With Class

User's Guide

MicroGold Software

696 Birch Hill Drive

Bridgewater, NJ, 08807

1 - 908 - 722 - 6438

CompuServe - 71543,1172

Internet - microgold@delphi.com

Table of Contents

Chapter 1 - Introduction

1.1 Description

With Class is an object-oriented CASE (Computer Aided Software Engineering) tool. There are three versions of With Class: 1) Education Version with a limited number of classes, 2) Professional Version, and 3) Network Version. Its goal is to provide automated support in S/W development projects. With Class supports the following major S/W development functions:
- creation of class diagrams,
- creation of the data dictionary,
- generation of C++ code,
- creation of class diagrams from C++ source code (reverse engineering).

Those of us who are C++ programmers know what a hassle it is to write header files, write ifdefs, look up what variables we used. Even the simple task of putting double colons everywhere in our CPP files is beyond annoying. This software tool was designed with the intent of simplifying the users job of designing a C++ program and at the same time providing a useful graphical means of documenting your code.

Specific diagramming capabilities of this tool are:
- point and click graphical interface,
- choice of O-O graphic notations, e.g. Rumbaugh, Coad-Yourdon, Booch, and Shlaer-Mellor,

- hide and show inheritance, aggregation, and association relationships,
- clip feature to edit, save, and print a diagram or portion of a diagram for export to a word processing program,
- zooming and scrolling features,
- browser to textually view each class with its attributes and operations,
- drawing tool to add lines, arrows, boxes, and circles to a diagram.
- preferences menu to set options (planned),
- automatic backup of files (planned),
- export diagrams in Windows Metafile format (.wmf) (planned),
- export diagrams with DDE or OLE (planned),
- integration of class diagram with state diagram and message diagram (planned),
- validation check of diagrams (planned).

Specific data dictionary capabilities for textual specifications are:
- fill-in form to specify the system,
- fill-in form to specify each class in a system,
- fill-in form to specify each attribute in a class,
- fill-in form to specify each operation in a class,
- create an ASCII file for the data dictionary,
- print the data dictionary as a report.
- provide data dictionary in dBASE or other format (planned),
- provide multiple reports, e.g. summary, class list, detailed, (planned),

Specific C++ code generation capabilities are:
- generate a C++ header and source file for each class,
- generate constructors (default, with arguments, copy),
- generate default destructors, including virtual destructors,
- generate assignment and equality operators (operator= and operator==),
- generate code for single and multiple inheritance,
- generate data members for 1:1 and 1:M aggregation and association relationships,
- generate static data members and function members,
- generate streamable classes with read, write, and other functions,
- generate insertion operator (operator<<) for cout statements,
- generate get and set accessor functions for all data members,
- generate get and set accessor functions for data member for associated and part objects (association and aggregation),
- support effective C++ coding standards, e.g. Scott Meyer's "Effective C++",
- ability to edit C++ source code files in With Class,
- support virtual base classes (planned),
- generate extraction operator (operator>>) for cin statements (planned),
- generate C++ exceptions in functions (planned).
- generate main function (planned),
- generate types file (planned),

- generate make or project file (planned).

Specific reverse engineering capabilities are:
- create class diagram from a set of C++ header and source code files that were originally created in With Class,
- show inheritance, aggregation, and association relationships,
- show class diagram in various O-O notations, e.g. Rumbaugh, Coad-Yourdon, Booch, and Shlaer-Mellor,
- reverse a single or multiple C++ files,
- interactive reverse engineering to clarify unknown C++ constructs (planned).

1.2 Installing With Class
To install With Class, run the install batch file on the floppy disk. If you are installing on hard drive C, you should type Install C: at the DOS prompt . The program will create directory WCLASS for your program just below the c:\ root directory.

When you have finished installing into DOS you must now install into Windows. Simply go to the New menu command in the Program Manager and browse for wc.exe under the WCLASS directory of the drive you installed the program on. The program will be added to your applications group. As an alternative, you can use the Windows Setup program to bring wc.exe into Windows in the same way.

The system requirements for With Class are as follows:
- 80386 processor or higher (80484 recommended),
- MS-Windows 3.1 or higher,
- Mouse that supports MS-Windows
- VGA monitor (1024x768 or higher resolution recommended).

Chapter 2: With Class Menu Commands

2.1 Introduction
The menu bar contains the following major menu item categories: file, draw, methodology, view, generate, specs, and help. There are four primary file extensions that are used in With Class:
- .omt file for each class diagram - graphic file,
- .dic file for the data dictionary - text file,
- .h file for the class header file - text file,
- .cpp file for the class source code file - text file.
The menu items in each category and what they all do are:

2.2 File- The file menu opens, saves, prints, and clips the class diagram files and text files. All class diagrams are saved under files with extension .omt. The File menu items are shown below.

WITH CLASS

File Draw Methodology View Generate Specs

Help
New
Open
Save
SaveAs
Save To Clipboard
Print
Print Text
Printer Setup
Edit File
Editor Setup
Set Directory
Exit

File New - This menu command clears the current diagram.

File Open - This menu option will open an .omt file.

File Save - This option saves an .omt file. The file contains all information about the current diagram.

File Save As - This command allows you to save your file under a different .omt filename.

