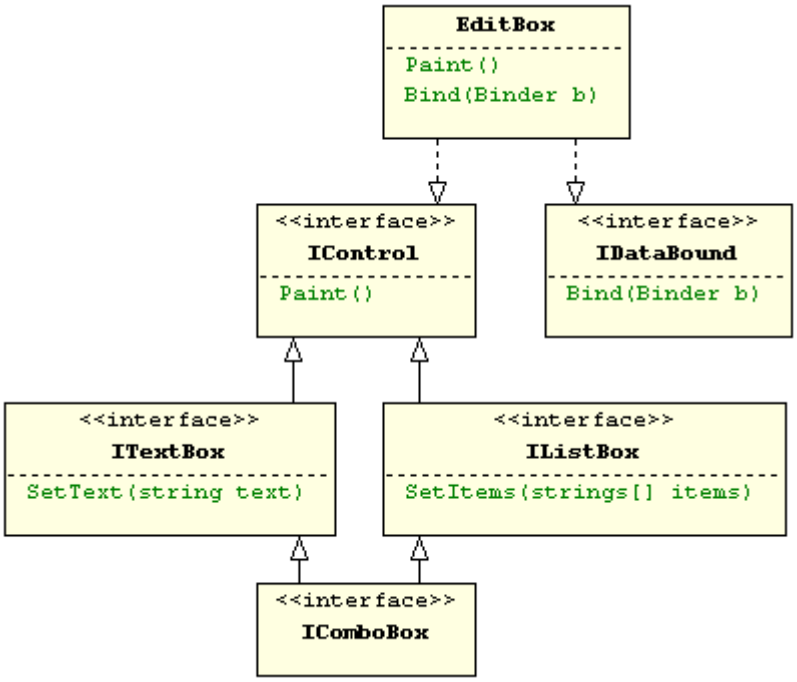
UML Class Diagram with C# Classes

Class Objects

Each class on a diagram is represented as a named box with dashed sections showing a list of class members. Classes are connected with relationship lines. In the diagram above, the Constant, VariableReference and Operation classes inherit from the abstract Expression class.

The presentation of a class diagram can vary widely based on user-specified criteria. In the diagram above, class attributes (C# fields) and operations (C# methods) are shown with access, data type and signature details. WinA&D gives the user a lot of flexibility to control how classes are presented. Class members can be shown or hidden based on member type or specific conditions based on access or modifiers on each class member. Members of a class instance on a diagram can show various levels of detail like its access type, data type or arguments. Presentation options can be easily applied across all diagrams, to specific diagrams or to individual instances of a class.

Information about classes, class member and C# details are entered into detail dialogs when drawing diagrams and stored in the global dictionary. Instances of the same class can be shown on many diagrams with different presentations.



Class and Interface Relationships

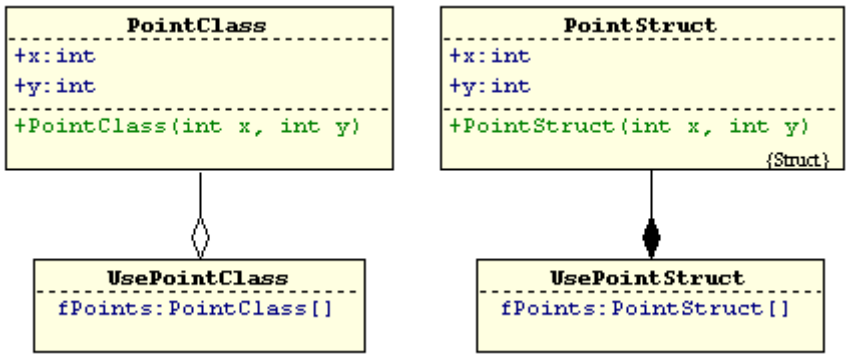
Interface Objects

An interface defines members without implementations providing a contract that can be implemented by classes and structs. An interface looks similar to a class box with the addition of the <<interface>> stereotype at the top. The ITextBox and IListBox interfaces inherit from the IControl interface shown with an open arrowhead on the line pointing at the inherited interface. Unlike classes, in C# an interface can inherit from multiple interfaces as IComboBox does. The EditBox class implements both the IControl and IDataBound interfaces as shown by the dashed line and open arrowheads.

