Symantec C++

More about Creating Programs

Part Four

More about Projects and Workspaces 15

This chapter continues the discussion of projects and workspaces that began in Chapter 3, "Starting a Project and Defining Workspaces." Here you find a detailed description of all commands and options associated with workspaces and projects. The first part of the chapter discusses the **Environment** menu and workspace options; the remainder of the chapter covers project files, the **Project** menu, project options, and the Project window.

Environment Menu Commands

The IDDE's **Environment** menu (Figure 15-1) contains commands with which you can modify the IDDE work environment.

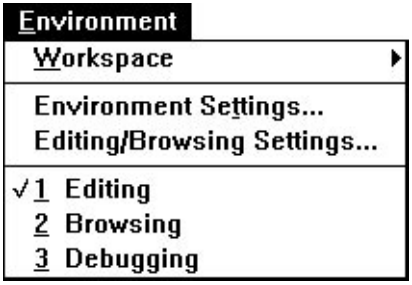

Figure 15-1 Environment menu commands

A list of available workspaces is added to the end of the **Environment** menu. The current workspace is checked. Choosing a name in this list is equivalent to clicking on the workspace tab in the Workspace toolbox.

Workspace

The **Workspace** submenu (Figure 15-2) contains commands for creating and editing workspaces.

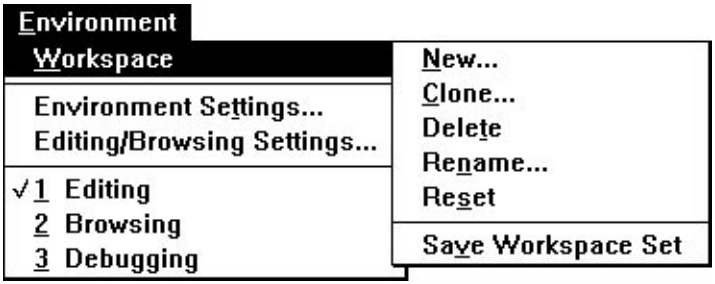

Figure 15-2 Workspace submenu

New

Opens the **Workspace Name** dialog box.

Figure 15-3 Workspace Name dialog box

To create a new empty workspace, type a name for the workspace and click OK. The Build and Views toolboxes open automatically.

The **New** command is disabled if you already have five workspaces.

Clone

Opens the **Workspace Name** dialog box. To create a copy of the current workspace, type a name for the new workspace and click OK.

The **Clone** command is disabled if you already have five workspaces.

Delete

Deletes the current workspace from the workspace set. The last remaining workspace cannot be deleted.

Rename

Opens the **Workspace Name** dialog box. Type a new name for the current workspace and click OK.

Reset

Resets the current workspace to the configuration it had when you started the session or when you last saved it during the current session with **Save Workspace Set**.

Save Workspace Set

Saves the current workspace configurations. These configurations can be restored with the **Reset** command.

Environment Settings

This command opens the **Environment Settings** dialog box with which you specify various environment options. Two pages of options are available: Workspace and Color.

Workspace

This page (Figure 15-4) provides options for controlling workspaces.

Figure 15-4 Workspace page of the Environment Settings dialog box

Save workspace set on exit
Saves the workspace set when the IDDE is closed.

Open last project on launch
When the IDDE is launched, automatically opens the project that was open when the IDDE was last closed.

Color

This page (Figure 15-5) provides options with which to customize the IDDE windows' colors.

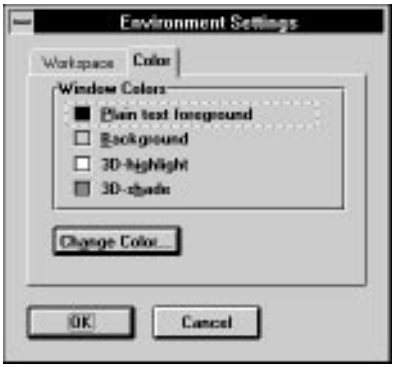

Figure 15-5 Color page of the Environment Settings dialog box

To change an item's color, first click on the item name. A dashed box appears around the item name. Then click on Change Color. You may then choose a new color from a Windows **Color** dialog box.

Editing/Browsing Settings

Opens the **Editing/Browsing Settings** dialog box, with which you can view and change the IDDE's editing and browsing options. For more information, see Chapter 19, "Class Editor Reference," Chapter 20, "Hierarchy Editor Reference," and Chapter 21, "Text Editor Reference."

More about Projects

This section continues the discussion of projects that began in Chapter 3, "Starting a Project and Defining Workspaces."

What a project contains

The project file contains a list of all the files in your project. It also contains information on how these files depend on one another. When the project manager creates a makefile (a file that builds your program), it uses a file's extension to decide what kind of file it is. The files you can put in a project are described here, along with how the IDDE uses them to build a program.

C and C++ files

The IDDE compiles C and C++ source files to produce object files and links the object files to produce the target.

C source files have the `.c` extension; C++ source files have either the `.cpp` or the `.cxx` extension. When compiled, they all generate object (`.obj`) files.

Header files

You do not need to add header files to a project; they are added automatically by the project system. See the section "Dependency tracking" later in this chapter.

If you do add a header file explicitly, it is flagged automatically for precompilation. See the description of the Header Files page of the **Project Settings** dialog box in Chapter 16, "More about Project Build Settings."

The header files that are included by C/C++ source files to provide common interface definitions are identified by the file extensions `.hpp`, `.hxx`, and `.h`.

In some situations, a C/C++ source file may be included by another source file. In this case, you probably do not want a separate object file created from that included source file. Do not add the file to the project. When you compile the target, the project system adds the included file but tracks it as an included object that should not be compiled or linked independently.

Assembly files

Your project may contain assembly source and header files. Assembly language source files are identified by the file extension `.asm`. Assembly language header files are identified by the file extension `.inc`.

Note

For assembly files to be built as part of a project, MASM must be in a directory specified by the PATH environment variable. For a NetBuild project, MASM must be in a directory specified by the PATH environment variable of every buildserver, and the buildclient.

Object files
Your project can include pre-existing object files—files for which you do not have the source or that were compiled outside the IDDE. Such object files are identified by the file extension `.obj`. These objects are linked into your executable.

Resource and dialog script files
Your project can include resource script files, dialog script files, and other binary resource files. The IDDE compiles these script files to produce a resource file. After the IDDE links the object files, it binds the resources from the resource files into the executable.

Table 15-1 Types of resource files

File Type	Description
`.rc`	Source scripts of resource files that are compiled by the Symantec C++ resource compiler
`.ico, .cur, .bmp`	Binary resources
`.res`	Compiled binary resources

Libraries
Your project can include libraries. The IDDE links the libraries with the project's object files to produce the executable file. If your program uses a dynamic link library (DLL), don't add the DLL to the project file. Instead, add the DLL's import library (`.lib`).

Libraries or library interfaces to a DLL that you want to link into your executable are identified by the file extensions `.lib`.

Linker definition files
Your project can include a linker definition (sometimes called module definition) file. Use a linker definition file to indicate to the linker how to build a library or executable. These files are identified by the file extension `.def`.

If you have your own `.def` file, you can include it in the project. The IDDE automatically modifies the `.def` file to change a linker option if necessary. If you do not specify a `.def` file, one is generated and maintained automatically for you.

Project files
The IDDE automatically generates and maintains project files. Project files can be nested to form a hierarchical project structure. Project files are identified by the file extension `.prj`.

Option set
You can include your own option set. Option sets that you create are identified by the file extension `.opn`.

Batch and makefiles
You can include a batch or makefile. Batch files are identified by a `.bat` file extension, and makefiles by a `.mak` file extension.

When you add a batch file or a makefile to a project, the **Build Order** dialog box (accessible via the Build Order button on the Make page under the Build tab of the **Project Settings** dialog box) becomes available to specify when to execute your batch files and makefiles.

Project-generated files

The project system generates and maintains the files listed in Table 15-2.

Table 15-2 Project-generated files

File Extension	Description
`.prj`	The project file that contains information on the base directory of the project, the option set, the Version Control System (VCS) configuration location, and each of the files and its attributes.
`.mak`	The makefile generated from the project options and files. You don't have to write and maintain a makefile—the project system does it for you.
`.def`	The linker definition (module definition) file. You can specify your own `.def` file or let the project system generate and maintain one for you.

File Extension	Description
.dep	The file that keeps the dependency listing.
.opn	The file in which project options are stored.
.lnk	A resource file that directs the linker to build your target. It is generated and maintained by the project system.

Hierarchical project structure

You can include projects within projects (to any depth) to create a hierarchical project structure. Hierarchical projects have many uses.

You can use hierarchical projects if you need to build more than one target as part of a system. For example, if your system includes an executable and a DLL, you can create a separate project for the DLL and include this new subproject in the master project. The library is built automatically when (and if) necessary. (Note that, since Windows relies on the module name to determine the uniqueness of modules, you need to give your targets different names. For example, do not name the executable generated by a project MYMOD.EXE and name a DLL created by a subproject MYMOD.DLL; an error results.)

In other cases, some files may need different compiler settings. You have two options. You can put those files in a separate subproject, setting compile options for the project as necessary. Or, you can override compile options on a file-by-file basis. To do this, right-click on the file in the Project window, and choose **Settings** from the pop-up menu. You can then set compile options for that file.

If you have a special preprocessing or translation step in your build process, you can create a subproject (a project within another project) for that make step. To accomplish the preprocessing, use the Make page under the Build tab in the **Project Settings** dialog box (see Chapter 16, “More about Project Build Settings”) to call your own makefile or batch file.

Use hierarchical projects to handle different releases or versions of a project. To build all the variants in one simple step, create a master

project that contains only projects. When you build the master, each of the variants is built.

These examples show that the hierarchical project system can be used in a variety of ways to customize the build process. Note that dependency tracking still works for subprojects—they are built as needed, before the master project is built.

Note

When you build a project that has a hierarchical structure, all subprojects are rebuilt with either the Debug or Release setting, whichever is applied to the master project. To ensure that each subproject is linked with the correct libraries (Debug or Release), you need to rebuild each subproject's `.LNK` file.

Dependency tracking

The project system automatically tracks dependencies among the components of a project. Dependency information is updated with each successive compilation. As you add or remove include files, for example, the corresponding dependencies are updated. You have the option of turning this tracking off.

The IDDE also tracks changes made to build options and determines how much of the project needs to be rebuilt based on those changes. For example, if you change a compiler setting, all the sources are rebuilt. If you change a resource compiler option, only the resource compiler and link steps are executed. If you change a link option or library directory, only the linker is run. By tracking these changes, the project system supports efficient and accurate builds.

In the Windows 3.1 version of the IDDE, the project system can track files in the project with respect to version control. For more information on project and source code control, see Chapter 22, "Using Version Control."

Project menu commands

The IDDE's **Project** menu (see Figure 15-6) contains commands to create, open, edit, and close projects; to build projects, to run the application; and to set project options. At the end of the menu, the

IDDE adds the names of the most recently opened projects so that you can switch between projects as you work.

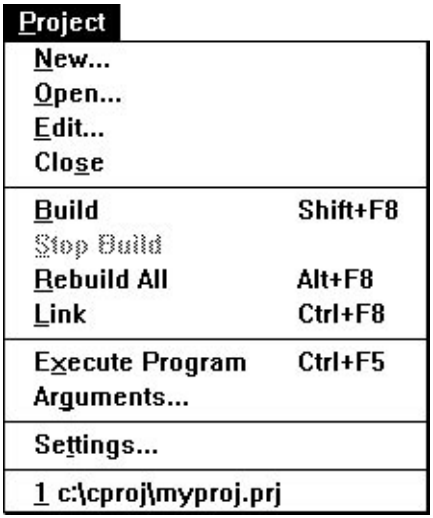

Figure 15-6 Project menu commands

New Opens the **ProjectExpress** dialog box, as described in Chapter 3, "Starting a Project and Defining Workspaces."

Open Opens the **Open Project** dialog box, as described in Chapter 3, "Starting a Project and Defining Workspaces."

Edit Opens the **Edit Project** dialog box, as described in Chapter 3, "Starting a Project and Defining Workspaces."

Close Closes the current project.

Build Builds your project. The IDDE examines all files in the project to determine whether they are up-to-date and recompiles only the necessary files.

Stop Build Stops a build in progress. You can also stop a build by choosing **Stop**! from the Output window's menu bar.

Rebuild All Rebuilds all files in your project, regardless of whether they are up-to-date.

Link Links all object files and libraries. Use **Link** instead of **Build** when adding `.lib` or `.obj` files to a project or subtracting them from a

project. You can also use this command if you know all files are up-to-date. There is no dependency checking with Link, so linking is faster than building. Also, you would use **Link** rather than **Build** if you changed source code but you wanted to generate an `.exe` with existing `.obj` files.

Execute Program

Executes your application. Command-line arguments may be set with the **Arguments** command.

Arguments

Opens the **Run Arguments** dialog box (Figure 15-7).

Figure 15-7 Run Arguments dialog box

Type the command-line arguments that you want passed to an application when you choose **Execute Program**.

Settings

Opens the **Project Settings** dialog box. This dialog box lets you set project options that specify how the IDDE builds a project. The following pages are available by clicking on the tabs at the top of the dialog box: Target, Build, Option Sets, VCS, and Directories.

Target settings

The options on the Target page (Figure 15-8) specify the target and platform for your project.

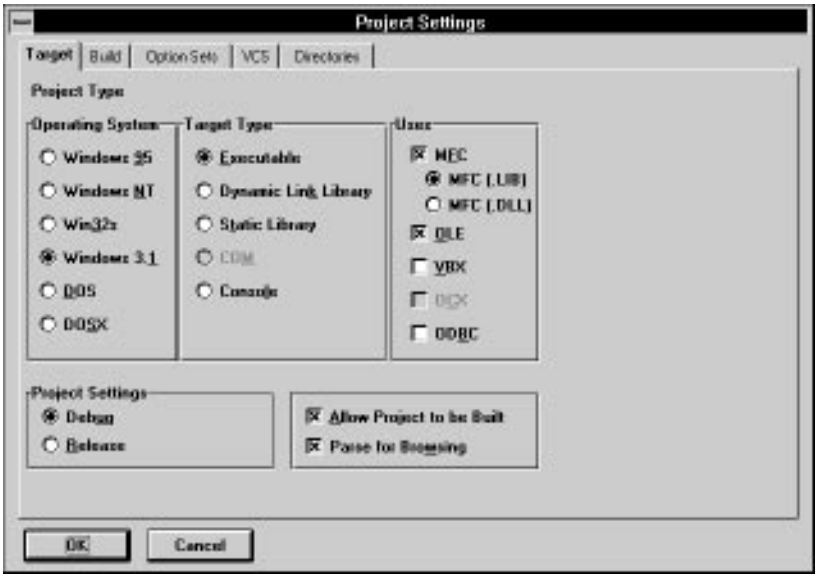

Figure 15-8 Target page of the Project Settings dialog box

Operating system

This set of options determines the target system for your project. Depending on the selection of the target system, different Target Type options become available. The target system can be one of the following:

- Windows 95
- Windows NT
- Win32s
- Windows 3.1
- DOS
- DOSX

All the systems are self-explanatory except DOSX. This option selects the DOSX 32-bit DOS Extender (which comes with Symantec C++) as the target operating system.

Target type
This set of options determines what your project will actually be: an executable, a library, a Windows DLL, or a simple console. The target type can be one of the following:

Executable: Builds an executable program (`.exe`).

Dynamic Link Library: Builds a Windows DLL. This option is not available when Operating System is set to DOS or DOSX.

Static library: Builds a static library (`.lib`)—that is, a library a program links to at compile time—as opposed to DLL, which links at run-time.

COM: Builds a DOS executable `.com` file. This option is only available when Operating System is set to DOS.

Console: Builds a Windows console program. In Windows 3.1 (and Win32s) this creates an SDI application in which the standard input and output functions are carried out through the SDI window. In Windows 95 and Windows NT, this creates an executable that runs as a console application, simulating an old-style teletype.

Uses
These options let you select the extension libraries that are linked to the executable.

MFC (Microsoft Foundation Class Library): Links the libraries needed for an application that uses classes from the MFC. Two methods of linking are available: static library (`.lib`) or dynamic link library (DLL).

OLE (Object Linking and Embedding): Links the libraries needed for creating an application that uses OLE.

VBX (Visual BASIC Control): Links the libraries needed to create a Visual Basic control.

OCX (OLE Control): Links the libraries for building an OLE control.

ODBC (Open Database Connectivity): Links the libraries needed to access an ODBC data source.

Project settings
Two options determine whether debugging information is included in the executable.

Debug: The executable contains debugging information and can be debugged by the IDDE debugger.

Release: The executable contains no debugging information. It cannot be debugged.

Allow project to be built
If this option is selected, the application can be built. Deselect this option if a project should not be built (for example, if you don't want a subproject rebuilt when you choose **Rebuild All** in the parent project).

Parse for browsing
If this option is selected, the project is automatically parsed. Deselect if you don't want the Browser to parse the source code. See Chapter 5, "Defining Classes and Their Hierarchies," for more information.

Build settings

The options on the Build page control how your project is compiled and linked. See Chapter 15, "More about Projects and Workspaces."

Option sets

The Option Sets page (Figure 15-9) lets you save and retrieve project options. This feature makes it easy to define options once for a particular kind of target and apply them later to another project.

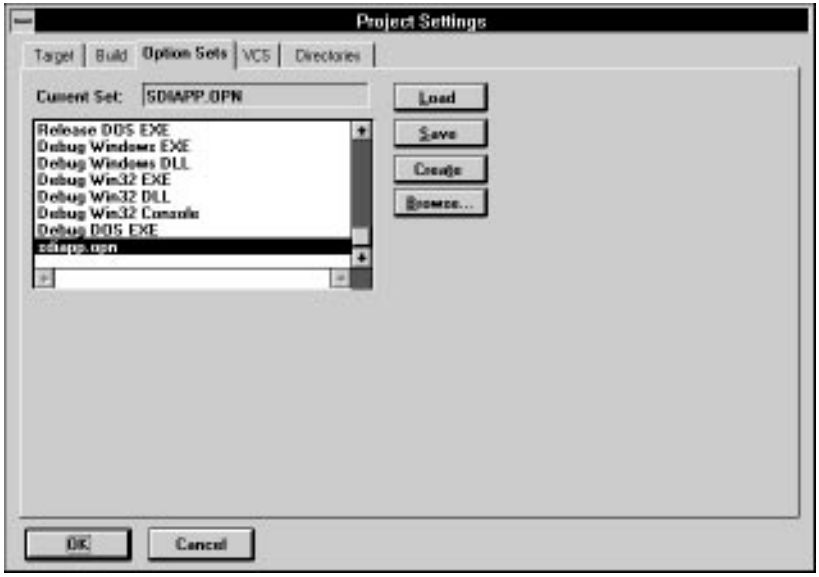

Figure 15-9 Option Sets page of the Project Settings dialog box

When you exit the IDDE or close a project, current options are saved in the project option file (`.opn`). This file has the same name as the project. For example, if the project name is `test.prj`, the option set associated with that project is named `test.opn`.

The list of option sets includes sets you define and several predefined option sets. Click on an option set name to select the option set; double-click on an option set name or click on Load to load the option set.

The predefined option sets are useful starting points for defining your own custom options. When you load one of these option sets, save any changes to another option set so the defaults are intact for later use. These option sets are named according to the target type and whether debugging information is placed in the executable.

The four buttons on the dialog box have the following functions:

Load: Loads the selected option set.

Save: Saves the current option set.

Create: Creates a new option set. You are prompted for the new set's name.

Browse: Opens the **Option Set Name** dialog box, from which you can select an option set file to load.

VCS options

The options on the VCS page control the version control system. These options are only available in the 16-bit IDDE. For information about version control see Chapter 22, "Using Version Control."

Directories

The options on the Directories page (Figure 15-10) specify various directories used by the compiler, linker, and browser.

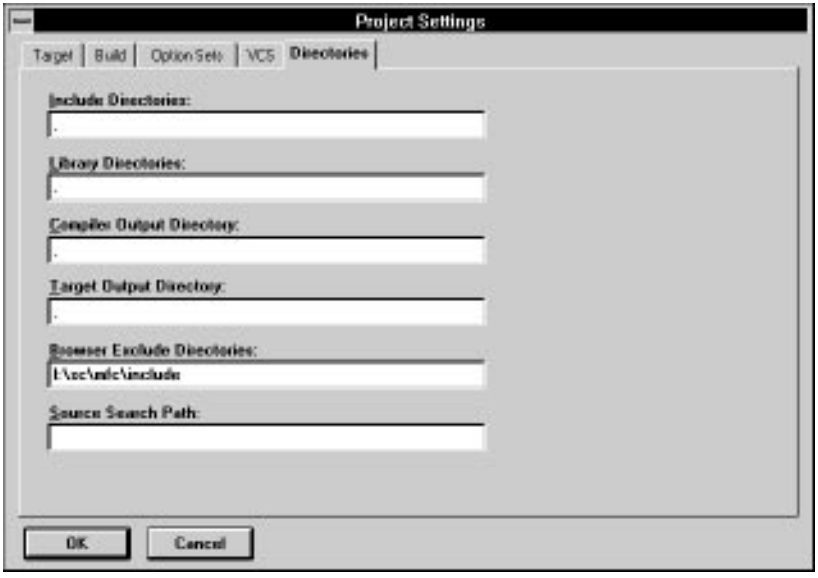

Figure 15-10 Directories page of the Project Settings dialog box

Include directories

Specifies directories to be searched for included files. You may specify multiple directories by separating each pathname with a semicolon. These directories are searched after those specified by the `INCLUDE` environment variable.

Library directories
Specifies which directories to search for libraries. You may specify multiple directories by separating each pathname with a semicolon. These directories are searched after those specified by the `LIB` environment variable.

Compiler output directory
Specifies the directory in which object files (`.obj`) are placed.

Target output directory
Specifies the directory in which the linked target is placed.

Browser exclude directories
Specifies the directories that are excluded from parsing by the browser. Use this option, for example, to exclude the MFC header file directory, and thus prevent the display of MFC classes in the Class and Hierarchy editors.

Source search path
Specifies which directories to search for source files while debugging.

The Project Window

This section describes all the menu commands available from the Project window.

Parse menu commands

The **Parse** menu (Figure 15-11) contains commands to control the parsing of source files. Parse information (information about the project's C++ classes and class members) is stored in a global pool that is accessible to and used by the Class and Hierarchy Editors.

Figure 15-11 Parse menu commands

Update All

Parses all unparsed files in the project.

Parse All Parses all files in the project.

Parse File Parses the selected file, adding information about classes and members in the file to the global parse information.

Unparse File Unparses the selected file. It removes classes and members in the file from the global parse information.

Stop Parse This command cancels a parse operation in progress. You can also stop a parse by choosing **Stop**! from the Output window's menu bar.

View menu commands

In debugging mode, commands in the **View** menu update other IDDE windows to show information pertaining to the selected file. For more information on this menu refer to Chapter 24, "Commands Available in Debugging Mode."

Trace menu commands

The **Trace** menu controls whether the debugger can step into, set breakpoints in, or watch data in a particular source file in debugging mode. See Chapter 24, "Commands Available in Debugging Mode," for more information.

VCS

The **VCS** menu controls the Version Control System operation. Version control options are only available in the 16-bit IDDE. The **VCS** menu commands and their functions are described in Chapter 22, "Using Version Control."

Project window left pane pop-up menu commands

This menu (Figure 15-12) contains commands that operate on the current project.

Figure 15-12 Left pane pop-up menu commands

New Subproject Opens the **New Sub-Project** dialog box, in which you select a project to be a subproject of the currently selected project.

Link Links the project. This is the same as choosing **Link** from the IDDE's **Project** menu.

Build Builds the project. This is the same as choosing **Build** from the IDDE's **Project** menu.

Rebuild All Rebuilds the entire project. This is the same as choosing **Rebuild All** from the IDDE's **Project** menu.

Parse Parses all files. This is the same as choosing **Parse All** from the **Parse** menu.

Edit Project Opens the **Edit Project** dialog box, as described in Chapter 3, "Starting a Project and Defining Workspaces." This is the same as choosing **Edit** from the IDDE's **Project** menu.

Settings Opens the **Project Settings** dialog box. This is the same as choosing **Settings** from the IDDE's **Project** menu.

Project window right pane pop-up menu commands

This menu (Figure 15-13) contains commands that operate on the selected file.

Figure 15-13 Right pane pop-up menu commands

Compile Compiles the selected file.

Parse Parses the selected file. This is the same as choosing **Parse File** from the **Parse** menu.

Other	Opens the **Other** submenu, which contains the **Preprocess**, **Disassemble**, and **Precompile** commands.
Preprocess	Preprocesses the selected file.
Disassemble	Disassembles the selected file.
Precompile	Precompiles the selected file.
Get	Is the same as choosing **Get** from the **VCS** menu.
Put	Is the same as choosing **Put** from the **VCS** menu.
Remove	Removes the selected file from the project. You can also remove a selected file from the project by pressing Delete.
Attributes	Sets read/write attributes for the selected file. They are mutually exclusive, so only one can be selected. A checkmark is displayed next to the selected item.
Read Only	Sets the attribute to read only.
Read/Write	Sets the attribute to read/write.
Don't Show	Filters the display of the project's files.
Modules	Does not list modules added by the debugger.
Parsed	Does not list files added by the parser.
Dependencies	Does not list files included through dependency relationships.
Settings	Lets you set certain compiler options for an individual source file, overriding those specified for the project as a whole. Choosing this command opens the **Project Settings** dialog box with only the Build tab available (see Chapter 16, "More about Project Build Settings"). Only the compiler-related subpages (Compiler, Code Generation, Header Files, Code Optimizations, Output, Warning, and Debug Information) are available. The Inherit from Project button resets this source file's individual options to those of the project.

"..." pop-up menu commands

This menu (Figure 15-14) is opened by clicking on the "..." box above the vertical scroll bar in the right pane of the Project window.

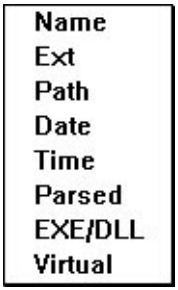

Figure 15-14 "..." pop-up menu commands

Use this menu to set up the display of project file information in the right pane. Information that can be displayed includes:

Name Displays the filename.

Ext Displays the file extension.

Path Displays the file path.

Date Displays the modification date.

Time Displays the modification time.

Parsed Displays whether or not the file has been parsed.

EXE/DLL Displays to which EXE or DLL the module belongs (in debugging mode).

Virtual Displays whether the module is virtual (in debugging mode).

For more information about the EXE/DLL and Virtual columns, see "The Project Window," in Chapter 24, "Commands Available in Debugging Mode."

Columns of information are removed from the display by dragging the column heading out of the column heading area. (The item then becomes available on the "..." pop-up menu.) The order of the columns can be changed by dragging the column headings to a new position.

Project window mouse functions

Use the mouse to open projects, select project files, open source windows, drag project files to other windows, open pop-up menus, and change the relative sizes of the right and left panes.

To resize the panes, first position the cursor on the dividing line between panes. The cursor changes to a two-headed arrow. Then click the left mouse button and drag the separator to the desired location.

The right mouse button opens the pop-up menus (see the sections "Project window left pane pop-up menu commands" and "Project window right pane pop-up menu commands" earlier in this chapter).

Click on a project or subproject in the left pane to open that project. Double-click on the current project in the left pane to toggle (expand or collapse) the display of its subprojects.

Click on a project file in the right pane to select it. Double-click on a file in the right pane, or drag it to an empty part of the workspace, to open the Source window to view and edit the file. (Double-clicking on a subproject in the right pane opens that subproject.)

You can drag files from the right pane to Source windows, Function windows, Data/Object windows, and Assembly windows.

Finally, to eliminate a column of information from the right pane, click on the title at the top of the column and drag it outside the column heading area. Columns of information are restored from the **"..."** pop-up menu, to the right of the column titles. To rearrange columns of information, drag the title at the top of the column to a new position. You can make columns wider or narrower by dragging the column title's right edge to the right or left. Clicking on a column title re-sorts the list of project files according to that column.

More about Project Build Settings 16

This chapter details the options for controlling how your project is built. This discussion began in Chapter 15, "More about Projects and Workspaces." This chapter lists and explains the options on the Build page of the **Project Settings** dialog box. You access these options by selecting **Settings** from the **Project** menu, then clicking on the Build tab.

The Build page of the **Project Settings** dialog box is composed of 18 subpages. The subpages are displayed in a listbox on the left of the window. To access a particular subpage, click on its name. The options displayed on the right change with each subpage. Subpages are organized hierarchically, as shown in the listbox.

This chapter covers all the subpages on the Build page of the **Project Options** dialog box. The first section introduces you to build settings and the **Project Settings** dialog box, and the later sections describe the subpages in the order in which they are listed on the Build page.

For more detailed information on how each of these options affects the compilation of your code, refer to the *Compiler and Tools Guide*. Appendix B, "IDDE Settings and Command-Line Options," details how each of these options map to the corresponding SC, OPTLINK, Make, or Librarian command line options.

Introducing Build Settings

Choose the **Settings** command from the IDDE's **Project** menu to open the **Project Settings** multipage dialog box. Using the tabs at the top of the dialog box, you can select different pages, each of which presents a set of options.

Click on the Build tab in the dialog box to open the Build page. The Build page is composed of subpages of options. Select a subpage by

clicking on its name in the listbox on the left of the Build page. The following subpages are available:

- Compiler
- Code Generation
- Header Files
- Memory Models
- Code Optimizations
- Windows Prolog/Epilog
- Output
- Warnings
- Debug Information
- Linker
- Packing & Map File
- Definitions
- Segments
- Imports/Exports
- Resource Compiler
- Make
- External Make
- Librarian

Each of the subpages is described in a separate section in this chapter.

Compiler

The Compiler subpage (Figure 16-1) contains a variety of parameters controlling compilation.