File Save to Clipboard - This menu option will save the text outlined by the clipping tool to the clipboard.

File Print - This prints out the entire class diagram in the file up to a 3 x 3 page diagram. The user can piece together the 8 1/2 x 11" pages to form a full diagram with all the details.

File Print Text - This allows you to print any text file. This includes *.h, *.cpp, *.hpp, and *.dic files.

File Printer Setup - This feature lets you set up your default printer.

File Edit File - This command allows you to edit a text file. The editor used depends on the editor specified in the Editor Setup. Any Windows editor may be used. The default editor is Windows Notepad.

File Editor Setup - This feature allows you to choose an editor in which you wish to edit your code. The command defaults to notepad.exe (the text editor shipped with Windows 3.x).

File Set Directory - This command allows the user to set a directory in which the With Class files and generated code will be stored. The user can also create a new subdirectory by entering it in the edit box.

File Exit - 1This menu option exits the program. It will ask for a save first.

2.3 Draw- Draw allows you to draw classes and relations by opening up class and relation dialog boxes. This feature is also accessible through accelerator keys, and from the tool palette. The Draw menu items are shown below.

WITH CLASS

File Draw Methodology View Generate Specs Help

Class
Relation
Draw Text
Draw Oval
Draw Rectangle
Draw Line
Draw Arrow
Draw CW Arc
Draw CCW Arc
Stretch Shape
Delete On
Clip On
Snap On
Show Tools

Draw Class - This menu command brings up the class dialog box. It allows you to draw a class.

Draw Relation - This menu command brings up the relation dialog box. It allows you to draw a relation. The following is a brief description of each relation:

1) Association - This is used to link one class to another. It is a means of allowing information to pass between classes. For example an employer hires an employee. You would need a link between employer and employee to execute hiring.

2) Aggregation - This is what a class is made up of, or often described as a 'part-whole' relationship. For example class car is made up of class door, class radio, class engine and class tire.

3) Inheritance - This is what allows the designer to go from general to specific between classes, and is often referred to as 'generalization - specialization' relation. For example class boat is a superclass for subclass oceanliner, subclass yacht, and subclass rowboat. Oceanliner, yacht, and rowboat inherit the properties of boat.

Draw Text - This feature allows the user to draw text on the

diagram and stretch the text to whatever rectangle the user wishes to fit the text in.

Draw Oval - The user uses this feature to draw an oval of any size by clicking down on the location and dragging to the appropriate size.

Draw Rectangle - The user uses this feature to draw a rectangle of any size by clicking down on the location and dragging to the appropriate size.

Draw Line - The user can draw a line in the same way the user has drawn relations.

Draw Arrow - The user can draw an arrow in the same way the user has drawn relations. The arrow may be used to show events or messages.

Draw CW Arc, CCW Arc - This feature allows the user to draw clockwise and counter clockwise arcs. The user may want to turn off the snap when using this feature.

Stretch Shape - This feature brings up a tool that the user can use to stretch polygons and text. Simply click on the polygon you wish to stretch. Release the mouse, and the dotted version of the polygon will appear. Now just drag on the shape to stretch it.

Delete On - This turns the delete tool on. The delete tool can be turned back off by toggling this menu option. It can also be turned on and off from the X in the tool palette.

Clip On - This turns the clipping tool on. The user simply clicks the left mouse button down in the corner in which the user wishes to clip and drags to the far corner. The dotted rectangle borders the area of clipping. The user then chooses the File Save to Clipboard.

Snap On - This function defaults to checked when the program is launched. The Snap On 'checked' indicates that the program will snap all lines and relations
horizontally or vertically to the attached class or polygon.

Show Tools - This feature hides and shows the tool palettes. The default is checked. This feature allows the user to hide the tool palettes.

2.4 Methodology- This menu option allows you to check which methodology you wish to use, Rumbaugh, Shlaer-Mellor, Booch, or Coad-Yourdon. Note: The menu will default to Rumbaugh during a save or print. However, there is no problem switching between methodologies in the middle of drawing. The Methodology menu items are shown below.

WITH CLASS

File Draw Methodology View Generate Specs Help

Rumbaugh
Shlaer-Mellor
Booch
Coad-Yourdon

Rumbaugh - This menu option selects the O-O graphic notation as described in "Object-Oriented Modeling and Design" by Rumbaugh, Blaha, Premerlani, Eddy, and Lorensen. This is one of the methodologies the user can choose as a basis for the shapes of relations. Clicking Rumbaugh in the menu changes all relations to the characteristic shapes for Rumbaugh.

Shlaer-Mellor - This menu option selects the O-O graphic notation as described in "Object Lifecycles Modeling the World in States" by Sally Shlaer and Steven Mellor. This is one of the methodologies the user can choose as a basis for the shapes of relations. Clicking Shlaer-Mellor in the menu changes all relations to the characteristic shapes for Shlaer-Mellor.

Booch - This menu option selects the O-O graphic notation as described in "Object Oriented Design with Applications" by Grady Booch. This is one of the methodologies the user can choose as a basis for the shapes of relations. Clicking Booch in the menu changes all relations to the characteristic shapes for Booch.

Coad-Yourdon - This menu option selects the O-O graphic notation as described in "Object Oriented Analysis" by Peter Coad and Edward Yourdon. This is one of the methodologies the user can choose as a basis for the shapes of relations. Clicking Coad-Yourdon in the menu changes all relations to the characteristic shapes for Coad-Yourdon.