Struct Objects

A struct can have data and function members similar to a class. A variable of type class stores a reference to an object dynamically allocated on heap. Unlike a class, a struct is a value type that doesn't allocate heap or allow user-specified inheritance.

In WinA&D terminology, UsePointClass and UsePointStruct are classes with an fPoints attribute (C# field). The X and Y coordinates for each point in the array can be stored using a class or struct.

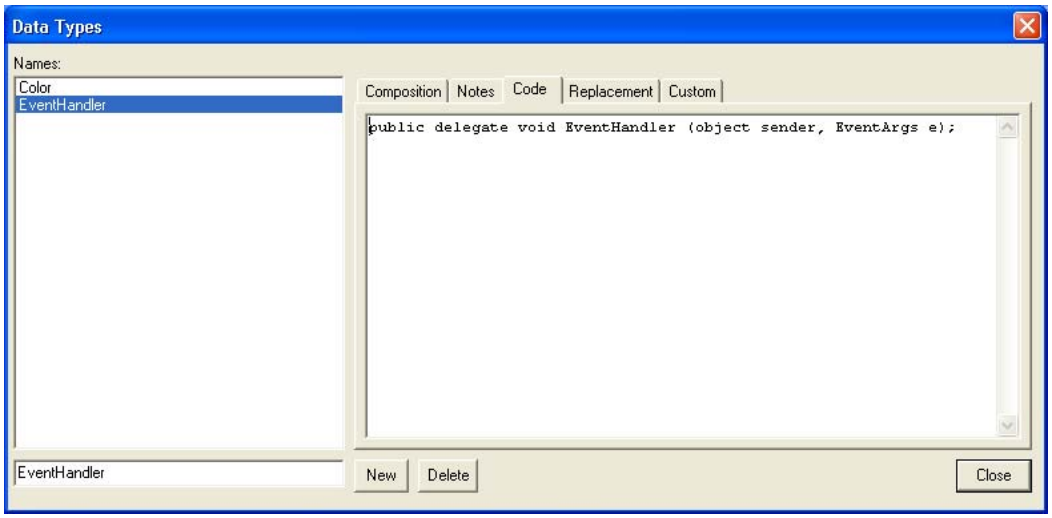


Similarities Between Class and Struct

When PointClass is used, each point and the array itself is stored in a separate class instance dynamically allocated on heap and represented on the diagram as aggregation by reference (hollow diamond on relationship line). When PointStruct is used, only one object for the array itself is instantiated and coordinates are stored in-line in the array. On the diagram, this is represented as aggregation by value (solid diamond on relationship line).

C# Delegates and Enums

Delegates and enums are type declarations stored in the dictionary and defined using the Data Types dialog. After typing the entry name at the bottom, press the Insert key or click the New button, then use the Code panel to enter the actual C# code of the delegate or enum entry.



