C++ Reverse Engineering Overview

Overview

The reverse-engineered code produces a complete UML Class Diagram. All objects also have all of their attribute information completely filled in from the source code.

The procedure outlined here can be used on any language and methodology. The best way to evaluate GD*Pro*'s reverse-engineering capabilities is by running your own code through it. GD*Pro* provides detailed and comprehensive support for reverse-engineering source code. Both header files and body files can be reverse-engineered.

What is Produced from Reverse Engineering

- UML Class Diagram
  - Classes
  - Class data members and methods
  - Inheritance
  - Class associations
- Implementation Diagram
  - Files processed
  - File "Include Tree"
- Repository completely populated with model data extracted from source code

C++ Reverse Engineering Overview

The primary purpose of reversing C++ code is to look at the structure of the classes from an analysis and design perspective. When you reverse C++ code using GD*Pro* a new system is created with two views, the view describing the classes is an Class Model view and the other is the Implementation Model view.

When GD*Pro* reverses C++ classes, the information about each class attribute is complete and there is no loss of any class information. Embedded information within the C++ class is maintained in the class object on the diagram. In this manner, all implementation details, those details that are not supported by a methodology, are stored for later use by code generation.

The Implementation Model is used to provide the second view that is generated when some C++ code is reverse engineered. It is a model used to represent all the files used in a reversed system. This includes both user and system include files. This model is used to provide support for seamless generation of C++ code from the system, minimizing adverse changes to the reversed code.

The Implementation Model represents all the files for a system in a hierarchical, left to right layout. For example, you can have a single C++ file representing a piece of legacy code. This file might include a single header file. That header file, however, can include quite a large number of files, one doesn't really know until they examine the include structure. GD*Pro* reverse engineering follows these includes and track them completely, showing you the resultant include hierarchy on an Implementation Model view.

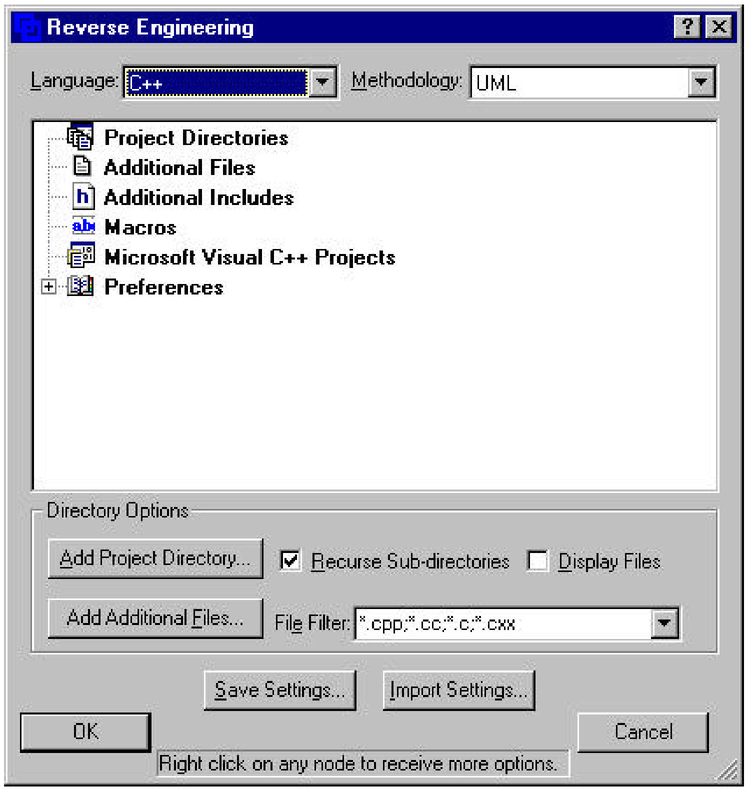
The Implementation Model does more than show a file hierarchy, it also holds all the non-class related information needed to regenerate the code. For example, each file object in this model stores what classes were generated from it enabling code generation to generate code for the classes back into the same files.

©2000 Advanced Software Technologies, Inc.

Starting the Reverse Engineering Process

The Reverse Engineering process is outlined below.

1. Choose TOOLS->REVERSE ENGINEER from the menu. The Reverse Engineering dialog box is displayed.



2. Make a choice from the "Language" drop down list. Your options are C++, Java or IDL. Choose C++ for this tutorial and the following information appears in the Directory List Box:

Project Directories	This shows the various directories that will be interrogated for files with extensions indicated in the File Filter text box. You can add directories by right-clicking the Project Directories title or by selecting the "Add Project Directory" button.
Additional Files	Right-click this option to add any files not found in the Project Directories tree that you would like to reverse engineer. You can also delete all additional files. Clicking the Add Additional Files button also allows you to add or delete files.
Additional Includes	Add any additional directories that should be included during the reverse engineering process.
Macros	Specify macros you would like the reverse engineering process to recognize.
Microsoft Visual C++ Project	Add files, macros and include directories from your MSVC projects and workspaces.

©2000 Advanced Software Technologies, Inc.

Preferences The following preferences can be set:

- Use System Includes
- Hide Class Attributes
- Hide Class Operations
- Save Intermediate Files
- Enable Comment Capture

3. Select UML from the "Methodology" drop down list. Currently UML is your only choice.
4. Check the Recurse Subdirectory option if you want added directories (both Project and Additional Includes directories) to recurse subdirectories when creating new nodes. If a recursion is done, only directories with files matching the filter are added.
5. Check the Display Files option if you want all the files displayed in the Project Directories tree. A ⊞ appears next to all the project directories. Click this icon to expand the tree and display all the files in the directory.

Project Directories

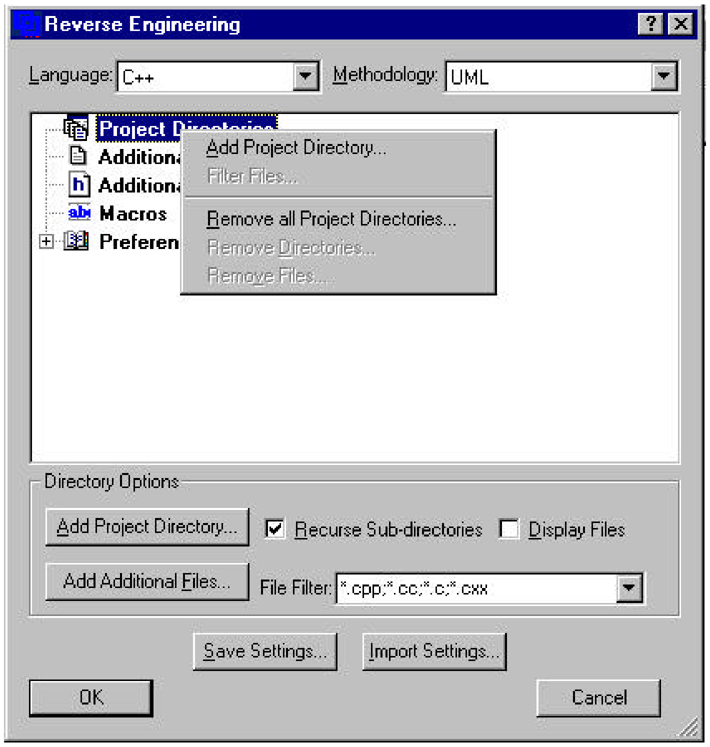
Note: The following section on Project Directories is for information only. Because of the size of the sample project directory we will not use a project directory, only selected files from this directory.

You can add project directories containing files to be reverse engineered by doing the following steps.

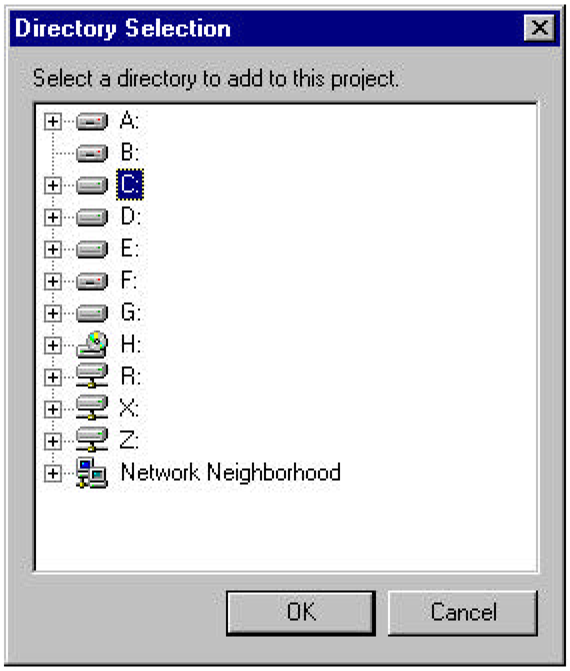
1. Right-click on the Project Directories title in the files list box and a background menu opens.

Note: You can also gain access to the Directory Selection dialog by by clicking the Add Project Directory... button. Another method of adding a directory is to "drag and drop" a directory directly from the Windows Explorer into the Project Directories tree.

©2000 Advanced Software Technologies, Inc.

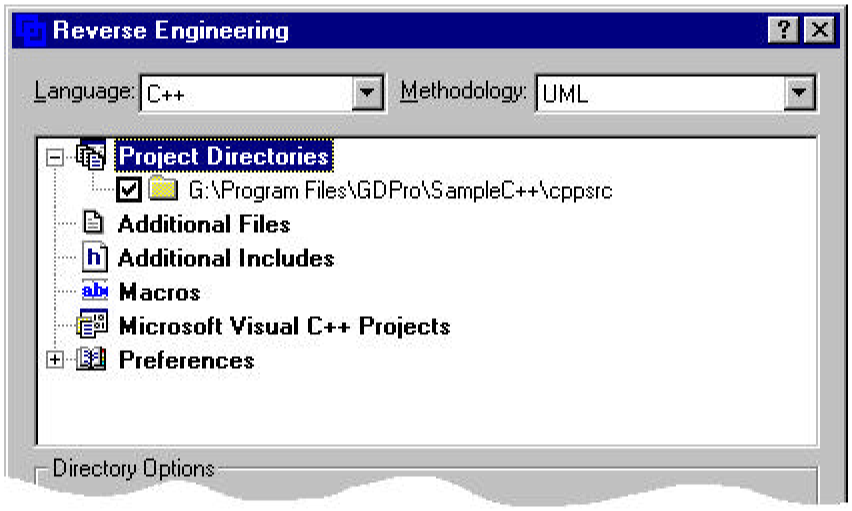


2. Choose Add Project Directory and the Directory Selection dialog box opens.



3. Select a directory and click OK. All files matching the File Filter are added. If the Recurse Subdirectories toggle is on then the selected directory and all sub-directories are searched. The Directory Selection dialog box closes and the selected directory appears in the list box of the Reverse Engineering dialog box.