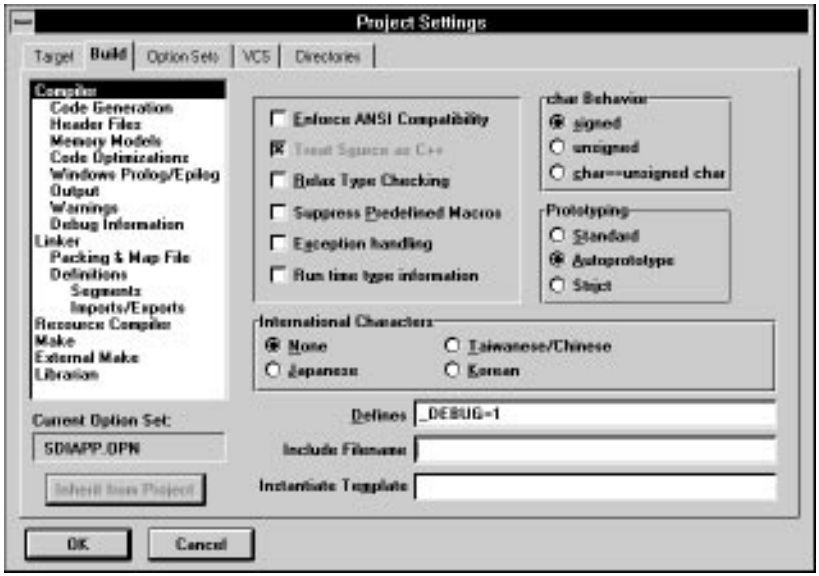

Figure 16-1 Compiler subpage

Enforce ANSI compatibility
Establishes the necessary parameters so the compiler generates C code that conforms to ANSI standards. Refer to the *Compiler and Tools Guide* for information on the restrictions that take effect when you enable this option.

Treat source as C++
With this option selected, the compiler treats C source files as C++ files. The option is useful if you want to:

- Compile the file to take advantage of type-safe linkage
- Link a C file to a C++ file without changing it or giving it a C++-compatible extension

You can also use this option on C++ header files with the .h extension.

Relax type checking
Causes the compiler to use relaxed (non-ANSI) type checking. The following data types are then treated as equivalent:

- `char == signed char == unsigned char`
- `short == unsigned short`
- `long == unsigned long`

In addition, for 16-bit compilations:

- `int == unsigned == enum == short`

And for 32-bit compilations:

- `int == unsigned == enum == long`

The option is useful for quickly porting code from compilers that do not obey the full set of ANSI type-checking rules.

Suppress predefined macros
Suppresses the definition of the non-ANSI predefined macros.

Exception handling
Enables implementation of exception handling.

Run-time type information
Enables implementation of run-time type information.

Enable new[], delete[] overloading
Enables overloading of operator `new[]` and operator `delete[]`. It also sets the predefined macro _ENABLE_ARRAYNEW to 1.

Compiling with Enforce ANSI compatibility automatically enables this option.

char behavior

This option controls the way the compiler treats a `char` type. By default, the compiler treats `char` types as `signed`. Set this option to unsigned to quickly port code that depends on `unsigned char` types. Note that the behavior of the run-time library routines is not affected by this option unless they are also recompiled.

signed: Makes `char` types behave as `signed char` types.

unsigned: Makes `char` types behave as `unsigned char` types.

char==unsigned char: Changes the type of `char` to be unsigned.

Prototyping

This option specifies how the compiler handles function prototypes. New code should always be fully prototyped, due to the requirements of type-safe linkage and the support for alternative linkage conventions.

There are three prototyping possibilities: Standard, Autoprototype, and Strict.

Standard: Turns off autoprototyping and strict prototyping.

Autoprototype: Enables the compiler to generate a prototype according to the way the function is used, even if one is not specified for a function. Subsequent uses are checked against the generated prototype. This is especially useful when compiling old C code that is not completely prototyped.

Strict: Requires that all functions be declared (prototyped) before being used. This declaration provides the compiler with the function name, return type, and storage class of a function, as well as the number and type of arguments that may be passed to it. Once the compiler encounters a function prototype, it can check each function call in the source file against that prototype and flag an error for the calls that do not match it.

International characters

This option specifies how the compiler interprets 2-byte Asian language character codes within character constants and strings. That is, if a character code represents the first byte of a 2-byte sequence, the second byte is not checked to see whether it is a backslash or a closed quote. The second byte cannot be a NULL (0), a carriage return (`0x0D`), or an end-of-file (`0x1A`).

None: Allows no 2-byte sequences.

Japanese: Signals the 2-byte sequence with a value in the range `0x81` ... `0x9F` and `0xE0` ... `0xFC`

Taiwanese/Chinese: Signals the 2-byte sequence with a value in the range `0x81` ... `0xFC`

Korean: Signals the 2-byte sequence with a value in the range `0x81` ... `0xFD`

Other options

The options below direct the compiler to define a macro, include a header file, or instantiate a template.

Defines
Allows you to specify macro definitions on the compiler command line. You should separate multiple Defines with a semicolon (;).

Include filename
Directs the compiler to include a header file for all modules in the project.

Instantiate template
Creates an instance of a template in your program.

Code Generation

The Code Generation subpage, shown in Figure 16-2, contains parameters that control how the compiler generates code.

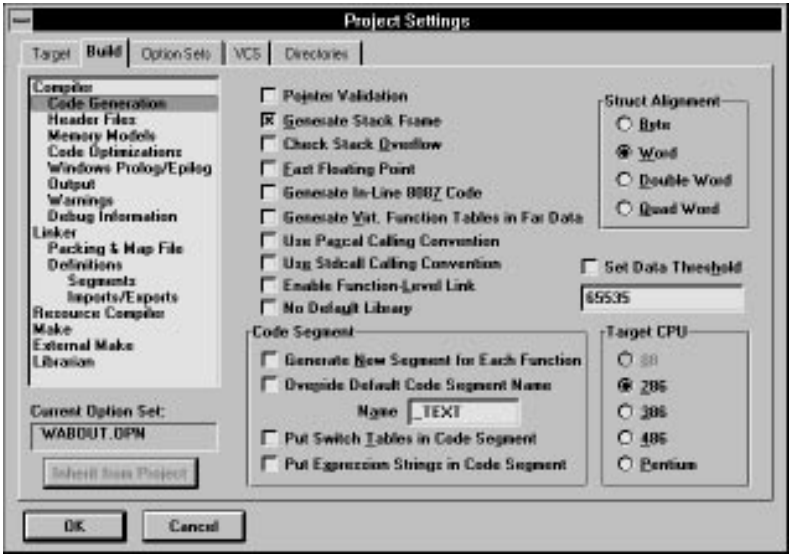

Figure 16-2 Code Generation subpage

Pointer validation
Makes the resulting program validate each pointer as it is dereferenced; if the pointer is invalid, a run-time error occurs. This slows and slightly increases the size of the resulting code.

Generate stack frame
Generates a stack frame for each function. The stack frame is generally for the use of the debugger.

Check stack overflow
Inserts stack overflow checking at the beginning of each function. The resulting program aborts with an error message if it detects a stack overflow.

Fast floating point
Directs the compiler to produce the fastest possible floating-point code. No compatibility checking is performed.

Generate inline 8087 code
Causes the compiler to generate inline 80x87 instructions. It significantly speeds up floating-point code, reduces its size, and improves its accuracy.

Generate virtual function tables in far data
Affects Compact and Large memory models only. It causes the virtual function tables to be placed in far data segments rather than in the code segment.

Use Pascal calling convention
Makes Pascal, instead of cdecl, the default linkage for all functions and global data. Because all C library routines have cdecl linkage, they must be prototyped as such before being called. Therefore, you must include the appropriate header files. The `main()` function must also be declared cdecl for the linker to find it.

Using Pascal as the default linkage type results in a roughly 3% code size reduction and a corresponding speed up in generated code.

Use Stdcall calling convention
Makes stdcall the default linkage for all functions and global data, instead of cdecl.

Note

Under Windows 95 and Windows NT, Symantec's name mangling scheme now appends the string "@*nn*" to the names of all stdcall functions (where *nn* is the number of bytes in parameters to the function). Previous versions of Symantec C++ did not append this string to mangled names.

Enable function-level link
Directs the compiler to encapsulate functions in initialized common blocks (COMDAT records). This allows the linker to perform function-level linking, which results in a smaller executable.

No default library
Prevents the compiler from embedding the default library record in the object file. This option can result in a significant decrease in program size when generating a large library.

Set data threshold
Places large arrays in far data segments. The threshold size is set in the adjacent textbox.

Code segment

These options govern the ways in which code segments are handled.

Generate new segment for each function
Causes the compiler to start a new code segment each time it encounters a global far function. The name of the segment is the name of the function with _TEXT appended.

Override default code segment name
Overrides the default code segment name. Type the new name in the Name textbox.

Put switch tables in code segment
Places switch tables in the code segment rather than in the data segment. It is useful when data segment space is critical. Do not use this option when the code segment in a Small or Compact model program is close to overflowing.

Put expression strings in code segment
Puts expression string literals into the code segment rather than wasting space in a group.

Struct alignment

This lets you control the boundaries for structure alignment. The default is to align members within a structure on word boundaries for 16-bit programs. This maximizes speed on computers with a 16-bit bus (such as a PC-AT). The default in 32-bit DOS programs is to align on double-word boundaries to maximize performance.

Byte: Aligns structures on byte boundaries.

Word: Aligns structures on word boundaries.

Double Word: Aligns structures on double-word boundaries. This is the default for 32-bit DOS applications.

Quad Word: Aligns structures on boundaries that are multiples of four words. This is the default for Win32 applications.

Target CPU

This option lets you create a compilation tailored for the instruction set of a specific CPU.

88: Generates 16-bit code using the 8088 instruction set.

286: Generates 16-bit code using the 80286 instruction set. Programs compiled with this option will not run on an 8088 or 8086 processor.

386: Generates code optimized for machines with an 80386 CPU. Programs compiled with this option require an 80386, 80486, or Pentium processor.

486: Generates code optimized for machines with an 80486 CPU. Programs compiled with this option require a 32-bit DOS extender or 32-bit operating system, and an 80386, 80486, or Pentium CPU.

Pentium: Generates code for the Pentium instruction set.

Header Files

The Header Files subpage (Figure 16-3) specifies compiler header file options.

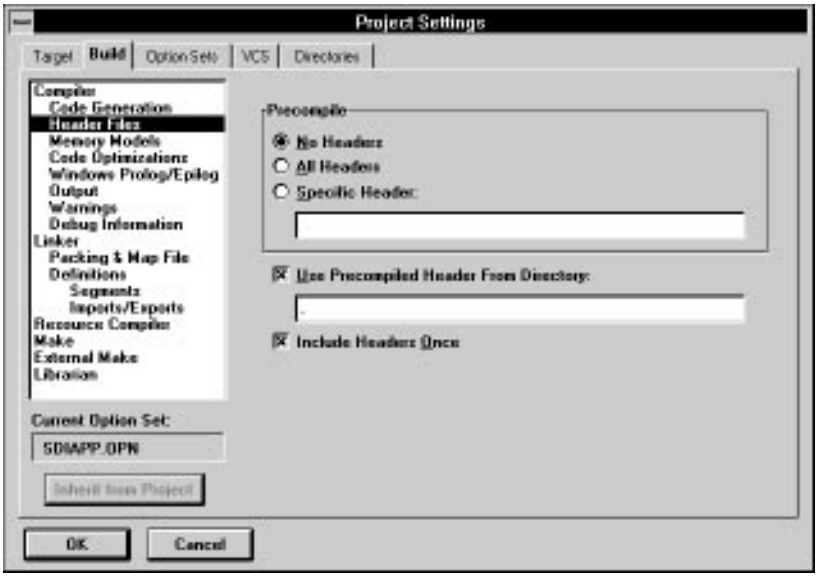

Figure 16-3 Header Files subpage

Precompile options

If your program uses a large header file or numerous small headers, the compiler spends considerable time compiling the same source code over and over again. To reduce compile time, you can precompile header files; the compiler can load a precompiled header faster than it can a text header file. It is especially useful to precompile large header files that seldom change, such as `windows.h`.

No headers

Disables generation and use of precompiled headers.

All headers

In most cases, using precompiled headers is convenient and fast. The All Headers option automatically precompiles all header files defined in the project source files into the file `scph.sym`. The file `scph.sym` is always in the current directory or in the directory specified as the compiler output directory.

Setting the All Headers option causes the compiler to generate the file `scph.sym` when it does not exist. If the `scph.sym` file does exist, the compiler assumes it is a precompiled header. The compiler will then test the header file to see whether all the headers it contains are older than the header file itself. If any are newer, or if the compiler flags have changed, a new `scph.sym` file is created; otherwise, the old file is loaded as a precompiled header. Setting the All Headers option also causes all files being compiled to use scph.sym as their precompiled header.

The compiler writes out the precompiled header when the first line in the top-level source file is encountered, and when that line is not a comment or an `#include` statement.

To avoid problems with precompiled headers:

- Do not write any declarations that cross boundaries of header files. Each header file should be self-contained.
- Do not write any `extern "C"` constructs that start in one file and end in another.
- Do not depend on header files being included more than once.
- Do not write any code or data definitions in header files, only declarations.

To maximize compile speed, set the directory to a RAM disk.

There are two circumstances in which using precompiled headers is not recommended:

- The file being compiled causes `scph.sym` to be regenerated, but that file contains a subset of the header files that other files also contain.
- Included files are wrapped in `#if blocks` or `extern "C"` blocks. (The `extern "C"` statement is a non-include statement, so the precompiled header is written out prior to the `extern "C"` block.) Embed the `extern "C"` blocks in the included files themselves.

Specific header
Lets you select a header file to precompile individually. Type the header file name into the textbox.

Use precompiled headers from directory
Tells the compiler to use precompiled headers from a specific directory. In the textbox, type the name of the directory in which the precompiled headers reside. If this directory is empty, no additional directories, other than the current or path directories, are searched.

Include headers once
Tells the compiler to include each header file only once, even if it is named in more than one source file. Otherwise, header files are included whenever they are named in a source file. This option can be used with or without precompiled headers.

Memory Models

The Memory Models subpage (Figure 16-4) specifies the memory model to be used.

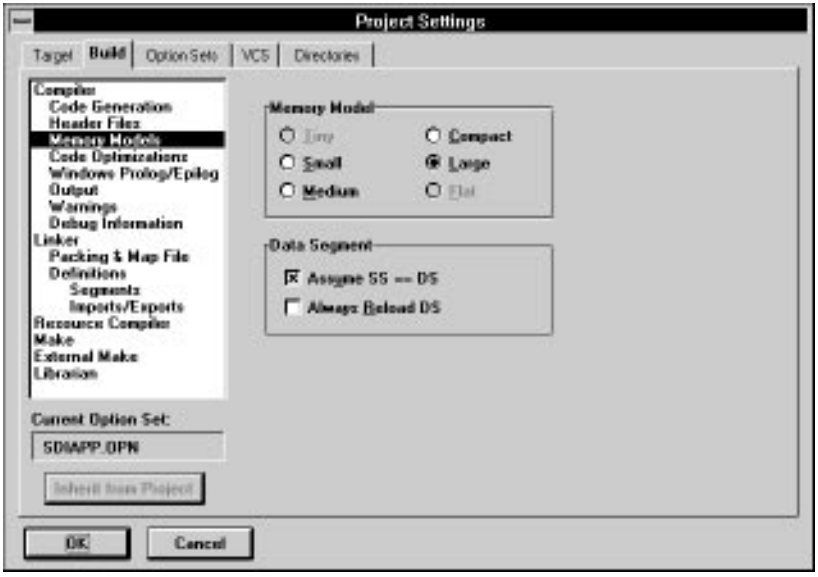

Figure 16-4 Memory Models subpage

Memory model

Controls the memory model the compiler uses by specifying its size:

Tiny: Tells the compiler to create a `.com` file active only for DOS compilations.

Small: Generates code with near pointers for the code segment and the data segment.

Medium: Generates code with far pointers for the code segment and near pointers for the data segment.

Compact: Generates code with near pointers for the code segment and far pointers for the data segment.

Large: Generates code with far pointers for both the code and data segments.

Flat: Generates code for a Win32 compilation; that is active only for Win32s and DOSX compilations.

Data segment

These options control the way the compiler treats the data segment register.p.

Assume SS==DS
Causes the compiler to generate code that assumes SS equals DS.

Always reload DS
Causes the compiler to generate code that reloads DS at the beginning of each function call.

Code Optimizations

Options on the Code Optimizations subpage (Figure 16-5) control how the compiler optimizes code.

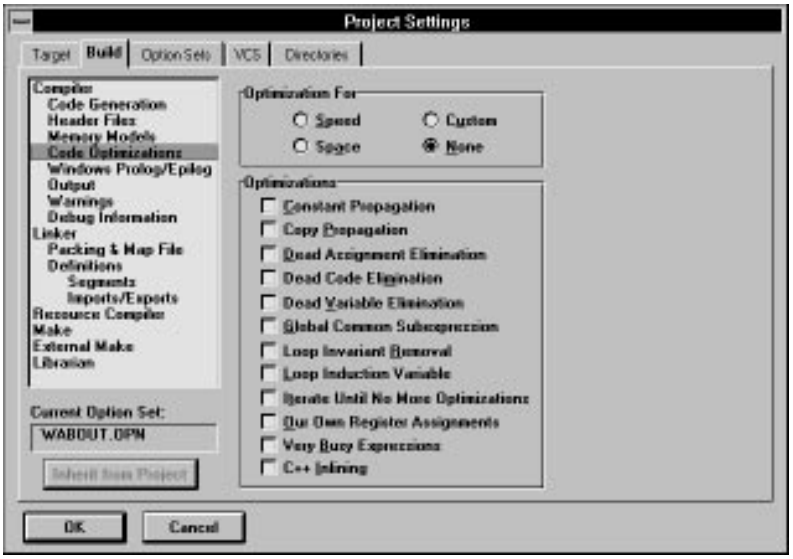

Figure 16-5 Code Optimizations subpage

Optimization for
These radio buttons control the type of optimization. You can optimize for speed or space, select a custom set of optimizations, or disable optimization.

Speed: Optimizes code for speed at the expense of program size. The code uses all optimizations.

Space: Optimizes code for space at the expense of execution speed. The code uses all optimizations.

Custom: Allows selection of optimizations using the Optimization check boxes.

None: Turns off all Optimization check boxes. No optimization is performed.

Optimizations
The Optimization check boxes control the individual optimizations. These options relate only to the Custom radio button.

For more information on how Symantec C++ optimizes code, see the *Compiler and Tools Guide.*

C++ inlining
Controls inline function expansion in C++. When you are debugging C++ files, the presence of inline code can present considerable problems to most symbolic debuggers. This option suppresses the production of inline code.

Windows Prolog/Epilog

The Windows Prolog/Epilog subpage (Figure 16-6) specifies the type of Windows prolog and epilog code that the compiler attaches to each far function in a compilation.

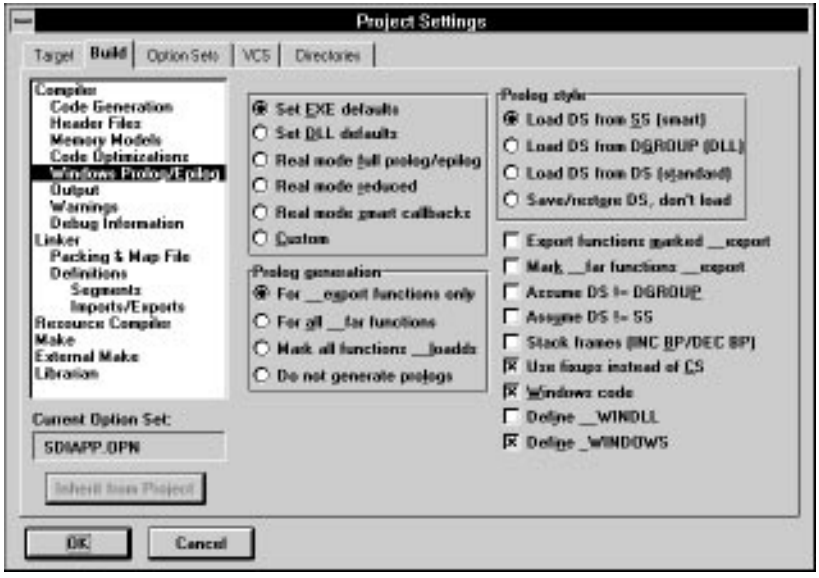

Figure 16-6 Windows Prolog/Epilog subpage

The first group of radio buttons selects from predefined sets of prolog/epilog options.

Set EXE defaults
Sets options to generate prologs and epilogs for a protected-mode Windows application, with callback functions all marked as `_export`.

Set DLL defaults
Sets options to generate prologs and epilogs for a protected-mode Windows DLL, with callback and exported functions all marked as `_export`.

Real mode full prolog/epilog
Sets options to generate prologs and epilogs for a real or protected-mode Windows application or DLL.

Real mode reduced
Sets options to generate prologs and epilogs for a real or protected-mode Windows application or DLL with exported and callback functions (marked with `_export`).

Real mode smart callbacks
Sets options to generate prologs and epilogs for a real or protected-mode Windows application with smart callbacks. In smart callbacks, the compiler compiles far functions with a smart prolog and epilog that loads the data segment from the stack segment. Use smart callbacks only with applications in which the data segment is the same as the stack segment (DS==SS). Do not use it with DLL files.

Custom
Lets you specify a nonstandard set of prolog/epilog options.

The remaining options on the Windows prolog/epilog subpage are discussed in the *Compiler and Tools Guide*.

Output

The Output subpage (Figure 16-7) controls the output generated by the compiler.

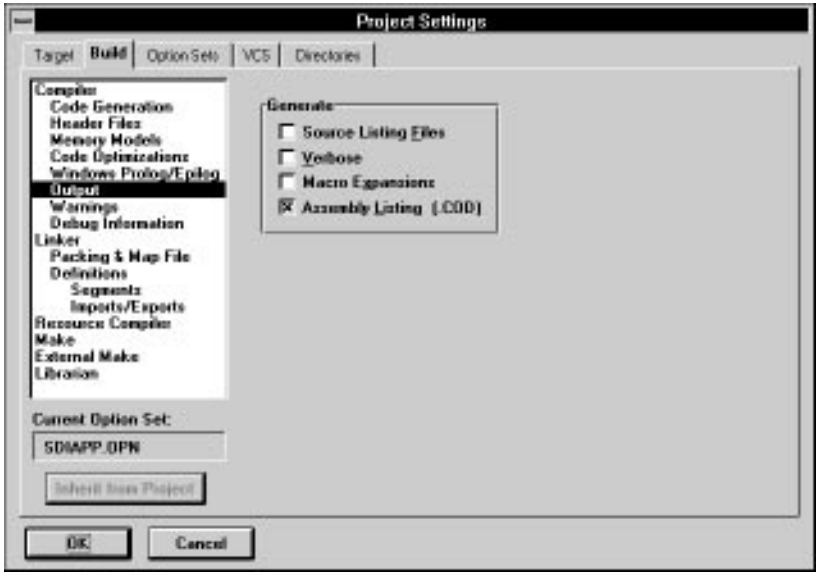

Figure 16-7 Output subpage

Source listing files
With this option on, the compiler creates a source listing file; its name is that of the source file with the extension `.lst`. The compiler inserts error messages and line numbers in the listing file.

Verbose
Displays source and header file names, classes, function prototypes, and time of execution during compilation.

Macro expansions
Tells the compiler to create macro expansions in error listings.

Assembly listing (.COD)
Causes the compiler to generate a `.cod` file containing an assembly language representation of the program.

Warnings

The options on this subpage, shown in Figure 16-8, control how the compiler produces warnings and let you enable the compiler to generate only specific warnings.

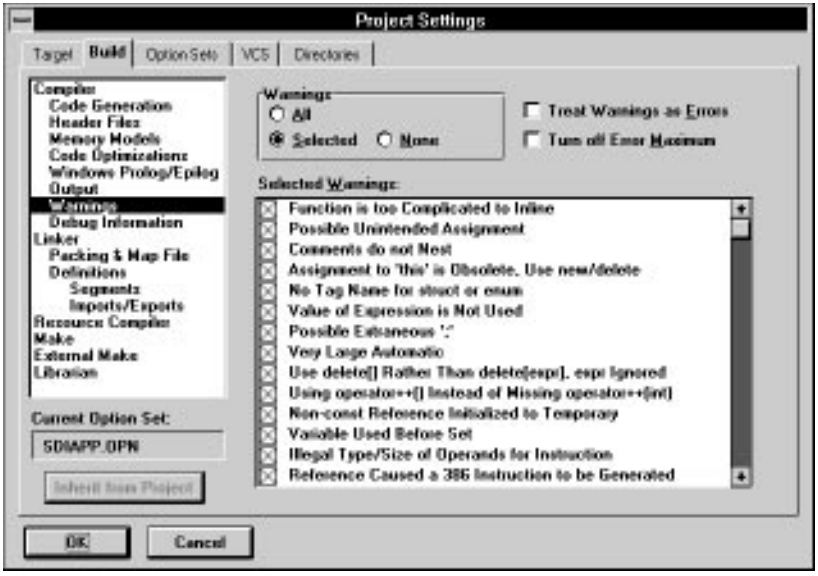

Figure 16-8 Warnings subpage

Warnings
Determines the warning messages that are produced:

All: All warnings are produced. Turns on all warnings in the Selected Warnings group.

Selected: Only warnings that have been checked in the Selected Warnings area are produced.

None: No warnings are produced. Turns off all warnings in the Selected Warnings group.

Treat warnings as errors
Causes the compiler to promote warnings to errors. Setting this option allows you to use the error window to find warnings in the source file.

Turn off error maximum
Makes the compiler continue rather than stop when its error limit is reached. The compiler processes the entire source file and displays all errors it has detected.

Selected warnings
Determines the warning messages that are produced. For more information on the specific warnings, refer to the *Compiler and Tools Guide*.

Debug Information

The Debug Information subpage (Figure 16-9) controls the information the compiler places into the code for debugging. These options specify the level of debugging, debug information, and other conditions.

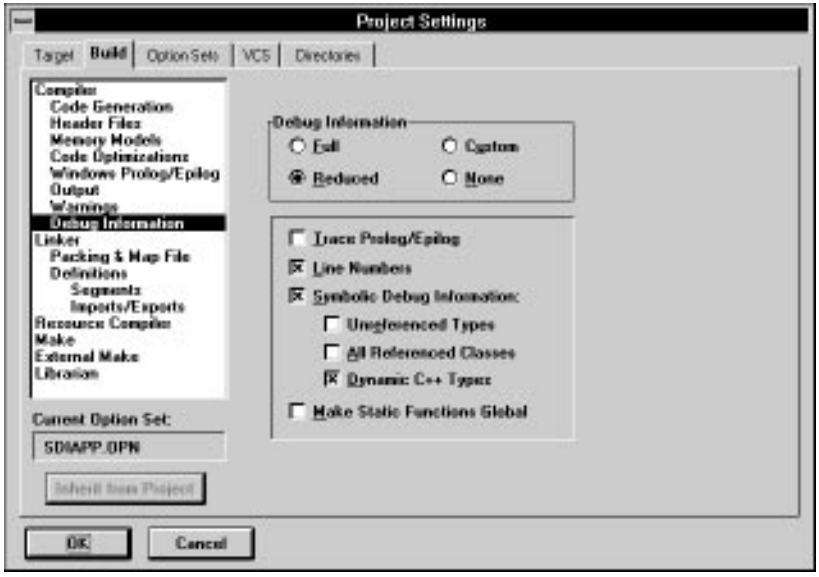

Figure 16-9 Debug Information subpage

Debug information
These radio buttons select from predefined sets of debug information options.

Full: Turns on all debug information. Use this set when the application uses a class library or DLL.

Reduced: Turns on the most frequently needed debug information. Use this set for most other programs.

Custom: Allows you to select the debug information to be included.

None: Turns off all debug information.

Trace prolog/epilog
Adds the user-defined function calls `__trace_pro_f` and `__trace_epi_f` to the prolog and epilog, respectively, for each function. The prolog function is called after the stack frame is set up; the epilog function is called just before the stack frame is destroyed.

Line numbers
Places line numbers corresponding to the source into the code. Line number information significantly increases the size of the object file.

Symbolic debug information
Includes symbol information for all public symbols. Symbols significantly increase the size of the object file.

Three additional options become available when Symbolic Debug Information is checked:

Unreferenced types: Generates symbols for unreferenced types (for example, `typedef`s).

All referenced classes: Forces symbols for all classes that are referenced in DLL files and class libraries to be generated.

Dynamic C++ types: Adds dynamic C++ class type information to classes with virtual functions. To force the production of typing information for a class with no virtual functions, add a dummy function such as `virtual void dummy(){}` to the class definition.

Make static functions global
Makes all static functions global. The linker then can enter the names of these functions into the map file and place global debugging information in the executable.

Linker

The Linker subpage (Figure 16-10) governs the overall behavior of the linker.

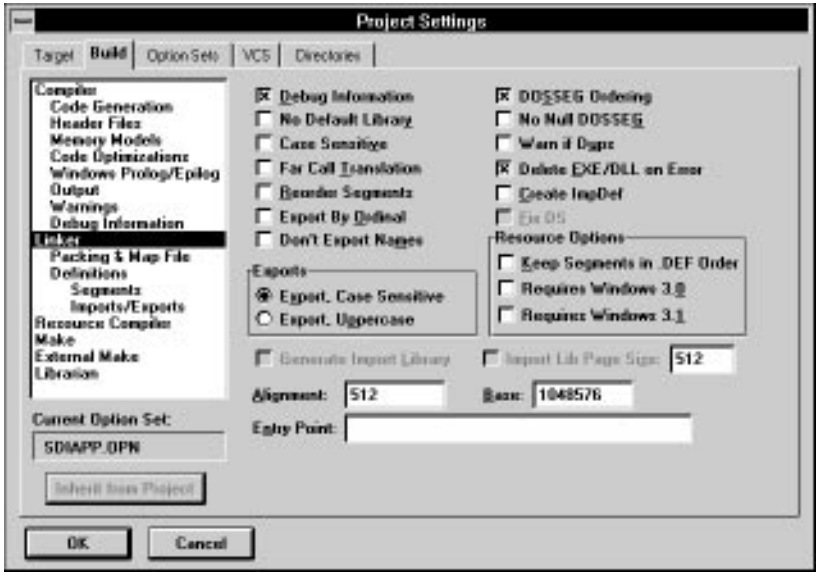

Figure 16-10 Linker subpage

Debug information
Places debugging information in the executable file. This is the normal option for linking an executable for debugging using the IDDE.

No default library
Causes the linker to ignore libraries specified in object files.

Case sensitive
Causes the linker to be case sensitive.

Far call translation
Causes the linker to convert intrasegment far calls to near calls.

Reorder segments
Places like segments in contiguous locations.

Export by ordinal
For 32-bit output, causes the linker to export symbols by ordinal. For 16-bit output, when this option is on, the name text for every exported symbol is moved from the resident name table to the nonresident name table.

Don't export names
Eliminates storage of name text for symbols exported by ordinal.

Export, case sensitive
Makes the linker treat the import and export symbols as case sensitive.

Export, uppercase
Forces the linker to convert import and export symbols to uppercase.

DOSSEG ordering
Causes the linker to perform the special segment ordering used by Microsoft high-level languages.

No null DOSSEG
Causes the linker not to offset the first segment by 10.