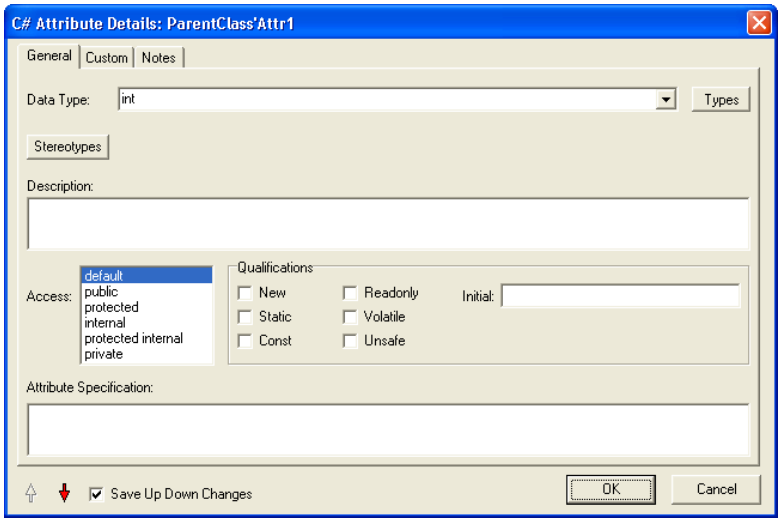
C# Delegates and Enums

Class Member Details

After adding objects on a class diagram and merging the diagram to generate corresponding dictionary entries, class members can be added and C# specific details can be defined using the Attribute Details, Operation Details, Property Details and Event Details dialogs.

C# Constants & Fields

Constants and fields of a C# class are represented as class attributes in WinA&D. Details of an attribute like its data type, access and qualifications (C# modifiers) are entered into the Attribute Details dialog and stored in the dictionary entry for that attribute.



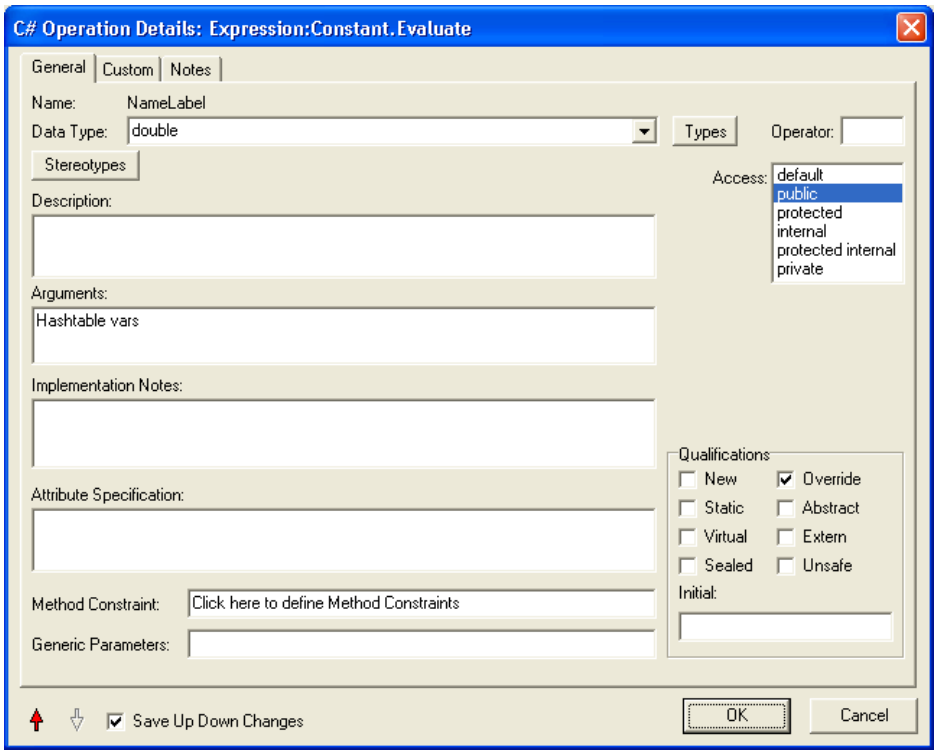
C# Constants & Fields

The drop down menu on the Data Type field shows a list of primitive data types for C#. That list can be user defined from the Document Defaults dialog of the Dictionary document. The Types button can also be used to select from user defined classes and data structures in the dictionary.

The bottom left corner of a class member detail dialog may have Up and Down arrows enabled. Click these arrow icons to navigate between items in a class member list (attributes, operations, etc.) to quickly make editing changes.