©2000 Advanced Software Technologies, Inc.

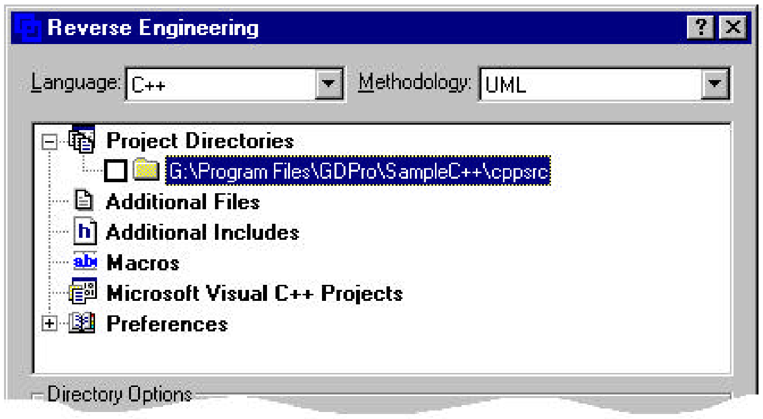

Reverse Engineering

Language: C++ Methodology: UML

- Project Directories
  - [✓] G:\Program Files\GDPro\SampleC++\cppsrc
- Additional Files
- Additional Includes
- Macros
- Microsoft Visual C++ Projects
- Preferences

Directory Options

4. You can make the Project Directory a Reference Include Directory. To have this directory included as a reference library only, remove the check mark from the option box to the left of the directory path.


Reverse Engineering

Language: C++ Methodology: UML

- Project Directories
  - [ ] G:\Program Files\GDPro\SampleC++\cppsrc
- Additional Files
- Additional Includes
- Macros
- Microsoft Visual C++ Projects
- Preferences

Directory Options

Adding Additional Files

You can add files not found in the Project Directory or select specific files within a Project Directory. You can select one or multiple files for inclusion from a file selection dialog box. The files selected are added whether or not they match the file filter.

1. Right-click on the Additional Files title in the files list box and a background menu opens.
2. Select the Add Additional Files command from the menu and the Open dialog box appears. Notice that the types of files listed in the Files of Type text box correspond to the Language selection you made in the Reverse Engineering dialog box.

Note: You can also click the

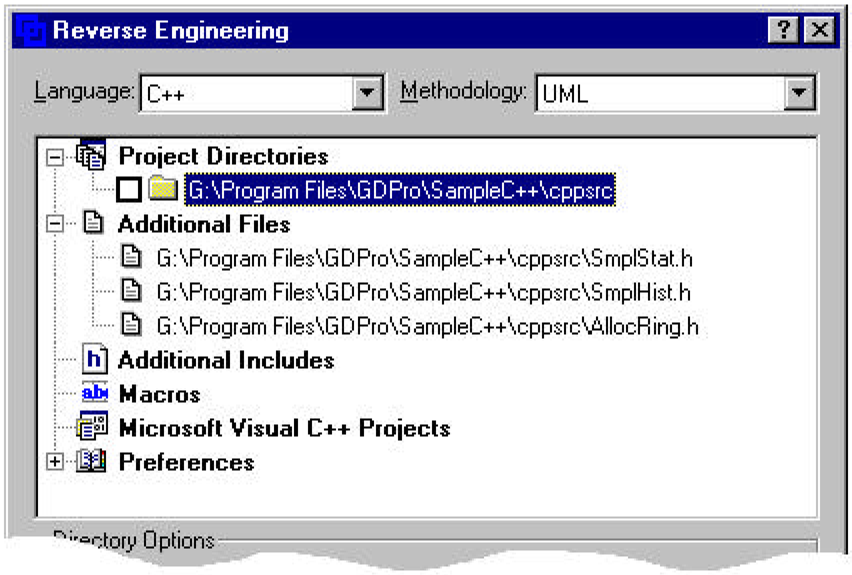
button.

3. Select the following files from *x:\<path>\GDPro\SampleC++\cppscr* and click Open. The dialog box closes and the files you selected appear in the list box of the Reverse Engineering dialog box. You can also "drag and drop" files directly from the Windows Explorer into the Additional Files tree.

SmplStat.h
SmplHist.h
AllocRing.h

©2000 Advanced Software Technologies, Inc.

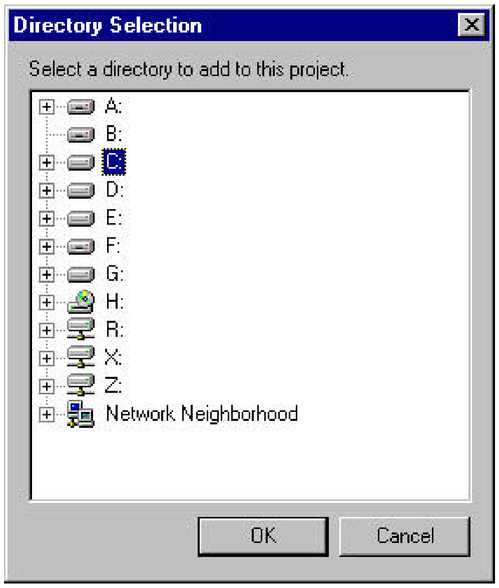
Note: Use the Control key to select multiple, non-adjacent files from the file list. The Shift key allows you to select a group of adjacent files in the list.


Reverse Engineering
Language: C++
Methodology: UML
Project Directories
G:\Program Files\GDPro\SampleC++\cppsrc
Additional Files
G:\Program Files\GDPro\SampleC++\cppsrc\SmplStat.h
G:\Program Files\GDPro\SampleC++\cppsrc\SmplHist.h
G:\Program Files\GDPro\SampleC++\cppsrc\AllocRing.h
Additional Includes
Macros
Microsoft Visual C++ Projects
Preferences
Directory Options

Include Files

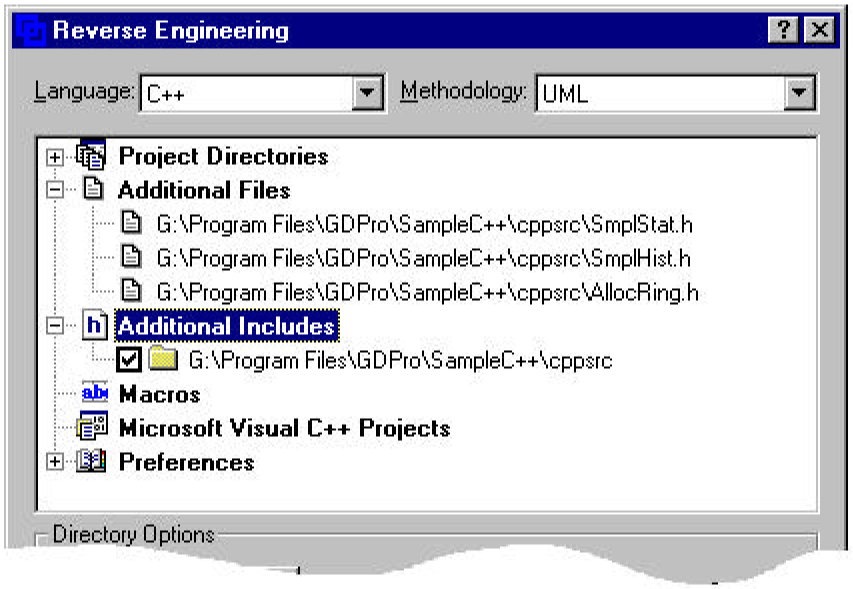
You can add include directories and defines for the current reverse engineering run.

1. Right-click on the Additional Include title in the files list box and a background menu opens.
2. Select the Add Additional Includes Directories command from the menu and the Directory Selection dialog box opens.



3. Select the directory *<path>/GDPro/Sample++/cppscr* directory and click OK. The Directory Selection dialog box closes and the selected directory appears in the list box of the Reverse Engineering dialog box. You can also "drag and drop" an include directory directly from the Windows Explorer into the Additional Includes tree.

©2000 Advanced Software Technologies, Inc.


Reverse Engineering
Language: C++
Methodology: UML
Project Directories
Additional Files
G:\Program Files\GDPro\SampleC++\cppsrc\SmplStat.h
G:\Program Files\GDPro\SampleC++\cppsrc\SmplHist.h
G:\Program Files\GDPro\SampleC++\cppsrc\AllocRing.h
Additional Includes
G:\Program Files\GDPro\SampleC++\cppsrc
Macros
Microsoft Visual C++ Projects
Preferences
Directory Options

To Preserve C++ Macros

You are able to select a small, well defined subset of C++ macro expansions for use in round trip engineering. Simple macro instance preservation functionality in now available.

A set of default macros suitable for use with Microsoft's MFC are included. You can specify that some of your own macros should be saved during round trip. These macros are specified by creating a text file containing one or more macro names, for example:

MySimpleMacros.txt

MY_MACRO_1

MY_MACRO_2

To select this list for use in reverse engineering, enter a pseudo-macro into the Macros area of the reverse engineering set-up dialog. This macro definition looks like:

GDPRO_SIMPLE_MACRO_LIST="<full path to the macro list file>"

There are no spaces allowed in this pseudo-macro definition except for any required spaces in the path to the macro list file. Macros selected for use in reverse engineering are only preserved if they look more or less like those in these two examples.

Example 1 - Simple Macro

```
class UsesMacro
{
  // normal class member
  int x;
  // macro
  MY_MACRO_1()
};
```

©2000 Advanced Software Technologies, Inc.

Example 2 - Function-like Macro

```
class UsesFunctionMacro
{
  // normal class member
  int x;
  // macro
  int MY_MACRO_1();
};
```

Parameters

The parameters for a simple macro are as follows:

1. The preceding non-whitespace non-comment character must be one of '{', '}', ';', or ')'.
2. The following non-whitespace non-comment character must be something other than ';', ':', or '{'.

The parameters for a function-like macro are as follows:

1. The macro cannot substitute for C++ tokens which separate declarations. These are curly brackets ('{' and '}') and semicolons (';').
2. If the macro takes arguments, none of the arguments can be literals (string, character, or numeric).

Examples of Acceptable Function-like Macros Include:

```
INT OPERATION_NAME(TYPE1 a);
(INT, OPERATION_NAME, and TYPE1 are macros)
TYPE1 ATTRIBUTE_NAME;
(TYPE1 and ATTRIBUTE_NAME are macros)
SPECIAL_OPERATIONS(A, B);
```

(SPECIAL_OPERATIONS is a macro. Note that this is just like a simple macro except that it ends in a semicolon.)