Warn if dups
Causes the linker to warn you if you have duplicate symbols in your code.

Delete EXE/DLL on error
Deletes target executable if a link error occurs.

Create ImpDef
Forces the linker to generate a `.din` file, which combines export information from your source, definition file, and options.

Fix DS
Turning this option on has the same effect as putting a FIXDS directive in the `.def` file.

Keep segments in .def order
Causes the linker to keep segments in the order in which they appear in the `.def` file (Windows only).

Requires Windows 3.0
Causes the linker to tag the executable as requiring Windows 3.0 or later to run.

Requires Windows 3.1
Causes the linker to tag the executable as requiring Windows 3.1 or later to run.

Generate import library
Directs the linker to build an import library (`.lib`) describing the exported symbols available to be imported from a DLL.

Import lib page size
Sets the page size for the Generate Import Library option.

Alignment
In conventional MS-DOS executables, causes the header to be rounded up to the specified size. In segmented `.exe` files, this option governs the page size.

Base
Sets the base address of the executable.

Entry point
Specifies the program entry point for Win32 applications.

Packing & Map File

The Packing & Map File subpage (Figure 16-11) controls the linker's output of cross-reference and map files.

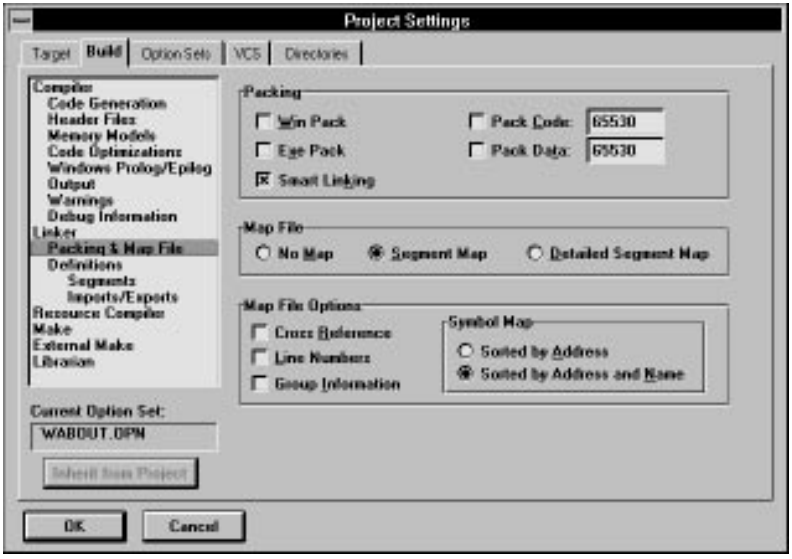

Figure 16-11 Packing & Map File subpage

Packing

This group of options controls how your target is packed.

Win pack
Packs Windows programs.

EXE pack
Compresses the executable file.

Smart linking
Enables smart linking of object files containing COMDAT records; only referenced COMDAT records are retained.

Pack code
Causes the linker to combine code segments. The size textbox specifies the maximum code segment size.

Pack data
Causes the linker to combine data segments. The size textbox specifies the maximum data segment size.

Map file

These options generate a file containing a list of segments, in the order of their appearance in the module. This group of options controls map file generation.

No map: No map file generated.

Segment map: Generates a list of segments, in the order of their appearance in the module.

Detailed segment map: Includes more detail about segment type, the modules that were added, the number of bytes per segment, and where each module begins.

Map file options

These options control the contents of the map file.

Cross reference
Causes the linker to generate a cross-reference list in the map file.

Line numbers
Controls whether line-number information is contained in the map file.

Group information
Enables output of group information.

Definitions

The Definitions subpage (Figure 16-12) contains the parameters necessary to create the `.def` file for an application.

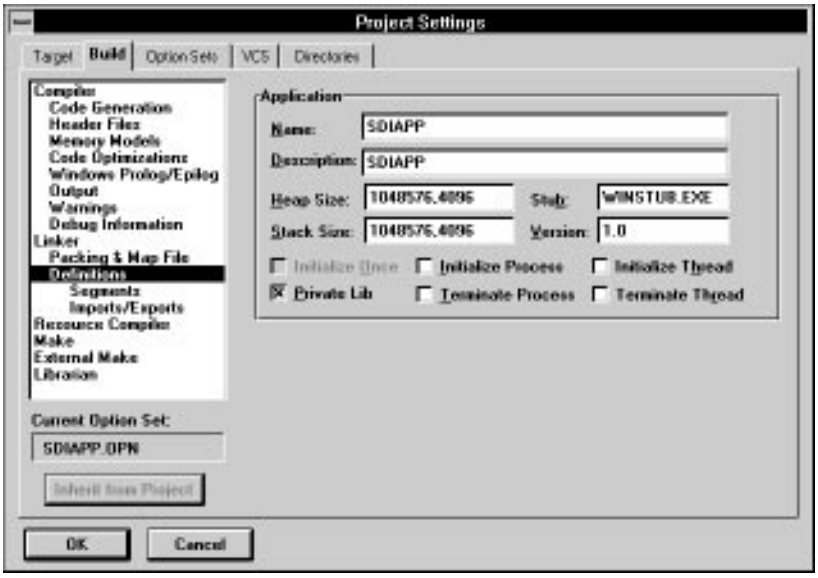

Figure 16-12 Definitions subpage

Name
Defines the name of the application, which is used by Windows to identify the application. A name is required for all Windows applications.

Description
Contains a description of the application. This optional string is placed in a Windows executable. It can be used for version control or to otherwise help identify the application.

Heap size
Specifies the size of the application's local heap. The default is 4096. If your application frequently uses the local heap, specify a larger heap size.

For Windows 3.1, heap size is a single text field. For Win32s, the field has two parts: Reserve and Commit, where Reserve is optional. Reserve tells Win32s how much heap space to try to get for this application. Commit specifies the amount of heap space the application actually needs. The two fields are separated by a comma (,). For example, 100000,4096 would specify 100000 for Reserve and 4096 for Commit.

Stack size
Defines the size, in bytes, of the executable's stack. The default is 4096 bytes. The stack is used for storing function arguments. You may need to increase the stack size, especially if your application contains heavily recursive functions.

For Windows 3.1, stack size is a single text field. For Win32s, the field has two parts: Reserve and Commit, where Reserve is optional. Reserve tells Win32s how much stack space to try to get for this application. Commit specifies the stack size the application actually needs.

Stub
Specifies an optional file that defines the executable stub to be placed at the beginning of the executable. When a user tries to run the application from DOS, the stub is executed instead. Many applications use the `winstub.exe` file supplied with the Windows SDK. You can use one of your own executables instead.

Version
Contains optional version information that becomes part of the executable file.

Initialize once
The DLL's initialization routine is called only when the module is initially loaded into memory.

Private lib
Creates a private DLL that is called.

Initialize process
The DLL's entry point is called when a process attaches.

Terminate process
The DLL's entry point is called when a process terminates.

Initialize thread
The DLL's entry point is called when a thread attaches.

Terminate thread
The DLL's entry point is called when a thread terminates.

Segments

This subpage, shown in Figure 16-13, provides segment information for the `.def` file.

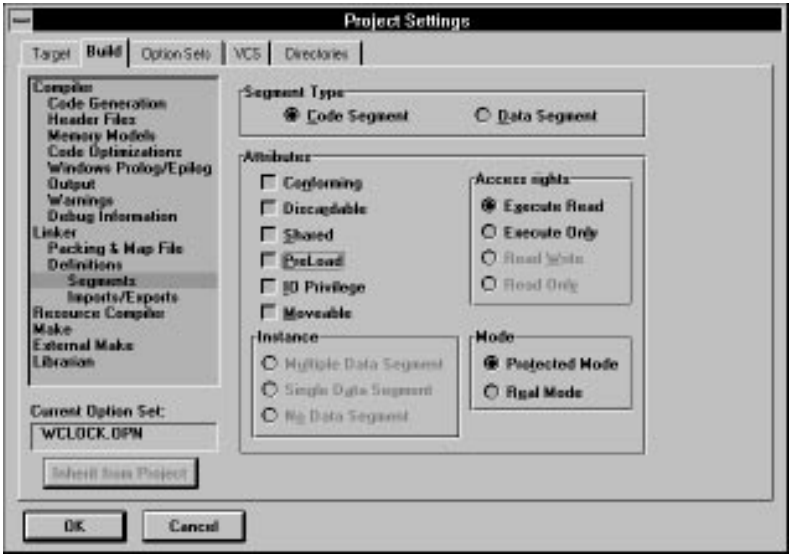

Figure 16-13 Segments subpage

Segment Type

This option toggles between two option sets: one for code segment, and one for data segment.

Code Segment: Shows options for the code segment.

Data Segment: Shows options for the data segment.

Attributes

These options define the attributes for the application.

Conforming
Turns on the conforming bits for the segment. This attribute can be set for code segments only.

Discardable
Lets the system flush the segment from memory. This attribute can be set for code segments only.

Shared
Lets multiple applications use this segment simultaneously (DLLs only). This attribute can be set for both code and data segments.

Preload
Loads the segment when the executable file or library is loaded. This attribute can be set for both code and data segments.

I/O privilege
Turns on the I/O privilege bit for the segment. This attribute can be set for code segments only.

Moveable
Lets the segment be moved when memory is compacted. This attribute can be set for both code and data segments.

Access Rights

These settings let you select access privileges for a segment.

Execute Read
Lets an executable read from or execute a segment, but not write to a segment.

Execute Only
Lets an executable execute, but not read or write, the segment. This attribute can be set for code segments only.

Read Write
Lets an executable read from a segment, or write to a segment, but not execute a segment.

Read Only
Lets an executable read from a segment, but not write to or execute the segment. This attribute can be set for data segments only.

Instance

These settings let you select the type of data segment generated.

Multiple data segments
Forces the generation of multiple data segments.

Single data segment
Forces the generation of a single data segment.

Mode

These settings let you select the type of executable file generated.

Protected mode
Causes the application to run in protected mode.

Real mode
Causes the application to run in real mode.

Imports/Exports

The Imports/Exports subpage (Figure 16-14) lets you define the names of routines in DLLs that your executable can use. It also lets you define the names of routines that your target (which must be a library) exports to other programs. The `IMPORTS` and `EXPORTS` statements are placed in the `.def` file.

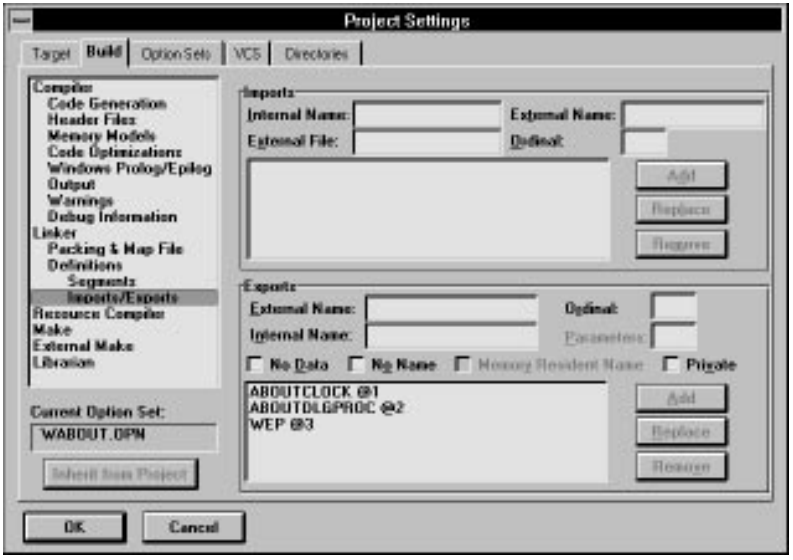

Figure 16-14 Imports/Exports subpage

Imports

This section defines the names of routines in DLLs that your application can use.

Internal name
Contains the name by which the imported routine is called internally. If omitted, the internal name is the same as the external name.

External file
Contains the name of the DLL from which the routine is imported.

External name
Contains the name of the DLL routine to be imported.

Ordinal
Specifies the ordinal number in the DLL of the routine to be imported. You can specify the ordinal or the external name, but not both.

Add, **Replace**, **Remove**
Clicking on the Add button adds the routine to the list of imported routines. Clicking on the Replace button uses the routine to replace the currently selected routine in the list. Clicking on the Remove button removes the currently selected routine from the list.

Exports

This section defines the names of routines that can be exported from your DLL.

External name
Contains the name by which the DLL routine will be known to other applications.

Internal name
Contains the name of the DLL routine to be exported.

Ordinal
Specifies the ordinal number by which the DLL can be referenced in other applications. This is optional unless No name is set.

Parameters
Specifies the total number of words occupied by the function's parameters. This option applies only to protected-mode functions with I/O privilege.

No data
Specifies that this function does not reference any data. This option applies only to real-mode Windows functions with I/O privilege.

No name
Specifies that the function can only be referenced by ordinal number.

Memory resident name
Makes the function name memory resident, even though an ordinal number is specified.

Private
Causes PRIVATE to be added to the names in your module definition file; this directs the IMPLIB utility to ignore the EXPORTS statements.

Add, **Replace**, **Remove**
Clicking on the Add button adds the routine to the list of exported routines. Clicking on the Replace button uses the routine to replace the currently selected routine in the list. Clicking on the Remove button removes the currently selected routine from the list.

Resource Compiler

The Resource Compiler subpage (Figure 16-15) contains options to control the resource compiler.

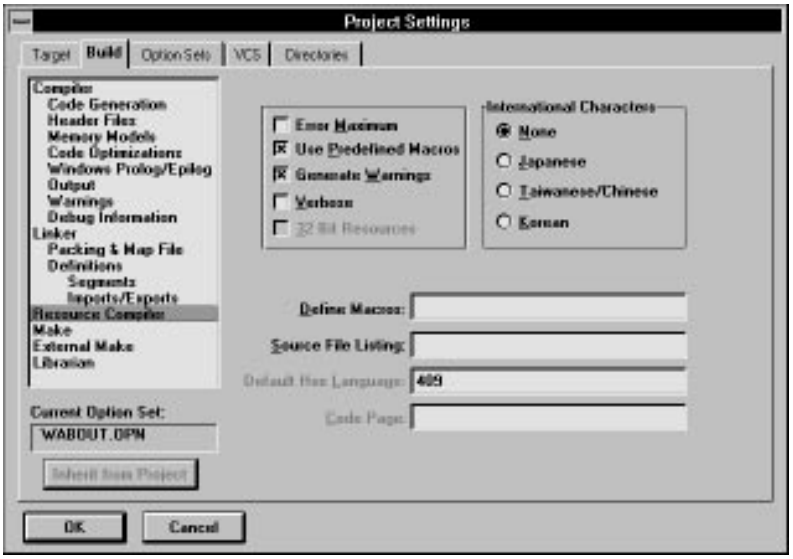

Figure 16-15 Resource Compiler subpage

Error maximum
Makes the resource compiler stop when its error limit is reached. If this option is off, the resource compiler processes the entire source file and displays all errors that have been detected.

Use predefined macros
Directs the resource compiler to use all predefined macros for the resource file.

Generate warnings
Shows warning messages.

Verbose
Shows greater detail when compiling resources.

32-bit resources
Specifies creation of 32-bit resource files.

Define macros
Specifies macros that should be defined for the resource compilers.

Source file listing
Specifies the name of the output listing file to create from the `.res` script.

Default hex language
Specifies the default hexadecimal language for 32-bit resources.

Code page
Specifies the code page used to convert strings in 32-bit resources to Unicode (currently unused).

International characters

This option specifies how the resource compiler interprets 2-byte Asian language character codes within character constants and strings. That is, if a character code represents the first byte of a 2-byte sequence, the second byte is not checked to see whether it is a backslash or a closed quote. The second byte cannot be a NULL (0), a carriage return (`0x0D`), or an end-of-file (`0x1A`).

None: Allows no 2-byte sequences.

Japanese: Signals the 2-byte sequence with a value in the range `0x81` ... `0x9F` and `0xE0` ... `0xFC`.

Taiwanese/Chinese: Signals the 2-byte sequence with a value in the range `0x81` ... `0xFC`.

Korean: Signals the 2-byte sequence with a value in the range `0x81` ... `0xFD`.

Make

The Make subpage (Figure 16-16) sets Make options for the IDDE built-in Make.

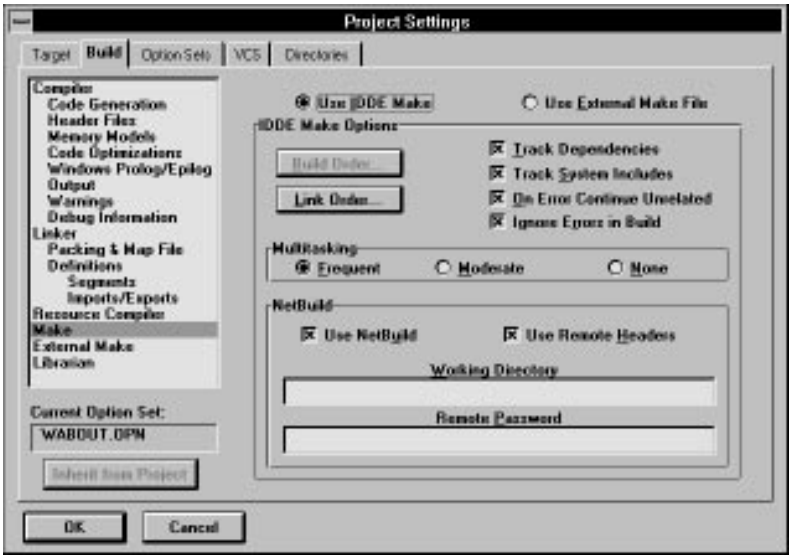

Figure 16-16 Make subpage

The radio buttons at the top of the dialog box control the Make program that will be run when the IDDE builds the project.

Use IDDE make: Use the built-in IDDE make tool.

Use external make file: Allows you to use an external Make program.

IDDE make options

This group of options governs the IDDE Make.

Build order
Opens the **Build Order** dialog box (see Figure 16-17).

Link order
Opens the **Link Order** dialog box (see Figure 16-18).

Track dependencies
Specifies whether or not to track dependencies (enabled by default). More time is required to track dependencies for large projects with many include files. The most effective way to use this option to establish the correct dependencies is to turn it on when you first build a project. Then turn it off, except when you change the dependency structure (by changing `#include` statements, for example).

If dependencies are tracked, the dependent files are shown in the Project window.

Track system includes
Specifies whether to track dependencies in system include files (disabled by default).

On error continue unrelated
Causes the IDDE Make to continue to build modules that are not dependent on the module in which the error occurred.

Ignore errors in build
Directs the IDDE Make to ignore errors and continue to build the target.

Multitasking
Affects the responsiveness of your system to a command to execute another task within Windows while the project is being built.

Frequent: Causes the IDDE to frequently give up time slices so other applications can execute faster.

Moderate: Causes the IDDE to give up some of the time to other applications.

None: Turns off multitasking. Other applications are suspended while the IDDE is building your project.

Netbuild

The IDDE allows the build process to be distributed among one or more remote servers. For more information, see Appendix C, “Using NetBuild.”

Use NetBuild
Enables distributed builds.

Use remote headers
Allows the remote server to use the header files provided with Symantec C++ on the build server. When this option is off, the build server takes the files from the local machine.

Working directory
Specifies the working directory on the remote server.

Remote password
Specifies the password for logging on to the remote server.

Build order

The Build Order button is only enabled when you have `.prj`, `.bat`, or `.mak` files in your project. Clicking on this button opens the **Build Order** dialog box, shown in Figure 16-17.

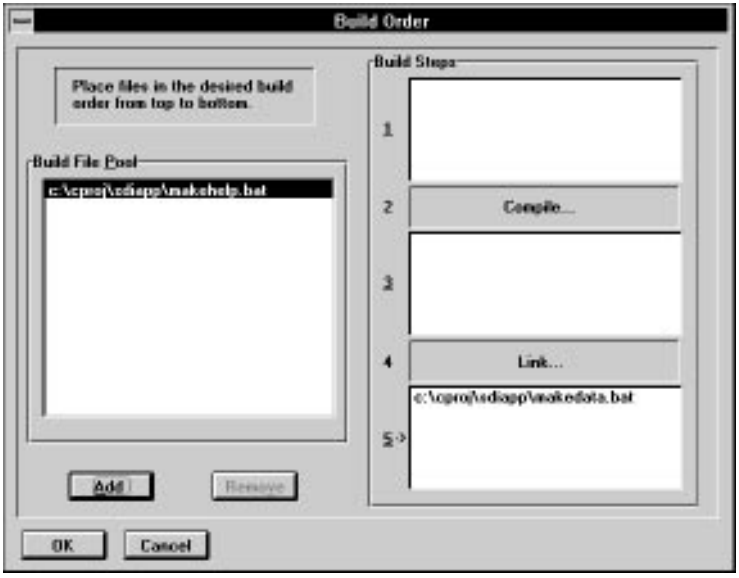

Figure 16-17 Build Order dialog box

The **Build Order** dialog box lets you specify the stage of the build in which you want your `.prj`, `.bat`, or `.mak` file executed. You do this by selecting a file from the Build File Pool list, clicking on one of the build steps on the right, then clicking on Add. The steps you can select are as follows:

Step 1: Happens before any compilation takes place.

Step 3: Happens after your files have been compiled into the object files, but before they have been linked to make the target.

Step 5: Happens after your final executable has been linked.

You can return any of the files to the Build File Pool list by selecting the filename in one of the steps and clicking on Remove.

When you are satisfied with the build order, click OK to return to the Make subpage.

Link order

The Link Order button opens the **Link Order** dialog box, shown in Figure 16-18.

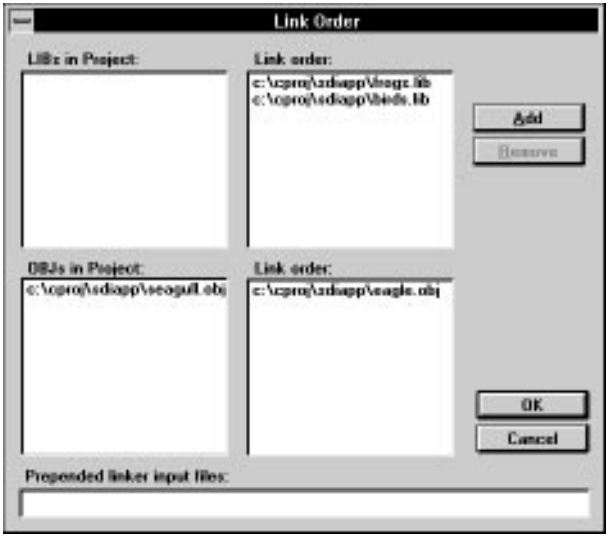

Figure 16-18 Link Order dialog box

The **Link Order** dialog box lets you specify the order in which libraries and object files added explicitly to the project are linked. You do this by iteratively selecting files from the LIBs in Project or the OBJs in Project listbox, then clicking Add to move the files to the Link Order listbox.

You can return a file to the LIBs in Project or the OBJs in Project listbox by selecting the filename and clicking on Remove.

The libraries added explicitly to the project are linked before other libraries. The object files added explicitly to the project are by default linked after the object files generated by compiling source files in the project. You can specify object files to be linked before any other object files in the Prepended linker input files textbox.

When you are satisfied with the link order, click OK to return to the Make subpage.

External Make

The options on this subpage govern the use of a make utility other than IDDE Make. Figure 16-19 shows the External Make subpage.

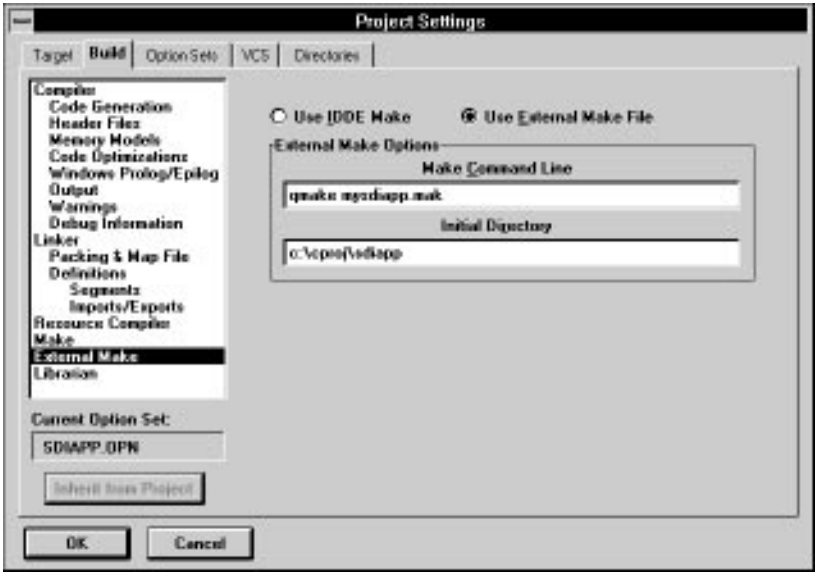

Figure 16-19 External Make subpage

Using external make

To use your own Make program:

1. Select Use External Make File.
2. Specify the `.exe` name of your Make program in the Make Command Line textbox, followed by arguments, if any. The IDDE expects the program's directory to be in the PATH environment variable.

3. If you want to change the Make's default directory, enter the change in the Initial Directory box. By default, this is the directory that contains Make.

4. You can use a Windows PIF file to further customize the way Make behaves.

The next time you use the **Build** or **Rebuild All** command, the IDDE runs your Make program from a DOS command-line window, which closes when Make is done.

The radio buttons at the top of the dialog box control the Make program that will be run when the IDDE builds the project:

Use IDDE make: Use the built-in IDDE make tool.

Use external make file: Allows you to use an external Make program.

Make command line
The Make Command Line textbox holds the `.exe` name of your Make program followed by arguments, if any.

Initial directory
Use this textbox to specify the directory from which the Make utility will run.

Librarian

The Librarian subpage (Figure 16-20) specifies options for building a library.

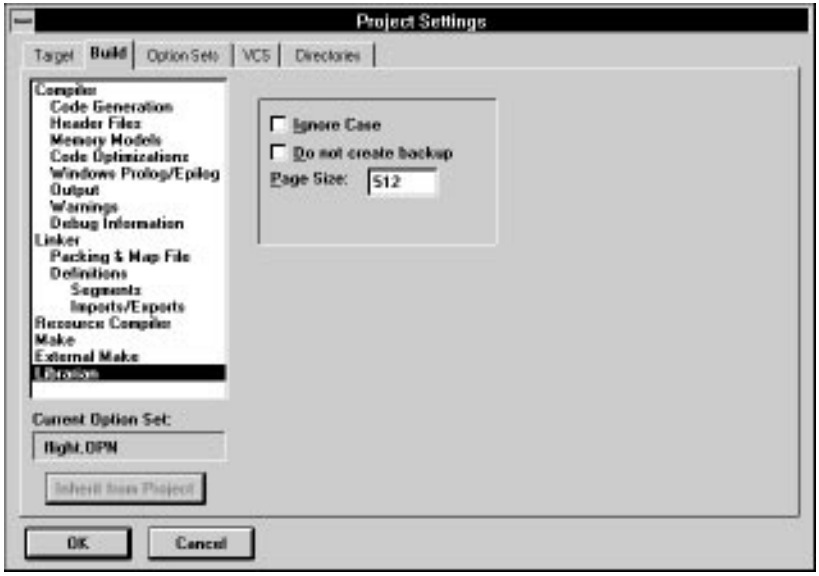

Figure 16-20 Librarian subpage

Ignore case
Directs the librarian utility to ignore case in symbols.

Do not create backup
Keeps the librarian utility from backing up the original library. When this option is turned off, the original library is saved in a backup file.

Page size
Specifies the library page-swapping size.

More about AppExpress 17

Chapter 4, “Generating an Application Framework,” defines an application framework and outlines the steps for generating a skeleton application using AppExpress. This companion reference chapter provides further detail concerning application types, program detail, program architecture, message maps, as well as generating and examining source files.

Selecting an Application Type

Select Application Type from the list of steps at the upper left of the AppExpress window. The options pane at the right contains three groups of controls, labeled Applications, OLE Options, and Project Options.

Figure 17-1 AppExpress application type options

Applications

The group of Applications radio buttons contains six categories of applications. Of these, only Quick Console does not use the MFC library. These categories are:

- Quick Console: The kind of application generated is determined by whether the 32-Bit Project box in the Project Options group is checked. See the information on the Project Option group later in this section.
- Dialog Box: This standard Windows dialog box can be either modal or modeless and can include dialog box controls such as push buttons, textboxes, and listboxes.
- Form or Database: This is a window with the functionality of a dialog box enhanced with scroll bars. It is appropriate for dialog boxes that have more than one screen of controls.
- Single Document Interface (SDI): This application uses a single window in which the application data is displayed. The OLE Options group of controls is enabled if this application type is selected.
- Multiple Document Interface (MDI): This application lets you display more than one window of data within a parent frame window. The OLE Options group of controls is enabled if this application type is selected.
- OCX Control in MFC: This template is for a custom control that is an OLE2 object.

The application types vary in the amount of functionality that AppExpress generates as part of the skeleton program. For example, SDI applications contain only one window, while MDI applications contain a main window and a variable number of child windows.

OLE Options group

The OLE Options group is enabled only if you select the SDI or MDI application type. The group contains four radio buttons:

- No Support: The application is not OLE-aware: it is neither a server nor a container. This option is the default.
- Server: The generated application acts as an OLE2 server. An OLE2 server can create or edit data, which OLE2 containers can link to or embed.
- Container: A host application. The generated application can contain OLE2 server data elements (OLE2 objects) within this host application's data.
- Server & Container: The generated application acts as both an OLE2 server and as an OLE2 container.

Project options group

The Project Options group contains two check boxes, Include Help and 32-Bit Project.

Checking Include Help tells AppExpress to generate the files necessary to build a Windows Help file for the specified application type. AppExpress creates a Help subdirectory named `hlp` beneath the project directory, which contains those files. It also creates the file `makehelp.bat`, which you run to compile a Windows Help file from the files AppExpress provides.

For all application types other than Quick Console, checking the 32-Bit Project box causes the MFC 3.0 to be used instead of the 16-bit MFC 2.5.

For Quick Console, leaving the box unchecked results in a skeleton WINIO program being generated. WINIO is a library that allows you to write simple Windows programs that perform input/output using standard C library functions (in other words, those functions prototyped in `stdio.h`.). If the 32-Bit Project box is checked, a Win32 console application is generated. Win32 console applications can only be run under Windows NT and Windows 95 (and not under Win32s).

If you check the 32-Bit Project box, your application can call the Win32 API.

Providing Miscellaneous Information

Selecting Miscellaneous in the steps list opens the Miscellaneous options page. This page lets you provide copyright information as well as a name for the project.