2.5 View - These menu options provide various choices to display the class diagram. The Methodology menu items are shown below.

WITH CLASS

File Draw Methodology View Generate Specs Help

Zoom In

Zoom Out
View All Attributes
View All Operations
View Attribute Type
View Operation Parameters
Color
Show Inheritance
Show Aggregation
Show Association
Show Drawing Elements
Browse Classes

View Zoom In - This feature expands the diagram until it is full size. The maximum sized diagram shows all class details of attributes and operations.

View Zoom Out - This feature shrinks the diagram. Zooming out can help you see more of the diagram, especially if your diagram is very large.

View All Attributes - Here the user can choose to view and print all the attributes of the class. This feature defaults to off, showing only 5 of the attributes.

View All Operations - This feature allows the user to view and print all operations of a class. The default is to only view 5 of the operations in the class.

View Attribute Type - Sometimes the user may wish to hide the attribute types in the class. Turning off this feature allows the user to show and print only the attribute names, making it more desirable for an analysis tool. The feature defaults to on.

View Operation Parameters - Sometimes the user may wish to hide the return types and parameters of the operations in the class. Turning off this feature allows the user to show and print only the operation names. The feature defaults to on.

View Color - The user has a choice of a color diagram or black and white. The diagram defaults to color.

View Show Inheritance - The user can hide the inheritance relations by turning off this feature. This shows or removes the inheritance relation symbol in printing.

View Show Aggregation - The user can hide the aggregation relations by turning off this feature. This shows or removes the aggregation relation symbol on the screen and in printing.

View Show Association - The user can hide the association relations by turning off this feature. This shows or removes the association relation symbol on the screen and in printing.

View Show Drawing Elements - The user can hide the shapes created

from the shape palette by turning off this feature. This shows or removes the drawing elements on the screen and in printing.

View Class Browser - This feature opens up a dialog in which the user can browse classes in the diagram. By click on a class, the attributes and operations owned by the class appear to the right. Clicking on a specific operation brings up the code for that operation. The code may be edited in this window.

2.6 Generate - These menu items provide options to generate C++ source code, create a data dictionary, and to reverse engineer class diagrams from C++ source code. The Generate menu items are shown below.

WITH CLASS

File Draw Methodology View Generate Specs Help

Generate Code
Make Data Dictionary
Options
Reverse

Generate Code - This option will generate C++ code from your current class diagram in the omt file.

Make Data Dictionary - This feature makes a data dictionary from the diagram. It is most useful if the user has descriptions for the various classes, relations, attributes, operations, and system specifications.

Options - This feature brings up the Options Dialog Box This dialog allows the user to check various options for code generation. The Options dialog has the following choices:

- Generate Header files
- Generate CPP files
- Generate Copy Constructors, Assignment Operator, Equality Operator
- Generate #ifdefs - #if define directive for header files
- Generate ClassDefs - Borland macro that inserts P for pointers and R for references
- Add Comments from Description - comments header and cpp files from attribute and operation descriptions
- Generate Streamable Classes - Borland feature to make classes persistent
- Generate ostream << functions for all classes. - Allows user to do things like cout << myClass
- Generate prefix in front of classes

- Make One-to-Many C_Array or Container Class (Borland BIDS or Visual C++ COb collections).

Reverse - This feature allows the user to reverse engineer already generated code.
The code should all reside in the same directory for this feature to work. Upon selecting this option, the user will be prompted to choose the directory of the existing code. Then after the user clicks ok the user will be prompted for the header files (*.h, *.hpp) and source files (*.cpp) which they wish to import into the diagram. The program will then proceed to reverse the code into a diagram on the screen which the user can choose to save. Only non-inline code will be brought into the diagram. With Class does not reverse comments, global variables, consts and types, friend functions and inline functions.

2.7 Specs System - This menu option brings up a dialog for specifying your system. The information from this dialog will appear in the data dictionary.

2.8 Help - This menu option brings up the With Class Help System.

Chapter 3 - Class Diagram and Drawing Palettes

3.1 Introduction - With Class has a class diagram and a drawing palette to create and embellish class diagrams. The mouse is an integral part of using With Class. The mouse is used as follows:
- single click left button to select a menu item or a tool palette icon, e.g. save,
- double click left button to invoke a dialog box, e.g. double click on the class symbol on a diagram invokes to class dialog box,
- single click right button on a class symbol to generate C++ or copy the class,
- drag with the left button down on a class or drawing symbol to move the symbol or after the stretch icon has been selected to stretch a drawing symbol..

3.2 Class Diagram Palette - This palette of icons permits the creation, editing, viewing, saving, and printing of class diagrams. The specific icons with the icon description in parenthesis are listed below:

Class Icon (Box with Class Label) - This is used to create a class symbol on the class diagram.

Relation Icon (Arrow with Relation Label) - This is used to connect to classes with a relationship, e.g. association, aggregation, or inheritance.

Zoom In Icon (Circle with Incoming Arrows) - This feature expands the diagram until it is full size.

Zoom Out Icon (Circle with Outgoing Arrows) - This feature shrinks the diagram

File Open Icon (Disk with Arrow to Right) - This feature opens an omt file to display a class diagram.