Examples of Non-acceptable Function-like Macros Include:

```
int myFunction() FUNCTION_BODY
int myAttribute SEMICOLON
SPECIAL_OPERATIONS("special ops", 3);
```

- An attempt to preserve the FUNCTION_BODY or SEMICOLON macro instances results in parse errors during reverse engineering.
- An attempt to preserve a macro which uses a C++ literal in its argument list results in either parse errors or in these macro instances being silently dropped during reverse engineering.
- A function-like macro appears in GD*Pro* as a class attribute or operation, depending on the similarity in appearance between the unexpanded macro and a normal C++ attribute or operation.
- Simple macro instances are not visible in the GD*Pro* class diagram. They are stored in a separate area for use with round-trip engineering.

Predefined Macros Under Reverse Engineering

©2000 Advanced Software Technologies, Inc.

WIN32

_WIN32

i386

WINNT

_X86_

unix

ALMOST_STDC

_MSC_VER (set to 800)

MFC Macros.h

ATL Macros.h

Reference Classes

Note: The following section on Reference Classes is for information only. The tutorial does not contain an example of reference classes.

Reference classes are classes that do not participate in the code generation process. They are automatically created by reverse engineering when classes in the reverse engineered set of files have inheritance relationships with classes that are not defined in the code that is being reverse engineered. Reference classes complete the relationships, but do not have any content (no attributes or operations defined), and will not have any code generated for them.

Once you have your library of reference classes, you can use them as components in creating a new system by:

a. opening the system with the reference classes

b. going to the navigator

c. clicking on the class you want to use in your new system

d. zooming to the class

e. then copy and paste it into your new class diagram

You can inherit or create associations to the reference class, and GD*Pro* knows not to generate the header and body files for the reference class.

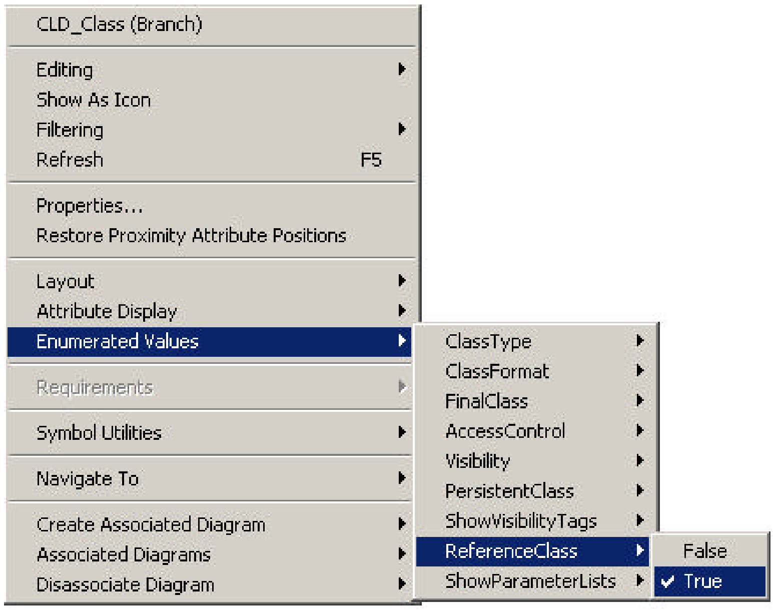
Setting a Reference Class

1. Open the class diagram in the view area
2. Select the class you want to make a reference class.
3. Right-click in the background. A background utilities menu opens.

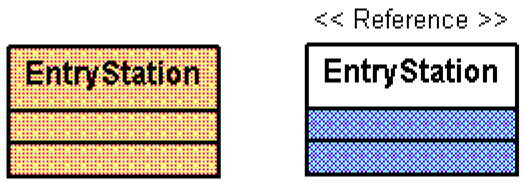
Note: Make sure you click in the background of the diagram. Do not click on the class itself.

3. Select ENUMERATED VALUES->REFERENCE CLASS->TRUE from the background menu.

©2000 Advanced Software Technologies, Inc.

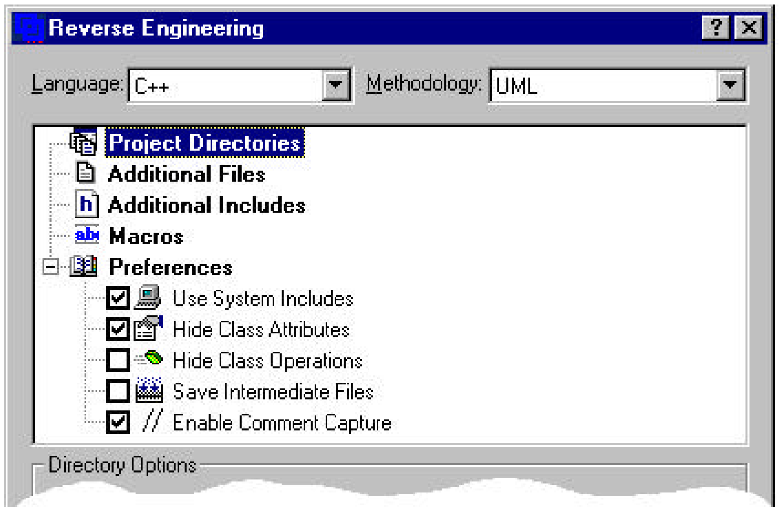


4. The selected class is now a reference class. You can also toggle the reference classes back to "live" classes by selecting ENUMERATED VALUES->REFERENCE CLASS->FALSE.



Setting the Preferences

1. Click the ⊞ located to the left of Preferences title. The list expands showing the available preferences.


Reverse Engineering

Language: C++ Methodology: UML

- Project Directories
- Additional Files
- Additional Includes
- Macros
- Preferences
  - [✓] Use System Includes
  - [✓] Hide Class Attributes
  - [ ] Hide Class Operations
  - [ ] Save Intermediate Files
  - [✓] Enable Comment Capture

Directory Options

©2000 Advanced Software Technologies, Inc.

Use System Includes	Currently this option provides functionality only for C++ and only on the UNIX platform.
Hide Class Attributes	Hides the class attributes. These class attributes are captured and can be accessed through the reporting function.
Hide Class Operations	Hides the class operations. These class operations are captured and can be accessed through the reporting function.
Save RE Intermediate Files	The Save RE Intermediate Files option allows you to save the preprocessed files so you can track any syntax errors in the code.
Enable Comment Capture	The reverse engineering process will capture comments from the source code and add them to the model as class, attribute and operation descriptions. The forward engineering process applies the current descriptions from the model to the source when doing code generation.

2. Click the option button to select the "Use System Includes" preference.
3. The "Hide Class Attributes", "Hide Class Operations" and "Save Intermediate Files" preferences should not be selected.
4. Make sure the "Enable Comment Capture" is selected.

   The preferences you set are implemented when you start the reverse engineering process.

Note: When reverse engineering large diagrams, you should hide the class attributes and operations because of the size of the diagram generated.

Comment Capture

The reverse engineering process will capture comments from the source code and add them to the model as class, attribute and operation descriptions. The forward engineering process applies the current descriptions from the model to the source when doing code generation.

- Most comments which precede or are contained by a C++ class definition are preserved.
- Most comments appear as part of a class symbol in the class diagram and can be viewed and edited using the GDPro attribute editor. Preserved comments will appear in reports and in code generated during forward engineering.

Simple Class Comment Capture

Here is a simple class which demonstrates GDPro 4.0's handling of comments.

```
// This comment goes into the Class Description.
class wellDocumented {

// This comment is not visible in the class diagram, but it is stored
// for use during forward engineering.
```

©2000 Advanced Software Technologies, Inc.

```
private:

  int x; // This comment goes into the attribute description
         // for int x.

  int y; // This comment goes into the attribute description.
         // for int y.

  // This comment goes into the attribute description for int z.

  int /* this embeddedcomment is lost */ z;

  // This comment goes into the operation description for wellDocumented().

  wellDocumented();

  // This comment, which is in the middle of an argument list,
  // is not preserved.

  int midOperationComment(int x, /* this embedded comment is lost */ int y);

  // This comment, and the macro instance which follows, are not visible in
  // the class diagram, but are stored for use during forward engineering.

  MY_MACRO()
};
```

Save Settings

Once you have set your preferences, includes, reference include libraries, and definition, you can save these settings for future use.

1. Click the "Save Settings" button. The Save As dialog box opens.
2. Select the directory and then type the name "Tutorial" in the file name text box. The file extension is ".re" and it is the default setting in the Save as Type text box.
3. Click Save, the Save As dialog box closes and a prompt dialog box opens telling you that your file was saved successfully.
4. Click OK to close the prompt dialog box.

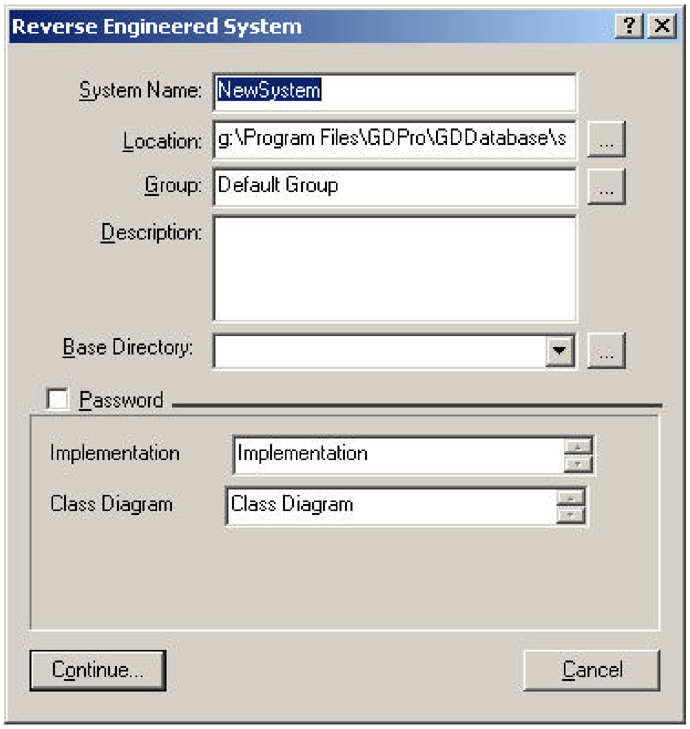
Import Settings

You can also use a previously created file by clicking the Import Settings button. The Open dialog box appears with all the available settings files shown in the list box. Select the file you want to use and click Open. The dialog box closes and the imported settings are in effect.

Naming the New System

1. Click OK at the bottom left of the Reverse Engineering dialog box. The Reverse Engineered System dialog box opens.

©2000 Advanced Software Technologies, Inc.