- In the first three fields, provide a company name, suffix, and copyright year.
- In the Project Name field, type a name for the project. (This name is also referred to as the Windows module name.)

Note

The module name is recorded in the module definition file generated by AppExpress. It can be changed by using the **Project Settings** dialog box, opened from within the IDDE by choosing **Settings** from the **Project** menu.

The Document/View Architecture

The MFC library provides a number of C++ classes that, when used together, create the object-oriented structure for your application. These are the document, view, frame window, and document template classes. The Form or Database, SDI, and MDI application types all use document/view architecture. This section introduces the MFC classes that implement this program structure.

Frame window

The frame window contains views on the data used by the application. In an SDI application, there is only one frame window, which is derived from the MFC class CFrameWnd. In an MDI application, there is a main frame window derived from CMDIFrameWnd, as well as document frame windows derived from CMDIChildWnd.

In addition to containing child view windows, the frame window handles all window management—for example, minimizing, maximizing, and closing the window. A standard toolbar and status bar also are displayed in this window.

View

Each frame window can contain a view on the data used by the application. A view is a C++ class derived from the class CView. Your application interacts with the user through this view class.

In the SDI frame window, AppExpress generates a child view window that takes up the client area of the frame window. (The client area refers to the part of the window in which the program's data is displayed. It excludes the window border, caption, and menu.) In an SDI application, this view window is given the default class name CSDIAPPView. All display and printing of the application's data is done using this view window and its class. The user's manipulation of the data is also done through the view.

Document

A document is a C++ class, derived from CDocument, that represents the data in your application. For example, a standard Windows application has a **File** menu that is a variant of the one shown in Figure 17-2.

Figure 17-2 Standard Windows file menu

When you choose **Open** from this menu, you are telling the program to open a document.

Note

The word document refers to whatever type of data is used by the application. It does not necessarily mean a text-based word processing document.

As the developer of the application, you write code in a CDocument-derived class to handle the operations on the **File** menu. An example of a skeleton CDocument-derived class follows.

```
class CSDIAPPDoc : public CDocument
{
protected: // create from serialization only
   CSDIAPPDoc();
   DECLARE_DYNCREATE(CSDIAPPDoc)
// Attributes
public:
// Operations
public:
// Implementation
public:
   virtual ~CSDIAPPDoc();
   virtual void Serialize(CArchive& ar);
#ifdef _DEBUG
   virtual void AssertValid() const;
   virtual void Dump(CDumpContext& dc) const;
#endif
};
```

Notice that no member variables are included with this class. It is up to you as the developer to add whatever data you need for your application.

In the example above, a `Serialize` method is included in the Implementation section of the class definition. This method, inherited from the base class, CDocument, performs object storage and retrieval to and from a disk file. In fact, the framework generated by AppExpress already handles the **File Open, File Save,** and **File Save As** operations by automatically calling the `Serialize` method in your CDocument-derived class. You can use the `Serialize` method to implement object persistence—the ability to preserve the complete state of objects across multiple executions of the program.

When you add your own member variables to this class, you should also override the `Serialize` method to read and write the added variables.

Pulling it all together: the document template

Creating and managing an application's frame window, views, and documents is the job of another C++ class, derived from the CDocTemplate class. In an SDI application, the CSingleDocTemplate class is used; in an MDI application, the CMultiDocTemplate class is used.

For more information, refer to the *Microsoft Foundation Class Library Reference*.

More about Message Maps

This section outlines the purpose of message maps and identifies the different parts of a message.

The rationale for maps

Using message maps saves you development time. The reason for this productivity improvement lies in the event-driven nature of Windows applications.

As a user of a Windows application clicks on buttons, selects menus, drags the mouse to highlight text, or performs any other mouse or keyboard action, the application is notified of this action through a Windows message. This message contains pertinent contextual information such as the screen coordinates at which the mouse was clicked, or an identifier indicating the button that was clicked.

The application developer decides which messages the application should respond to and how. If the developer decides not to write code to handle a particular message, the message can still be passed back to Windows to perform default processing.

If you write a Windows application using the Windows SDK, these decisions are most likely implemented as a switch statement in the main window procedure (usually referred to as a `WndProc`). For example, your window procedure might look like this:

```
LRESULT CALLBACK WndProc (HWND hwnd, UINT
message, WPARAM wParam, LPARAM lParam)
{
   switch (message) {
      case WM_PAINT:
         PAINTSTRUCT ps;
         BeginPaint(hwnd, &ps);
         MyPaintProc(hwnd, ps.hdc);
         EndPaint(hwnd, &ps);
         return(0);
      case WM_CREATE:
         hmenu = GetSystemMenu(hwnd, FALSE);
         AppendMenu(hmenu, MF_SEPARATOR, 0,
                    (LPSTR) NULL);
         AppendMenu(hmenu, MF_STRING, IDM_ABOUT,
                    "About...");
         break;
      case WM_DESTROY:
         PostQuitMessage(0);
         return(0);
   }
   return DefWindowProc(hwnd, message, wParam,
                        lParam);
}
```

If your application must respond to many types of messages, the window procedure can get quite large and become difficult to maintain. One of the advantages of the MFC library is the use of message maps, which drastically reduce the amount of code required to process messages.

Components of the message map

A message map is composed of the three components described in this section.

BEGIN_MESSAGE_MAP, END_MESSAGE_MAP macros
All message maps must begin and end with these macros. At run-time, the expanded macro sets up the message mapping between events and the code to handle the events.

ClassExpress-specific comment sections
AppExpress and ClassExpress add special-purpose comments to the message map so that ClassExpress knows where to add or remove mapping macros. Because ClassExpress provides an easy-to-use interface to your C++ class mappings, you should not manually edit the comments or code in a message map.

Message-mapping macros
If a C++ class has a method (that is, a class member function) to respond to a message, ClassExpress writes a message-mapping macro for that message in the class's message map. This macro begins with the prefix `ON_`, usually followed by the macro name for the Windows message. The mapping macro takes two parameters:

- The message identifier.
- The method that is called when the message event occurs at run-time.

  For example:

```
ON_COMMAND(ID_FILE_PRINT,
            CView::OnFilePrint)
```

  This macro sets up a linkage between the Windows message `WM_COMMAND` and the method `OnFilePrint`. This method is only called, however, if the `WM_COMMAND` parameter (in this case, the `wParam`) is equal to `ID_FILE_PRINT`. This mapping macro is equivalent to the following case statement in a window procedure:

```
case WM_COMMAND:
   if ( wParam == ID_FILE_FORMAT)
      CView::OnFilePrint ();
```

  Having the Express tools generate message maps lets you concentrate on the specific function of the application without having to worry about syntactical issues.

Generating and Examining the Source Files

Generating the source files of the application framework is as easy as clicking on a button. After the files are generated, you may want to examine the header and implementation files. AppExpress typically generates one header and one implementation file for each class. (An exception is the CAboutDlg class, which shares files with the application class.) Examining a few generated files in the IDDE, setting breakpoints on methods, and tracing through their code will help you understand the internal workings of the frameworks that

AppExpress generates. At that point you will then be ready to edit the code in order to enhance it as needed.

To generate and examine sample source files:

1. Launch AppExpress from the IDDE **Tools** menu, and make all the selections necessary to create an SDI application. When you click on Finish, AppExpress generates all the source files for this skeleton program. When it is done, AppExpress gives control to the IDDE (with the new project open), and then closes.

2. Open the Project window in the IDDE by clicking the Project View icon and dragging it onto the desktop, or by pressing Ctrl+Shift+P.

3. Double-click on the filename `mainfrm.h`. A Source window opens containing the header file `mainfrm.h`. (For more information on Source windows, see Chapter 2, “Introducing the IDDE.”)

This file, which AppExpress generated, contains the definition of the class CMainFrame. It is reproduced in its entirety below.

```
// mainfrm.h : interface of the CMainFrame class
//
// Copyright (c) XYZ Corporation, 1994. All Rights Reserved.
//
//
class CMainFrame : public CFrameWnd
{
protected: // create from serialization only
     CMainFrame();
     DECLARE_DYNCREATE(CMainFrame)
// Attributes
public:
// Operations
public:
// Implementation
public:
     virtual ~CMainFrame();
#ifdef _DEBUG
     virtual void AssertValid() const;
     virtual void Dump(CDumpContext& dc) const;
#endif
protected: // control bar embedded members
     CStatusBar m_wndStatusBar;
     CToolBar m_wndToolBar;
// Generated message map functions
protected:
     //{{AFX_MSG(CMainFrame)
     afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
          // ClassExpress will add and remove member functions here.
          // DO NOT EDIT these blocks of generated code !
     //}}AFX_MSG
     DECLARE_MESSAGE_MAP()
};
```

The file contains:

- A file header that uses the company name, suffix, and copyright year that you specified in AppExpress
- The CMainFrame class declaration

In the class declaration, notice that the status bar and toolbar are represented by class member variables, each of which is an instance of yet another C++ class. When you write code to manipulate the toolbar and status bar, you reference these member variables.

In the protected section of the class declaration is a prototype for a function that is called as part of the class's message map. As indicated, you should not edit the code in this section because it is reserved for ClassExpress**.**

AppExpress also generates a `.cpp`, or implementation, file for the CMainFrame class. This file has the same base name, `mainfrm`, as the class header file, but has the `.cpp` extension. To examine `mainfrm.cpp`, open this file in an IDDE Source window. This implementation file contains the following components:

- A file header.
- Preprocessor include statements for the required header files.
- A declaration of the CMainFrame message map containing a single entry (for the Windows `WM_CREATE` message).
- Static array data for initialization of the toolbar and status bar.
- The definitions of the CMainFrame constructor and destructor functions. Notice the comment in the constructor indicating where to add member initialization code. You may edit these functions to insert initialization and shutdown code for object instantiation.

- The definition of the class's method, `OnCreate`, for handling `WM_CREATE` messages.
- The definitions of two functions—`AssertValid` and `Dump`—that may be used during debugging.
- A final comment indicating where ClassExpress will add stub methods for new entries in the CMainFrame message map.

The next chapter discusses capabilities of ClassExpress, one of the tools (along with the Resource Studio) that you can use to enhance an application generated by AppExpress.

More about ClassExpress 18

Chapter 4 defined the concept of an application framework and outlined the steps for building on your skeleton application using ClassExpress. This companion reference chapter provides more detail about using ClassExpress, including:

- Deriving a class to handle user interface events in your program
- Working with Dialog Data Exchange (DDX) and Dialog Data Validation (DDV)
- Enabling a C++ class as an OLE2 automation server or client
- Deriving a C++ class from an existing Visual Basic custom control (VBX)

Deriving a Class to Handle User Interface Events

With ClassExpress, you can derive a new class designed to handle user interface events such as menu selections and button clicks directly from a Microsoft Foundation Class (MFC) library class. Suppose, for example, that you want to add a new dialog box to your application. To derive a class from the CDialog class using ClassExpress, follow these steps:

1. Create an application framework for a standard SDI program using AppExpress (for details, see Chapter 4, “Generating an Application Framework”).
2. Use ResourceStudio to create a new dialog box resource (see Chapter 7, “Adding Look and Feel with Resources”).
3. Launch ClassExpress from ResourceStudio or from the IDDE’s **Tools** menu.

4. Add a new class to your program by clicking on the Add Class button. (Follow the instructions in Chapter 4, "Generating an Application Framework.") Be sure to derive the new class from the Dialog class type.

   In ClassExpress, all of the base classes from which you derive new classes are themselves derived from the MFC class CCmdTarget. The MFC Library Reference defines CCmdTarget as the base class for the message map architecture. Any class derived from CCmdTarget inherits the ability to respond to user interface events such as menu and toolbar selections and dialog box actions.

5. In the ClassExpress main window, verify that your class has been created by browsing through the Class drop-down list.

6. Select the new class name from the Class list. Note that the list of Control IDs and Windows messages for your derived dialog box class is different from the list for non-dialog classes.

Note

ClassExpress filters out the Control IDs and Windows messages that do not apply for the selected class name. For example, dialog box classes can handle the `WM_INITDIALOG` message, but this message is not handled by any class derived from CFrameWnd.

The ability to add new classes that derive specialized message-handling functionality from the base MFC classes is one of the benefits of using ClassExpress**.** The following material summarizes the type of functionality that a new class inherits if it is derived from CCmdTarget. For more information on any of these base classes, refer to the *Microsoft Foundation Class Library Reference.*

- CmdTarget: The base class for all MFC classes that offer support for Windows message handling. You probably will not derive a new class directly from CmdTarget; instead, use the other base classes in this list.

- Dialog: This class implements dialog boxes, either modal or modeless. It usually is associated with a dialog box resource template created in ResourceStudio. Member variables of this class typically are mapped to fields or controls in the dialog box. For details on how this mapping is established, see the next section, "Working with Data Transfer: DDX and DDV."

- Document: The application's data is represented by the document class. The data can be anything the programmer chooses. All file input and output should be handled within the document class.

- FormView: A class of views that has built-in support for scrolling and for child controls. FormViews typically are combined with a dialog box resource template. One way in which the FormView class differs from the Dialog class is its added support for scrolling.

- FrameWnd: The main window class for single document interface (SDI) applications.

- MDIChildWnd: The child document window class for multiple document interface (MDI) applications.

- ScrollView: A class of view window that supports scrolling.

- View: The base class that provides the connection between the document class representing the program data and the user interface to that data.

- Wnd: The base class for any window, including dialog boxes, frame windows, views, and dialog box controls. Because this class is used to derive many of the other classes listed here, choose Wnd as a base for a new class if other choices do not satisfy your programming requirements.

- Splitter: This special type of window can contain multiple panes. A pane is usually a window that is associated with a View-derived class in the application.

Working with Data Transfer: DDX and DDV

The previous section discussed the architecture classes that you use to build an object-oriented Windows program with the MFC library. The application, document, view, template, and frame window objects are the key components of a standard Windows, MFC-based application.

This section describes MFC library support for data transfer between dialog boxes or other windows and your C++ objects that store that data. This is referred to as Dialog Data Exchange (DDX) and Dialog Data Validation (DDV). You use the Data Transfer options of ClassExpress to bind class member variables to dialog box or window controls.

Note

Data transfer using DDX/DDV can be applied to any window that is derived from the MFC base class, CWnd. It is not restricted to dialog boxes derived from CDialog.

Dialog Data Exchange with Windows 95 Common Controls is also supported.

Implementing Dialog Data Exchange (DDX) using ClassExpress

To implement DDX using ClassExpress, use the following steps. Before performing these steps, generate a dialog box application with AppExpress, then:

1. Add a few edit controls to your skeleton dialog box using ResourceStudio.

2. Launch ClassExpress from ResourceStudio or from the IDDE's **Tools** menu. ClassExpress loads the project and displays the Message Maps options.

3. In the upper-left listbox, click on Data Transfer. From the Class drop-down list, select the CMainDialog class.

The ClassExpress window should look similar to that shown in Figure 18-1.

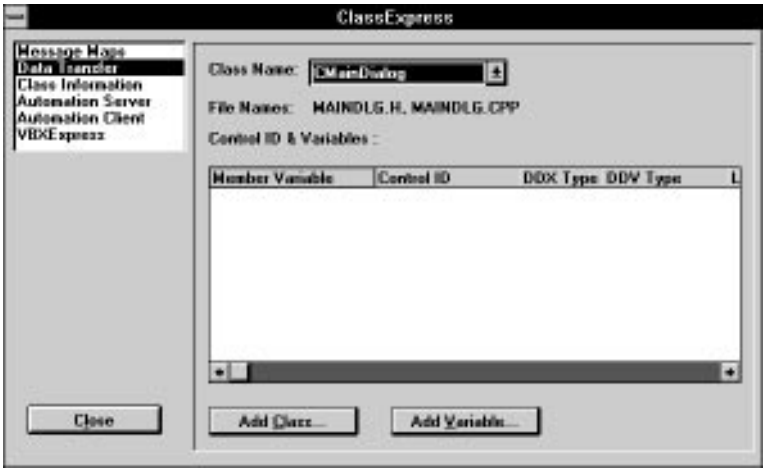

Figure 18-1 Data Transfer page in ClassExpress

4. Click on the Add Variable button. The **Add Member Variable** dialog box shown in Figure 18-2 opens.

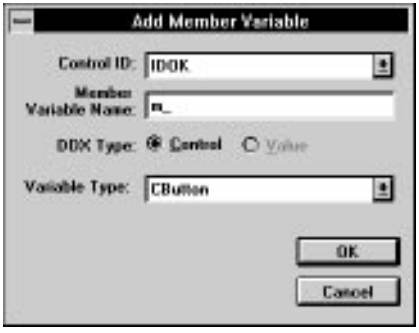

Figure 18-2 Add Member Variable dialog box

5. From the Control ID drop-down list, select a dialog box control that you want to map to a class member variable.

6. Edit the Member Variable Name to specify the name of a variable to be mapped to the selected Control ID. The variable does not have to exist in the class. (ClassExpress adds it automatically to the class for you.)

Note

Nonstatic class member variable names are usually prefixed with `m_` for easy identification; however, you are not required to follow this convention.

7. For DDX Type, select Control to map the control ID and variable name to a control class (such as CButton or CEdit). Select Value to map to a CString or to a numeric type.

8. Select a Variable Type for this member variable from the Variable Type drop-down box. The available types depend on the type of control being mapped and the DDX Type option.

9. If you selected Value from the Type radio buttons, one or two additional fields are displayed along the bottom of the dialog box. If the added member variable is of a numerical variable type, two fields are displayed, which allow you to set the minimum and maximum ranges for the variable's value. If the variable type is CString, then only one field is displayed, in which you specify the maximum number of characters that the CString can contain. (You cannot specify a minimum number of characters.)

   Figure 18-3 shows an example using a CString variable type.

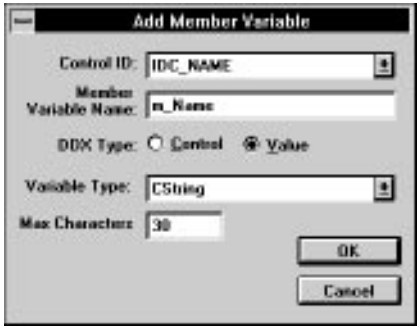

Figure 18-3 Adding a CString member variable

10. Click OK. You are now back in the ClassExpress window, and the member variable you just added is displayed in the spreadsheet. If you added minimum or maximum values for your member variable, a dialog data validation function name is displayed in the DDV type field.

Understanding data transfer at the source code level

The procedure in the preceding section instructs ClassExpress to establish a link between your CMainDialog class and a control within a dialog box. In the implementation file `maindlg.cpp` are calls to virtual functions that CMainDialog inherits from other MFC classes. These functions perform the dialog data exchange and validation (DDX/DDV) for your program.

This section explores the source code generated by ClassExpress and explains the implementation of data transfer—the first step in how DDX functions bind member variables to dialog box objects to transfer data between the variables and the controls. Next, the validation of values entered into dialog box controls is explained. The third section describes how data transfer functions are invoked by the `UpdateData` function.

Dialog Data Exchange (DDX)

To implement DDX, follow these steps:

1. Select **Open** from the IDDE's **Project** menu to open the dialog box project you created earlier.

2. Open the `maindlg.cpp` file in a Source window. Either double-click on `maindlg.cpp` in the Project window or open an empty Source window, then open the source file using the **Open** command in the **File** menu.

3. From the **Edit** menu, choose **Find** and search for `DoDataExchange`. Look at the definition for the method `CMainDialog::DoDataExchange`. ClassExpress has overridden the `CWnd` method `DoDataExchange` in your CMainDialog class. A sample implementation of DoDataExchange follows.

```
void
CMainDialog::DoDataExchange(CDataExchange* pDX)
{
   CDialog::DoDataExchange(pDX);
   //{{AFX_DATA_MAP(CMainDialog)
   DDX_Control(pDX, IDABOUT, m_About);
   DDX_Control(pDX, IDOK, m_OKButn);
   DDX_Control(pDX, IDCANCEL, m_CancelButn);
   DDX_Control(pDX, IDHELP, m_Help);
   //}}AFX_DATA_MAP
}
```

DDX lets you copy data easily from class member variables to dialog box controls, then from the controls back to the member variables. This is accomplished by implementing an override for the `CWnd::DoDataExchange` method in your dialog box class. For each control that is mapped to a member variable, ClassExpress generates a call to a DDX function. There are four DDX function calls in the code sample shown above.

DDX functions take the form:

```
DDX_xxx(pDX, nIDC, Data);
```

where:

- `pDX` is a pointer to a `CDataExchange` object. This object contains context information such as the dialog box instance and whether the data exchange is from the member variable to the control or vice versa.
- `nIDC` is the dialog box control ID.
- `Data` is the member variable in your dialog class.

Note

DDX functions that exchange data with Visual Basic custom controls (VBXs) take an additional parameter, nPropIndex—the property index being exchanged. This parameter is shown before the Data parameter.

The preceding sections cover how ClassExpress prompts you for new member variables, then generates code that performs automatic data exchange between those variables and their respective dialog box or window controls. The next section describes how to enhance data exchange by using data validation.

Dialog Data Validation (DDV)

When adding a new variable using the Data Transfer options in ClassExpress, you can define minimum and maximum values for numeric variables and maximum lengths for CString variables. This is illustrated in Figure 18-4 below.

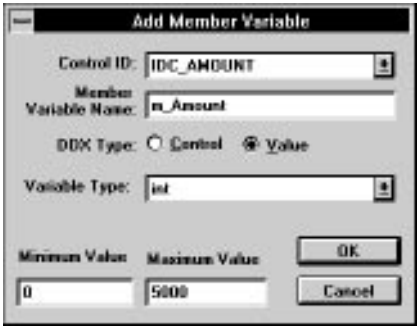

Figure 18-4 Adding a numeric member variable

In this example, the member variable `m_PayTo` is defined as an integer and is limited to values between 100 and 5000. This variable is bound to a dialog box edit control identified by `IDC_PAYTO`. The user of the application is not allowed to enter a value in that control that is outside the minimum and maximum bounds.

You do not have to write a single line of code to enforce this rule. The DDV functions in the MFC library do this for you. After adding the variable `m_PayTo`, as shown earlier, and clicking Close in ClassExpress's main window, ClassExpress writes the following lines to the `DoDataExchange` method of your dialog box class:

```
DDX_Text(pDX, IDC_PAYTO, m_PayTo);
DDV_MinMaxLong(pDX, m_PayTo, 100, 5000);
```

The first line binds the edit control to your `m_PayTo` member variable. This uses a DDX function, as discussed earlier in this chapter. The second line is the DDV function that limits values in the textbox control to between 100 and 5000.

Note

For each member variable, the DDV function call should immediately follow the DDX function call in your `DoDataExchange` function. This is a requirement of the application framework and is enforced when ClassExpress writes new DDX/DDV function calls to the source code file.

DDV functions take the following form:

```
DDV_xxx(pDX, Data, ...);
```

where:

- `pDX` is a pointer to a `CDataExchange` object. This object contains context information such as the dialog box instance and whether the data exchange is from the member variable to the control or vice versa.
- `Data` is the member variable in your dialog class.
- `...` indicates the remaining arguments: minimum and maximum values for numerical variables, and maximum number of characters for strings.

Calling UpdateData

Your dialog class's `DoDataExchange` method is called by another CWnd method, `UpdateData`, whose prototype follows:

```
BOOL UpdateData(BOOL fSaveOrValidate);
```

If the parameter to `UpdateData` is `FALSE`, then the function updates the dialog box controls with data from class member variables that have been mapped to the controls. If the parameter is `TRUE`, then the member variables are updated with data from the controls and validated.

You call `UpdateData` from the places in your program at which you want to exchange data with the dialog box. `UpdateData` is called for you automatically in only one place in the dialog initialization.

In response to the WM_INITDIALOG message, the CDialog::OnInitDialog method calls UpdateData with a parameter equal to FALSE, indicating that the controls are being set. Initialize the values of a dialog class's member variables in your OnInitDialog method. For example:

```
BOOL CMainDialog::OnInitDialog()
{
   m_ColorIsRedCheckBox = TRUE;
   m_Filter = FILTER_NONE;
   CDialog::OnInitDialog();
}
```

Here two member variables, m_Color and m_Filter, are set to initial values. When the CDialog::OnInitDialog method is called, it uses those values to set the state of the dialog box controls mapped to these member variables.

Making Your Application an OLE Automation Server

This section covers the following topics:

- The definition of an OLE automation server
- The difference between creating an automation server in ClassExpress and a standard OLE server in AppExpress
- The mechanics of creating an OLE automation server using ClassExpress
- The source code that ClassExpress generates to implement OLE automation

Note

Use of the acronym OLE (object linking and embedding) in this section refers to version 2 of OLE, which includes automation support.

What is an OLE automation server?

OLE automation is an architecture that allows programs to manipulate objects within other applications. The application that defines the objects is called the automation server. Any application that uses OLE to manipulate another application's objects is called an automation client. For example, Microsoft Excel is an automation server, and Microsoft Visual Basic is an automation client. From within Visual Basic, you can write a program that loads an Excel spreadsheet, runs Excel macros, and saves the spreadsheet.

OLE automation server vs. OLE server

OLE automation is an extension of the original object linking and embedding technology that originated in version 1 of OLE. The linking and embedding features determine how data from one application is used in another. For example, you can embed a spreadsheet document within a word processing document, or link a word processing document to a spreadsheet document that exists in a file.

Note

An embedded object's data is saved as part of the client application's data. A linked object's data is saved independently of the client's data; the client's data contains a reference to the filename of the linked data.

In these examples, the application that is the container for the spreadsheet is called an OLE client or a container application. The application that originally created the spreadsheet and that is used to edit the spreadsheet is called the OLE server application. The server application is responsible for creating and maintaining data objects embedded in or linked to another application.

When you create an OLE server application with AppExpress, you are making it possible for your application's data to be embedded in or linked to another application's data.

OLE automation was introduced in the second version of the OLE technology. Automation has nothing to do with embedding or linking to data objects. It is used to manipulate objects that an OLE automation server has created.

Using OLE automation to manipulate an object allows you to do some or all of the following:

- Query and change the properties of an object
- Call functions that are defined in the object
- Be notified when an object triggers a specific event

Although an OLE server might provide embedding, linking, and automation support, all three options do not have to be provided. For example, you could create an automation server that only performs mathematical calculations, has no user interface, and is accessible only by function calls through the OLE automation interface. In this case, you only need an OLE automation server. Linking and embedding technology, which is packaged in a standard OLE server, is not required.

Enabling your application to be an OLE automation server

This section assumes that you have generated a sample MFC-based application framework with AppExpress. You use that framework in this section.

To enable your application to be an OLE automation server:

1. With the project containing your sample application framework loaded in the IDDE, launch ClassExpress by choosing it from the **Tools** menu in the IDDE's main window.

2. To act as an OLE automation server, your application needs a C++ class that defines an automation object. In ClassExpress, you take care of this by adding a class. Click on the Add Class button.

3. Select a Class Type. This specifies the MFC class from which your new class is derived.

4. Type the name of the class in the New Class Name textbox.

5. Check the OLE automation box. If your class is derived from CCmd Target or CWnd, the Creatable check box and the External Name textbox become visible.

6. For CCmdTarget- and CWnd-derived classes, if you want OLE client applications to be able to create instances of your OLE automation object, check the Creatable box as well.

7. For CCmdTarget- and CWnd-derived classes, enter an External Name that will be used by OLE automation client applications to identify your automation object.

8. Click OK. At this point, ClassExpress generates the new class in your project's source code and reports that the classes were generated correctly. You are returned to the main ClassExpress window.

Completing these steps creates the basic source code structure for your OLE automation class. However, you may also want to take advantage of ClassExpress's ability to add functions and properties to your class that will be exposed by OLE automation.

Adding exposed functions to an automation server class

First, select the class from the Class Name drop-down list. To add a function, follow these steps:

1. Click the Add Function button on the Automation Server page of ClassExpress. The **Add Function** dialog box opens.

2. In the External Name textbox of **Add Function**, type the name by which automation clients will refer to the new function. As you type, the Internal Name textbox mirrors the name you are entering.

3. If you want your new function to have an Internal Name different from its External Name, type the desired Internal Name in that textbox. The Internal Name is the name the function will have in your source code.

4. Select the function's Return Type from the drop-down list with that label. In addition to the standard types `void`, `short`, `long`, `float`, and `double`, this list contains the OLE types `CY`, `DATE`, `LPDISPATCH`, `SCODE`, `BOOL`, `VARIANT`, and `LPUNKNOWN`. If you are unfamiliar with these latter types, consult the *OLE2 Programmer's Reference* for their definitions.

5. If your function takes arguments, add them one at a time by clicking on the Add button at the bottom of the Parameters List listbox. The **Add Parameter** dialog box that is displayed lets you specify the Name and Type of a parameter. Enter the parameter's name in the Name textbox. Select its type from the Type drop-down list. (The Type list contains more OLE types. See the *OLE2 Programmer's Reference* for details.) Click OK. You are returned to the **Add Function** dialog box, in which the parameter you just specified is displayed in the Parameters List listbox.

   Follow the procedure just described to add any other parameters the function requires.

6. Click OK in the **Add Function** dialog box. You are returned to the Automation Server page of ClassExpress. The function you just created is added to the Name listbox, and is referred to using its external name. When this function is selected in the Name listbox, the Implementation listbox displays the prototype of the function that ClassExpress generates to implement it.

Adding properties to an automation server class

First, select the class from the Class Name drop-down list. To add a property, follow these steps:

1. Click the Add Property button on the Automation Server page of ClassExpress. The **Add Property** dialog box opens.

2. Select one of the radio buttons to the right of the Implementation label. If you want to grant automation clients read-only access to the property you are adding, select Variable. If you want to grant read/write access to the property, select Get/Set Function.

3. In the External textbox, type the name by which automation clients will refer to this property.

As you type, the values of the Get Function and Set Function fields are automatically filled in using the external name *extname* you specify. If you are creating a read-only property, it will be implemented as a member variable in your automation class. The Get Function textbox will contain the proposed name of that variable, `m_`*extname*. If you are creating a read/write property, it will be implemented by a pair of member functions—a Get and a Set function—in your automation class. The Get Function and Set Function textboxes will contain the proposed names of those functions, `Get`*extname* and `Set`*extname*.

4. Select the Type of the property from the drop-down list with that label. In addition to the standard types `short`, `long`, `float`, and `double`, this list contains the OLE types `CY`, `DATE`, `LPDISPATCH`, `SCODE`, `BOOL`, `VARIANT`, and `LPUNKNOWN`. If you are unfamiliar with these latter types, consult the *OLE2 Programmer's Reference* for their definitions.