File Save As Icon (Disk with Arrow to Left) - This feature saves the omt file.

Generate Code Icon (Diagram with Arrow to C) - This feature brings up the generate code dialog box.

Save to Clipboard Icon (Arrow) - This feature permits the drawing of a rectangle in the portion of the display area to be clipped into the clipboard.

Delete On Icon (X) - This feature permits the deletion of any graphic or text symbol in the display area.

Print Icon (Printer Symbol) - This feature permits the printing the class diagram.

3.2 Class Diagram Palette - This palette of icons permits the creation and placement of drawing symbols and text on class diagrams. The specific drawing icons are:

Rectangle Icon (Box) - This is used to draw a rectangle.

Circle Icon (Circle) - This is used to draw a circle.

Line Icon (Line) - This is used to draw a line.

Arrow Icon (Arrow) - This is used to draw an arrow.

Clockwise Arc Icon (Arrow Arc) - This is used to draw an arc in the clockwise direction.

Counter Clockwise Arc Icon (Arrow Arc) - This is used to draw an arc in the counter clockwise direction.

Text Icon (Text) - This is used to create text.

Stretch To Icon (Box Next to Text Icon)- This is used to stretch drawing and text symbols.

Chapter 4 - Creating Diagrams, Data Dictionary, and C++ Source Code

4.1 Creating a Class Diagram - The class diagram displays classes with their attributes, operations, and relationships. The steps to create a class diagram using With Class with a Car class example are:

>> Create a directory for Car Products, e.g., c:\md car

>> Run With Class from Windows
>> Select "File - New"
>> Select "File - SaveAs" e.g., c:\car\car.omt
>> Select Class Icon
>> Enter the class name, e.g., Car
>> Enter each attribute in the form <Class/Type> <Attribute Name>, e.g., int gasQty and select Add
>> Enter each operation in the form <return class/type> <operation name> <argument class/type argument name>, e.g., int getGasQty () and void setGasQty (int aGasQty) and select Add
>> Double click OK to create the class
>> Place the class symbol on the page
>> Select "File - Save" to save the diagram
>> Select "Print" to print the diagram

There are four major dialog boxes to enter information about the class, attributes, operations, and relationships. These are described below:

Class Dialog - Double clicking on a class brings up the class dialog box. The class dialog box has several features: the ability to provide a class name, list attributes, list operations, define description for attributes and operations, list include files, and provide a class description(through the Spec button). Attributes should be entered in a C code format as follows:
<type> <name>
Examples:

int Number;
BOOL IsOn
float Distance

If you just enter a <name> then the program will default to an int type;

Operations are entered in a similar fashion:
<return type> <name>(<parameters>);

Examples:

int draw(int length, int width)
void execute()

If you just enter a procedure name, the program will default to a void return type.

Double clicking on either an operation or an attribute will bring up information about the respective operation or attribute. Here you can enter a description of the operation or attribute in the Operation Dialog or the Attribute Dialog. Hitting OK in this class dialog brings up the placement crosshair for a class. The user can then place the class where he or she wishes by moving the mouse to the desired location, and then clicking the left

mouse button. Double Clicking on the inside of the class figure will bring up the class dialog again if you wish to make changes.

Attribute Dialog - The attribute dialog is invoked by double clicking any attribute. It provides the user with the ability to enter information about each attribute. A form is provided to enter the attribute type, name, description, visibility, initial value, minimum value, maximum value, and constraint. A constraint is a rule or limitation on a attribute value, e.g. worker_salary must be less than the boss_salary.

Operation Dialog - The operation dialog is invoked by double clicking any operation. It provides the user with the ability to enter information about each operation. A form is provided to enter the operation name, return type or class, arguments, description, visibility, classification (modifier or accessor), precondition, postcondition, invariant, exception, concurrency, and transformation (formula). A user may enter C++ source code by hitting the Code Button. This button opens up a simple editor in which the user only need enter the lines of code contained inside the brackets of the function. Hitting exit will save the code to the program. Check boxes are provided to designate an operation as virtual, static, const, or pure virtual.

Relation Dialog - The relation dialog is invoked by selecting the relation icon in the tool palette, selecting the relation menu item in the draw menu, or double clicking on a existing relation symbol on a diagram. This dialog box allows the user to define the type of relation he or she wants (whether it is association, aggregation, or inheritance), the relation name, and basic cardinality (whether it is one or many) for association and aggregation. One can also provide a description of the relation.

Hitting OK in this dialog brings up the relation pencil tool for a relation. The user can then click on the class they want to connect from and drag to the class they want to connect to. The relation will automatically be fitted between classes.

Double clicking on the name of the relation will bring up the relation dialog again for changes.