Reverse Engineered System

System Name: NewSystem

Location: g:\Program Files\GDPro\GDDatabase\s

Group: Default Group

Description:

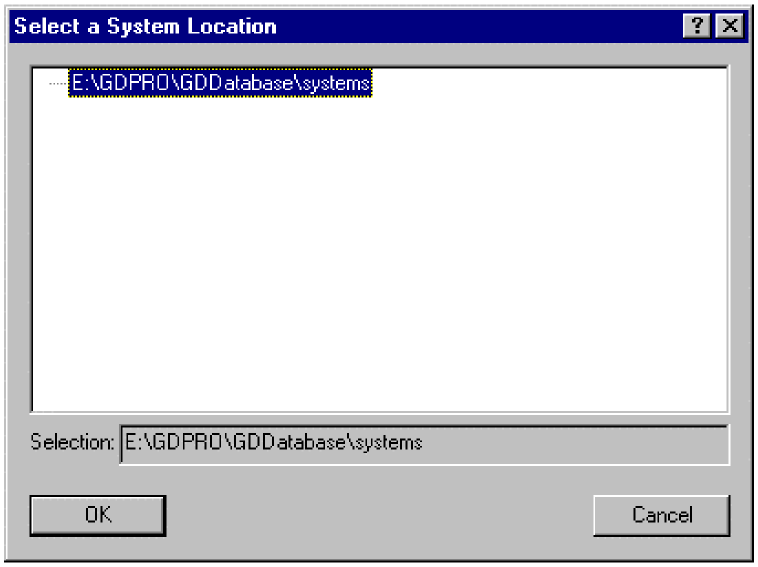
Base Directory:

Password

Implementation: Implementation

Class Diagram: Class Diagram

2. Type the name "RE Tutorial" for your new system. The default name is NewSystem.
3. The default location for your system is *<path>\GDPro\GDDatabase\systems*. If you want to store your system in another location click ⬚ next to the Location text box. The Select a System Location dialog box opens and all available system directories are listed.

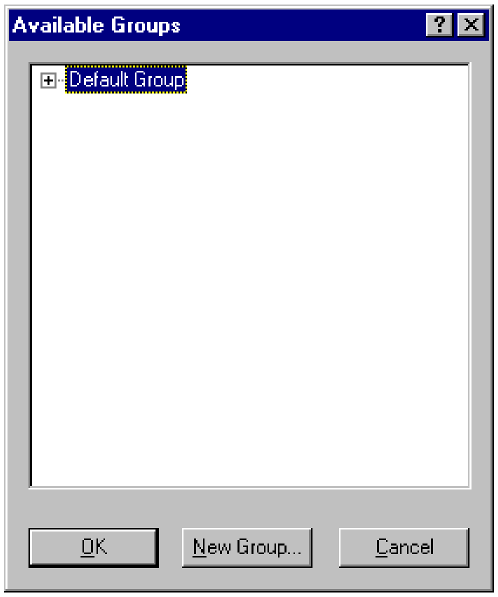

Select a System Location

E:\GDPRO\GDDatabase\systems

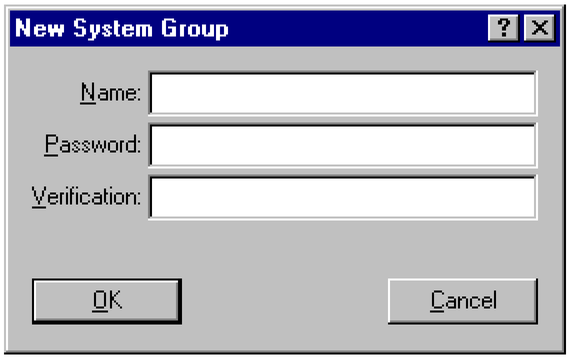
Selection: E:\GDPRO\GDDatabase\systems

4. For this tutorial we will not change the system directory. Click OK and the Select a System Location dialog box closes.
5. The reverse engineering system is automatically placed in a group named Default Group. We will put the system in another group so click ⬚ located next to the Group text box. The Available Groups dialog box opens.

©2000 Advanced Software Technologies, Inc.



6. Create a new group for your reverse engineered system, by clicking New Group... . The New System Group dialog box opens.



7. Type the name "RE_Test" in the Name text box.
8. Type the password "tutorial" in the Password text box. Reenter the same password in the Verification text box and click OK. The New System Group dialog box closes.
9. Select the name "RE_Test" from the Available Groups list box and click OK. The Available Groups dialog box closes and the name RE_Test appears in the Group text box.
10. In the Reverse Engineered System dialog box type the following description: "This is an example of Reverse Engineering". This field is optional.
11. Enter a base directory for this system. If you click the browse button, a Search for Folder dialog box opens. You can select your base directory from this dialog box.
12. The default name for the generated Implementation diagram is Implementation. You can edit this name if you want.
13. The default name for the generated Class Diagram is Class Diagram. You can edit this name if you want.

©2000 Advanced Software Technologies, Inc.

Reverse Engineering Process

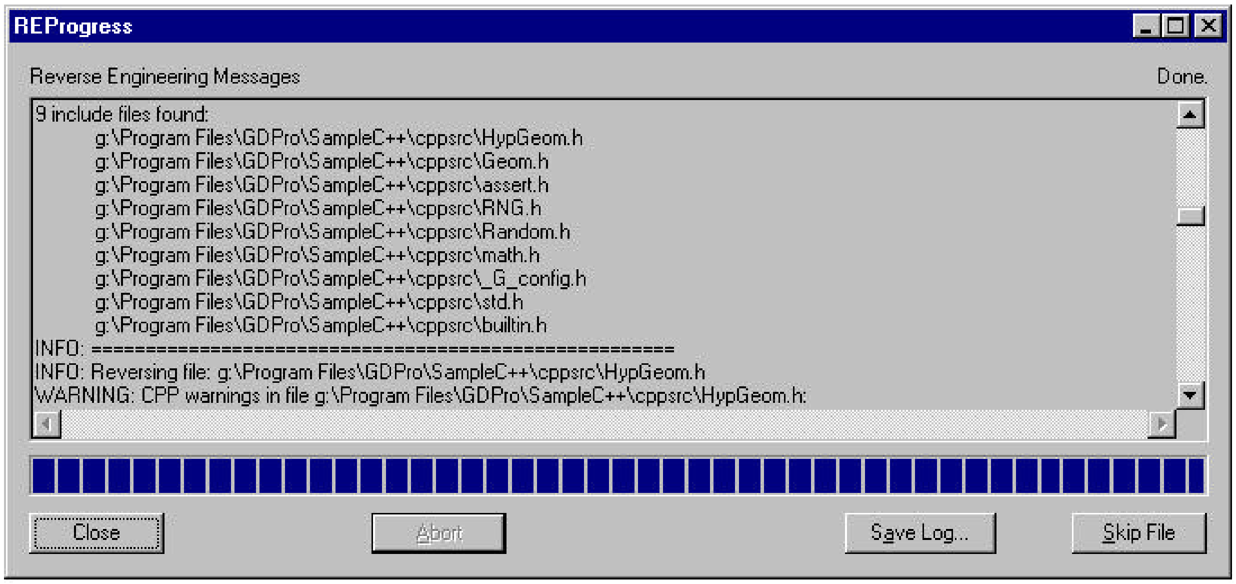
1. After you have completed all the requested information in the Reverse Engineered System dialog box click the Continue button. The REProgress dialog box opens. This displays messages of the reverse-engineering process. The status bar across the bottom of the dialog box displays the progress of the reverse engineering process.

   The following files are included in the reverse engineering process:

   (1) Include files found in a selected include directory

   (2) System Include files

   (3) "double quoted" include files in the source file's home directory.



```
9 include files found:
        g:\Program Files\GDPro\SampleC++\cppsrc\HypGeom.h
        g:\Program Files\GDPro\SampleC++\cppsrc\Geom.h
        g:\Program Files\GDPro\SampleC++\cppsrc\assert.h
        g:\Program Files\GDPro\SampleC++\cppsrc\RNG.h
        g:\Program Files\GDPro\SampleC++\cppsrc\Random.h
        g:\Program Files\GDPro\SampleC++\cppsrc\math.h
        g:\Program Files\GDPro\SampleC++\cppsrc\_G_config.h
        g:\Program Files\GDPro\SampleC++\cppsrc\std.h
        g:\Program Files\GDPro\SampleC++\cppsrc\builtin.h
INFO: ==================================================
INFO: Reversing file: g:\Program Files\GDPro\SampleC++\cppsrc\HypGeom.h
WARNING: CPP warnings in file g:\Program Files\GDPro\SampleC++\cppsrc\HypGeom.h:
```

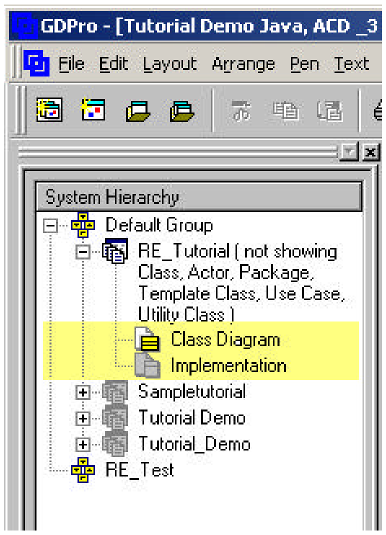
   As the reverse engineering process continues, you can skip any files you don't want included. Click Skip File and the file that is currently being processed is skipped and a message appears in the REProgress dialog box. We will not skip any files for the tutorial.

2. You can save the messages/error file by clicking Save Log... . A Save As dialog box opens. Select a directory for the message/error log file and give the file a name. The default name is GDREResults.txt. Click OK and the Save As dialog box closes.

Completed Reverse Engineering Process

1. Once the reverse engineering process is complete the REProgress dialog box becomes an icon in the lower left corner of the screen, and the class diagram opens in the design area. The generated diagram names also appear in the System Hierarchy Window under the group and system name.

©2000 Advanced Software Technologies, Inc.