5. If you want, change the names of the Get and Set Function.

6. Click OK in the **Add Property** dialog box. You are returned to the Automation Server page of ClassExpress. The property you just created is added to the Name listbox and is referred to using its external name. When this item is selected in the Name listbox, the Implementation listbox displays the member variable or the pair of functions that ClassExpress generates to implement the property.

OLE automation server source code

This section examines some of the source code generated by ClassExpress that enables an application to be an OLE automation server.

As explained in the previous section, you create a new class using ClassExpress and indicate that the class should support OLE automation. When the source code is generated, the constructor for your new class contains the following code:

```
   EnableAutomation();
   // To keep the application running as long as
   // an OLE automation
   // object is active, the constructor calls
   // AfxOleLockApp.
   AfxOleLockApp();
```

OLE automation is first enabled for this object using the MFC function `EnableAutomation`. This function should only be called if there is a dispatch map declared for the class (discussed in more detail below). If your new class objects are creatable by other applications, then the MFC function `AfxOleLockApp` is called in your constructor.

This function increments a global count of the number of times this object has been activated by OLE clients. It is the MFC library's way of ensuring that an object is not destroyed by one OLE client, if it is in use by another client. In the destructor for your class, ClassExpress has written a call to the function `AfxOleUnlockApp`, which decrements the global count for this object.

ClassExpress also creates a new macro structure called a dispatch map, as shown in the following example:

```
BEGIN_DISPATCH_MAP(COLEAutoObject, CCmdTarget)
   //{{AFX_DISPATCH_MAP(COLEAutoObject)
      // NOTE - the ClassExpress will add and
      // remove mapping macros here.
   //}}AFX_DISPATCH_MAP
END_DISPATCH_MAP()
```

As you can see, a dispatch map is similar to a message map, the MFC library macro for routing Windows messages to your class methods. Like message maps, you do not edit the dispatch map directly; AppExpress and ClassExpress do that for you. The dispatch map is a macro that generates dispatch table information used by the MFC library's OLE classes to route automation requests.

In addition to the OLE initialization code in the constructor and the dispatch map, ClassExpress optionally writes the following macro to your class implementation file (`.cpp`):

```
IMPLEMENT_OLECREATE(COLEAutoObject,"MYOBJ",
        0xd73cfd60, 0x3cea, 0x101b, 0x80, 0x60,
        0x4, 0x2, 0x1c, 0x0, 0x94, 0x2)
```

This macro is written if you selected the Creatable check box when you added the new class in ClassExpress. The macro allows your OLE automation object, as defined by your new C++ class, to be created dynamically by an OLE client application. If you do not have this macro, your automation object would have to be created by your application before any OLE client could manipulate it using the automation interface.

The `IMPLEMENT_OLECREATE` macro takes these arguments:

- The new class name
- The external name of the object
- The components of the class's OLE class ID

The last parameters, the components of the OLE class ID, together represent a 128-bit value that uniquely defines the OLE object within Windows.

Another source code file created by ClassExpress has a filename consisting of the project name with the extension `.odl`. This file contains the Object Description Language implementation for your automation object class. You run the utility program `mktyplib.exe`, passing the `.odl` file as an argument. The result is a type library file with an extension of `.tlb`.

The type library file is used by OLE automation clients to query the objects, properties, and functions exposed by your application.

This section has given you a view of the source code generated for an OLE automation server. OLE automation has many more facets that you should explore. Refer to the *Microsoft Foundation Class Library Reference* and the *OLE2 Programmer's Reference* for additional details.

Making Your Application an OLE2 Automation Client

The Automation Client selection of ClassExpress lets you make an application an OLE2 automation client for type libraries that export an OLE2 automation interface. ClassExpress creates a C++ class in your program that acts as an interface to the type library class.

This section uses the OLE2 sample type library `hello.tlb`, found in `samples\ole16\hello` below your Symantec C++ installation directory. If you do not have this sample file installed, you can select any other type library file that exports an OLE2 automation interface. Otherwise, you should install the OLE2 samples.

To make your application an OLE2 automation client:

1. Launch ClassExpress and, if necessary, open a project. For this example, the application type of the project does not matter.

2. Select Automation Client from the list at the upper left of the ClassExpress window.

3. Click the "..." button to the right of the Type Library File field. The **Open** dialog box is displayed.

Note

This dialog box has filtered out all filenames that do not end with `.tlb` or `.olb`. These extensions are used for OLE type library files.

---

4. Browse to the type library file `samples\ole16\hello\hello.tlb` within the Symantec C++ installation directory.

5. Click OK in the **Open** dialog box. The type library file should now be displayed in the Type Library File field in the ClassExpress window.

Note

Class names representing exported OLE2 automation interfaces in the type library are displayed in the Class list.

6. Select `_DHello` from the Class list.

7. ClassExpress fills in the New Header File and New Implementation File fields with the suggested file names for the new C++ class that will be generated in your framework.

8. Click on the Generate button. ClassExpress creates the header and implementation file for your new class.

The C++ class `_DHello` that was generated by ClassExpress represents the client OLE automation interface to the `_DHello` type defined in `hello.tlb`. This class provides a number of methods that simplify the processes of attaching to and detaching from an OLE automation dispatch connection. ClassExpress generates additional methods that simplify the interface to `IDispatch::Invoke`. If you are not familiar with the OLE automation architecture, refer to the *OLE2 Programmer's Reference*.

Creating a C++ Wrapper Class for an Existing VBX

The VBXExpress section of ClassExpress lets you incorporate VBX controls into an MFC application. ClassExpress makes it easy to use a VBX by generating a C++ wrapper class through which you can directly program the VBX. The wrapper class makes use of the MFC base class CVBControl, which provides access to VBX properties and events.

You can use VBXExpress with any 16-bit MFC application. (The restriction arises because VBXs themselves are inherently 16-bit.) Follow these steps to create a wrapper class for a VBX and add it to the class hierarchy of a project:

1. Launch ClassExpress—either from the IDDE or standalone. If you launch ClassExpress standalone, specify your project in the **Open** dialog box.

2. Select VBXExpress from the list at the upper left of the ClassExpress window.

3. The control labeled VBX File displays the name of the currently selected VBX file. Click the button labeled "...", to the right of the control. An **Open** dialog box is displayed. Select a VBX file and click OK.

   VBX File displays the name of the file selected.

   ClassExpress examines the VBX file to determine the name and capabilities (events and properties supported) of every VBX control contained in the file. (There can be more than one.) The drop-down list labeled VBX contains the names of all VBXs in the file. The capabilities of the control selected from this list are displayed in the Property and Event list.

4. In the textbox labeled ClassName, specify the name of the C++ wrapper class that will be generated for the currently selected VBX control. In the textboxes labeled Header File, specify the names of the C++ files that will contain the definition and implementation of the class. Default names are provided in all three fields; you don't have to change them.

5. Repeat the above step for each VBX control in the VBX file.

6. Click on the Generate button. ClassExpress creates the C++ wrapper classes.

Examine the interface to the VBX control(s) generated by ClassExpress by opening the header file in an IDDE source window. You will see that the methods of the C++ class correspond in a natural way to the properties and events of the VBX.

Summary

This chapter covers ClassExpress in detail discussing these main points:

- Deriving a new class in an application from one of the many MFC library base classes

- Establishing links between class member variables and dialog box or window controls, and validating user entry without having to write a single line of code
- Manipulating objects within an application from any OLE client application
- Conversely, enabling an application to manipulate OLE automation objects in other applications
- Extending the power of an application with Visual Basic custom controls, without having to write the control yourself

With this information, you are able to add significant new functionality in a short period of time to the skeleton application framework that AppExpress generates.

Class Editor Reference 19

This chapter describes menu operations and mouse functions available in the Class Editor. For an introduction to this tool, see Chapter 5, “Defining Classes and Their Hierarchies.”

The Class Editor Window

There are several ways to open a Class Editor window:

- Choose **Class Editor** from the **Goto View** submenu of the IDDE’s **Window** menu.
- Double-click on the Class Editor icon in the Views toolbox, or drag the icon from the toolbox to the desktop.
- Choose **New!** from another Class Editor window’s menu bar.
- Double-click on a class in the Hierarchy Editor graphical display.
- In the Class pane, move the cursor towards the left margin until its orientation changes and it points to the right. Then highlight one or more classes and drag them onto the desktop.
- In the Members pane, move the cursor towards the left margin until its orientation changes and it points to the right. Then highlight one or more members and drag them onto the desktop.
- Double-click on an entry in the Query Implementors window.

- In the Query Implementors window, move the cursor towards the left margin until its orientation changes and it points to the right. Then highlight one or more entries and drag them onto the desktop.

Multiple instances of the Class Editor may be open at the same time. These windows share data among themselves and with Hierarchy Editor windows, so changes made in one window are transmitted instantly to the others.

Edit menu commands

With one exception, the commands in the **Edit** menu (Figure 19-1) operate on text in the Source pane and are identical in function to the corresponding **Edit** menu commands in Source windows. For a description, see Chapter 21, "Text Editor Reference."

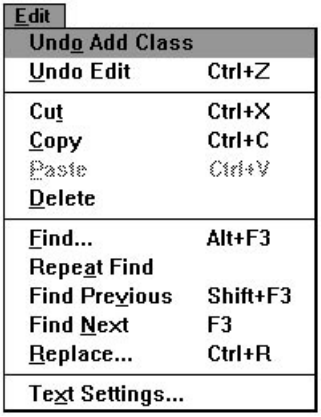

Figure 19-1 Edit menu commands

Global Undo

Cancels the last operation performed in the Class Editor or Hierarchy Editor (such as Add Class or Add Member). The text of this menu item changes to reflect the previous operation performed in the Class Editor.

Goto menu commands

The commands in the **Goto** menu (Figure 19-2) are used to move around within the Source pane and are identical to the corresponding **Goto** menu commands in Source windows. For a description, see Chapter 21, "Text Editor Reference."

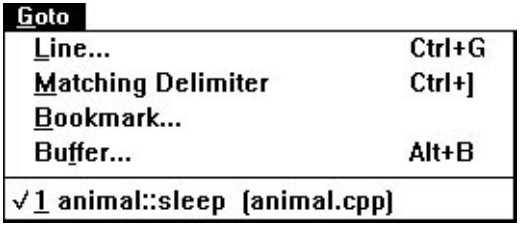

Figure 19-2 Goto menu commands

Macro menu commands

The commands in the **Macro** menu (Figure 19-3) are identical to **Macro** menu commands in Source windows. For a description, see Chapter 21, “Text Editor Reference.”

Figure 19-3 Macro menu commands

New! command

This command opens another Class Editor window.

Classes pane pop-up menu commands

The Classes pane pop-up menu (Figure 19-4) contains commands to add and modify classes and inheritance relationships, and to access Class Editor options.

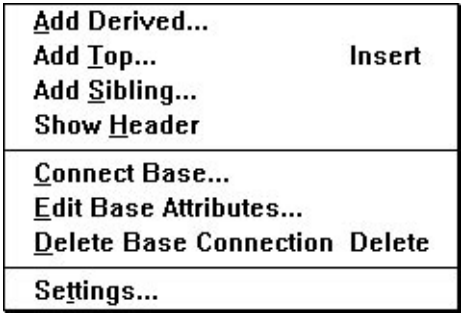

Figure 19-4 Classes pane pop-up menu commands

Add Derived

Opens a dialog box (Figure 19-5) that lets you add a new class to the hierarchy. The new class is a derived class of the currently selected class(es), whose name is shown in the dialog box's title.

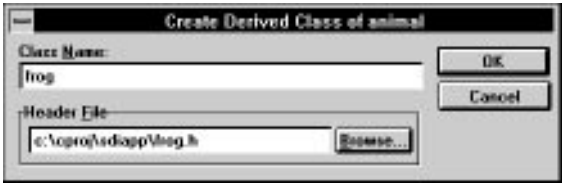

Figure 19-5 Create Derived Class dialog box

Class name
Specifies a name for the new class.

Header file
Contains the name of the header file into which the class declaration is placed. By default, the first eight characters of the class name (with `.h` appended) are used as the header file name. You may type an alternative filename into the textbox, or click on Browse to select a filename from the **Class Header File** dialog box.

Add Top

Adds a top-level (baseless) class. The dialog box that opens as a result of selecting this command is identical in function to the one that opens when you choose the **Add Derived** command.

Add Sibling

Adds a class that is a sibling of (derived from the same base class as) the currently selected class. The dialog box that opens as a result of selecting this command is identical in function to the one that opens when you choose the **Add Derived** command. You cannot add siblings to classes with multiple base classes.

Show Header

Opens a new Source window to edit the header file containing the declaration of the currently selected class.

Note

You can also perform this action by moving the cursor towards the left margin of the Class pane until its orientation changes and it points to the right. Then select a class and drag it onto the desktop while holding down the Control key.

Connect Base

Opens the **Add Base** dialog box (Figure 19-6) that lets you add a base class to the selected class.

Figure 19-6 Add Base dialog box

Access
These options indicate the base class access specifier:

Public: Public members of the base class are public members of the derived class, and protected members of the base class are protected members of the derived class.

Protected: Public and protected members of the base class are protected members of the derived class.

Private: Public and protected members of the base class are private members of the derived class.

Virtual
Specifies the base class to be virtual.

Base class
Specifies the class that is made a base of the selected class. You can select more than one class to make a base class by holding down the Control key while clicking on the class name, or by holding down the Shift key while selecting a range of classes.

Edit Base Attributes

Opens the **Edit Base Attributes** dialog box (Figure 19-7) with which you change the attributes of the connection between the selected class and its base class.

This command is disabled if classes are sorted alphabetically rather than hierarchically.

Figure 19-7 Edit Base Attributes dialog box

Access
These options indicate the base class access specifier:

Public: Public members of the base class are public members of the derived class, and protected members of the base class are protected members of the derived class.

Protected: Public and protected members of the base class are protected members of the derived class.

Private: Public and protected members of the base class are private members of the derived class.

Don't change: Appears only if you are editing the attributes of more than one connection and the base class access specifiers are not all identical. It means that the base class access specifiers are not altered. It lets you change the Virtual specifier of all connections without affecting their individual access specifiers.

Virtual
If selected, this option specifies the base class to be virtual. If grayed, the virtual setting is left unchanged.

If the selected class is derived from multiple bases, only the attributes of the connection with the base class immediately above in the Classes pane are changed.

Delete Base Connection Deletes the connection between the selected class and its base class.

If the class is derived from multiple bases, only the connection with the base class immediately above it in the Classes pane is removed.

You will be prompted to confirm this if the Confirm Inheritance Changes option on the General page of the **Editing/Browsing Settings** dialog box is selected.

This command is disabled if classes are sorted alphabetically rather than hierarchically.

Settings Opens the **Editing/Browsing Settings** dialog box, in which you set general Class Editor and text editor options. Details of this dialog box are discussed in "Class Editor Settings," later in this chapter.

Members pane pop-up menu commands

The Members pane pop-up menu (Figure 19-8) contains commands to add, delete, and modify class members, and to access Class Editor options.

Figure 19-8 Members pane pop-up menu commands

Add

Opens the **Add Member** dialog box (Figure 19-9). The new member is added to the class whose members are currently displayed in the Members pane.

Figure 19-9 Add Member dialog box

Access
These options specify the member access control:

Public: Public members are accessible from outside the member's class.

Protected: Protected members are accessible only within the member's class, derived classes, and their friends.

Private: Private members are accessible only within the member's class and its friends.

Storage
These options specify the member storage class:

Normal: The member has no storage modifiers.

Static: Only one copy of the member exists; it is shared by all objects of the member's class.

Virtual (functions only): The function *may be* overridden in derived classes.

Pure virtual (functions only): The function *must be* overridden in derived classes; the class is abstract.

Friend (functions only): The named function is allowed access to the class's private and protected members.

Inline
Requests inline implementation of a function. Causes its definition to be placed in the `.h` file.

Declaration
Contains the member declaration. For data items, enter the type and member name (for example, `int nCats`). For functions, enter the return type, function name, and argument types (for example, `void AddCats(int)`). Do not precede the declaration with storage specifiers; use the option buttons above. Trailing semicolons are optional.

Source file
Contains the name of the source file into which the member definition is placed. By default, the first eight characters of the class name (with `.cpp` appended) are used as the source file name. You may type an alternative filename into the textbox, or click on Browse to select an alternative filename from the **Member Source File** dialog box.

The member declaration is placed in the class declaration. If the specified source file does not already exist, it is created and added to the project. By default, empty definitions of inline functions are placed in the header file. Empty definitions of normal, static, and virtual functions, as well as static data members, are placed in the source file.

Delete

Deletes the currently selected member. If the Confirm Member Delete option on the General page of the **Editing/Browsing Settings** dialog box is selected, you are asked to confirm the deletion. The member's declaration and definition (if applicable) are removed from the header and source files.

Edit Attributes

Opens the **Change Member Attributes** dialog box (Figure 19-10), which you use to edit access and storage specifiers of the selected member.

Figure 19-10 Change Member Attributes dialog box

Access
These options specify the member access control:

Public: Public members are accessible from outside the member's class.

Protected: Protected members are accessible only within the member's class, derived classes, and their friends.

Private: Private members are accessible only within the member's class and its friends.

Don't change: Only appears if you are editing the attributes of more than one member and their access controls are not all identical. It means that the access control is not altered. It lets you change other member attributes without also affecting their individual access controls.

Storage
These options specify the member storage class:

Normal: The member has no storage modifiers.

Static: Only one copy of the member exists; it is shared by all objects of the member's class.

Virtual (functions only): The function *may be* overridden in derived classes.

Pure virtual (functions only): The function *must be* overridden; the class is abstract.

Friend (functions only): The named function is allowed access to the class's protected and private members.

Don't change: Only appears if you are editing the attributes of more than one member and their storage classes are not all identical. It means that the storage class is not altered. It lets you change other member attributes without affecting their individual storage classes.

Inline
Requests inline implementation of a function. If grayed, the current setting is left unchanged.

Source file
If applicable, this option contains the name of the source file holding the member definition. You may change the filename in the textbox, or click on Browse to select an alternative filename from the **Member Source File** dialog box.

Show Source

Opens a new Source window to show the source file containing the definition of the currently selected member.

If you add new functions or static data definitions to the source file or alter function argument or return types, you must update the class header to reflect the changes.

Settings

Opens the **Editing/Browsing Settings** dialog box that lets you set general Class Editor and text editor options. Details of this dialog box are discussed in "Class Editor Settings," later in this chapter.

Source pane pop-up menu commands

With one exception, commands in the Source pane pop-up menu (Figure 19-11) are identical to the corresponding commands in Source windows. For a description, see Chapter 21, "Text Editor Reference."

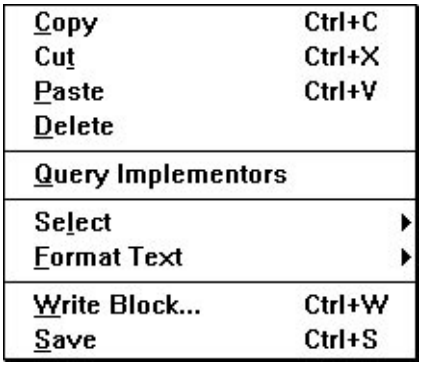

Figure 19-11 Source pane pop-up menu commands

Save

Saves the program code shown in the Source pane and places it in the appropriate source or header file.

Changes to static data and to function argument and return types made in the source code are updated automatically in the class declaration and Members pane.

Class editor mouse functions

The mouse is used to select classes and members, perform editing operations, open pop-up menus, and change the relative sizes of Class Editor panes. As mentioned in "The Class Editor Window" above, drag and drop operations are supported.

To resize the panes, first position the cursor over the dividing line between panes. The cursor changes to a two-headed arrow. Then, press the left mouse button and drag the separator to the desired location.

Classes pane

The right mouse button opens the pop-up menu (see "Classes pane pop-up menu commands," earlier in this chapter).

Select a class by clicking on it. Several classes may be selected by clicking on each one while holding down Control. The members of

the class last selected appear in the Members pane. You may drag and drop a class into the source pane; the class name is inserted into the buffer.

Members pane
The right mouse button opens the pop-up menu (see "Members pane pop-up menu commands," earlier in this chapter).

Select a member by clicking on it. Several members may be selected by clicking on each one while holding down Control.

Double-clicking on a member causes its definition (if appropriate) or declaration to appear in the Source pane. If the source is not available, the declaration is displayed.

You may drag and drop a member into the source pane; the member declaration is inserted into the buffer.

Note

In addition to clicking on it, you can select a member when the Members pane is active by typing its name. As you type, the Class Editor attempts to automatically complete the name. (Depending on the rate at which you type, it may consider a character to be the first character of a new selection, rather than the next character in the current select operation.)

Source pane
The right mouse button opens the pop-up menu (see "Source pane pop-up menu commands," earlier in this chapter).

The source pane supports typical Source window mouse and cursor operations, as described in Chapter 21, "Text Editor Reference."

Toolbar commands

The Class Editor toolbar (Figure 19-12) offers quick access to several menu options.

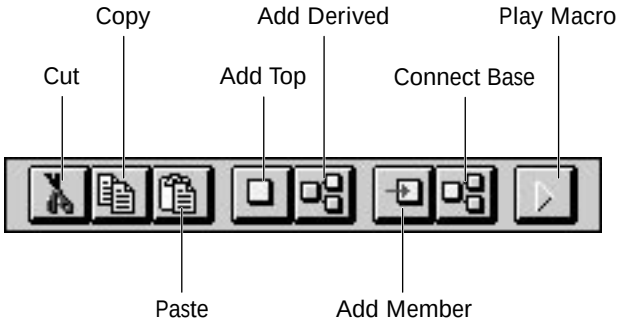

Figure 19-12 Class Editor toolbar

These options are the same as those in other menus:

Cut: Same as choosing **Cut** from the **Edit** menu.

Copy: Same as choosing **Copy** from the **Edit** menu.

Paste: Same as choosing **Paste** from the **Edit** menu.

Add top: Same as choosing **Add Top** from the Classes pane pop-up menu.

Add derived: Same as choosing **Add Derived** from the Classes pane pop-up menu.

Add member: Same as choosing **Add** from the Member pane pop-up menu.

Connect base: Same as choosing **Connect Base** from the Classes pane pop-up menu.

Play macro: Same as choosing **Play** from the **Macro** menu.

Class Editor Settings

You can access Class Editor settings by choosing **Text Settings** from the **Edit** menu, or **Settings** from the Classes or Members pane pop-up menus. These commands open the **Editing/Browsing Settings** dialog box, a workspace with tabs along the top margin. The tabs are used to switch between several sets of options. The Class, Member, and General options are discussed in this section; the remaining options pertain to text editing and are discussed in Chapter 21, “Text Editor Reference.”

General options

The General options set (Figure 19-13) contains options for undo levels, class/member bar confirmations, output window actions, and key bindings, as well as some options related to the text editor.

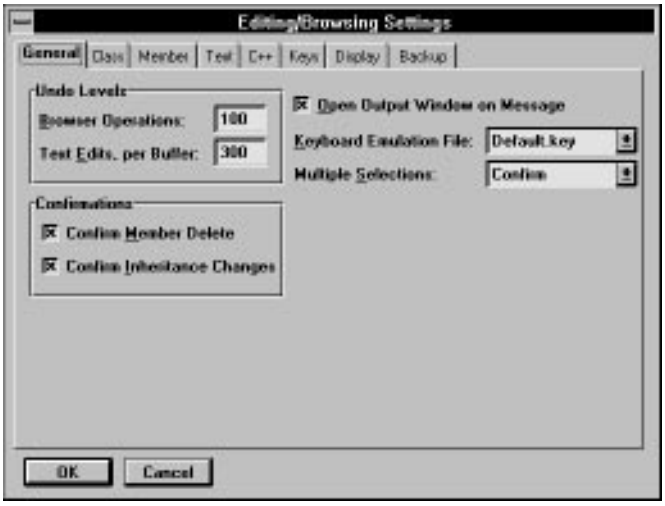

Figure 19-13 General options

Browser operations

Specifies the number of operations that can be undone in the Class and Hierarchy Editors with the Global Undo command.

Text edits, per buffer

Specifies the number of edit operations that can be undone, per buffer.

Confirmations

These options enable confirmation requests for various operations in the Class and Hierarchy Editors. You can enable confirmations of:

- Member deletions
- Inheritance changes

Open output window on message

Lets the IDDE open an error window whenever there is an error of any kind (during compilation, during parsing, and so on.)

Keyboard emulation file

Specifies the key bindings set to be used by the text editor.

Multiple selections
This option enables multiple selections in lists in the Class and Hierarchy editors:

Yes: Multiple selections are allowed.

No: Multiple selections are not allowed.

Confirm: Multiple selections are allowed, but a confirmation request is displayed each time an operation is performed on multiple classes or members.

Class options

The Class options set (Figure 19-14) lets you specify the display order and font of classes in the Classes pane.

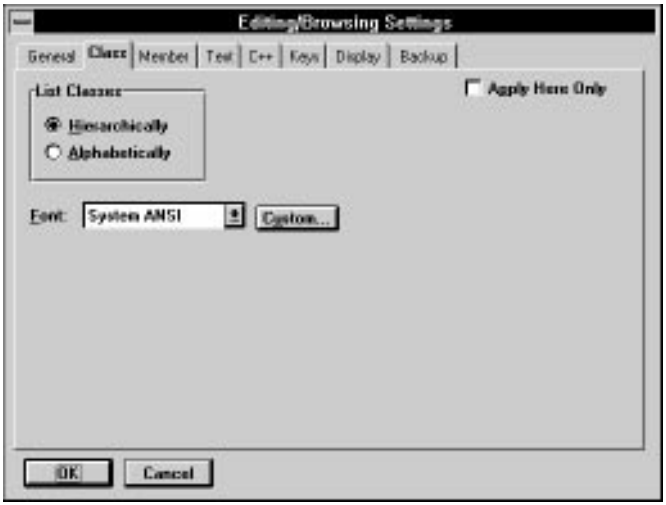

Figure 19-14 Class options

List classes
This option defines how classes are arranged in the Classes pane:

Hierarchically: Base classes are arranged alphabetically, with derived classes placed below and indented relative to their base classes. If a class has multiple bases, that class is listed below each base class.

Alphabetically: Classes are arranged in alphabetical order, and each class is listed only once.

Font
Specifies the font used to display class names in the Classes pane. You can select a predefined font from the drop-down list, or you may click on Custom and select any installed font from a Windows **Font** dialog box.

Apply here only
Indicates that the settings specified here should be applied only to the current Class Editor window.

Member options

The Member options set (Figure 19-15) lets you specify the display parameters of class members in the Members pane.

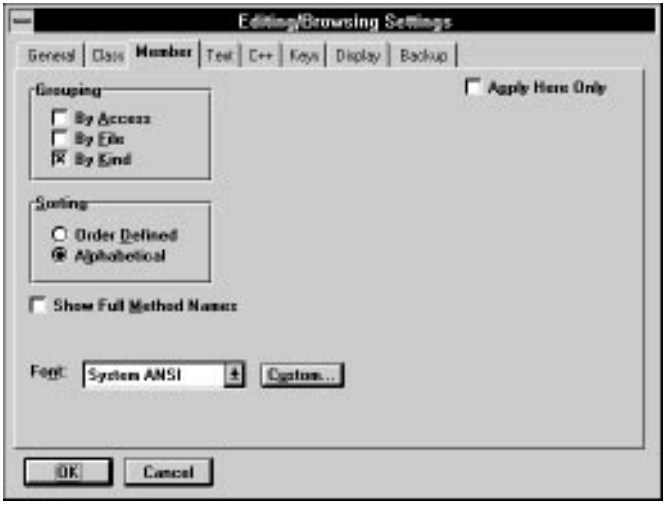

Figure 19-15 Member options

Grouping
This option indicates how class members are grouped. Any or all of the following options may be selected:

By access: Members are grouped into Public, Protected, and Private. If By Access is not selected, each member is preceded by a colored diamond indicating its access specifier (green for public, yellow for protected, red for private).

By file: Members are grouped by the source file containing their definitions. When this option is selected, only functions and static data are shown.

By kind: Members are grouped into Data, Functions, and Typedefs.

If more than one Grouping option is selected, members are grouped by kind within files, and by files within access category.

Sorting
This option indicates how members are arranged within each group:

Order defined: Members are arranged in the order they are declared in the class header.

Alphabetical: Members are arranged alphabetically.

Show full method names
Specifies that member names are displayed with their type specifiers and (if a method) their parameter type list (for example, `int Multiply(int, int, int)`). If this option is not selected, only the identifier name is displayed (for example, `Multiply`).

Font
Specifies the font used to display member names in the Members pane. You can select a predefined font from the combo box, or you may click on Custom and select any installed font from a Windows **Font** dialog box.

Apply here only
Indicates that the settings specified here should be applied only to the current Class Editor window. Otherwise, the settings are applied to all Hierarchy and Class Editor windows.

Hierarchy Editor Reference 20

This chapter describes menu operations and mouse functions available in the Hierarchy Editor. For an introduction to this tool, see Chapter 5, “Defining Classes and Their Hierarchies.”

The Hierarchy Editor Window

To open a Hierarchy Editor window you have several choices:

- Choose **Hierarchy Editor** from the **Goto View** submenu of the IDDE's **Window** menu.
- Double-click on the Hierarchy Editor icon in the Views toolbox, or drag the icon from the toolbox to the desktop.
- Choose **New!** from another Hierarchy Editor window's menu bar.

Multiple instances of the Hierarchy Editor may be open at the same time. These windows share data among themselves and with Class Editor windows, so changes made in one window are transmitted instantly to the others.

Edit menu commands

Commands in the **Edit** menu (Figure 20-1) let you undo Hierarchy Editor operations and print the graphical display.

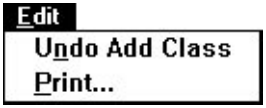

Figure 20-1 Edit menu commands

Global Undo Cancels the last operation performed in the Hierarchy Editor (such as Add Class or Add Member). The text of this menu item changes to reflect the previous operation performed in the Hierarchy Editor.

Print Opens the **Print** dialog box. See Chapter 21, "Text Editor Reference."

Macro menu commands

The commands in the **Macro** menu (Figure 20-2) are identical to **Macro** menu commands in Source windows. For a description, see Chapter 21, "Text Editor Reference."

Figure 20-2 Macro menu commands

New! command

Opens another Hierarchy Editor window.

Pop-up menu commands

Contains commands to add and modify classes and inheritance relationships, and to access Hierarchy Editor options.