4.2 Creating the Data Dictionary - The data dictionary displays text specification information. The steps to create the data dictionary using With Class are:

>> Run With Class from Windows
>> Select "File - Open" e.g., c:\car.omt
>>Select "Specs - System"
>> Enter system specification information, e.g. system name, enclosing system, access, imports
>> Double click on a class, e.g., Car
>> Select "Specification"
>> Enter class documentation, e.g., description, visibility, concurrency, cardinality, persistence

>> Double click on an attribute to bring up the Attribute Specification Form
>> Enter attribute documentation, e.g. visibility, initial value, minimum value, maximum value
>> Double click on an operation to bring up the Operation Specification Form
>> Enter operation documentation, e.g. precondition, postcondition, transformation
>> Select "Generate - Make Data Dictionary" and enter the dictionary file name, e.g., c:\car\carspec.dic
>> Select "Generate - Edit File" and enter the dictionary file name, e.g., c:\car\carspec.dic
>> In the Edit Box, select "File - Exit"

4.3 Generating C++ Source Code - With Class has a C++ code generator to create C++ header and source code files. The steps to generate C++ code using With Class are:

>> Run With Class from Windows
>> Select "File - Open", e.g., c:\car\car.omt
>> Select "Generate - Options" and select C++ code generation options
>> Select "Generate - Generate Code"
>> Select "Generate - Edit File" and select a ".h" or ".cpp" file for review

Options Dialog Box
if you wish certain options on your code generation, go to the options menu item. This brings up a dialog box with several options for your code generation.

Header Files - This generates C++ header files (.h) based on your diagrams.

#ifdefs - This is if you want to use Borland's #ifdef feature for includes in your header files. It prevents multiply defined headers in a link.

Get and Set Methods - These are methods in a class that are generated when this option is turned on. They will allow you to access attributes of your class via operations.

Copy Constructor and Assignment Operator - This is a constructor that will allow you to make a copy of an instance of the class. This feature will only generate copying for attributes, but not for relations.

ClassDef Macro - This is Borland's feature for making pointers and references look nicer by using a capital R for references and capital P for pointers.

Example:

Instead of TString*, you can have PTString

Putting T's in front of class names - This is a feature which puts a capital T in front of all class names.

Add Comments from Description - This inserts the description from the class, attribute, and operation dialog box into the C++ source code.

CPP Files - This generates C++ source code files (.cpp) based upon your diagram.

Insert the prefix - This inserts the designated letter in the class name.

Make All Classes Streamable - This makes the classes persistent as Borland Stream derived classes.

ostream << to Display Classes - This creates the operator<< for use with cout to display attribute values.

Many Options - If you have a one to many relation, you may want to generate a C array, or you may have a collection library that you would rather use. This option lets you choose which one you will use for all your code. The collections given will generate Borland's template classes and include files. You can choose your own template collection class, but don't forget to provide an include file.

Guidelines for Selecting Generate Options

1. Select defaults for new users. Turn off defaults to see the minimum C++ code to be generated by With Class.

2. Select #ifdefs for projects with shared files. With Class generates the compiler directives #ifndef, #define, #include, and #endif to avoid "multiple declaration errors".

3. Select Get Method and Set Method to generate get and set accessor functions for attributes (data members), e.g. setGasQuantity (15.0). Accessor functions are used in place of assignment statements, e.g. gasQuantity = 15.0. Note: accessor functions also allow you to access aggregation and association related attributes.

4. Select "Copy Constructor and Assignment Operator" to generate the following:
copy constructor to create an object with a copy of another object, e.g. Car car1 (car2).
= assignment operator to assign one object with another object, e.g. car1 = car2.
== is equality operator to compare two objects, e.g. if (car1 == car2) .

5. Select "CLASSDEF Macro" if you desire to have P before

pointer variables and R before reference variables.

6. Select "Add Comments from Description" to have text descriptions from the data dictionary transferred to the C++ source code.

7. Select "Make All Classes Streamable" to generate streamable C++ classes for persistent storage with C++ file streams.

8. Select "ostream<< to Display Classes" with the cout stream object, e.g. cout << car1.

Guidelines for C++ Compiler Directives

With Class automatically generates the following C++ compiler directives from the class diagram and class specification: #ifndef, #define, #include, #endif. These avoid "multiple declaration errors". If you add files to your C++ project after code generation you must enter the compiler directives in the source code. The following is an example of the added compiler directives for a types.h file that was added in the Borland C++ environment.

The types.h file appears as follows:

```
#ifndef __TYPES_H
#define __TYPES_H
#endif
```

The following compiler directives were added to each header file that required access to the types. h file:

```
#ifndef __TYPES_H
#include "types.h"
#endif
```

4.4 Using the C++ Development Environment (Compiler)

Once the code is generated, then the user may go to a C++ compiler to compile, link, and execute the code. With Visual C++ use the default project settings. If you use <<ostream option, use Quick Win settings. To use the Borland C++ environment, the following are sample environment settings for Borland C++ Options:

Application: Large Memory Model
Linker Link Libraries: Static for all libraries
Include Directories: c:\bc\include;
c:\bc\owl\include;c:\bc\classlib\include;
Library Directories: c:\bc\lib; c:\bc\owl\lib;
c:\bc\classlib\lib;
Compiler - Code Generation - Defines WIN31

The steps to Compile the C++ Source Files in Borland C++ are:

>> Run Borland C++ (BCW) from Windows
>> Select "Project - Open Project"
>> Enter a project file name, e.g., carproj.prj
>> Select "Project - Add Item"
>> In the Directories Box, change the directory where the C++ files are located, e.g., Car Directory
>> Enter *.cpp in the File Name Box
>> Select the .cpp files, e.g., main.cpp and car.cpp and click on the "Add Button" to add each file to the project
>> Select "Done"
>> Select "Options - Directories" to update the Include Directories List
>> In the Include Directories Box, add the directory where the C++ files are located, e.g., c:\car
>> Select "OK"
>> Select "Compile - Compile" to compile the C++ source code
>> Select "Run - Run" to compile, link, and execute