GDPro - [Tutorial Demo Java, ACD _3
File Edit Layout Arrange Pen Text
System Hierarchy
Default Group
RE_Tutorial (not showing Class, Actor, Package, Template Class, Use Case, Utility Class)
Class Diagram
Implementation
Sampletutorial
Tutorial Demo
Tutorial_Demo
RE_Test

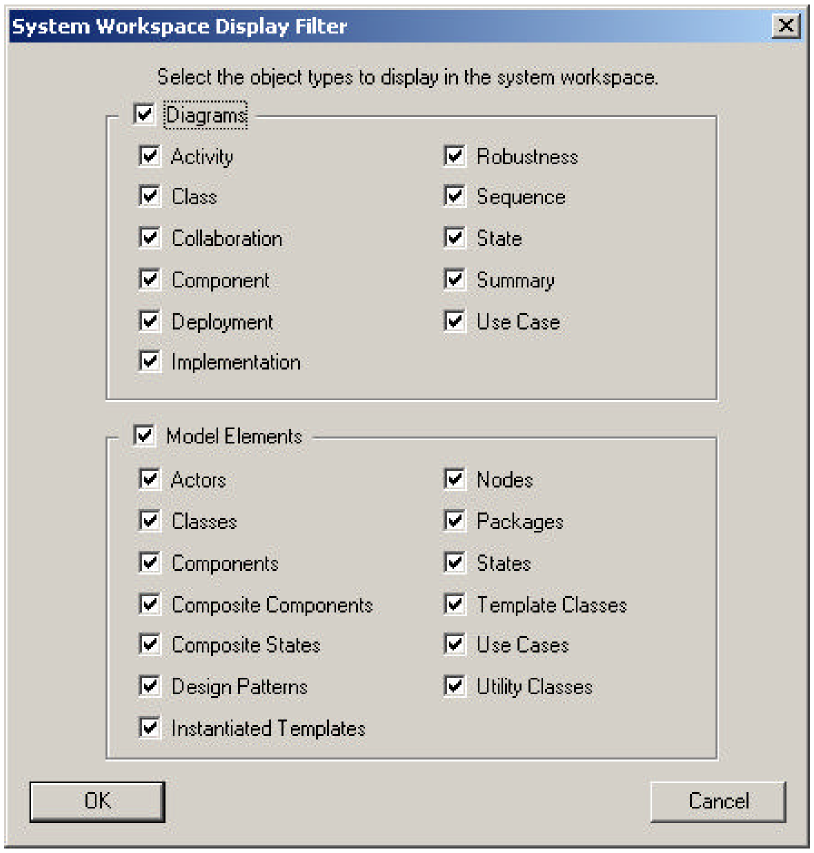
Notice that when the reverse engineering process is complete the system hierarchy tree does not show classes, actors, packages, template classes, use cases and utility classes. To display these objects in the window right click in the area of the opened system in the System Hierarchy Window and the background menu opens.

2. Choose DISPLAY FILTER from the menu and the System Workspace Display Filter dialog box opens. This dialog box displays two different objects to be filtered: Diagrams and Model Elements.

Diagrams:	All the UML diagram types are listed here. Each diagram type is represented by an icon in the System Hierarchy Window. You can display all diagrams or choose to just display diagrams you are currently working with.
Model Elements:	The major modeling elements that are displayed in the System Hierarchy Window are listed here. Each modeling element is represented by an icon in the System Hierarchy Window. You can display all model elements or choose to just display the elements you are currently working with.

You can select an individual object type to be filtered or you can select the option box next to the heading to filter all associated types.

©2000 Advanced Software Technologies, Inc.


System Workspace Display Filter

Select the object types to display in the system workspace.

☑ Diagrams

☑ Activity	☑ Robustness
☑ Class	☑ Sequence
☑ Collaboration	☑ State
☑ Component	☑ Summary
☑ Deployment	☑ Use Case
☑ Implementation	

☑ Model Elements

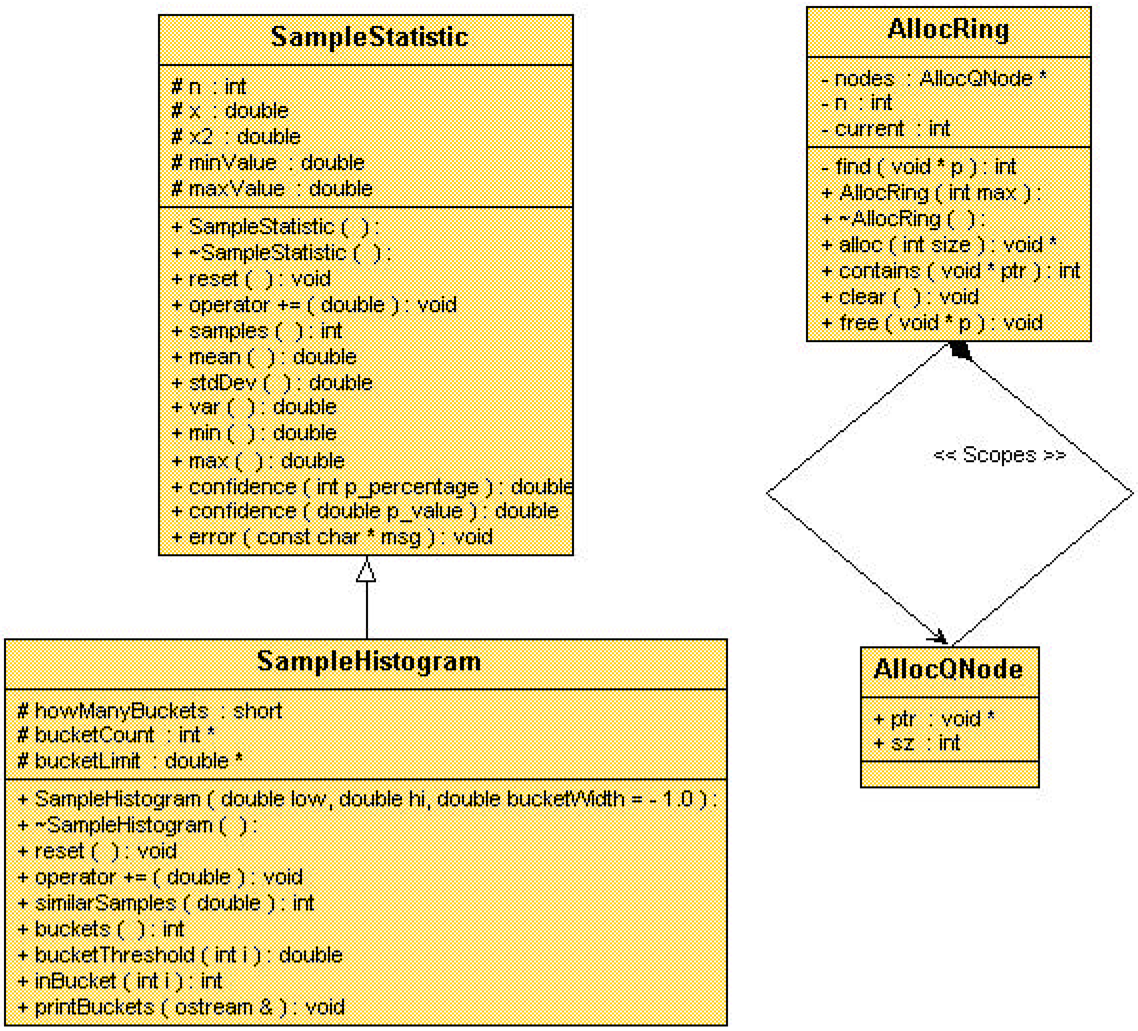
☑ Actors	☑ Nodes
☑ Classes	☑ Packages
☑ Components	☑ States
☑ Composite Components	☑ Template Classes
☑ Composite States	☑ Use Cases
☑ Design Patterns	☑ Utility Classes
☑ Instantiated Templates	

3. Click the object type "Class" to be displayed and click OK. The dialog box closes and the System Hierarchy Window is updated.

The Class Diagram

When the reverse engineering is completed, you should have a UML Object model that describes the C++ source code you reverse engineered.

©2000 Advanced Software Technologies, Inc.



The Class Diagram Reviewed

Let's take a closer look at some of the objects on your UML diagram.

1. When the reverse engineering process is complete, the class diagram created now appears in the design window.

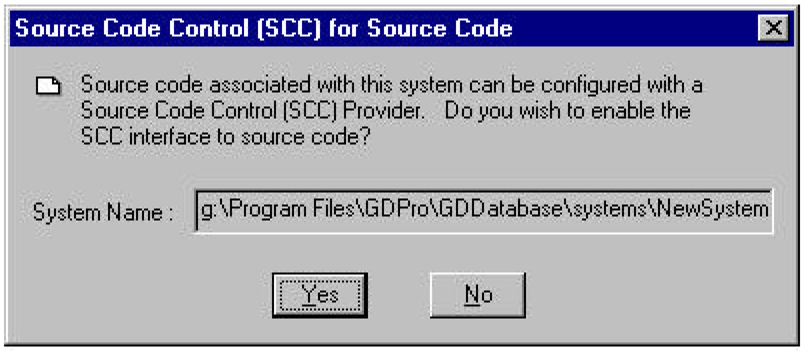
    - All classes are represented on the diagram
    - The classes have their class attributes and operations filled in.
    - Inheritance among classes is shown via the Generalization symbols. For example, the "SampleHistogram" is a subclass of the "SampleStatistic" class.
    - UML Associations are used to represent classes that are associated with each other by pointers or by reference.

©2000 Advanced Software Technologies, Inc.

The Class Diagram Attributes

Unlike many other reverse-engineering tools, GD*Pro* fully reads and parses **ALL** of your C++ code and can therefore completely fill in the attribute editors for all objects.

1. Double-click on the "SampleStatistic" class and Source Code Control dialog box opens.



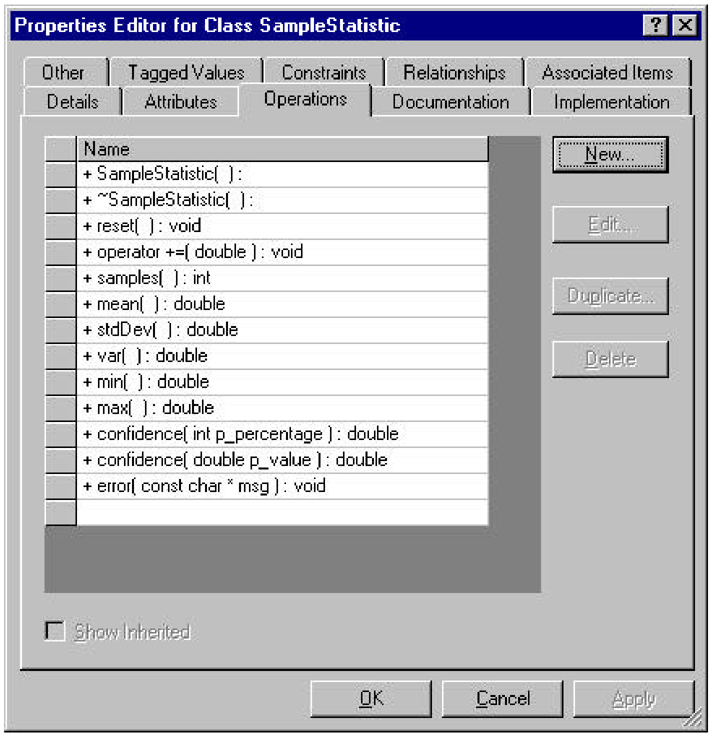
Source Code Control (SCC) for Source Code

Source code associated with this system can be configured with a Source Code Control (SCC) Provider. Do you wish to enable the SCC interface to source code?