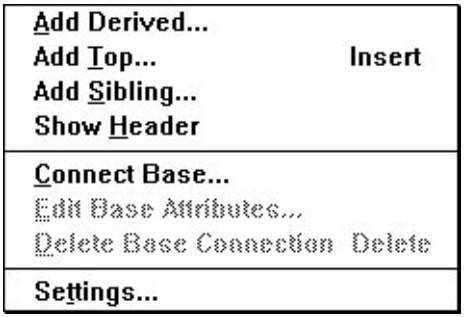

Figure 20-3 Hierarchy Editor pop-up menu commands

Add Derived

Opens a dialog box (Figure 20-4) that lets you add a new class to the hierarchy. The new class is a derived class of the currently selected class, whose name is shown in the dialog box's title.

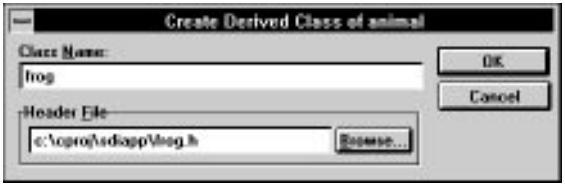

Figure 20-4 Create Derived Class dialog box

Class name
Specifies a name for the new class.

Header file
Contains the name of the header file into which the class declaration is placed. By default, the first eight characters of the class name (with `.h` appended) are used as the header file name. You may type an alternative filename into the textbox, or click on Browse to select a filename from the **Class Header File** dialog box.

Add Top

Adds a top-level (baseless) class. The dialog box that opens as a result of selecting this command is identical in function to the one that opens when you choose the **Add Derived** command.

Add Sibling

Adds a class that is a sibling of (derived from the same base classes as) the currently selected class. The dialog box that opens as a result of selecting this command is identical in function to the one that opens when you choose the **Add Derived** command.

Show Header

Opens a new Source window to edit the header file containing the declaration of the currently selected class.

Connect Base

Opens the **Add Base** dialog box (Figure 20-5), with which you can add a base class to the selected class.

Figure 20-5 Add Base dialog box

Access
These options indicate the base class access specifier:

Public: Public members of the base class are public members of the derived class, and protected members of the base class are protected members of the derived class.

Protected: Public and protected members of the base class are protected members of the derived class.

Private: Public and protected members of the base class are private members of the derived class.

Virtual
Specifies the base class to be virtual.

Base class
Specifies the class that is made a base of the selected class. You can select more than one class to make a base class by holding down the Control key while clicking on the class name.

Edit Base Attributes

Opens the **Edit Base Attributes** dialog box (Figure 20-6), with which you can change the attributes of the selected connection between a derived class and its base class.

Figure 20-6 Edit Base Attributes dialog box

Access
These options indicate the base class access specifier:

Public: Public members of the base class are public members of the derived class, and protected members of the base class are protected members of the derived class.

Protected: Public and protected members of the base class are protected members of the derived class.

Private: Public and protected members of the base class are private members of the derived class.

Don't change: Appears only if you are editing the attributes of more than one connection and the base class access specifiers are not all identical. It means that the base class access specifiers are not altered. It lets you change the Virtual specifier of all connections without affecting their individual access specifiers.

Virtual
If selected, this option specifies the base class to be virtual. If grayed, the virtual setting is left unchanged.

Delete Base Connection

Deletes the selected connection between a derived class and its base class.

You are prompted to confirm this if the Confirm Inheritance Changes option on the General page of the **Editing/Browsing Settings** dialog box is selected.

Settings

Opens the **Editing/Browsing Settings** dialog box, in which you set global Hierarchy Editor and text editor options. Details of this dialog box are discussed in "Hierarchy Editor Settings," later in this chapter.

Mouse functions

The right mouse button opens the pop-up menu.

A class is selected by clicking on it. Additional classes may be selected by clicking while holding down Control. Members of the class last selected appear in the Members child window.

Double-clicking on a class opens a Class Editor window.

A connection is selected by clicking on it. Additional connections may be selected by clicking while holding down Control.

A new derived class is added by clicking on the base class and dragging the cursor to an empty area of the display.

A new base-to-derived connection is made by clicking on the base class and dragging the cursor to the derived class.

A derived class's connection to a base may be moved to a different base by clicking the connection, then dragging the connection's handle to the new base class. (This is a shortcut for deleting one connection, then establishing a new one.)

Toolbar commands

The Hierarchy Editor toolbar (Figure 20-7) offers quick access to several menu choices.

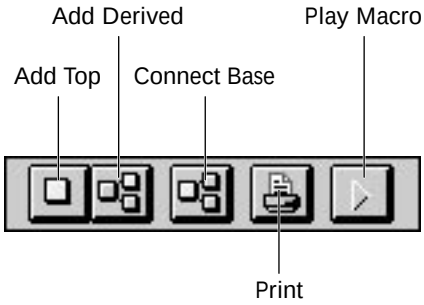

Figure 20-7 Hierarchy Editor toolbar

The following options are the same as those on other menus:

Add top: Same as choosing **Add Top** from the pop-up menu.

Add derived: Same as choosing **Add Derived** from the pop-up menu.

Connect base: Same as choosing **Connect Base** from the pop-up menu.

Print: Same as choosing **Print** from the **Edit** menu.

Play macro: Same as choosing **Play** from the **Macro** menu.

Members Child Window

The Hierarchy Editor's Members child window is enabled by setting the Members option on the Hierarchy page of the **Editing/ Browsing Settings** dialog box. You can open this dialog box by choosing **Settings** from the Hierarchy Editor's pop-up menu.

Member menu commands

The Member menu (Figure 20-8) contains commands to add, delete, and modify class members, and to access Hierarchy Editor options.

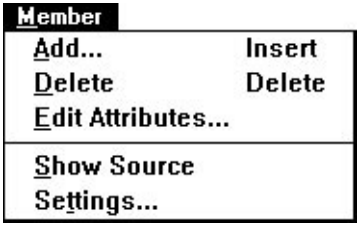

Figure 20-8 Member menu commands

Add

Opens the **Add Member** dialog box (Figure 20-9). The new member is added to the class whose members currently are displayed in the Members child window.

Figure 20-9 Add Member dialog box

Access
These options specify the member access control:

Public: Members are accessible from outside the member's class.

Protected: Members are accessible only within the member's class, derived classes, and their friends.

Private: Members are accessible only within the member's class and its friends.

Storage
These options specify the member storage class:

Normal: The member has no storage modifiers.

Static: Only one copy of the member exists; it is shared by all objects of the member's class.

Virtual (functions only): The function *may be* overridden in derived classes.

Pure virtual (functions only): The function *must be* overridden in derived classes; the class is abstract.

Friend (functions only): The named function is allowed access to the class's private members.

Inline
Requests inline implementation of a function.

Declaration
Contains the member declaration. For data items, enter the type and member name (for example, `int nCats`). For functions, enter the return type, function name, and argument types (for example, `void AddCats(int)`). Do not precede the declaration with storage specifiers; use the option buttons above. Trailing semicolons are optional.

Source file
Contains the name of the source file into which the member definition is placed. By default, the first eight characters of the class name (with `.cpp` appended) are used as the source filename. You can type an alternative filename into the textbox, or click on Browse to select an alternative filename from the **Member Source File** dialog box.

The member declaration is placed in the class declaration. If the specified source file does not already exist, it is created and added to the project. By default, empty definitions of inline functions are placed in the header file. Empty definitions of normal, static, and virtual functions, as well as static data members, are placed in the source file.

Delete

Deletes the currently selected member. If the Confirm Member Delete option on the General page of the **Editing/Browsing Settings** dialog box is selected, you are asked to confirm the deletion. The member's declaration and definition (if applicable) are removed from the header and source files.

Edit Attributes

This command opens the **Change Member Attributes** dialog box (Figure 20-10), which you can use to edit access and storage specifiers of the selected member.

Figure 20-10 Change Member Attributes dialog box

Access
These options specify the member access control:

Public: Members are accessible from outside the member's class.

Protected: Members are accessible only within the member's class, derived classes, and their friends.

Private: Members are accessible only within the member's class and its friends.

Don't change: Appears only if you are editing the attributes of more than one member and their access controls are not all identical. It means that the access control is not altered. It lets you change other member attributes without also affecting their individual access controls.

Storage
These options specify the member storage class:

Normal: The member has no storage modifiers.

Static: Only one copy of the member exists; it is shared by all objects of the member's class.

Virtual (functions only): The function may be overridden in derived classes.

Pure virtual (functions only): The function must be overridden; the class is abstract.

Friend (functions only): The named function is allowed access to the class's protected and private members.

Don't change: Appears only if you are editing the attributes of more than one member and their storage classes are not all identical. It means that the storage class is not altered. It lets you change other member attributes without affecting their individual storage classes.

Inline
Requests inline implementation of a function. If grayed, the current setting is left unchanged.

Source file
If applicable, this option contains the name of the source file holding the member definition. You may change the filename in the textbox, or click on Browse to select an alternative filename from the **Member Source File** dialog box. If you change the source file, the member definition moves.

Show Source

Opens a new Source window to show the source file containing the definition of the currently selected member.

If you add new functions or static data definitions to the source file, or alter function argument or return types, you must update the class header to reflect the changes.

Settings

Opens the **Editing/Browsing Settings** dialog box, in which you set global Hierarchy Editor and text editor options. Details of this dialog box are discussed in "Hierarchy Editor Settings," later in this chapter.

Pop-up menu commands

The Members child window pop-up menu is identical to the **Member** menu.

Mouse functions

The right mouse button opens the pop-up menu.

Select a member by clicking on it. Additional members may be selected by clicking on them while holding down Control.

Double-clicking on a member causes its definition (if appropriate) or declaration to be displayed in the Source child window.

You may drag and drop a member into the Source child window; the member declaration is inserted into the buffer.

Source Child Window

The Hierarchy Editor's Source child window is enabled by setting the Source option on the Hierarchy page of the **Editing/Browsing Settings** dialog box. You can open this dialog box by choosing **Settings** from the Hierarchy Editor's pop-up menu.

Edit menu commands

Commands in the **Edit** menu (Figure 20-11) are used to perform text editing operations, and are identical to the corresponding **Edit** menu commands in Source windows. For a description, see Chapter 21, "Text Editor Reference."

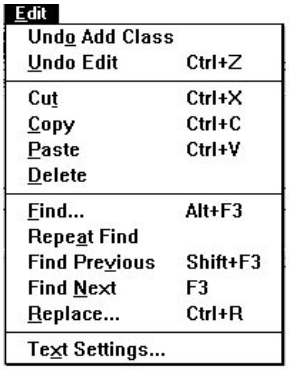

Figure 20-11 Edit menu commands

Goto menu commands

The commands in the **Goto** menu (Figure 20-12) are used to move around within the Source child window and are identical to the corresponding **Goto** menu commands in Source windows. For a description, see Chapter 21, "Text Editor Reference."

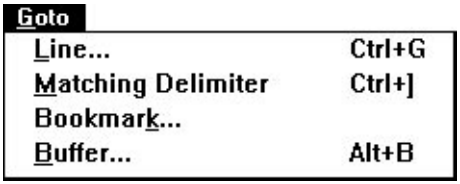

Figure 20-12 Goto menu commands

Pop-up menu commands

With one exception, commands in the Source child window pop-up menu (Figure 20-13) are identical to the corresponding commands in Source windows. For a description, see Chapter 21, “Text Editor Reference.”

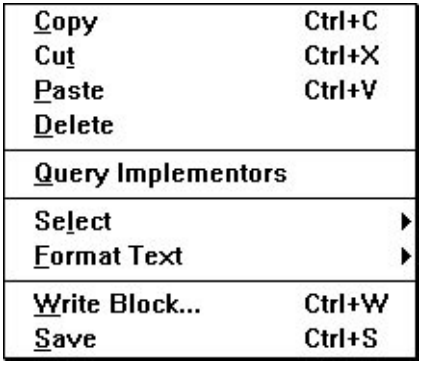

Figure 20-13 Pop-up menu commands

Save

Saves the program code shown in the Source child window and places it in the appropriate source or header file.

Changes to static data and to function argument and return types made in the source code are updated automatically in the class declaration and Members child window.

Mouse functions

The right mouse button opens the pop-up menu.

The Source child window supports typical source window mouse and cursor operations, as described in Chapter 21, “Text Editor Reference.”

Hierarchy Editor Settings

You can access Hierarchy Editor settings by choosing **Settings** from the Hierarchy Editor pop-up menu, **Settings** from the Members child window **Member** or pop-up menus, or **Text Settings** from the Source child window **Edit** menu. These commands open the **Editing/Browsing Settings** dialog box, a workspace with tabs along the top margin.

The tabs are used to switch between several sets of options. The Hierarchy, Member, and General options are discussed in this section; the remaining options pertain to text editing and are discussed in Chapter 21, “Text Editor Reference.”

General options

The General options set (Figure 20-14) contains options for undo levels and confirmations, as well as options related to the text editor.

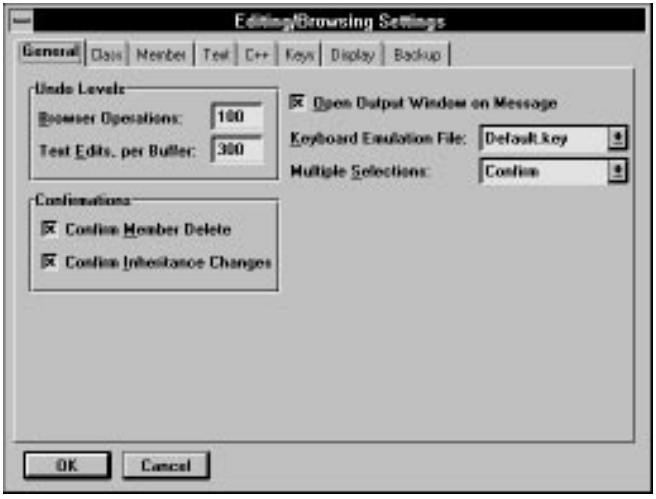

Figure 20-14 General options

Browser operations
Specifies the number of operations that can be undone in the Class and Hierarchy Editors.

Text edits, per buffer
Specifies the number of edit operations that can be undone per buffer.

Confirmations
These options enable confirmation requests for various operations in the Class and Hierarchy Editors. You can enable confirmations of:

- Member deletions
- Inheritance changes

Open output window on message
Lets the IDDE open an error window whenever there is an error of any kind (during compilation, during parsing, and so on.)

Keyboard emulation file
Specifies the key bindings set to be used by the text editor.

Multiple selections
Enables multiple selections in lists in the Class and Hierarchy editors.

Yes: Multiple selections are allowed.

No: Multiple selections are not allowed.

Confirm: Multiple selections are allowed, but a confirmation request is displayed each time an operation is performed on multiple classes or members.

Hierarchy options

The Hierarchy options set (Figure 20-15) lets you enable the Members and Source child windows and specify the font used in the graphical display.

Figure 20-15 Hierarchy options

Pop-up windows
These options enable the Hierarchy Editor child windows:

Members: Enables the Members child window.

Source: Enables the Source child window.

Font
Specifies the font used to display class names. You can select a predefined font from the drop-down list, or you can click on Custom and select any installed font from a Windows **Font** dialog box.

Apply here only
Indicates that the specified settings should be applied only to the current Hierarchy Editor window.

Member options

Lets you specify the display parameters of class members in the Members child window.

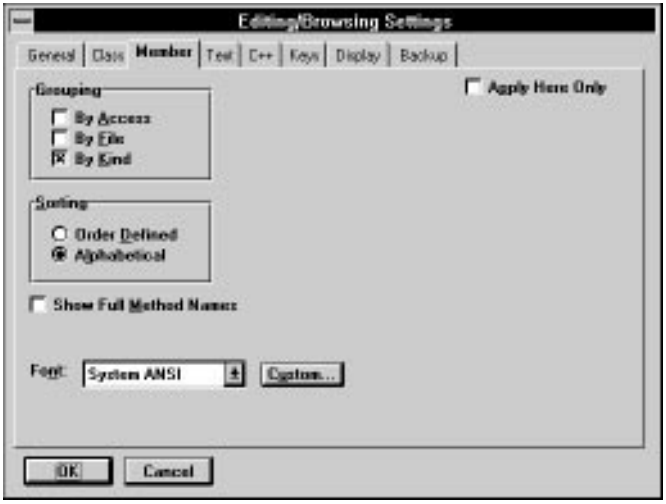

Figure 20-16 Member options

Grouping
Indicates how class members are grouped. Any or all of the following options may be checked:

By access: Members are grouped into public, protected, and private. If By Access is not selected, each member is preceded by a colored diamond indicating its access specifier (green for public, yellow for protected, red for private).

By file: Members are grouped by the source file containing their definitions. When this option is selected, only functions and static data are shown.

By kind: Members are grouped into Data, Functions, and Typedefs.

If more than one Grouping option is selected, members are grouped by kind within files and by files within access category.

Sorting
Indicates how members are arranged within each group.

Order defined: Members are arranged in the order in which they are declared in the class header.

Alphabetical: Members are arranged alphabetically.

Show full method names
Specifies that member names are displayed with their type specifiers and (if a method) their parameter type list (for example, `int Multiply(int, int, int)`). If this option is not selected, only the identifier name is displayed (for example, `Multiply`).

Font
Specifies the font used to display member names in the Members child window. You can select a predefined font from the combobox, or you can click on Custom and select any installed font from a Windows **Font** dialog box.

Apply here only
Indicates that the specified settings should be applied only to the current Hierarchy Editor window. Otherwise, the settings are applied to all Hierarchy and Class Editor windows.

Text Editor Reference 21

This chapter describes commands and options available in the text editor, including global search functions, key binding options, and macro functions. For an introduction to text editing, see Chapter 6, "Editing Program Code."

The Source Window

To open a Source window, you have several choices:

- Choose **New** or **Open** from the IDDE's **File** menu.
- Choose **Source** from the **Goto View** submenu of the IDDE's **Window** menu.
- Double-click on the Source icon in the Views toolbox, or drag the icon from the toolbox to the desktop.
- Double-click on a filename in the Project window.
- Choose **Show Source** or **Show Header** from pop-up menus in the Class and Hierarchy Editors.
- Double-click on an error message in the Error window.
- Choose **New** or **Open** from another Source window's **File** menu.
- Choose **New!** from another Source window's menu bar.
- Select text in one Source view, then drag it onto the desktop. This results in an untitled Source window containing the selected text.

The Hierarchy Editor's Source child window, as well as the Source pane of the Class Editor, contain a subset of the standard Source window functionality. See Chapter 5, “Defining Classes and Their Hierarchies,” Chapter 19, “Class Editor Reference,” and Chapter 20, “Hierarchy Editor Reference.”

File menu commands

The **File** menu (Figure 21-1) contains commands to open, save, and print files, as well as other useful file-related commands. Note that the **Save and Add to Project** command changes to **Add to Project**, **Save and Parse**, or **Parse**, depending on whether the file is part of the project, up to date, or parsed.

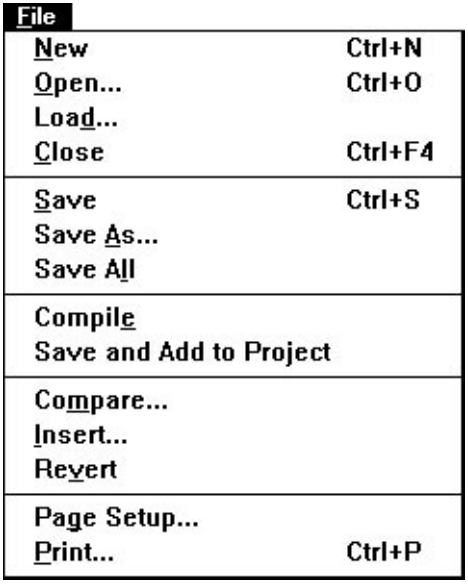

Figure 21-1 File menu commands

New Opens a new, empty, and untitled Source window.

Open Opens a Windows **File Open** dialog box, then creates a new Source window containing the selected file. If the **Open** command is chosen from an untitled, unmodified Source window, the selected file is opened in that Source window.

Load Opens a Windows **File Open** dialog box, then loads the selected file into the current Source window. If there are unsaved changes in the previous file, you are asked if you would like to save the changes before loading the new file.

Close Closes the Source window. If there are unsaved changes, you are asked if you would like to save the changes before closing the file.

Save Saves the current buffer to disk. If the file is untitled, this command executes **Save As**.

Save As Saves the current buffer using the Windows **File Save As** dialog box. This dialog box contains a check box—Add to Project, which lets you add the file to the current project.

Save All Executes **Save** for every open Source window.

Compile Saves the buffer and compiles the file.

Save and Add to Project Saves the file to disk, adds the file to the project, and reparses all files in the project.

The Save and Add to Project command changes to Add to Project, Save and Parse, or Parse, depending on whether the file is part of the project, up to date, or parsed. These are the four changes:

1. A new source file is created but not saved, or flagged as modified since the last save. The menu reads “Save and Add to Project.”
2. A source file which has not been added to the project is saved. The menu reads “Add to Project.”
3. A source file which exists in a project is loaded in the source editor. The menu reads "Parse."
4. A source file exists in a project and has been flagged or modified. The menu read “Save and Parse.”

Add to Project Adds the file to the project and reparses all files in the project.

Save and Parse Saves the file to disk and reparses all files in the project.

Parse Reparses all files in the project.

Compare

Opens the **Compare Files** dialog box (Figure 21-2), which lets you select two files to compare.

Figure 21-2 Compare Files dialog box

File 1 and File 2
Specifies the files to be compared. You may type the file names, select names from the drop-down lists, or click on Browse to select files from standard Windows filename dialog boxes.

If either file is open in a Source window, the editor uses the version in memory rather than the one in the disk file.

Line
Specifies the line number at which the comparison starts.

Display
Specifies the arrangement of the windows in which the two files are displayed:

Horizontal: The files are displayed one above the other.

Vertical: The files are displayed side-by-side.

The editor performs the comparison on a line-by-line basis. When it finds a mismatch, it highlights the appropriate lines in both files. The **Compare** dialog box (Figure 21-3) indicates where the mismatch was found.

Figure 21-3 Compare dialog box showing mismatch

Next match: Click on this button to resynchronize the comparison. The editor highlights the next set of matching lines and the **Compare** dialog box indicates where the match occurs.

Figure 21-4 Compare dialog box showing match

Next difference: Click on this button to find the next mismatched line. You can continue the comparison in this way until no more differences are found.

Insert

Opens an **Insert File** dialog box. The editor inserts the contents of the file you select at the current insertion point.

Revert

Rereads the file from disk, abandoning any changes made since the last time it was saved.

Page Setup

Opens the **Page Setup** dialog box (Figure 21-5), which sets parameters for printing.

Figure 21-5 Page Setup dialog box

Header and footer

The header and footer are single lines of text displayed at the top and bottom (respectively) of each page of the printed output. To omit the header or footer, leave the textbox empty.

You can embed the following special-purpose codes in the header and footer text:

- %f gives the full path and filename of active file
- %d gives the current date and time
- %p gives the current page number

Margins
Sets the page margins. The units are the standard units of measurement used in your country (inches or centimeters), set in the Windows Control Panel.

Font
Opens a Windows **Font** dialog box, with which you select a typeface, style, and size for the printed output. The entire document, including headers and footers, is shown in the selected font.

Printer
Opens a Windows **Print Setup** dialog box, with which you can set additional printing parameters.

Print

Opens the **Text Print** dialog box (Figure 21-6), to let you set additional print options and print the current file or text selection. After setting print options, click OK to print.

Figure 21-6 Text Print dialog box

Print range
Specifies whether the entire file or just the current text selection is printed:

All: Prints the entire file.

Selection: Prints the highlighted text selection. (It is disabled if no text is selected.)

Print quality
This drop-down listbox specifies print quality. Options include the specified printer's output capabilities, expressed in dots per inch.

Copies
Specifies the number of copies to print.

Setup
Opens a Windows **Print Setup** dialog box, with which you can set additional printing parameters.

Edit menu commands

The **Edit** menu (Figure 21-7) contains standard edit and search commands, as well as commands to access text editor options.

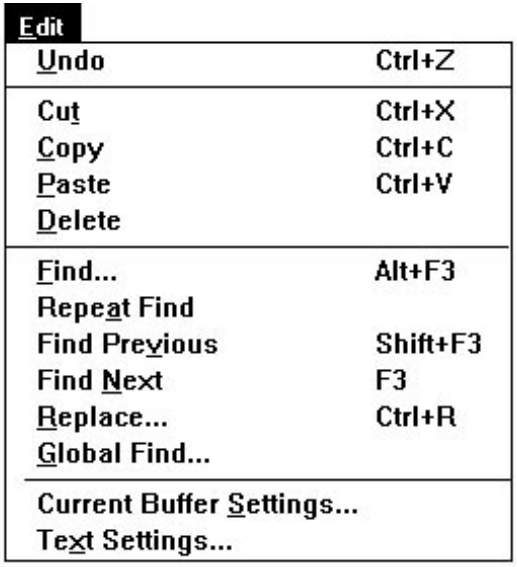

Figure 21-7 Edit menu commands

Undo Reverses the last cut, paste, replace, or typed character. By repeatedly choosing this command, you can undo previous commands, up to the limit of the undo buffer.

Cut Copies the selected text to the Clipboard, then deletes it from the buffer.

Copy Copies the selected text to the Clipboard.

Paste Inserts the text from the Clipboard at the insertion point.

Delete Deletes the selected text from the buffer.

Find Opens the **Find** dialog box (Figure 21-8), used to search the file for specified text.

Figure 21-8 Find dialog box

Pattern
This drop-down listbox contains the text to be found.

You can initialize the pattern by selecting the text before choosing **Find**. (The text must not span a line break.) Otherwise, you may type the text you want to find into the textbox, or select text from previous search strings, stored in the drop-down listbox.

The pattern may also be a regular expression (text containing wildcard characters) if the Regular Expression option is selected. Regular expression syntax is discussed in "Using Global Find," later in this chapter.

Ignore case
If this option is turned on, the search is not case sensitive.

Whole words only
If this option is turned on, a string is considered a match only if it is not part of a larger alphanumeric string.

Regular expression
This option enables regular expression matching.

The search begins at the current insertion point. Click Next to search forward in the file, or Previous to search backward in the file. If the search is successful, the matching text is highlighted. Otherwise, the status line displays the message "Pattern not found."

Repeat Find

Continues the search begun by **Find**. The search resumes from the current insertion point and proceeds in the direction previously specified.

If the search is successful, the matching text is highlighted. Otherwise, the status line displays the message “Pattern not found.”

Find Previous

Searches backward from the current insertion point for the text to be found.

Find Next

Searches forward from the current insertion point for the text to be found.

Replace

Opens the **Replace** dialog box (Figure 21-9), which lets you find and replace occurrences of text with different text.

Figure 21-9 Replace dialog box

Pattern
This drop-down listbox contains the text to be found and replaced.

You can initialize the pattern by selecting text before choosing **Replace**. (The text must not span a line break.) Otherwise, you may type the text you want to find into the textbox, or select text from previous search strings, stored in the drop-down listbox.

The pattern may also be a regular expression (text containing wildcard characters) if the Regular Expressions option is selected. Regular expression syntax is discussed in the section “Using Global Find,” later in this chapter.

Replacement
This drop-down listbox contains the text with which you replace occurrences of Pattern. Type the replacement text into the textbox, or select text from previous replacement strings, stored in the drop-down listbox.

Ignore case
If this option is turned on, the search is not case sensitive.

Regular expressions
Enables regular expression matching.

Confirm changes
Causes the text editor to request confirmation before each replacement. If this option is not selected, the editor replaces all occurrences of Pattern, starting at the current insertion point, without prompting you.

Restrict changes to selected text
Instructs the text editor to perform replacements only within the selected block of text.

Whole words only
If selected, the text editor considers text a match only if it is not part of a larger alphanumeric string.

The search/replace operation begins at the current insertion point. If you have selected Confirm Changes, the **Confirm Replacement** dialog box (Figure 21-10) is displayed when a match is found.

Figure 21-10 Confirm Replacement dialog box

You may click Yes to make the replacement, No to skip this replacement, or Cancel to end the search/replace operation. Also, if you uncheck Confirm, then click Yes, the editor replaces all remaining occurrences of Pattern without prompting you.

Global Find

Opens the **Global Find** dialog box. See “Using Global Find,” later in this chapter.

Current Buffer Settings

Opens the **Current Buffer Options** dialog box (Figure 21-11), with which you can change editing options for the current Source window.

Figure 21-11 Current Buffer Options dialog box

Tab spacing
Specifies the number of columns between tab stops.

Right margin
Specifies the column that acts as the right margin.

Word wrap
Enables word wrap. While typing, lines extending beyond the right margin are broken automatically at the last word boundary before the margin.

Autoindent
Enables automatic indentation on newline. When you press Enter, the editor positions the cursor directly below the first nonblank character in the previous line.

Read only
Sets the read-only flag on the buffer, so the buffer may not be changed.

Use as default for ...
Saves the Current Buffer Options settings so they become the defaults for any subsequent file with the same file extension that is loaded. For example, if the file that is currently open is “test.cpp,” the check box reads, “Use as default for .cpp.” If this is an untitled buffer, the check box is disabled.

Expand tabs with spaces
Tabs are inserted into the text as an appropriate number of spaces rather than as tab characters.

C++ mode
The buffer is treated as C++ code. Special options for C++ code are set in the **Editing/Browsing Settings** dialog box.

Persistent
Causes the buffer options for this file to be saved during the current IDDE session, even if the file is closed. Otherwise, buffer options are set to their global defaults if a file is closed and reopened.

Text Settings

Opens the **Editing/Browsing Settings** dialog box, in which you set global text editing options. Details of this dialog box are discussed in the "Text Settings" section later in this chapter.

Goto menu commands

The **Goto** menu (Figure 21-12) contains commands to move within the source file.

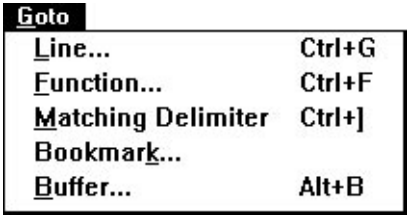

Figure 21-12 Goto menu commands

Line

Opens the **Goto Line** dialog box (Figure 21-13). Type the line number and click OK, and the insertion point is moved to the specified line.

Figure 21-13 Goto Line dialog box

Function

Opens the **Goto Function** dialog box (Figure 21-14).

Figure 21-14 Goto Function dialog box

The Function Name listbox holds the available function names. Either type in the function name or scroll and select the function name from the list. When you click OK, the insertion point moves to the beginning of the specified function.

Member functions in the list typically are displayed as *member*::*class*. To change the format to *class*::*member*, deselect the Reverse Class::Member Format option.

Matching Delimiter

Finds the delimiter that matches the one to the right of the current insertion point. The insertion point is moved to the front of the matching delimiter. This command can find matching parentheses, square brackets, or braces.