To execute the program the user must create a main function. The steps to create a main function and execute the program in Borland C++ are listed below. To execute the C++ source code with messages, you must update the main module with an object declaration and messages to the object.
>> Select "File - Open"
>> Select the main function, e.g., c:\car\main.cpp
>> Enter the C++ statements for the main, e.g., statements shown below
>> Select "File - Save"
>> Select "Compile - Compile" to compile the main function
>> Select "Run - Run" to compile, link, and execute the program

```
#include "car.h"    //Added code to use generated C++
main ()
{
Car car11;//object declaration
car11.setGasQty (10.0);  //function call
car11.start ();//function call
int aGasQty = car11.getGasQty ();  //function call
return (0);
}
```

4.5. Reverse Engineering Class Diagrams from C++ Code
After generated C++ code has been updated in the C++ development environment with algorithmic and other statements, a new class diagram can be reverse engineered.

After C++ code has been generated from a class diagram, many additions must be made to the C++ code for messages, transformations (rules, expressions, equations, algorithms), correctness assertions (preconditions, postconditions, invariants), comments, etc. Later in a project you may desire to create a class diagram from the code you're working on. This is called reverse engineering. It is the creation of a diagram from source code. With Class only reverses updated C++ code that was

originally generated using With Class.

The steps to reverse engineer a class diagram from C++ code using With Class:
>> Ensure all applicable C++ .h and .cpp files are in a single directory
>> Run With Class from Windows
>> Select "Generate - Reverse"
>> Choose the directory of the C++ files
>> Choose the header and cpp files you wish to reverse and add them
>> Select "File - SaveAs" xxx.omt to save the new diagram

Guidelines for Reversing C++ Not Generated in With Class
The reverse engineering capability was designed to work with C++ files that were originally generated in With Class. These files may be updated in an editor or in the C++ environment. However, a user may desire to reverse C++ files that were not originally generated in With Class. Such files may or may not reverse properly. The following are several guidelines to for attempting to reverse such files:
1) Ensure the files compile, link, and execute properly.
2) Ensure there are separate header (.h) and source code (.cpp) files.
3) Ensure there is only one class in each header or source code files.
4) Try reversing only header files first.
5) If there is a problem try reversing each header file one at a time to find the file that is causing a problem. If you find the problem file try commenting out any suspicious looking code.
6) Send any problem files to MicroGold S/W for evaluation and suggested actions to reverse the files.

4.6 Modeling Persistent Objects Using With Class

A persistent object retains its values when the program is not running, e.g., the system is not active. The three primary ways to create persistent objects are with flat file storage, relational data base management system storage, or object oriented data base storage. With Class generates C++ streamable classes for persistent objects with the following steps:

>> Select "Make All Classes Streamable", "Get Methods", and Set Methods" from the Options Menu.

>> Select "Generate Code" from the Generate Menu.

>> Update the generated code as follows:

Create a stream object, e.g. inputStream or outputStream.

Open the stream object, e.g. inputStream.open ("CAR.STM").

Read from or write to the stream object, e.g.

inputStream >> car1 or outputStream << car1.

Close the stream object, e.g. inputStream.close ().

As an example, a class diagram was created with a single class Car with the data member "float gasQuantity". With Class automatically generated the get and set functions: "getGasQuantity" and "setGasQuantity". The following main function shows how to create, open, read from, write to, and close a C++ a file stream. In this example, the car1 object was written to and read from a C++ file stream for persistent storage.

```
#include "car.h"    //Added code to use generated C++

#include <iostream.h>
int main ()    //Implements MainUser
{
Car car1;
car1.setGasQuantity (20.0);
ofpstream outputStream;  //Creates file stream
outputStream.open ("CAR.STM");
//Opens file stream
if (outputStream.bad () )
cout << "Could not open file/n";
outputStream << car1;    //Writes to file stream
outputStream.close ();   //Closes file stream
Car car2;
ifpstream inputStream;   //Creates file stream
inputStream.open ("CAR.STM");//Opens file stream
if (inputStream.bad())
cout << "Could not open file/n";
inputStream >> car2;//Reads from file stream
cout << car1.getGasQuantity ();
cout << car2.getGasQuantity ();
inputStream.close();//Closes file stream
return 0;
}
```

Appendices

References

[Booch-91] Booch, Grady Object Oriented Design with Applications. The Benjamin/Cummings Publishing Co 1991.

[Booch-92] Booch, Grady The Booch Notation: Part I and II Computer Language Sept and Oct 1992.

[Coad-91a] Coad, Peter, and Yourdon, Edward, Object Oriented Analysis, Second Edition, Prentice Hall, Englewood Cliffs, N.J. 1991.

[Coad-91b] Coad, Peter, and Yourdon, Edward, Object Oriented

Design, Prentice Hall, Englewood Cliffs, N.J 1991.

[Coleman-94] Coleman, Derek, Arnold, Patrick, Bodoff, Stephanie, Dollin, Chris, Gilchrist, Helena, Hayes, Fiona, Jeremaes, Paul, Objet-Oriented Development The Fusion Method, Prentice Hall, Englewood Cliffs, N.J. 1994.