C# Methods, Constructors, Destructors and Operators

Methods, constructors, destructors and operators of a C# class are represented as operations in WinA&D. Details of an operation like its data type, access, arguments and qualifications (C# modifiers) are entered into the Operation Details dialog and stored in the dictionary for that operation.



C# Methods, Constructors, Destructors and Operators

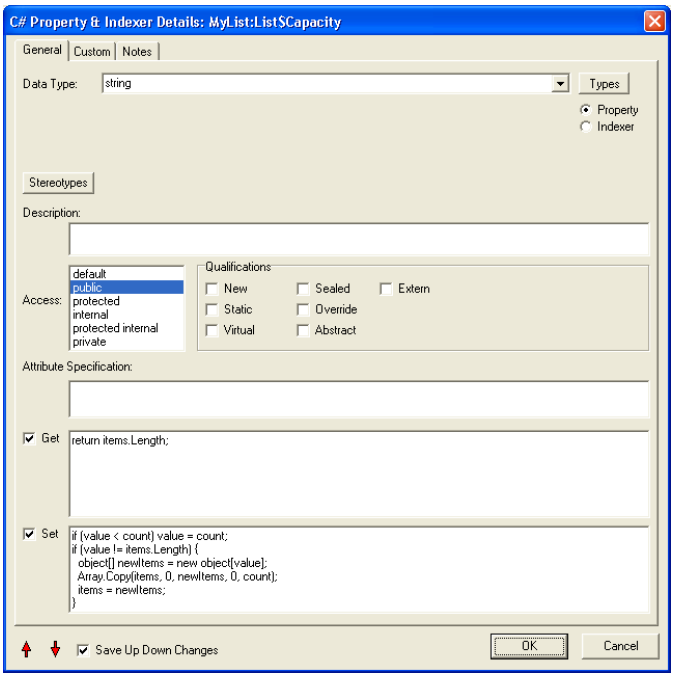
A C# constructor has the same name as the class itself. A destructor has the same name as the class with the ~ prefix. A C# operator has an alphanumeric operation name and uses the Operator field of the dialog to store its real name like +, -, etc.

C# supports overloaded function names. Each overloaded function in C# can have the same name with a different argument list. In the design model, name the operations (C# methods) using the convention MethodName$1, MethodName$2, MethodName$3. WinA&D stores the details of each method like its unique signature in a separate dictionary entry named Class.MethodName$1, Class.MethodName$2, Class.MethodName$3 and during code generation strips off the $ and number from the method name.

C# Properties and Indexers

Properties and Indexers of a C# class are represented as properties in WinA&D. In the details dialog, set the Property or Indexer radio button. Next assign the data type, access and qualifications (C# modifiers).

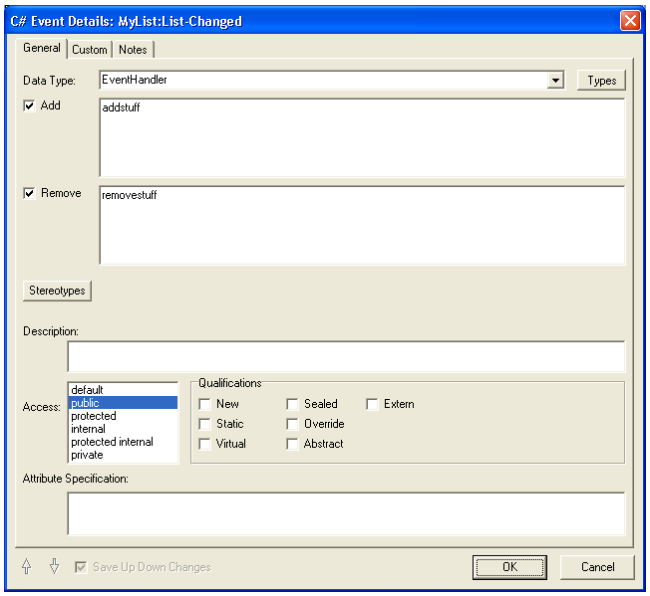
Use the Get and Set edit field to enter the actual code for getting or setting the property or indexer. When the Indexer radio button is selected, an Arguments edit field is visible in the dialog.