Bookmark

Opens the **Bookmarks** dialog box (Figure 21-15), which you can use to set and move to as many as ten different locations in your source files. Bookmarks are saved through the current IDDE session only.

Figure 21-15 Bookmarks dialog box

Bookmark list
Shows the locations of the ten bookmarks by file, line, and column. Click on an entry to select it; double-click on an entry to go to it.

Goto
Moves the insertion point to the selected bookmark. This command opens a file if it is not already open. You can also double-click on the bookmark in the list.

Clear
Removes the selected bookmark.

Drop
Sets the selected bookmark to the current insertion point. The entry in the bookmark list is updated to show the file, line, and column.

Buffer

Opens the **Edit Buffers** dialog box (Figure 21-16). This dialog box presents a list of files currently open in Source windows; it allows you to view and change each buffer's editing options and to perform various file-related operations.

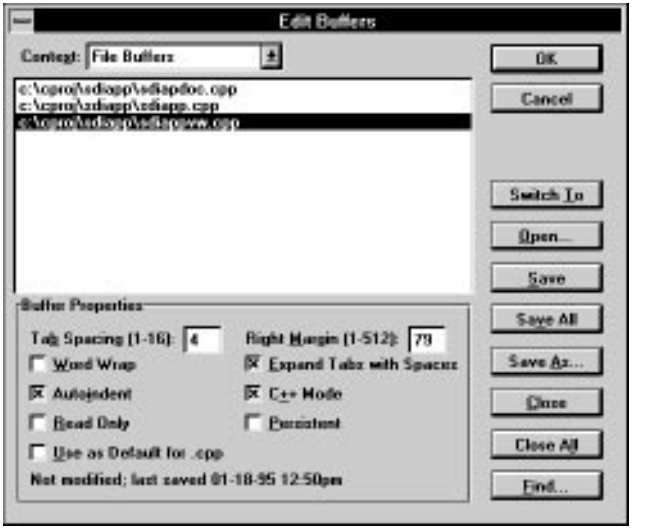

Figure 21-16 Edit Buffers dialog box

Context
This drop-down listbox specifies File Buffers or Member Buffers. File Buffers are Source windows open to edit an entire file. Member

Buffers are Class Editor Source panes and Hierarchy Editor Source child windows, open to edit a particular member definition.

Buffer list
Contains the names of files currently open in Source windows (or, if the Member Buffers context is selected, the names of member functions open in Class Editor Source panes and Hierarchy Editor Source child windows). Click on a filename to select it; double-click on it to bring the corresponding Source window to the front.

Buffer properties
Lets you view and set options for each individual buffer listed in the buffer list.

Tab spacing: Specifies the number of columns between tab stops.

Right margin: Specifies the column that acts as the right margin.

Word wrap: Enables word wrap. While typing, lines that extend beyond the right margin are broken automatically at the last word boundary before the margin.

Autoindent: Causes the text editor to indent automatically on newline. When you press Enter, the editor positions the cursor directly below the first nonblank character in the previous line.

Read only: The buffer may not be changed.

Use as default for ...: Saves the Current Buffer Options settings as the defaults for any subsequent file with the same file extension that is loaded. For example, if the currently open file is `test.cpp`, the check box reads, “Use as default for .cpp.” If the currently open file is `test.txt`, the check box reads, “Use as default for .txt.” If the currently open file has no extension, the check box reads, “Use as default.” If this is an untitled buffer, the check box is disabled.

Expand tabs with spaces: Tabs are inserted into the text as an appropriate number of spaces, rather than as tab characters.

C++ mode: Text is treated as C++ code. Special options for C++ code are set in the **Editing/Browsing Settings** dialog box (see “Text Settings,” later in this chapter).

Persistent: Causes the buffer options for this file to be saved during the current IDDE session, even if the file is closed. Otherwise, buffer

options are set to their global defaults if a file is closed and reopened.

Switch to
Brings the Source window containing the file selected in the listbox to the front.

Open
Opens a Windows **File Open** dialog box, with which you can select a file to open for editing.

Save
Saves the file selected in the listbox. If the file is untitled, this command executes **Save As**.

Save all
Saves all files in the listbox.

Save as
Opens a Windows **File Save As** dialog box, with which you can save the file selected in the listbox under a new name.

Close
Closes the file selected in the listbox.

Close all
Closes all files in the listbox.

Find
Opens the **Global Find** dialog box (see "Using Global Find," later in this chapter).

Macro menu commands

The **Macro** menu (Figure 21-17) allows you to record, play, and edit macros.

Figure 21-17 Macro menu commands

Record Macro
Starts the recording of the default macro. While a macro is being recorded, this menu choice is replaced in the menu by **Stop**

Recording. Menu and keystroke recording is limited to the current source window only.

To record a macro:

1. Choose **Record Macro**.

   If a default macro exists, you are asked to confirm that you want to record over the default macro. Click OK.

2. Enter the sequence of keystrokes and menu selections you want to record.

3. Choose **Stop Recording** to end the macro.

Play Macro

Plays back the default macro.

ScriptMaker

Opens the **ScriptMaker** dialog box (Figure 21-18), with which you copy, name, and edit macros.

Figure 21-18 ScriptMaker dialog box

Existing macros
This is the list of macros. Click on a macro to select it; double-click on the macro to edit it.

Menu order
These buttons allow you to change the order of the macros listed in the **Macro** menu. Click on the Up Arrow to move the selected macro up in the menu; click on the Down Arrow to move it down in the menu. The default macro always remains at the top of the list.

Put in menu
Causes the selected macro to be listed in the **Macro** menu.

Edit
Opens the Macro Editor window. For information about the macro language and the Macro Editor window, see the Symantec C++ IDDE Help.

Rename
Opens the **Rename/Clone Script** dialog box (Figure 21-19), with which you can change the selected macro's menu name or filename.

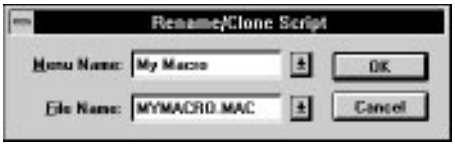

Figure 21-19 Rename/Clone Script dialog box

Menu name: Name under which this macro is listed in the Macro menu.

File name: Name of the file in which the macro is saved.

You can change either or both names. Note that you cannot rename the default macro.

Clone
This button opens the **Rename/Clone Script** dialog box. This command makes a copy of the selected macro.

Menu name: Name under which this macro is listed in the Macro menu.

File name: Name of the file in which the macro is saved. When cloning, this filename must be different from that of any other macro.

Note

To create a new macro, first use the **Record Macro** command to record it as the default macro. Then choose **Scriptmaker** and use Clone to make a copy of the default macro under a new name.

Delete
Deletes the selected macro.

New! command

Opens another Source window on the current file.

Changes made to a file in one Source windows are made automatically in other Source windows containing the same file.

Pop-up menu commands

The pop-up menu (Figure 21-20) is opened by clicking the right mouse button in the edit area of the Source window.

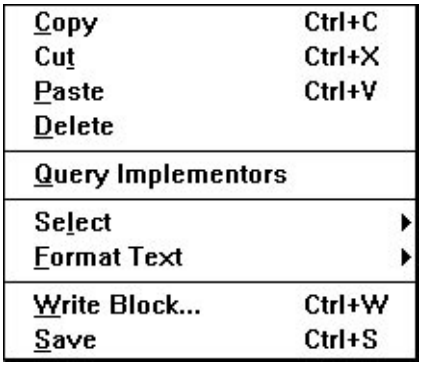

Figure 21-20 Source window pop-up menu commands

Copy Copies the selected text to the Clipboard.

Cut Copies the selected text to the Clipboard, then deletes it from the buffer.

Paste Inserts the text from the Clipboard at the insertion point.

Delete Deletes the selected text from the buffer.

Query Implementors Interprets the tokens or symbols surrounding the insertion point as a C++ class member name and locates all classes with a member

of this name. The results are displayed in the Members window (Figure 21-21).

Figure 21-21 Members window

For example, if the token is `Test`, **Query Implementors** shows a list of all implementors of `Test`, such as `One::Test`.

In the Members window, you can select an implementor and choose **Show Source** from the **Member** menu to open a Source window to the corresponding source code or, double-click an implementor to open a Class Editor window to the member source. Note that if only one implementor of a token is found, the **Query Implementors** command opens the Class Editor window directly, without first opening the Members window.

Select

Opens the **Select** submenu (Figure 21-22).

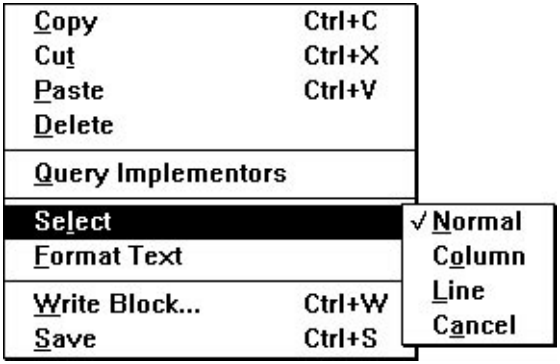

Figure 21-22 Select submenu commands

Normal

Restores the original text select block to normal mode, undoing changes caused by **Column** and **Line** (see the following).

Column Changes the text selection block to a column-oriented select block, in which only the characters in the columns between the start and end of the original text block are selected.

Line Changes the text selection block to a line-oriented select block, in which all characters in the lines between the start and end of the original text block are selected.

Cancel Deselects the current select block.

Format Text Opens the **Format Text** submenu (Figure 21-23).

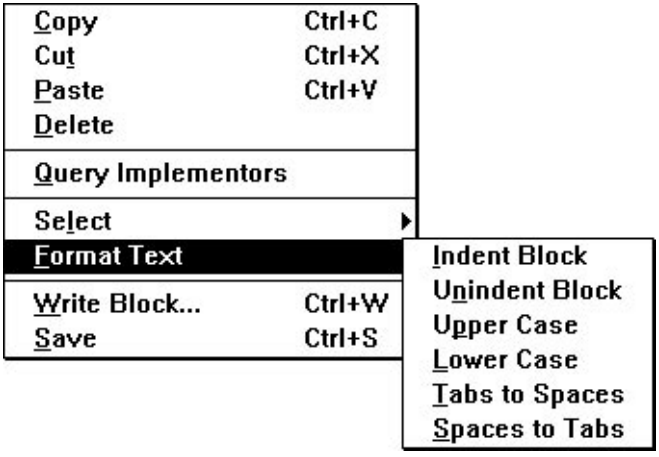

Figure 21-23 Format Text submenu commands

Indent Block Indents all nonblank lines in the selected text by one tab stop. Tabs are inserted as tab characters or as spaces, depending on the current buffer option settings. All text in a region will be indented if a region is selected upon issuing the command.

Unindent Block Unindents all lines in the selected text by one tab stop. All text in a region will be unindented if a region is selected upon issuing the command.

Upper Case Changes all alphabetic characters in the selected text to uppercase.

Lower Case Changes all alphabetic characters in the selected text to lowercase.

Tabs to Spaces Changes all tab characters in the selected text to spaces. The number of spaces used to replace each tab character depends on the Tab spacing option.

Spaces to Tabs Changes spaces in the selected text to tab characters. The number of spaces used to create each tab character depends on the Tab spacing option.

Write Block Opens a **Write Block** dialog box. Select a file or type a new name; the editor writes the currently selected text block to this file. To append the selection block to a file, check Append.

Save Saves the current buffer to disk. If the file is untitled, this command executes **Save As**.

Toolbar commands

The Source window toolbar (Figure 21-24) offers quick access to several menu choices.

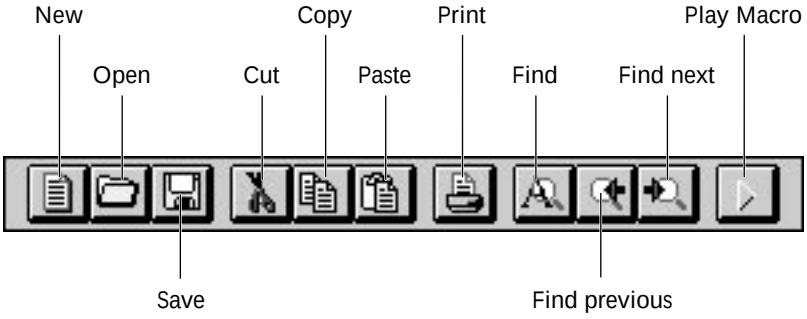

Figure 21-24 Source window toolbar

New: Same as choosing **New** from the **File** menu.

Open: Same as choosing **Open** from the **File** menu.

Save: Same as choosing **Save** from the **File** menu.

Cut: Same as choosing **Cut** from the **Edit** menu.

Copy: Same as choosing **Copy** from the **Edit** menu.

Paste: Same as choosing **Paste** from the **Edit** menu.

Print: Same as choosing **Print** from the **File** menu.

Find: Same as choosing **Find** from the **Edit** menu.

Find previous: Searches backward in the file for the search string.

Find next: Searches forward in the file for the search string.

Play macro: Same as choosing **Play Macro** from the **Macro** menu.

Note that the Find previous and Find next buttons are subtly different from **Find Again** in the **Edit** menu, which can only repeat the search in the original direction.

Text Settings

Choosing **Text Settings** from the Source window's **Edit** menu opens the **Editing/Browsing Settings** dialog box, a workspace with tabs along the top margin. The tabs are used to switch between several sets of options. Each set of options is described below.

General options

The General options set (Figure 21-25) contains options for undo levels and the key binding file, as well as some options related to the Class and Hierarchy Editors.

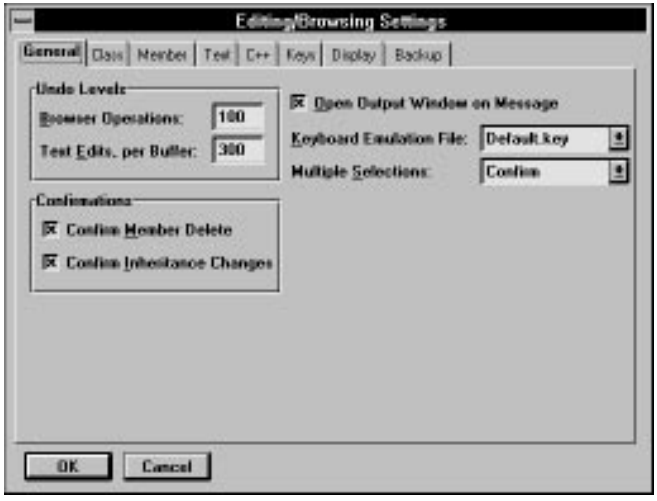

Figure 21-25 General options

Browser operations
Specifies the number of operations that can be undone in the Class and Hierarchy Editors. See Chapter 19, "Class Editor Reference," and Chapter 20, "Hierarchy Editor Reference."

Text edits, per buffer
Specifies the number of edit operations that can be undone per buffer.

Confirmations
Enables confirmation requests for various operations in the Class and Hierarchy Editors. See Chapter 19, “Class Editor Reference,” and Chapter 20, “Hierarchy Editor Reference.”

Open output window on message
Lets the IDDE open an error window whenever there is an error of any kind (during compilation, during parsing, and so on.)

Keyboard emulation file
Specifies the key bindings set to be used. Key bindings allow you to associate keystroke sequences with functions and macros. Information about key bindings sets can be found in the Symantec C++ IDDE Help.

Multiple selections
Enables multiple selections in lists in the Class and Hierarchy editors. See Chapter 19, “Class Editor Reference,” and Chapter 20, “Hierarchy Editor Reference.”

Text options

The Text options set (Figure 21-26) contains options for indentation, cursor styles, keyboard emulation, and text editor font.

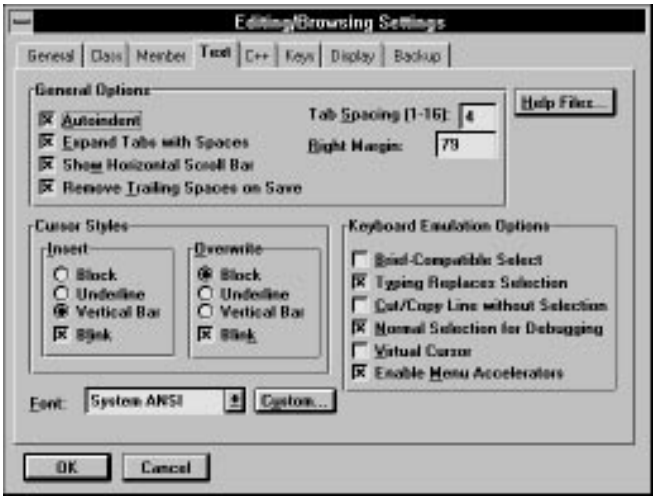

Figure 21-26 Text options

Tab spacing

Specifies the default for the number of columns between tab stops. This value may be overridden locally in each buffer.

Right margin
Specifies the default for the column that acts as the right margin. This value may be overridden locally in each buffer.

Autoindent
Indents automatically on newline. When you press Enter, the editor positions the cursor directly below the first nonblank character in the previous line. This option may be overridden locally in each buffer.

Expand tabs with spaces
Tabs are inserted into the text as an appropriate number of spaces, rather than as tab characters. This option may be overridden locally in each buffer.

Show horizontal scroll bar
Enables the horizontal scroll bar at the bottom of the Source window.

Remove trailing spaces on save
Trailing spaces and Tabs are removed from the end of each line when a file is saved.

Cursor styles
Specifies caret style. You may set styles individually for the caret in Insert and Overwrite modes. Styles are:

Block: The current character is displayed in inverse video.

Underline: The current character is underlined.

Vertical bar: A vertical bar appears to the left of the current character.

Blink: The cursor blinks. The blink rate is specified in the Windows Control Panel.

Font
Specifies the text font. You can select a predefined font from the drop-down list, or you can click Custom and select any installed font from a Windows **Font** dialog box.

Brief-compatible select
If you choose this option, then enter the “Toggle Mode Select” mode. The editor remains in selection mode when you use the arrow keys.

Typing replaces selection
Enables the Windows standard convention of replacing selected text with any typed character or pasted text. If this option is not selected, typing or pasting inserts the text to the left of the current selection.

Cut/copy line without selection
If no text is selected, **Cut** and **Copy**, respectively, cut and copy the current line. If text is selected, **Cut** and **Copy** work as usual.

If this option is not selected, **Cut** and **Copy** have no effect if no text is selected.

Normal selection for debugging
Enables normal selection of text when in debugging mode. If disabled, you can drag from the source window to the Assembly, Data/Object, and Function windows while debugging.

Virtual cursor
Enables virtual cursor mode, in which you can position the caret anywhere in the window, regardless of line endings. Note that even with this option enabled, you still cannot position the caret beyond the last line in the file.

Enable menu accelerators
Enables menu accelerator keys. With this option selected, new windows have underscores beneath the top-level menu items to show the Alt key combination you can use to access the menu.

Help Files

Clicking on Help Files opens the Text Help File Configuration dialog box, as shown in Figure 21-27.

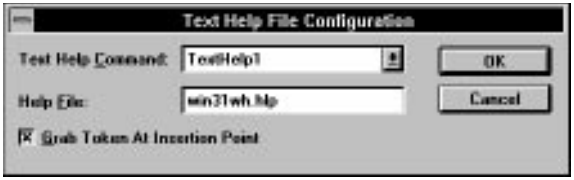

Figure 21-27 Text Help File Configuration dialog box

This dialog box lets you associate particular Windows Help files with each of the four Text Help commands. The Text Help commands are run by key sequences such as Shift+F1. (The exact mapping of key sequences to Text Help commands depends on the current key mapping. See "Keys options," later in this chapter.) The Text Help commands call the Windows API function Windows Help, passing it the name of a Windows Help file and (optionally) a keyword. The Text Help commands are useful, for example, for gaining access to help on a particular API function or MFC class directly from the source code.

Text Help Command: Specifies a Text Help command (TextHelp1, TextHelp2, TextHelp3, TextHelp4).

Help File: Specifies the Windows Help file to be associated with the selected Text Help command. This filename is passed to Windows Help when the Text Help command is run.

Grab Token At Insertion Point: If this box is checked, the Text Help commands pass the token at the insertion point in the text buffer to Windows Help as a keyword. This causes Windows Help to search for the associated topic, and, if found, to immediately display help on the keyword topic.

C++ options

The C++ options set (Figure 21-28) contains options to check delimiters, indent after braces, and auto-align comments. It also allows you to add custom keywords to the keyword dictionary.

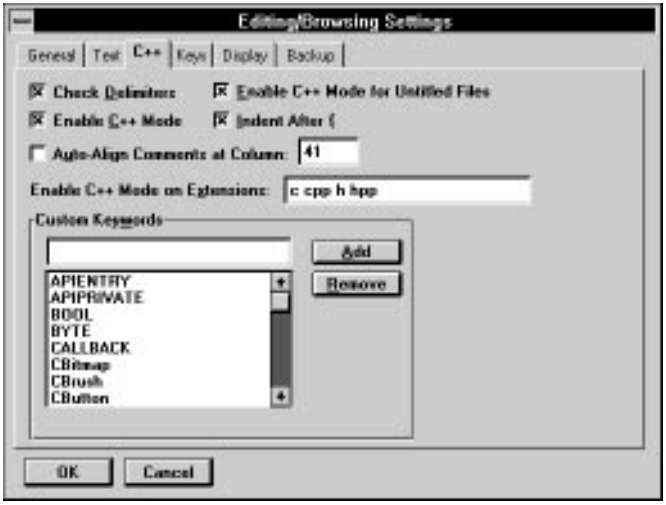

Figure 21-28 C++ options

Check delimiters

If you type a right parenthesis, square bracket, or brace, the editor briefly highlights the corresponding left delimiter. If no matching delimiter is found, an error message is displayed.

Enable C++ mode

This option enables C++ mode globally. If it is not selected, C++ mode features are disabled for all buffers, regardless of local buffer option settings.

Enable C++ mode for untitled files

When this option is on, new files that have not yet been given a name are assumed to be C/C++ source. This box should be checked in most circumstances, so that keywords can be recognized in new files, for example.

Indent after {

If the last character typed on a line is a left brace, the next line is indented automatically by an extra tab stop. This option works only if Autoindent is enabled in the buffer. Also, if the first character on a

line is a right brace (}), the line is unindented automatically; this is independent of the Autoindent option.

Auto-align comments at column
If this option is enabled, when you type "//" to start a C++ comment, the editor automatically indents the comment to a specified alignment column. You can specify the alignment column in the adjacent textbox.

Enable C++ mode on extensions
Specifies the file extensions for which C++ mode is automatically enabled. Type the extensions into the textbox, separated by spaces.

Custom keywords
You can maintain a set of custom keywords that are highlighted in the edit window in a manner you specify (see "Display options," later in this chapter).

To add a new keyword, type the keyword into the textbox and click on Add. To remove a keyword from the list, click on the keyword in the list and click on Remove.

Keys options

The Keys options set (Figure 21-29) lets you customize key bindings and assign key combinations to macros.

Figure 21-29 Keys options

A Key Bindings file (`.key`) associates keystroke sequences with editor commands and user-defined macros. You can select the particular key binding set you want to use either with the Key File option below, or with the Keyboard Emulation file option under the General tab (see "General options," earlier in this section).

Commands are grouped into functional categories. The groups are:

- Global: Global commands, used anywhere (includes all user-defined macros)
- Member: Member-related commands, used in the Class and Hierarchy Editors
- Text: Text editor functions and commands, used in Source windows
- Class: Class-related commands, used in the Class and Hierarchy Editors
- Project: Project-related commands, used in the Project window

You may assign more than one keystroke sequence to a command. However, within a category, only one command may be associated with a particular keystroke sequence.

Key file
Specifies the key bindings set to be used. Select a keyboard emulation file from the drop-down list.

Keys
Specifies a key sequence. This is not an ordinary textbox; it can display any keystroke sequence.

There are two ways to enter a keystroke sequence into the textbox:

- Click in the textbox and type the keystroke sequence. You can enter most sequences in this way. (Keys you cannot enter directly into the box include Home, End, Delete, Backspace, Right Arrow, Left Arrow, and Tab.)
- Use the Recorder buttons. Click on the green arrow to start recording. Enter your keystroke sequence. Click on the red box to stop recording.

Commands/macros
The textbox displays the currently selected command (or macro). You can type in the command name or select a command from the list by clicking on it.

The list displays commands and associated key sequences. If more than one sequence is assigned to a command, it is listed as many times as necessary. The content and sorting of the list is determined by the Commands/Macros List Options.

Commands/macros list options
Determine the filtering and sorting of commands and macros shown in the Commands/Macros list.

Scope: Displays commands only in the specified functional category. Categories are Global, Member, Text, Class, and Project. Specify All to see all commands.

Show bound keys only: Shows only those commands with an associated keystroke sequence.

Sort by command: Displays the list alphabetically by command. If this option is not selected, the list is ordered by key sequence.

Copy to clipboard: Copies the current contents of the commands/ macros list to the Clipboard.

Assign
Assigns the keystroke sequence to the selected command.

If another command in the same functional category is already bound to this keystroke sequence, you are asked if you want to reassign the sequence to the new command.

Unassign
Dissociates the selected command from its keystroke sequence.

Save as
Saves the key bindings to a new file. When prompted, type the name of the new file, or select a file from the drop-down list.

Display options

The Display options set (Figure 21-30) lets you select special font colors and styles for keywords, comments, preprocessor symbols, and other special text.

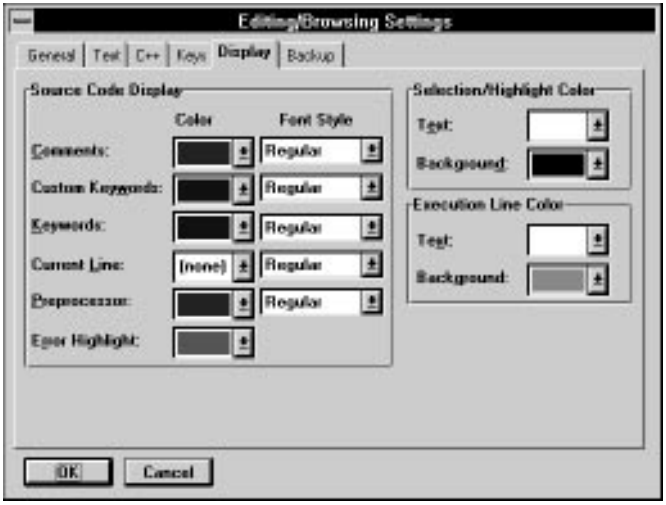

Figure 21-30 Display options

Source code display
Allows you to customize syntax highlighting. You can specify the highlighting for:

Comments: C/C++ comments

Custom keywords: Special keywords you specify. (See "C++ options," earlier in this chapter.)

Keywords: C/C++ keywords

Current line: The line containing the insert point

Preprocessor: C/C++ preprocessor directives

Error highlight: Lines on which compiler errors were found

Select Color and Font Style from the drop-down lists next to each item. (If you select the first color, which is labeled "none," the default text color is used for that item.)

Selection/highlight color
Specifies the text and background color of selected text.

Execution line color
Specifies the text and background color of the current execution line during debugging.

Backup options

The Backup options set (Figure 21-31) contains options for backing up files.

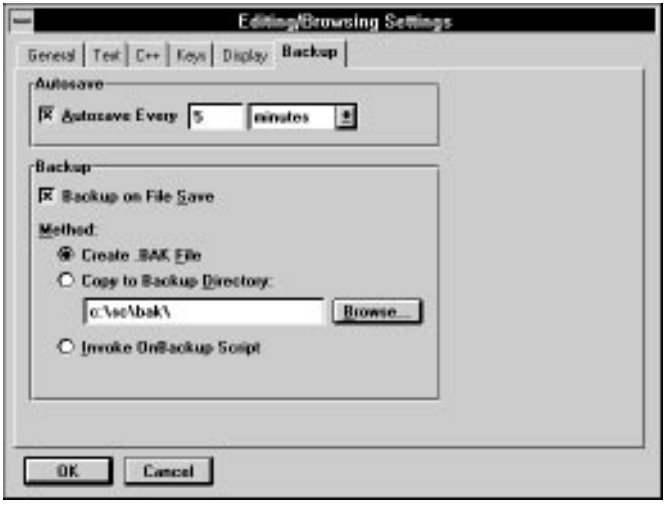

Figure 21-31 Backup options

Autosave
Causes the editor to save your work automatically after a certain number of changes, or after a certain number of minutes since the last save. Specify the number of changes or minutes in the textbox, and select changes or minutes from the drop-down list.

Backup on file save
This enables automatic backup on save. You must also select the backup method:

Create .bak File: Copies the previous saved version to file with the extension `.bak`.

Copy to backup directory: Copies the previously saved version to another directory. Type the directory into the textbox, or click on Browse to select a directory from a **Directory** dialog box.

Invoke OnBackup script: Runs a macro called OnBackup.

Note

The Autosave option also provides some protection against data loss in the event of a system crash. When you check Autosave, the editor saves the contents of each modified buffer to a temporary file on disk. If the editor does not exit normally (as with a crash), these files (named ~<*num*>.SAV, where <*num*> is a unique number) will not be deleted. Line 1 of a .SAV file specifies the drive, directory, name, and extension of the buffered file, also with the data and time it was last saved. The rest of the file stores the contents of the buffer, which you may be able to recover; use the editor's **Save As** option to save the .SAV file as a source file.

Using Global Find

Global Find is a multi-file search facility. You specify the files to be searched and the string or regular expression to be searched for. Global Find presents a list of files in which a match was found and allows you to view and edit the files, add files to the project, or refine the search criteria and search again.

Defining the search

You open the **Global Find** dialog box (Figure 21-32) by choosing **Global Find** from a Source window's **Edit** menu, choosing **Global**

Find from the IDDE's **Tools** menu, or clicking on the Find button in the **Edit Buffers** dialog box.

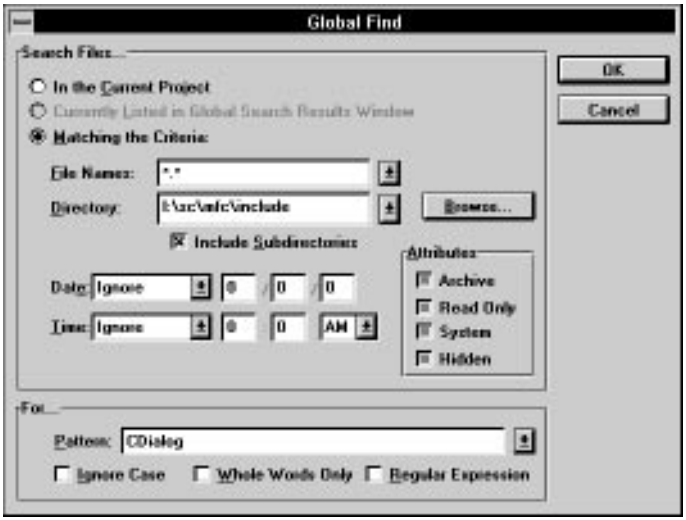

Figure 21-32 Global Find dialog box

The **Global Find** dialog box has two sections. The upper section specifies the files to be searched, and the lower section specifies the pattern to be searched for.