System Name : g:\Program Files\GDPro\GDDatabase\systems\NewSystem

2. Please refer to the Source Code Control Overview for information on this function. For purposes of this tutorial click the No button to close this dialog box and to open Properties dialog box for the SampleStatistic class.

3. Click on the Operations tab. All the class operations are filled in.

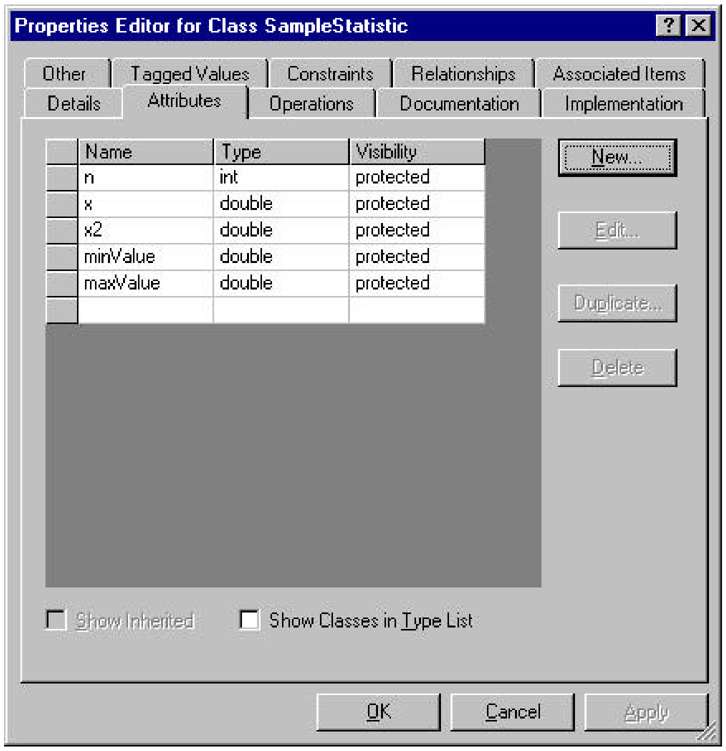


Properties Editor for Class SampleStatistic

Name
+ SampleStatistic( ) :
+ ~SampleStatistic( ) :
+ reset( ) : void
+ operator +=(double) : void
+ samples( ) : int
+ mean( ) : double
+ stdDev( ) : double
+ var( ) : double
+ min( ) : double
+ max( ) : double
+ confidence(int p_percentage) : double
+ confidence(double p_value) : double
+ error(const char * msg) : void

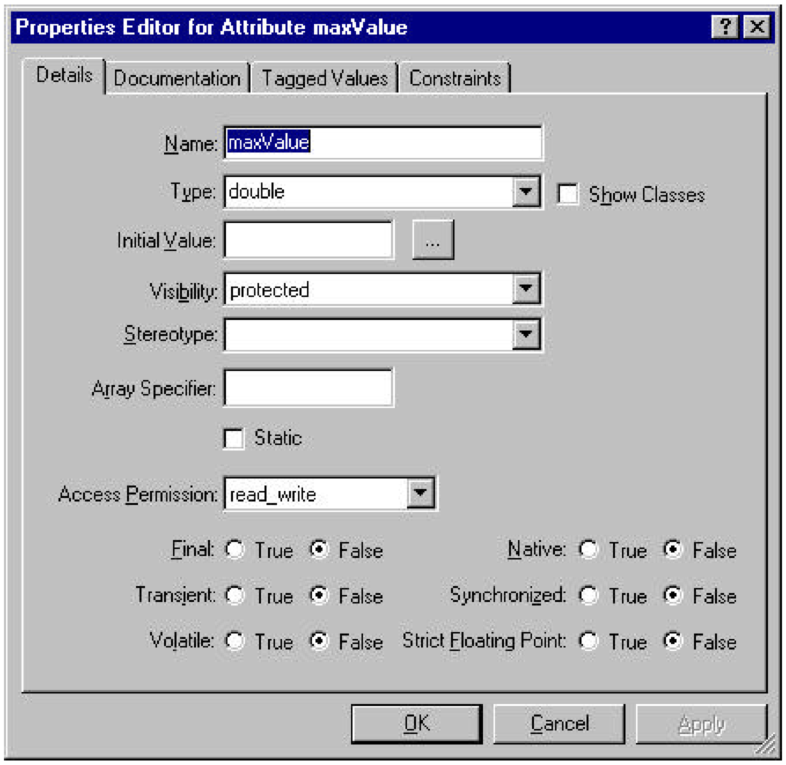
3. Click on the Attributes tab to view all the attributes.

©2000 Advanced Software Technologies, Inc.



Name	Type	Visibility
n	int	protected
x	double	protected
x2	double	protected
minValue	double	protected
maxValue	double	protected

2. Select "maxValue" from the list in the Attributes box and then click the Edit button to view the details of the attributes. The maxValue attribute editor dialog box opens.

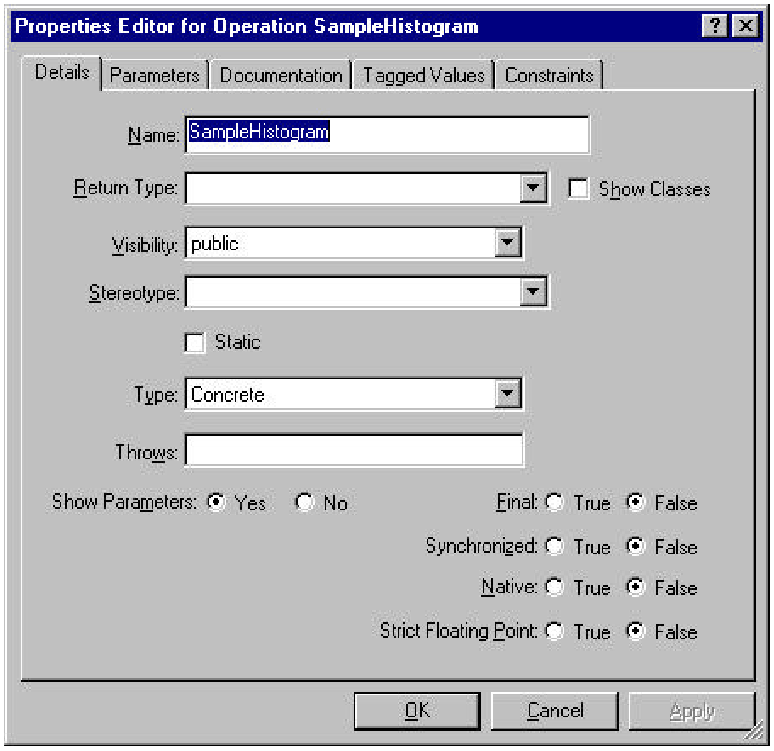


Name: maxValue
Type: double
Initial Value:
Visibility: protected
Stereotype:
Array Specifier:
Static
Access Permission: read_write
Final: True [✓ False]
Native: True [✓ False]
Transient: True [✓ False]
Synchronized: True [✓ False]
Volatile: True [✓ False]
Strict Floating Point: True [✓ False]

Notice that this attribute is of type "double". As you scroll through the dialog box, notice also that it is accessible for reading and writing, indicating it was not declared to be a constant. Constants are read-only.

©2000 Advanced Software Technologies, Inc.

3. Click OK to close the maxValue Properties dialog box.
4. Click OK once again to close the Properties Editor for SampleStatistic.
5. Double-click on the Class labeled SampleHistogram and click the Operations Tab. Select "SampleHistogram (double lo, double hi, double)" from the list box.
6. Click the Edit button the Properties Editor for the Operation SampleHistogram opens.



Name: SampleHistogram
Return Type:
Show Classes: ☐
Visibility: public
Stereotype:
Static: ☐
Type: Concrete
Throws:
Show Parameters: [✓ Yes] No
Final: True [✓ False]
Synchronized: True [✓ False]
Native: True [✓ False]
Strict Floating Point: True [✓ False]

Notice the operation visibility is public and the operation type is concrete.

7. Close this dialog box by clicking OK.
8. Close the Properties dialog box for the class by clicking OK once again.

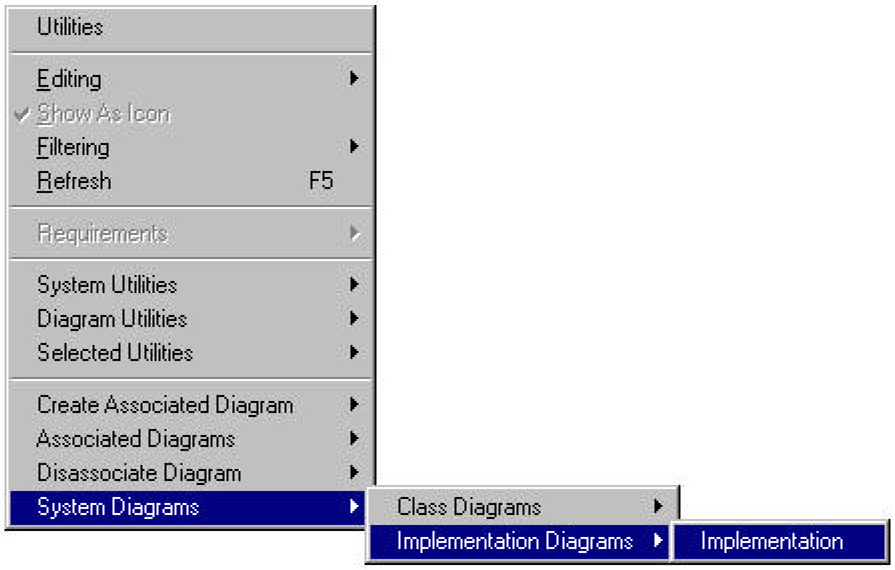
The Implementation Diagram

The reverse engineering process also automatically creates an Implementation diagram. This diagram shows you all the files that were reverse-engineered and the dependencies among them. It displays these files as an “include tree” showing which files include others.

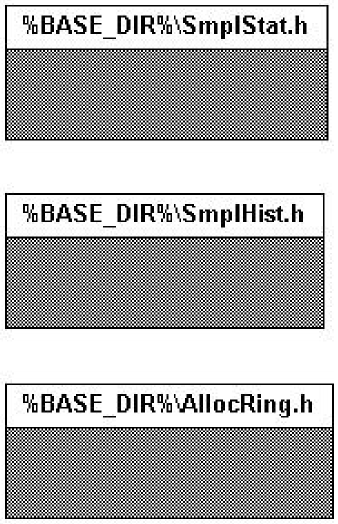
This view is useful to identify which files use a particular file and also to locate circular includes among files. This is particularly important when examining third-party class libraries and creating your own libraries.