C# Properties & Indexers

C# Events

An Event member enables a class to provide notifications. Other classes can handle the event by assigning an event handler. When a class raises the event, those event handlers get executed to process the event.



C# Events

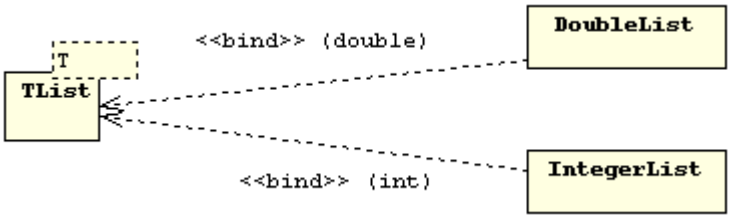
Generics

A generic class declaration is a template for creating a class type by providing one or more actual parameter types. Generic classes, interfaces and structs are added in C# 2.0. Generic methods and constraints can be declared inside any class, interface or struct declaration.

Classes, Interfaces and Structs

The UML representation of a generic class is a class box with an overlapping dashed box at the upper right corner showing its formal parameters.

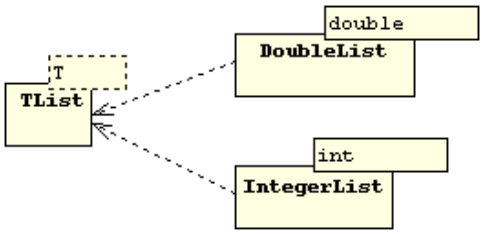
Generic classes, interfaces and structs can be created in C# using a similar representation with the <<interface>> stereotype or {struct} property highlighting interface or stuct objects, respectively.



Explicit Binding of Formal and Actual Generic Classes

An actual class (C# constructed type) is created from the generic class by providing one or more data types. This can be represented on the diagram as a dependency relationship using the <<bind>> stereotype and listing the supplied parameter types.

Alternatively, the actual parameters can be shown in a solid box at the upper right corner of the actual class (C# constructed type). Formal and actual parameters are defined in the Parameters field of the Class Properties dialog. The text entered into this field gets shown in the parameter box on the diagram.



Implicit Binding of Formal and Actual Generic Classes

The Class panel of the Class Attributes & Operations dialog has a field for defining the type parameter list that appears in generated source code.

Generic Methods and Constraints

The C# 2.0 language adds generic methods and constraints. A generic method has normal parameters plus type parameters that are provided when using the method.

If the name of the method itself uses a type parameter, then enter that type parameter in the Generic Parameters field of the C# Operation Details dialog. If the method uses constraints, then click the Method Constraints field and enter each constraint on a separate line of the Method Constraints dialog that is presented.

Other C# Constructs

C# introduces concepts not directly comparable to constructs in other OO languages.

Attributes

In C#, programmers can invent new kinds of declarative information called attributes. These attributes are not be confused with the conventional meaning of the term "attribute" as data or fields of a class.

In C# lingo, the term attribute refers to text within brackets that is attached to program entities like classes, interfaces, structs and class members. Within WinA&D, this attribute information is entered into an Attribute Specification field of a detail dialog for a class member. For example, the C# Attribute Details, C# Operation Details, C# Property & Indexer Details and C# Event Details dialogs all have an Attribute Specification field.

C# attributes of a class, interface or struct are stored in the External Declarations field on the Class panel of the Class Attributes & Operations dialog. The External Declarations field is visible by selecting Tab 2 at the bottom of the Class panel.

During code generation, WinA&D adds this attribute information prior to generated items. Likewise, WinTranslator captures this information from C# code.