[Embley, David, Kurtz, Barry, Woodfield, Scott, Object-Oriented Systems Analysis A Model-Driven Approach, Prentice Hall, Englewood Cliffs, N.J 1991.2

[Felsinger-93] Felsinger, Richard Object Oriented Analysis and Design Seminar Course Notes

[Jacobson-92] Jacobson, Christerson, Jonsson, Overgaard Object-Oriented Software Engineering A Use Case Driven Approach,Prentice Hall, Englewood Cliffs, N.J

[Meyer-88] Meyer, Bertrand Object-Oriented Software Construction Prentice-Hall, Englewood Cliffs, N.J. 1988.

[Meyer-92] Meyer, Scott Effective C++ Addison-Wesley Publishing Company, Reading MA.

[Rumbaugh-91] Rumbaugh, J.,Blaha, M., Premerlani, W., Eddy, K. and Lorensen, W. 1991. Object-Oriented Modeling and Design, Prentice Hall, Englewood Cliffs, N.J.

[Shlaer-88] Shlaer, Sally and Mellor, Stephen 1988 Object Oriented Systems Analysis Modeling the World in Data, Prentice Hall, Englewood Cliffs, N.J.

[Shlaer-92] Shlaer, Sally and Mellor, Stephen 1992 Object Lifecycles Modeling the World in States Prentice-Hall, Englewood Cliffs, N.J.

[Winblad-90] Winblad, A., Edwards, S. and King, D. 1990 Object-Oriented Software Addison-Wesley Publishing Company, Reading MA.

[Wirfs-Brock-90] Wirfs-Brock, R, Wilderson, B, Wiener, L 1990 Designing Object-Oriented Software, Prentice-Hall, Englewood Cliffs, N.J.

Glossary of Key Terms

Action - any response to an event. Actions may be updating an attribute, sending a message, or similar action.

Aggregation (Strong Association) - "Part of" or "bill of material" connection between an assembly and parts that has special semantics for "part of" (transitivity, antisymmetric, and propagation) and for the creation, copy, and deletion of an assembly and parts

Assertion - a rule or expression for correctness, such as a data value must always be greater than zero.

Association - A link, connection, or mapping between two or more objects, such as "has a", "knows about", "part of", or "bill of materials".

Atomic Class - An atomic class is a primitive data type whose objects are not logically decomposable, such as character, boolean, integer, and floating point types.

Attribute - A characteristic or property of an object. An attribute has a name (ID), class or type, and a value.

Class - A definition for one or more objects that have common attributes, common behavior, common relationships, and common semantics. In S/W a module that encapsulates attributes, operations, exceptions, and relationships.

Cohesion - the degree of internal relatedness of elements within a larger, more complex entity.

Collection Class - A collection class defines an object that holds other objects called elements. Elements may be added or removed from the collection object. Elements are stored in the collection object with an index. Sample collection classes are array, sets, bags, lists, stacks, queues, rings, trees, etc.

Composite Class - A composite class defines an object that has attributes and associated/part objects.

Condition - a guard or boolean that may affect the stimulus response logic. A condition is a guard or boolean expression signifying OK or NOTOK that are used in IF Condition = True THEN DoSomeAction. Examples of conditions in a Temperature Class might be "temperature high" and "temperature OK".

Control Class - defines time oriented or state based stimulus response behavior including rules and logic to respond to events.

Coupling - the degree of interconnectivity, interdependence, joining and linking of entities.

Dynamic Binding - Late binding (association) of an object name with an object and its class at run time. The object name may later be associated with a different object and its class. Also, the run time look-up of the correct version of a polymorphic operation.

Entity Class - manages and computes application information. An entity class is independent of protocol details of other interacting systems.

Event - an occurrence or physical phenomenon in the external environment that occurs ar a point in time such as a user pressing a button to which one or more systems must respond. For finite state machines, an event is any stimulus to an object that results in some action and that may result in a transition to a new state.

Exception - An abnormal, unusual error condition that may result in an operation performing incorrectly. An exception check, such as a precondition or postcondition check, is an expression that detects the presence of an exception and invokes an exception handler. An exception handler takes some action, such as attempts correct the abnormal condition or notify a user.

Finite State Machine - an entity that has state based stimulus response behavior in which there may be different actions from the same event depending upon the state of the entity.

Generalization Specialization - An "is a" or "type of" connection between superclasses and subclasses.

Generic Parameterized Class (Template) - a class that can be modified with parameters to contain or operate on objects of the parameter class, such as a parameterized Stack Class.

Inheritance - The capability of a subclass receive for use attributes and operations defined in a superclass.

Initializer Class - a top level class that initializes the system. It becomes the "main" in a C++ program.

Interface Class - defines communications with another interacting system. An interface class handles events and/or responses. An interface class isolates protocol details to communicate with other interacting systems.

Invariant - a general rule or expression that must be satisfied at all times by all applicable operations.

Message - A call to an object of a class to invoke one of its operations. A one way message is in one direction only from a requester to a server. A two way message is a peer to peer message objects both send and receive messages from each other, i.e. each can initiate a message to each other.