Search files

Three options are available for specifying the files to be searched.

In the current project
All files in the current project are searched.

Currently listed in global search results window
Enabled only after an initial global search has been performed. It limits the search to files in which a match was found in the previous global search.

Matching the criteria
Allows you to specify files by filename patterns, directory, date, time, and attributes.

File names: Comma-separated list of filenames and patterns. To search all files, use `*.*`.

Directory: Search files in the specified directory. You can click on Browse to select a directory from the **Choose Directory** dialog box. Check Include Subdirectories to search files in subdirectories as well.

Date: Select Ignore to ignore the date. Otherwise, specify a date and one of the relational options. For example, specify On and 11/6/94 to search files last modified on November 6, 1994, or After 4/1/90 and to search files last modified after April 1, 1990.

Time: Select Ignore to ignore the time. Otherwise, specify a time and one of the relational options.

Attributes: Search files with the given attributes. File attributes are Archive, Read Only, System, and Hidden.

The Attributes check boxes are three-state check boxes. If an attribute is checked, files with the given attribute are searched. If an attribute is cleared, files without the given attribute are searched. If an attribute is grayed, the attribute is ignored when deciding which files to search.

Search Pattern

Type the pattern to search for into the textbox. Three options can be used to modify the search:

Ignore case
Do not consider case when searching for a match.

Whole words only
Consider text a match only if it is not part of a larger alphanumeric string.

Regular expression
Enables regular expression matching.

Regular Expressions

A regular expression is a string with wildcards. The following wildcards are supported:

Wildcard	Meaning
?	Matches any character.
*	Matches zero or more occurrences of any character.

Wildcard	Meaning
@	Matches zero or more occurrences of the previous character or expression. For example, Ax@B matches AB, AxB, AxxB, and so on.
% or <	Matches the beginning of a line. For example, <{ finds lines that start with left braces.
$ or >	Matches the end of a line. For example, %$ finds blank lines.
[...]	Matches any of the characters listed between the square brackets. A hyphen can be used to specify a range of characters. For example, [axz] matches a, x, or z; [a-z] matches any lowercase letter.
[~...]	Matches any characters except those listed.
\	Escape character indicates that the following character should be taken literally rather than used as a wildcard character. For example, \% matches a percent sign.
\t	Matches a tab character.
\f	Matches a formfeed character.

The Search window

After specifying the search criteria, click OK to start the search. As the search begins, the editor opens the Search window (Figure

21-33). This window contains a list of files in which a match is found.

Figure 21-33 Search window

Also during the search, the **Search Progress** dialog box displays statistics (Figure 21-34). Click Stop to terminate the search at the current point, or click Revert to return to the **Global Find** dialog box.

Figure 21-34 Search Progress dialog box

When the search is complete, the **Search Progress** dialog box closes and the Search window contains a list of files in which at least one match occurred.

Search menu commands

The Search window's **Search** menu (Figure 21-35) contains commands to open files in Source windows, add files to the project, and continue the global search.

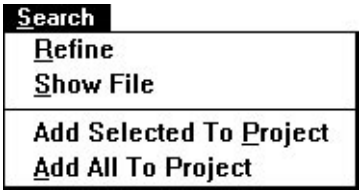

Figure 21-35 Search menu commands

Refine Reopens the **Global Find** dialog box. You may refine your search criteria and continue the global search. This command is disabled if no matches were found.

Show File Opens the selected file in a Source window. You can also open a file by double-clicking on the file in the list. The file is positioned to the first match of the search pattern.

Add Selected To Project Adds the selected file to the current project.

Add All To Project Adds all files listed in the Search window to the current project.

Toolbar commands

The Search window toolbar (Figure 21-36) offers quick access to menu choices.

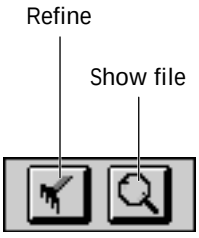

Figure 21-36 Search window toolbar

Refine: Same as choosing **Refine** from the **Search** menu

Show file: Same as choosing **Show File** from the **Search** menu

Using Version Control 22

The IDDE's Version Control System (VCS) allows programmers to work safely with the same source files simultaneously, integrate their changes, rebuild their own sources, and store updated master files in a common master archive.

The VCS works in combination with INTERSOLV's PVCS, a popular version control program for the PC. However, VCS offers significant functional improvements over PVCS alone, because it is integrated with the IDDE's project system and has access to information stored in the project (`.prj`) file. Symantec's VCS guards against accidentally overwriting changes made by others and provides the highest level of support possible for keeping both your private project directory and the master project directory fully synchronized.

Note

If you do not have PVCS, you need to purchase it from INTERSOLV before you can use Symantec's VCS. See the PVCS documentation for information on PVCS features and commands.

This chapter assumes a knowledge of PVCS and a familiarity with basic version control concepts. The chapter focuses on using the group-oriented model of software development that VCS makes possible, and it explains how to use VCS to build and maintain projects.

Note

The IDDE's Version Control System (VCS) is supported only in the 16-bit version of Symantec C++.

Overview of VCS Concepts

This section defines basic terms used in this chapter and compares the parallel model of version control to the linear model supported by other version control systems.

The next section, “Setting Up Version Control with VCS,” presents an overview of how you or your project team can use VCS to establish and maintain version control on software development projects of any size.

Terminology

The following terms are used throughout this chapter:

- Archive (or master archive): All revisions of a source file, stored in a commonly accessible directory structure, usually on a network.
- Revision: A revised archived source file that has a revision number and timestamp that identifies when the revision was created.
- Workfile: Your private copy of the revision that you are currently working on.
- Private project directory: Your personal working directory to which others do not have access. You work with workfiles in your private project directory.
- GET: The process of obtaining a private copy of a revision of an archive (usually the latest revision).
- MERGE: The process of collating changes to a revision into the archive, without overwriting or corrupting any changes made by others since you got the revision.
- PUT: The process of creating a new revision of a file in the archive.
- Version: A group of revisions containing all source files that make up a project. Typically you create versions to correspond to milestones in the development process, such as the weekly build or the beta build.

Version control models

There are two basic models for version control:

- Linear: Only one person can work with a file at any one time.
- Parallel: Groups of people can work with the same set of files simultaneously.

Most version control systems support a linear model of version control. In practice, few support a parallel version control system because additional support is required to prevent losing changes. The Symantec C++ IDDE, however, fully supports the parallel model.

Setting Up Version Control with VCS

This section gives you an overview of how you can use VCS on your software development project. The remainder of the chapter explains how to use VCS to perform the operations introduced here.

Using the linear model with VCS

In the linear model, VCS locks a revision when someone GETs it; that is, VCS prevents others from making a private copy. Thus, only one person may work with a revision at a time. It is even possible to lock all revisions that make up a particular version.

After changing the revision, the person who receives it must PUT it back into the master archive before others can make changes to it.

The advantage of the linear method of version control is that you do not have to collate (merge) sets of changes to a revision. The obvious disadvantage is that only one person can work on a file at a time. In large development projects in which many people are making changes to related groups of files, this model is often unworkable.

Using the parallel model with VCS

The parallel version control model allows two or more people to work simultaneously with copies of the same revision—a clear advantage even in a small work group. The disadvantage of this model is that each person's changes must be merged with changes made by others, so the master archive of a revision incorporates everyone's work and the related changes made to other files (sometimes called "dependencies").

When working in a parallel version control environment, the revisions and versions used by the development team are stored in a master archive, usually on a network. Developers GET a revision (or more typically a group of related revisions) from the master archive and work with the copies (workfiles) in their own private project directory.

After making changes to a group of workfiles, developers must MERGE their changes into the latest version of revisions of the files. This is done in the private project directory because it often involves identifying changes made by others in the interim, along with resolving dependencies in other revisions that were affected by other people's changes.

Note

In your revised file, be sure to identify the revision with which you want your changes merged. You make this reference by adding a `$revision` entry in the file.

When changes are merged successfully, and all related changes have been made to affected revisions in the private project directory, each developer PUTs the new revisions back into the master archive.

If a revision has been modified during the MERGE, it must be remerged before it can be PUT into the archive. VCS warns you of this automatically and gives you a way to prevent other people from merging changes into the archived version of a revision until you have successfully PUT it.

The parallel model also supports branching of revisions. When you create a branch of a revision, you can PUT it back into the archive without performing a MERGE. This allows others to GET your changes to a revision. You use the branching mechanism if you want a set of changes available to you but you do not want to MERGE the changes until your work is complete. For more information, see "Creating a new branch," later in this chapter.

Setting VCS Options

This section explains how to use the options in the VCS page of the **Project Settings** dialog box to set up version control parameters that specify the rules VCS follows when you GET and PUT files.

To open this dialog box, open the project you want to work with, and choose **Settings** from the IDDE's **Project** menu. In the **Project Settings** dialog box, select the **VCS** tab. The **VCS** page is shown in Figure 22-1.

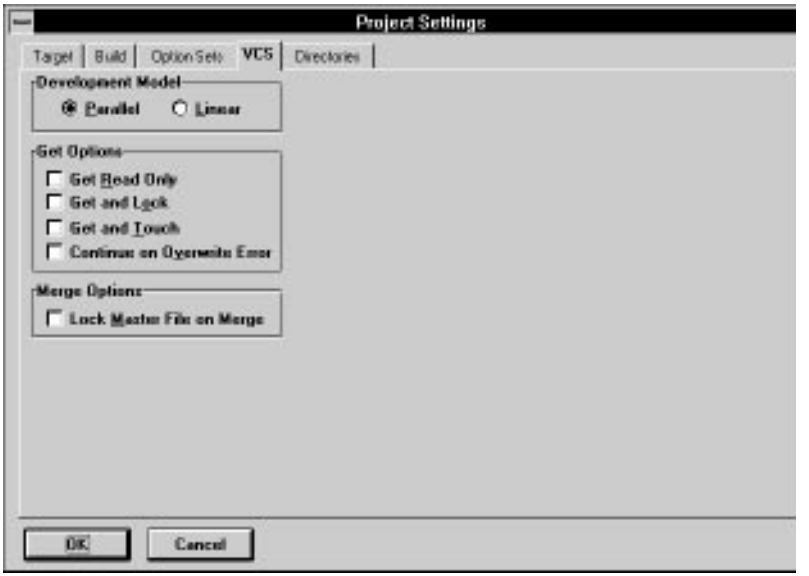

Figure 22-1 VCS page in Project Settings dialog box

Choosing a Development Model

The Development Model radio buttons specify the development model that VCS uses. Select the Parallel model, described earlier in this chapter, unless you are the only person working on the files in a project.

Select Linear if you are working alone, or if you need to deny other developers access to revisions you are working on.

You can switch from the parallel to the linear mode when, for example, you need to prohibit GET operations on revisions you are merging (see "Merge Options," later in this chapter).

The IDDE sets the rest of the VCS options for you, based on the development model you select, as described later in this chapter. If you need more control over the version control process, you may change the default settings. This is not usually necessary.

Get Options

These options specify how VCS handles your requests to GET revisions from the master archive:

Get read only: Lets you GET files from the master archive, but with read only access; you cannot modify the files you GET. Check this option if you want to view a revision (for example, an old revision), or if you want to ensure that you do not accidentally change the files. This option is off by default in both development models.

Get and lock: Prevents others from successfully executing a GET operation on a revision once you GET it. The revision remains locked until you PUT it back in the archive again. This option is on by default when you choose the Linear development model.

Get and touch: Sets the time-stamp for any revision you GET to the current date and time. This setting forces recompilation of your files. It is off by default in both development models.

Continue on overwrite error: Tells the IDDE to overwrite a file on a GET operation without displaying a warning. Use this setting to wipe out all changes you have made to a revision and GET a new copy from the master archives. This option is off by default in both development models.

Merge Options

The Lock Master File on Merge Setting prohibits others from executing a PUT operation on a revision in the master archive while you are performing a MERGE on it. This option is off by default in both development models.

Creating a VCS Configuration File

A VCS configuration file defines a series of variables that tell VCS where to look for and create files. You can create a VCS configuration (`.cfg`) file with the integrated editor or any other text editor. You need a `.cfg` file for each project in which you use VCS.

A sample configuration file, `vcs.cfg`, comes with Symantec C++. (It is installed in `\sc\samples\windows\wclock` by default.) The sample `vcs.cfg` contains information that VCS needs to work:

- `vcsdir`: The directory in which VCS stores master archives. This variable must be defined for VCS to work. (All others are optional and default to the current directory.)
- `journal`: The file name and path in which VCS creates a journal (`journal.vcs`) that contains the information about changes to revisions in an archive.
- `archivework`: The directory in which VCS keeps backup copies of revisions as they are updated. (This command corresponds to the `LOGWORK` command in older versions of PVCS.) Do not specify a RAM disk as the `ARCHIVEWORK` directory.
- `workdir`: The VCS temporary directory.

It is a good idea to give the `.cfg` file for a project the same name as the project itself and to save it in your private project directory.

Selecting the configuration file

Before you use VCS for the first time on a project, you need to select the configuration file for the project:

1. In the Project window, choose **Configuration** from the **VCS** menu. The **Set VCS Configuration** dialog box opens.

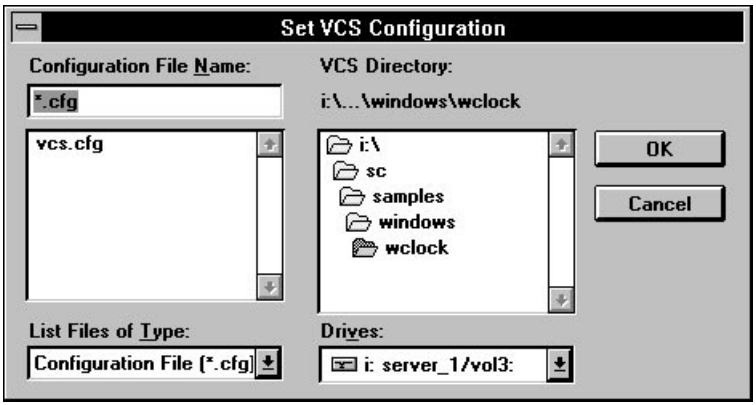

Figure 22-2 Set VCS Configuration dialog box

2. Select the configuration file you want and click OK. You need to specify its location if it is not in the current directory.

The PVCS Registration dialog box

The first time you open the **Set VCS Configuration** dialog box, the **PVCS Registration** dialog box is displayed. You must have an INTERSOLV LAF (License Administration Facility) code to use PVCS in the IDDE. INTERSOLV supplies the LAF code to licensed users. For more information, please contact INTERSOLV. Type the code in the License Code textbox, then click OK to continue. After the **PVCS Registration** dialog box disappears, a message informs you that you must restart Windows.

The User database directory textbox shows the database directory in which the license database is placed. This directory should be a shared directory on the network if multiple developers are using the same copy of the IDDE from the network. If developers have individual copies of the IDDE, do not change the text in this box.

Note

If you have PVCS version 5.0 or later, the IDDE does not display the **PVCS Registration** dialog box.

Putting Revisions into the Archive

When you PUT a revision in the master archive, VCS updates its revision number and time stamp and marks it as “not changed” with respect to the corresponding workfile in your private project directory.

Note

If, while you were performing a MERGE, another person has PUT a new revision of a file you want to PUT, VCS warns you with a modal dialog box that you must perform another MERGE operation on the revision.

To prevent others from working on a revision while you are merging it, check Lock Master File on Merge. See the section “Merge Options,” earlier in this chapter, for more information.

To begin a PUT operation, choose **Put** from the Project window’s **VCS** menu. The **Put** dialog box is displayed.

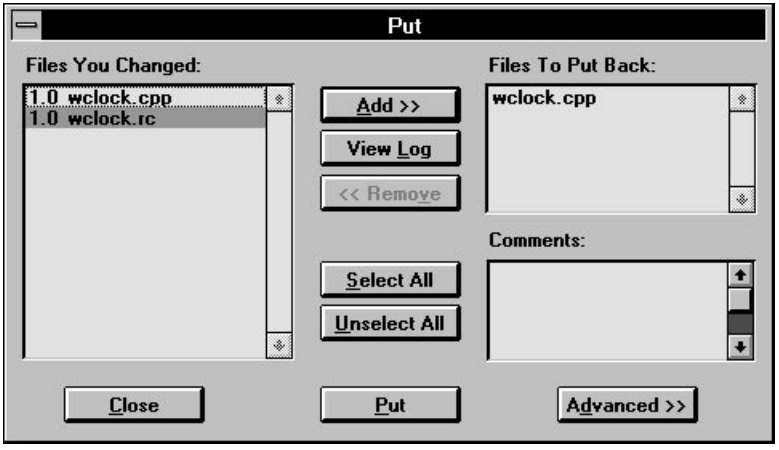

Figure 22-3 Put dialog box

The Put Candidate Files listbox shows those workfiles in your private project directory that have changed since you checked them out. Click the Add button to add files from this list to the Files to Put Back listbox; this shows the files you want to PUT into the master

archive. To remove a revision from the Files to Put Back list, select it and click on Remove.

Click on Select All to select all files in the current listbox. Click on Unselect All to deselect them.

Select a file name and click on View Log for a summary of who changed the latest revision of that file and the changes that were made.

To associate comments with all selected files in the Files to Put Back listbox, type them into the Comments textbox.

Click on Put to PUT the files listed in the Files to Put Back listbox into the master archive.

Click on the Advanced button and the **Put** dialog box expands to show additional options.

When you PUT files into the archives, you can specify a group of files using the Put Special textboxes. The Put Special options are:

Revision: Specifies the revision number you want to give the workfiles you are putting into the archives. The revision number has the format *nn.xx*, where *nn* is the major number and *xx* is the minor number. The numbers can range from 0 to 65,535.

If you do not explicitly assign a revision number to a revision, VCS assigns one automatically by incrementing the minor number of the most recent revision by one.

Version: Assigns the name you want to a group of revisions, such as beta or alpha. VCS associates the version name with all revisions listed in the Files To Put Back listbox.

Date/Time: The date and time of a version or revision you archive. The default date format is *mmm*/*dd*/*yy*; *mmm* is the month, *dd* the day, and *yy* the year. The time format is *hh*[:*mm*[:*ss*]]; *hh* is the hour, *mm* is the minute, and *ss* is the second.

Creating a new branch

Use the Create New Branch option to maintain a private set of revisions in the master archives without having to MERGE them. When you have a private branch, you can change a file and perform PUT operations as often as necessary, without affecting other developers.

When you are ready, you can MERGE your private revisions back into the master archive.

Note

A MERGE is the only way to make the changes in a branch available to other developers.

Getting Revisions from an Archive

Before you can work with archived revisions, you need to GET them from the archive and copy them to your private project directory using the **Get** dialog box, shown in Figure 22-4. (Both the archive directory and your private project directory are specified in the VCS configuration file for the project.)

When you GET a revision from an archive, VCS records the revision identification numbers of the files, marks them as unchanged and not new, and adds them to your project (if you so specify in the **Get** dialog box).

If you are using the linear development model, VCS locks the archived revisions when you GET them so nobody else can GET them until you PUT them back. If you are using the parallel model, more than one developer can check out the revision unless you specify Get and Lock on the VCS page in the **Project Settings** dialog box.

To GET revisions from the master archive:

1. In the Project window, choose **Get** from the **VCS** menu. The **Get** dialog box is displayed.

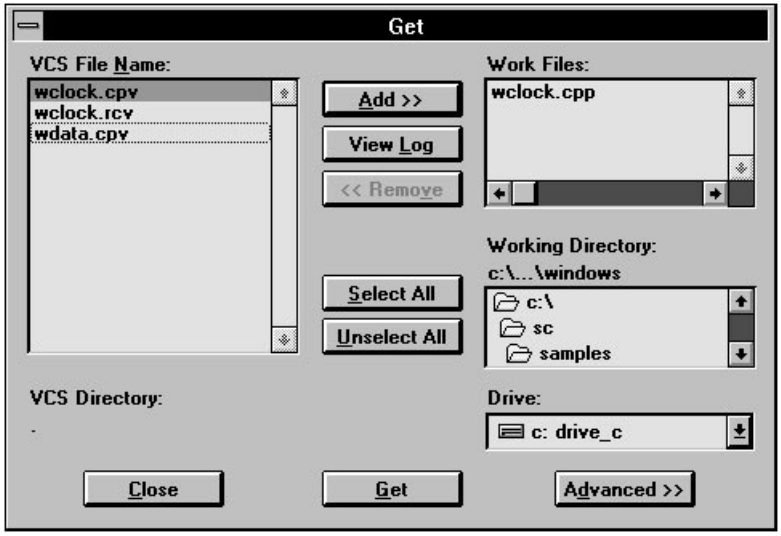

Figure 22-4 Get dialog box

2. The VCS File Name listbox shows the names of all revisions in the archive. Select from this list the files you want to move to your private project directory.

   To add one or more files, click on the file name(s), then click on the Add button. To add all files in the archive, click on the Select All button. VCS moves the files you select to the Work Files listbox.

   To view a summary of who made certain changes to a revision, click on View Log. (This is the same log that is available from the **Merge** dialog box.)

3. When all the files you want are in the Work Files list, click the Get button. VCS copies those files to your private project directory.

   To remove files from the Work Files list, click on the file name(s), then click on the Remove button.

Click on the Advanced button to display the following additional options: Show Archives, Get Special, and Add Files to Project.

Show Archives provides the following settings:

All archives: Displays all archives in the VCS File Name listbox. This is the default setting.

New archives: Displays in the VCS File Name listbox the names of the new archives and archives that have changed since the last time you performed a GET operation. Choose this option to update your private copy of the project with the most current revision(s).

Except merges: Displays in the VCS File Name listbox all the new revisions in the archive except those you just merged or need to merge. Select this option to choose only those additional archives you need to build and to test merged revisions.

The Add Files to Project option adds the revisions automatically to the current project when VCS gets them from the archive.

To retrieve a specific revision or version, use the Get Special textboxes:

- Revision is the number of the revision you want to get from the archives. The revision number has the format *nn.xx*, where *nn* is the major number and *xx* is the minor number.
- Version is the name of the version you want to GET. The version name identifies a particular version, such as alpha or beta.
- Date/Time is the date and time of a particular version or revision. The default date format is *mmm/dd/yy*; *mmm* is the month, *dd* the day, and *yy* the year. The time format is *hh*[:*mm*[:*ss*]]; where *hh* is the hour, *mm* is the minute, and *ss* is the second.

Merging Revisions

If you are working with the parallel development model, after you GET a revision and make changes to it, you may need to MERGE those changes into the master archive. The master archive presumably contains changes made by others that your revision does

not include. If no other developer has changed the file, then you do not need to MERGE; you simply PUT the new revision in the master archive.

Note

Remember that in your revised file, you must refer to the revision with which you want your changes merged. You do this by adding a `$revision` entry in the file.

When you MERGE a revision, VCS creates a backup of your revision in your private project directory. In addition, VCS provides all possible support for performing MERGE operations safely.

To merge your changes into another revision in the archive, choose **Merge** from the Project window's **VCS** menu. The **Merge** dialog box opens.

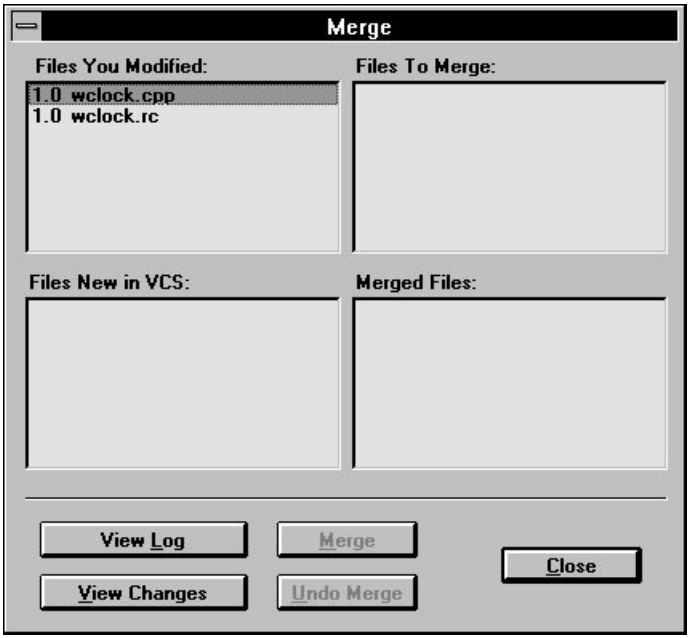

Figure 22-5 Merge dialog box

The **Merge** dialog box contains four listboxes:

Files you modified: Shows the files you modified in your private project directory since the last time you performed a GET operation.

Files new in VCS: Shows archives that someone else has changed since the last time you performed a GET operation. The current (most recent) revision number is displayed. These are files you might need to GET to rebuild the project with your changes and to PUT your new revisions back into the archive.

Files to merge: Shows files that both you and another person changed since the last time you performed a GET operation. These are the workfiles you need to MERGE.

Merged files: Shows files that you merged during the current MERGE session.

To display a log showing who changed an archive and what changes were made at each revision, click on a file name in one of the listboxes and click on the View Log button. See the PVCS documentation for information on the display format.

Click the View Changes button to show the changes you need to resolve when you MERGE the selected revision. When you click View Changes, an annotated change file appears in an IDDE editor window. See the PVCS documentation for information on the format of the display and the meanings of symbols.

When you are ready to perform the MERGE operation, click on Merge. VCS guides you through a three-way merge, incorporating your changes and those made by others into a new revision.

Because the IDDE automatically makes backup copies of both the revision in your private project directory and the revision you want to MERGE it with, you can undo a MERGE (before closing the **Merge** dialog box) by clicking Undo Merge.

Testing the MERGE operation

To test the MERGE, return to the **Get** dialog box, click on Advanced, and select Except Merges. GET the files in this list and build them using the IDDE (see Chapter 8, “Testing an Application,” for information).

Using the VCS Manager

Use the VCS Manager for additional flexibility in managing revisions and versions. You can apply a version name to a set of archives when the software reaches a significant release step, such as beta or final release. You can also delete a named version or revision from an archive. Before deleting a version or revision, make sure that you are deleting the right one, as a deleted version or revision cannot be recovered. Finally, the VCS manager lets you lock or unlock a version or revision.

To use the VCS Manager in the Project window, choose **Manager** from the **VCS** menu.

The **VCS Manager** dialog box shown in Figure 22-6 is displayed.

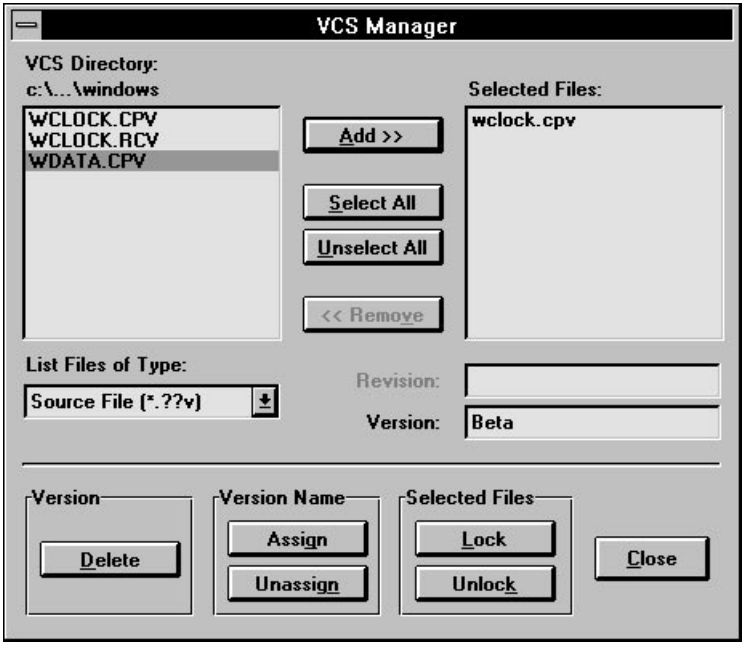

Figure 22-6 VCS Manager dialog box

Note

You probably will not need to use the **VCS Manager** dialog box on most projects.

The VCS Directory listbox contains the names of revisions to add to a version or revision. Double-click on the file name to add it to the Selected Files listbox. Click on Select All to select all files.

To assign a version name, type the name, such as `beta`, in the Version textbox, and click on the Assign button. This assigns the version name to all files in Selected Files. The Assign and Unassign buttons are available only if you specify a version name in the Version textbox.

To control access to a particular version or revision, specify the revision (its number) or version (its name) in the appropriate textbox, then click on Lock to deny access or Unlock to allow access.

To delete a version or revision, specify the appropriate information in the Revision or Version textbox, and click on the Delete button.

Creating a Master Archive

When you begin using VCS, it may be your job to create the master archive to and from which others GET and PUT files. This section describes the process of creating a master archive.

The master archive contains the revisions to all source files in a project. It also records information about changes from the previous revision, a history of the changes since the first revision, a record of who changed the files, the comments attached to the revisions as they were changed, and the date and time that each revision was PUT into the archive. See the PVCS documentation for additional information.

To set up a master archive, you PUT the workfiles in your project directory in the archives using the **Put** command. This command takes files from the original directory and creates the archives automatically. See “Using the VCS Manager,” earlier in this chapter, for more information on the **Put** dialog box and how to use it.

When you PUT files in an archive, VCS changes their extensions automatically. The last character of the extension on each file is changed to `V`. For example, the file extension on `abc.cpp` becomes `abc.cpv`. If the file does not have an extension, VCS adds the extension `.__v` to the filename (two underscores and a `v`).

To create a master archive from the workfiles in your private project directory:

1. Choose **Put** from the **VCS** menu in the Project window.

2. PUT the workfiles from your private project directory into the archive by double-clicking on the file in the Put Candidate Files listbox.

   The Put Candidate Files listbox displays the names of workfiles that you changed since the last time you performed a GET operation. The listbox also shows the files that do not yet have archives. Because you are creating the archives, all files in the project are displayed. Files you select are added to the Files To Put Back listbox.

3. To add all your workfiles to the master archive, click on Select All. To remove a workfile from the list of files to archive, select the file in the Files To Put Back list and click on the Remove button.

4. Optional: Type a comment in the Comments textbox. VCS applies your comment to all selected files in the Files to Put Back listbox.

5. Click the Put button to create the master archives and PUT your workfiles in them.