Partial Types

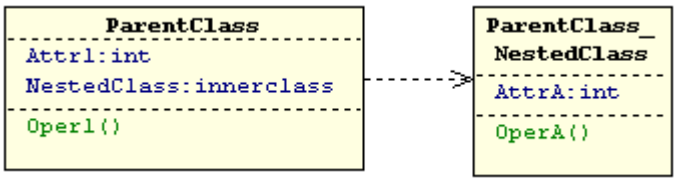
Some types can be defined in multiple parts. In C#, the partial modifier word identifies one of these types which gets merged together at compile time.

Within the WinA&D design environment, each class, interface or struct gets a unique name and its own dictionary entry. To support C# partial types, a ~ and number is added to items names. For example, if three partial types collectively define class X, they would be named X~1, X~2 and X~3.

During code generation, WinA&D will drop the ~ character and trailing number from the class name and add the partial modifier to the generated code. Likewise, when scanning C# code with WinTranslator to generate UML class diagrams in WinA&D, the same convention is used for partial types.

Nested Types

A type declared within a class or struct is called a nested type. Nested types are modeled in WinA&D as separate class or struct objects on the diagram. Set the Inner checkbox in the details dialog for the nested object. In the parent object (the enclosing class or struct), add an attribute that references the nested object with the data type of innerclass, innerstruct or innerinterface.



UML for Nested C# Class

In the UML class diagram shown above, ParentClass is the name of a class containing NestedClass. The name of a nested class is prefixed with the name of its parent class and an underscore. For example, if the parent class in the code is named A and its inner class is named B, then the required name of the inner class in the model is A_B. From the diagram we see that the parent class is dependent on the nested class as shown with the dependency relationship drawn from the parent to the nested class.

Namespaces

The WinA&D modeling tool has a namespace feature that serves several purposes. Namespaces can partition the models and dictionary information for a large project into different domains like communication, interface, database or control. Many features can be driven by namespace like report generation, import/export and naming conventions. Namespaces are also used to identify the paths to code folders that contain source code associated with a project.

The Java language uses the package concept to group related classes into a logical package with a source file for each class in the package physically stored in one folder on disk. When modeling a Java application, each package translates into a WinA&D namespace that identifies both its logical organization and physical folder path of related code files.

The C# language uses a namespace construct to logically group classes. Physically, however, the classes in a namespace can be stored in the same file, different files or even different files of different folders. If a program uses 7 C# namespaces and 20 code folders, it may require up to 27 WinA&D namespace definitions to define both the logical and physical organization of the code.

The WinA&D namespace definition for a logical C# namespace consists of its name and the Description field which holds the full namespace declaration as illustrated below. The Path and Access fields of the namespace definition are unused.

Namespace Name:	`Customize`
Namespace Description:	`wiki.editor.customize`

Each namespace name is limited to 20 characters and is usually the same as the last dot separated part of a hierarchical C# namespace. In an application with C# namespaces of “wiki.editor.customize” and “wiki.user.customize” you could use namespace names of cutomize1 and customize2.

The WinA&D namespace definition for a physical folder location is illustrated below. The Description field can be used to document the folder contents or left blank.

Namespace Name:	`~Customize`
Namespace Path:	`c:\wiki\editor\customize`

By WinA&D convention, the first character is a tilde “~” of a namespace name used solely for identifying a folder path. WinTranslator also uses this convention when generating namespaces from C# code.