Object Oriented Modeling - A term referring to the modeling phases of object oriented S/W development including analysis, design, and prototyping. It does not include implementation, productization, testing, etc..

Object Oriented Design - A software development methodology (set of steps) to build systems consisting of classes and objects.

Object - A thing; an instance of a class. An entity that has

state (retained information), has behavior (responds to messages), sends and/or receives messages from other objects, and has relationships with other objects. In S/W a variable defined by a class.

Object Oriented Programming - A method to develop software using inheritance, dynamic binding, and polymorphism with object oriented languages such as Smalltalk, Actor, C++, Eiffel, Object Pascal, etc.

Operation precondition - a rule or expression that must be satisfied before the execution of an operation for correct results.

Operation postcondition - a rule or expression that is satisfied upon the correct execution of an operation.

Operation - An action, service, procedure, function that performs some action in response to a message.

Pattern - Two or more entities with a well defined purpose, behavior, connections, and structure, such as a tree pattern.

Polymorphism (one name many forms) - An object name may refer to objects of different classes. An operation name may refer to different implementations.

Problem Domain Object - an object that exists in the system environment that is passed in a message to or from the system.

Relationship - A link or connection between classes or objects e.g. association "has a", aggregation "part of", and generalization specialization "is a" or "type of".

Response - a message from the system to other H/W and S/W systems. A response implements some action requested in an event.

State represents a mode of behavior that has a unique combination of events, conditions, actions, and next state. A state is static, i.e. waiting for an event to arrive. While in a state, a defined set of rules, laws, and policies apply. A state is like a manager or coordinator that knows how to respond to each event according to his rules, laws, and procedures.

Static Binding - The association of an object name (ID) with an object and its class at compile time. The object name (ID) is permanently bound to the object and its class for the life of the program.

Subclass - A refined, more specific class of a superclass. It defines more specific, attributes, operations, and exceptions.

Subsystem - a component of a larger system environment. A

subsystem has components, such as smaller subsystems or classes that are connected together.

Subtype class - defines a specialized class. An object of the subtype may be substituted for an object of the supertype. Objects of supertype and subtype respond to the same messages.

Superclass - A general class that defines the most general attributes, operations, and exceptions which may be inherited by subclasses.

Supertype class - defines a general class that has the same operations (same protocol) as the specialized subtype classes.

System Environment - a complex system that has other systems (subsystems) as components.

System - a general term for a complex entity that can be treated as a unit and that has simpler components that work together to perform a function. The system is the the S/W system to be developed. The system which is the center of focus and which becomes a single program (.EXE file). The system generally consists of 10 to 100 classes as a very rough order of magnitude.

Transformation - a description of how a data value may be correctly changed in a formula, expression, table, etc.

Transition - a unique pattern of an event, conditions, actions, and a destination state. For each state identify applicable events. Then for each event identify the applicable conditions, actions, and the destination state.

Weak Association - "Has a" or "knows about" connection between associated objects that does not have aggregation semantics.

Frequently Asked Questions

1. What do I do if I get a General Protection Fault? On rare occations With Class has a memory conflict with another program or video driver. If the GPF occurs on start up, try different screen resolutions (starting with standard VGA 640x480) and the lowest number of screen colors (16). Then increase the screen resolutions. Try to obtain the latest video driver from the manufacturer. Also, remove any memory resident software. Problems have been reported with PC Tools, QEMM, and Diamond Video cards. Notify Microgold of any GPF's especially the GPF number.

2. What do I do if the generated C++ source code is placed in the root directory. Save your diagram in the directory that you desire. Select "File Edit File" to ensure you are in the correct directory. Use "File Set Directory" to create a new directory.

3. How do you integrate With Class with StateMaker?

Use StateMaker in C++ mode to generate a method (e.g. process)
Make a method called process in your concurrent class using With Class.
Cut and paste the generated code into With Class's process operation
either using the class dialog or the browser.

4. How do I bring new C++ classes into With Class?
Use the reverse feature in With Class to reverse the new class code
into an existing diagram. Attach any necessary relations.

Examples and Product Advisory provided by

Richard Felsinger
RCF Associates
Object-Oriented Software
960 Scottland Drive
MT PLEASANT, SC, 29464
(803)881-3648
CompuServe 71162,755
Internet - felsingerr@citidel.edu

Ordering Info

To Order Call 1 - 908 - 722 - 6438
Visa/MC accepted.

Or send order to:
MicroGold Software
696 Birch Hill Drive
Bridgewater, NJ, 08807
1 - 908 - 722 - 6438
CompuServe - 71543,1172
Internet - microgold@delphi.com

merchantablility, or fitness for any purpose. It is the sole responsibility of the purchaser to determine the suitablility of the products for any purpose. To the original purchaser only, MICROGOLD SOFTWARE warrants the media to be free from defects in material for 90 days. If during the 90 day period a defect should occur, the software may be returned to MICROGOLD software, who will replace the media at no charge. If after the 90 day period you media becomes defective, the media may be returned to MICROGOLD SOFTWARE for replacement at a $10 service charge. In no case will MICROGOLD SOFTWARE be held liable for direct, indirect or incidental damages resulting from any defect in the disk or omission in the manual, or other related items and processes, including, but not limited to any interruption of service, loss of business, anticipated profit, or other consequential damages.