1. If the implementation diagram is not the current view, right click anywhere in the background of the class diagram. The Utilities background menu opens.
2. Choose SYSTEM DIAGRAMS->IMPLEMENTATION DIAGRAMS->IMPLEMENTATION.

©2000 Advanced Software Technologies, Inc.



The implementation diagram you created opens.



Round Trip Engineering

With GD*Pro's* Round Trip Engineering you can incrementally develop software, starting either from a new design or from an existing body of code.

- ? You can change the source code and keep design diagrams up to date, using any editor you like.
- ? You can change the design diagrams and keep the source code up to date.
- ? GD*Pro* does not use code markers, leaving your source code just as you wrote it.
- ? Code generation merges new design changes with existing header and body code.
- ? You can edit code to flesh out function implementations.
- ? When you add new structural components, they are synchronized with GD*Pro* through reverse engineering

©2000 Advanced Software Technologies, Inc.

- ? Retains compiler directives and comments
- ? Use whatever source editor you wish - we do not restrict your choice of source editing tools.
- ? We tell you what we are doing - a report is generated with forward engineering that lets you know exactly what changes are made, and what modules they were made in.

Source Editor Pane

The Source Editor Pane allows you to make changes directly to the code. The diagram we just created using Reverse Engineering has read-only code so for purposes of this tutorial, we cannot make changes using the Source Editor Pane.

The Design and Code Generation Process

Once a system has been reverse engineered, GD*Pro* can be used to iterate on the design. The changes can encompass:

- adding new classes
- adding or removing class inheritance
- creating new class associations
- adding, changing and deleting operations and attributes

When the design iteration is complete, code generation can be run on all classes in a system or view, or for specific selected classes.

Code generation merges the current design information for the class into the existing header and body files for the class, if they exist. If the class has been added during the design process, code generation creates header and body files that contain all the information defining the class captured during the design process.

When merging the current design information into existing code, GD*Pro* does the following:

Header File:	GD*Pro* replaces the existing class definition with the class definition generated from the design information. All code outside the class definition section will be left intact.
Body File:	GD*Pro* merges the changes into the body file; adding the framework for new functions, and deleting the definitions of old functions that are no longer in the design.

Reporting Changes to Code

GD*Pro* produces a report during the code generation process that describes the changes made to the old source modules. The report is called CodeGen_Report and contains the following information:

Attribute Added	The attributes are added to the class definition in the header file and attribute accessor functions are created as specified in the design.
Attribute Deleted	The attribute is not included as part of the class definition the class definition.
Attribute Modified	The new definition of the attribute is included as part of the class definition.
Operation Deleted	The operation is not included in the generation of the header file, and the existing operation is not copied into the new version of the body file.

©2000 Advanced Software Technologies, Inc.

Operation Added	The new operation is included as part of the class declaration in the header file, and a stub function is added to the body file.
Operation Modified	(1) The new operation signature is included as part of the class declaration in the header file; (2) the old operation is not coped into the new version of the body file; and (3) a stub function is added to the new body file.
Association Added	A forward reference to the associated class and the associated class elements are added to the header file.
Inheritance Deleted	The class definition no longer references the old base class in the new header file.
Inheritance Added	The reference to the new base class is added to the class definition in the new header file.
Association Added	Adds pointers
Association Deleted	If not reverse engineered, associations are removed.

The code generation process also creates backup files for all source modules it generates. The original file name is appended with an "#orig" notation.

The Reverse Engineering Process

Reverse engineering is the process of evaluating an existing body of code to capture interesting information describing a system, and representing that information in a format useful to software engineers and designers. Advanced Software's reverse engineering for the C++ language processes the source code. It then applies a sophisticated grammatical analysis to capture the elements of class structure and definition, class inheritance, and associations between classes. The captured elements are then rendered in an Object Model representation. This model is a graphical depiction of the class structure, and you can gain access to all of the data elements describing the class and its operations and attributes. Network layout logic is used to create diagrams that are clear and easy to read.

Another product of reverse engineering is the Implementation Diagram. This diagram shows the source and header files that were processed during the reverse engineering process. The Implementation Diagram can give invaluable information and insight into the include file dependency structure of a body of code.

After a system has been reverse engineered, you can generate reports in a variety of formats, including ASCII, HTML and RTF. The report set includes an Object Model Report in HTML format that permit rapid navigation through the class structure, and is linked directly to the source.

Trouble Shooting

Here are most of the errors which GD*Pro* can output during an RE run. The listed errors are those that you can actually do something about.

ERRORS

The environment variable "GDBIN" is not set.	You will need to set your environment variable 'GDBIN', to point to the location of your GDPro \bin directory.

©2000 Advanced Software Technologies, Inc.

Unable to access the methodology.	GDPro 4.x utilizes the UML.gdmeth to create your system. This file must be in a directory that you have read privileges to. GD*Pro* knows how to find this file because of the GDHOME variable. Make sure that the path to GDHOME is correct and that GDDatabas e\methods, contains the file UML.gdmeth.
Insufficient memory.	As this error implies, your system has run low on memory. You can shut down some of the other application which may be running, in order to free some memory. If this does not alleviate the error, you will need to either break your system into more manageable parts, or add more memory to your system. Another thing you may wish to explore, is to set your swap space to a higher value.
Unable to create a temporary file for the gdcpp project.	Check the permissions that have been set on your GDHOME directory. They should be Read/Write.
Unable to create a temporary file for the RE PID file.	Check the permissions that have been set on your GDHOME directory. They should be Read/Write.
Unable to delete some temporary files.	Check the permissions that have been set on your GDHOME directory. They should be Read/Write.
Unable to delete the preprocessed file.	Check the permissions that have been set on your GDHOME directory. They should be Read/Write.
$GDBIN/gdcpp could not be found.	Check to make sure that GDBIN has been defined as an environmental variable and that it is pointing to the correct location.
The environment variable "GDBIN" is not defined.	Check to make sure that GDBIN has been defined as an environment variable and that it is pointing to the correct location.
The temporary directory could not be found. Please set TMP or TEMP in your environment to point to your temporary directory.	TMP or TEMP is not set so it needs to be set.
No file entered.	GDPro Reverse Engineering requires that at least one file be specified for the RE process to continue.

©2000 Advanced Software Technologies, Inc.

WARNINGS

Source file not found for Java class	It is possible that you have not specified the correct location for this class, or you did not specify the directory in which this class is located in your include path.
Unable to locate IDL type "..." for a sequence association.	

WRAPLINE

These are all of the errors which can be reported in GDWrapline.

Unable to process line.	This is a syntax error. You need to review your code, to ensure that it is correct. The following token at line ... of ... is too long to fit into the 250 character limit. String truncated at line ... of .... For the last two error messages, they are caused by a single token (word) being >250 characters long or by a single string literal being >250 characters long. The remedy is to shorten token. Sting truncation is safe, this is just a warning.

CPP

Listed below are most of the errors and warnings which can be output from CPP. The bulk of these, if they appear in RE, will also appear when the user puts the code through a C++ compiler.

Note: All CPP errors are reported under the WARNING: header in the RE output window.

ERRORS

```
      recursive use of macro `%s'
      invalid preprocessing directive name
      Predefined macro `%s' used inside `#if' during precompilation.
      cccp error: not in any file?!
      `defined' without an identifier
      cccp error: invalid special hash type
      `#%s' expects \"FILENAME\" or <FILENAME>"
      empty file name in `#%s'
 (1)  No include path in which to find %s
      directory `%s' specified in #include
      badly punctuated parameter list in `#define'
      unterminated parameter list in `#define'
      duplicate argument name `%.*s' in `#define'
      invalid %s name
      invalid %s name `%.*s'
      `##' at start of macro definition
      `##' at end of macro definition
      `#' Operator is not followed by a macro argument name
      `#' operator should be followed by a macro argument name
      missing token-sequence in `#assert'
      empty token-sequence in `#assert'
      `defined' redefined as assertion
      empty token list in `#unassert'
      unterminated token sequence in `#assert' or `#unassert'
```

©2000 Advanced Software Technologies, Inc.

```
    invalid format `#line' directive
    `#elif' not within a conditional
    `#elif' after `#else'
    `#%s' not within a conditional
    `#else' or `#elif' after `#else'
    `#else' not within a conditional
    `#else' after `#else'
    unbalanced `#endif'
    arguments given to macro `%s'
    macro `%s' used without args
    macro `%s' used with just one arg
    macro `%s' used with only %d args
    macro `%s' used with too many (%d) args
    macro or `#include' recursion too deep
    unterminated string or character constant
    possible real start of unterminated constant
    unterminated character constant
    unterminated comment
    unterminated `#%s' conditional
    unterminated comment
    string constant runs past end of line
(2) #error output
```

Project File Errors

(3) Can't save the project in \"...\".

(3) Unexpected EOF writing project file.

(3) Error writing project file.

(3) Unexpected EOF reading project file.

(3) Error reading project file.

Warning

```
    preprocessing directive not recognized within macro arg
    `/*' within comment
    invalid preprocessing directive
    using `#import' is not recommended
    VAX-C-style include specification found, use '#include <filename.h>' !
    Header file %s exists, but is not readable
    No include path in which to find %s
    missing white space after `#define %.*s'
    macro argument `%.*s' is stringified.
    macro arg `%.*s' would be stringified with -traditional.
    undefining `%s'
    `#pragma once' is obsolete
    `#pragma implementation' for `%s' appears after file is included
    `/*' within comment
(2) #warning output
    file does not end in newline
    %s in preprocessing directive
    `#' followed by integer
    file does not end in newline
    another parameter follows `%s'
    invalid character in macro parameter name
```

©2000 Advanced Software Technologies, Inc.

```
      missing white space after `#define %.*s'
      ANSI C does not allow `#assert'
      ANSI C does not allow `#unassert'
      ANSI C does not allow testing assertions
      line number out of range in `#line' directive
      garbage at end of `#line' directive
      garbage after `#undef' directive
      ANSI C does not allow `#ident'
      ANSI C does not allow `#sccs'
      pedwarn (end == limit ? "`#%s' with no argument"
      `#%s' argument starts with a digit
      garbage at end of `#%s' argument
      `#' followed by integer
      invalid preprocessing directive
      invalid preprocessing directive name
      text following `#else' violates ANSI standard
      text following `#endif' violates ANSI standard
      text following `#else' or `#endif' violates ANSI standard
  (4) `%.*s' redefined
```

Other

I/O error on output

1. This error can appear in RE, but not during C++ compilation, if the include paths specified in the C++ makefile are different from those entered during RE.
2. Errors and warnings which are the result of #error or #warning directives in the source code can appear during RE if the predefined macro definitions used during RE are different from those used in C++ compilation.
3. These project file errors won't appear in normal operation, but can show up as a result of permissions problems, disk space restriction, or abnormally terminated RE runs.
4. This macro redefinition warning shows up frequently for Windows NT users due to our use of the MFCMacros.h and ATLMacros.h header files. These files are used during the reverse engineering of every source file. Frequently the source file will redefine macros which are already defined in these headers. This is not generally a problem.