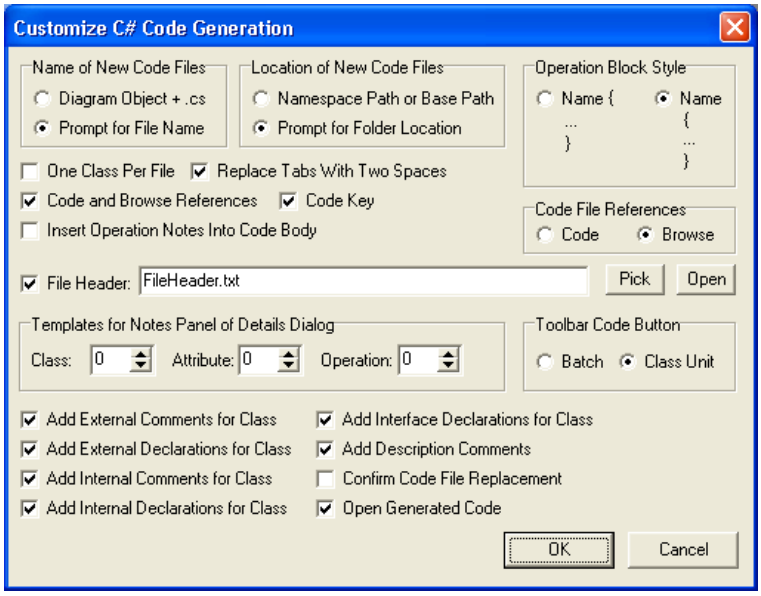
Code Generation

C# code generation in WinA&D uses information from a class model and its associated dictionary entries. The code generation process is similar to that used for C++, Delphi or other object-oriented languages. The resulting code includes a declaration for each class, interface or struct and empty function frames for each method.

A checkbox on the customization dialog allows text from the Notes panel of the Operations Details dialog to be inserted into the generated function frame, thus making it easy to include programming comments or source code into the function frames.

Customize Code

The **Generate->Code->Customize** command presents a dialog to customize the generated code.



Dialog to Customized Generated C# Code

The Customize dialog provides some control over what gets generated. For example, you can include a predefined file header at the beginning of each generated code file and automatically insert fields like the current time stamp, user name, organization, etc.

Generate Class, Interface and Struct

WinA&D has two commands for generating code from a class model, the **Batch** and **Unit Code** commands from the **Generate->Code** submenu. The **Unit Code** command for a selected class A automatically generates the code file named A.cs and prompts for a folder to store the resulting file. The **Batch** code command generates code for one or more selected class, interface or struct objects and prompts for a file name and location.

Generate Types

During design, delegate and enum definitions can be created in the Data Types dialog. During the implementation phase of a project, these type definitions can be output to a code file.

Place the insertion point in the code view of a Code or Browse window and choose the **Paste->Data Types from Dictionary Entry** command from the **Option** menu. The Code panel of the selected data types will be output to that location in the file.

Nested Types

To generate code for a parent class or struct and its nested types, first select and generate code for the parent object. A stub will be placed within the code generated for the parent object. Now select the nested object and generate its code with the **Unit Code** command. The stub will be replaced by code representing the selected nested object.

Reverse Engineering

WinTranslator is a reverse engineering tool that scans source code and extracts design information to a text file. The reengineering process is fully automated.

 The user simply clicks the Reengineer Project button to present a step-by-step dialog that identifies the source language (like C#), source code folders and other options. WinTranslator scans the code and outputs a text file of design information.

The output from WinTranslator for a C# project consists of a Dictionary.rp file in each code folder referenced by a Dictionary.list file in the designated WinTranslator project folder.

Within WinA&D, use the New Project dialog to create a new project with a Dictionary and Class Model document and set the language to C#. Open the generated project documents. From the Dictionary window, import the Dictionary.list file generated by WinTranslator. The project now has a dictionary populated with design information extracted from the source code. WinA&D has options to color dictionary entries based on type or to structure entries hierarchically to more easily identify entries for classes, interfaces, structs and various kinds of class members.

From the Class window, use the **Generate->Class Model->From Dictionary** command to present the Class Model From Dictionary dialog. From this dialog, a class diagram can be generated for each logically related cluster of classes.

Summary

This paper describes the primary constructs in the C# language and how they are modeled in WinA&D using the UML notation. The model can be used to generate C# code. WinTranslator scans source code to generate a UML model for an existing project.

Excel Software
Ph. 505-771-3719
info@excelsoftware.com
www.excelsoftware.com