TXL

TXL errors are almost always the result of bad syntax in the source file or unexpanded macros. They all look something like this:

TXL ERROR : (Fatal) Empty repeat in define 'repeat__member_list'

could not be resolved with lookahead 'CArray'

TXL ERROR : Syntax error on line 6 of test.C++, at or near:

: CArray < CPoint , >>> , <<< CPoint > export1 ; }

©2000 Advanced Software Technologies, Inc.

EXAMPLE OF OUTPUT

Here is example output from RE, demonstrating how these errors look in context. It includes errors and warnings reported from WRAPLINE, CPP and TXL.

```
Preparing to Reverse Engineer 1 file:

        /users/james/C++/test.C++

Include files found during RE will be reversed after the above file has been successfully processed.

INFO:
============================================================
INFO: Reversing file: /users/james/C++/test.C++
ERROR: Word wrap errors in file "/users/james/C++/test.C++":
ERROR: String truncated at line 20 of /users/james/gdtmp/foo7981.gdpro.
ERROR: End word wrap errors for file: "/users/james/C++/test.C++"
WARNING: CPP warnings in file "/users/james/C++/test.C++":
WARNING: /fake/path/for/exercise/file:3: missing.h: No such file or directory
WARNING: /fake/path/for/exercise/file:7: #error This is a #error directive.
WARNING: /fake/path/for/exercise/file:10: warning: #warning This is a #warning directive.
WARNING: End CPP warnings for file: "/users/james/C++/test.C++"
ERROR: Compile/transform errors in file "/users/james/C++/test.C++":
ERROR: TXL ERROR : (Fatal) Empty repeat in define 'repeat__level_1_declaration'
ERROR:   could not be resolved with lookahead 'class'
ERROR: TXL ERROR : Syntax error on line 14 of /users/james/gdtmp/test7981.gdpro.proc, at or near:
ERROR: # 1 "/users/james/C++/test.C++" class >>> class <<< { } ; void foo1
ERROR: End compile/transform errors for file: "/users/james/C++/test.C++"
INFO: Temporary files are preserved.
INFO: Preprocessed file preserved in /users/james/gdtmp/test7981.gdpro.proc.

INFO:
============================================================
INFO: Determining what include files to reverse...
INFO: No unprocessed include files were found.
INFO: ****************************************************
INFO: FINISHED PARSING. PREPARING TO LAYOUT...
INFO: ****************************************************
INFO: Creating semantic relationships for the Implementation Diagram view...
INFO: Creating semantic relationships for the Class Diagram view...
INFO: Saving the system...
```

©2000 Advanced Software Technologies, Inc.

```
INFO: The system has been saved.
********************************************

********************************************

** Reverse engineering is complete.
** (3 warnings, 5 errors)
********************************************
********************************************
```

Summary

/users/james/C++/test.C++: 3 warnings, 5 errors

Here's the source used to generate this:

```
// Trigger #error reporting.
// causes reverse engineering to skip this file.
#error This is a #error directive.

// Trigger #warning reporting.
#warning This is a #warning directive.

// Trigger syntax error from TXL with invalid C++ syntax.
class class {
};

// Trigger gdwrapline warning for string truncation.
void foo1()
{
  printf("this is a really long repetitive string this is a really long repetitive string this is a really long repetitive string this is a really long repetitive string this is a really long repetitive string this is a really long repetitive string this is a really long repetitive string");
}
```

Tips and Techniques

In what order are include directories searched?

The include directories are searched in the order listed. Although you cannot easily reorder entries in the GUI, you can save an RE configuration file using the "Save Settings..." option in the RE GUI. This creates an ASCII file with a ".re" extension. Open this in your favorite text editor and reorder or edit with ease.

©2000 Advanced Software Technologies, Inc.

What is the syntax for "Definitions", especially if you need to set something to a value. for example: #define MYVAL 3

The syntax for "Definitions" is as follows:

Simple Assertion

C++ syntax : #define ASSERTION

In GDPro, enter: ASSERTION

Simple Macro Definition

C++ syntax : #define MYVAL 3

In GDPro, enter: MYVAL=3

Complex Macro Definition

A complex macro is defined as any macro whose replacement contains parentheses or spaces.

C++ syntax : #define MYVAL(x, y) x + y

In GDPro, enter: MYVAL="x + y"

What do the numbers "184:730" mean in the line

"WARNING: from F:\Program Files\GDPro\GDTemp\someSource.cpp184:730:" They do not appear to be line numbers since the file does not have 730 lines in it.

184 is part of the temporary file name. 730 is a line number in that temporary file.

Since most reverse engineering errors occur as the C++ code is being parsed, GD*Pro* makes it easy to preserve the temporary files which are produced by the C preprocessor, via the "Save Intermediate Files" option in the RE Preferences area.

Reverse engineering should now preserve all temporary files. Now, open for example the file "F:\Program Files\ GDPro\GDTemp/addpicdlg.cpp184" and go to line 730. This should be the line indicated in the warning message.

Conclusion

This concludes our brief tour of the GD*Pro* reverse-engineering facility.

We encourage you to give us feedback as to what you like and dislike about the product, as well as any input for additional features. You may use the e-mail feedback box located under HELP->COMMENTS.

We can be contacted at:

WWW:	http://www.advancedsw.com
FTP:	ftp.csn.org:/ASTI
E-MAIL	info@advancedsw.com
Phone:	(303) 730-7981
FAX:	(303) 730-7983

©2000 Advanced Software Technologies, Inc.

Reference Include Libraries

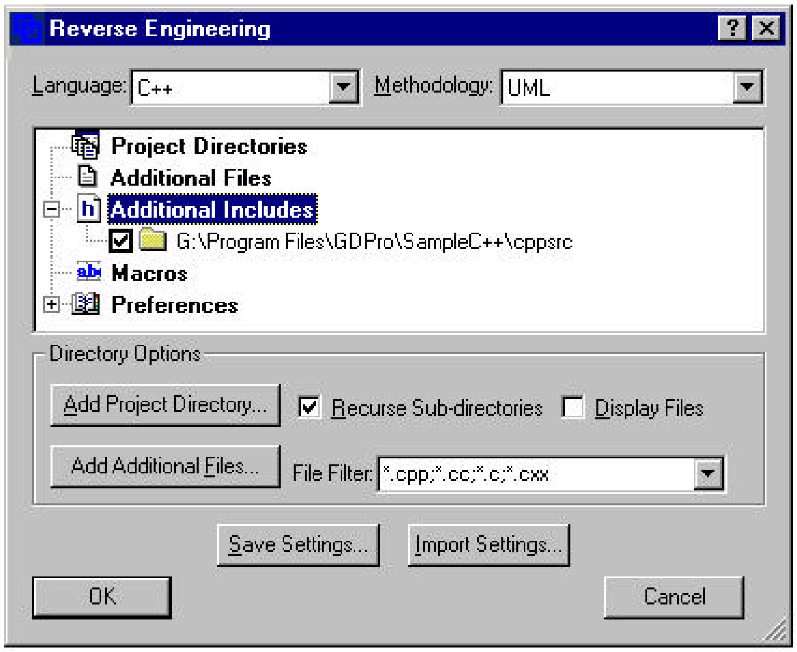
Reference Include directories are available for reverse engineering. They allow reverse engineering to process macro definitions from reference class libraries, like Rogue Wave or the Microsoft Foundation Class Library, without fully reverse engineering the classes in the reference library. You can specify as many reference include directories as you wish, to make sure that reverse engineering is able to locate all header files.

Reference and Normal Include Directories

- Reverse Engineering uses a pre-processing phase to process include dependencies
- The pre-processor searches the Include Directories for source files included by the files being reverse engineered
- Included source files that are found in "normal" include directories are also reverse engineered, and the class structure is added to the model
- Included source files that are found in "reference" include directories are not reverse engineered, and their class structure is not added to the model

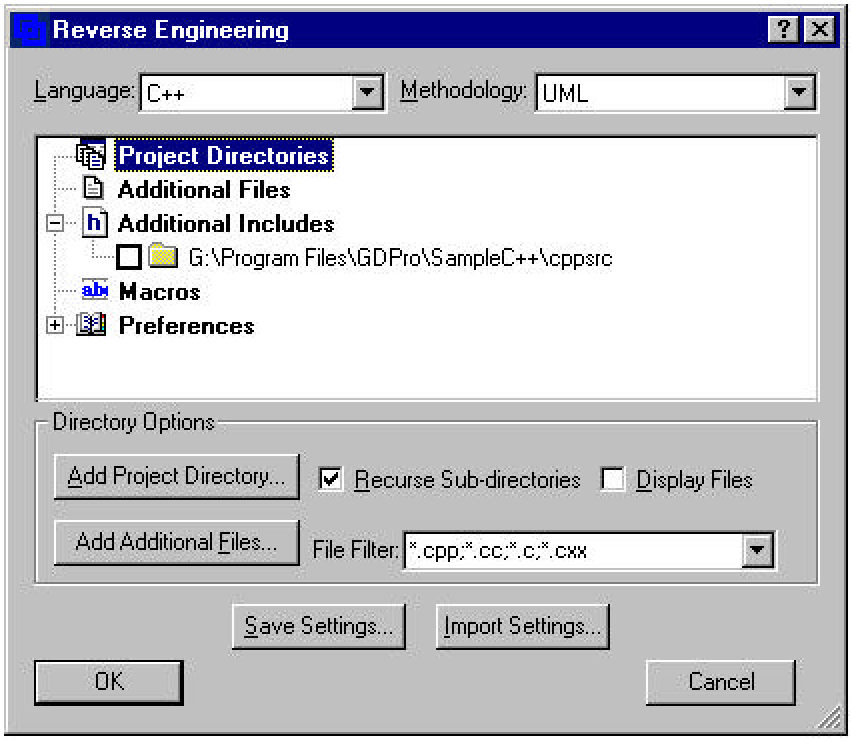
To add a reference include directory to the reverse engineering process:

1. Add an include directory to the Additional Includes portion of the tree. When this directory is added, the option box to the left of the directory path is checked.



2. To have this directory included as a reference library only, remove the check mark from the option box to the left of the directory path.

©2000 Advanced Software Technologies, Inc.



The reverse engineering process identifies these directories as referenced.

©2000 Advanced Software Technologies, Inc.